

Trio Motion Technology
Motion Coordinator MC464
Technical Reference Manual

Seventh Edition • 2010
Revision 2

All goods supplied by Trio are subject to Trio's standard terms and conditions of sale. This manual applies to systems based on the *Motion Coordinator MC464* with system software 2.0087 or higher.

The material in this manual is subject to change without notice. Despite every effort, in a manual of this scope errors and omissions may occur. Therefore Trio cannot be held responsible for any malfunctions or loss of data as a result.

Revision 1 December 2010

Copyright (C) 2000-2010 Trio *Motion* Technology Ltd.
All Rights Reserved

UK

Trio *Motion* Technology Ltd.
Shannon Way
Tewkesbury
GL20 8ND
United Kingdom
Phone: +44 (0)1684 292333
Fax: +44 (0)1684 297929

USA

Trio *Motion* Technology LLC.
1000 Gamma Drive, Suite 206
Pittsburgh
PA 15238,
USA
Phone: +1 412 968 9744
Fax: +1 412 968 9746

CHINA

Trio Shanghai
Tomson Centre
B1602, 188 Zhang Yang Road,
Pudong New Area, Shanghai,
200122,
CHINA
Tel: +86 21 5879 7659
Fax: +86 21 5879 4289

Contents

INTRODUCTION TO THE MC464	1-3
Features	1-5
The Trio Motion Technology Website.....	1-6
HARDWARE OVERVIEW.....	2-3
<i>Motion Coordinator</i> MC464	2-3
Connections to the MC464.....	2-5
Battery.....	2-9
Backlit Display	2-10
MC464 Feature Summary.....	2-11
INSTALLATION OF THE MC464.....	3-3
Packaging.....	3-3
Mounting.....	3-5
EMC considerations	3-7
Background to EMC Directive.....	3-9
Installation Requirements to Ensure EMC Conformance	3-10
MODULE ASSEMBLY	4-3
Fitting Expansion Modules	4-4
RTEX Interface (P871)	4-5
SERCOS II Interface (P872)	4-7
SLM Interface (P873).....	4-9
FlexAxis Interface (P874 / P879).....	4-11
Anybus-CC Module (P875)	4-14
EtherCAT Interface (P876)	4-16
INPUT / OUTPUT MODULES	5-3
General Description	5-3
CAN 16-I/O Module (P316).....	5-4
CAN 16-Output Module (P317)	5-5
CAN 16-Input Module (P318)	5-6
Alternative connection protocols	5-10
Software Interfacing P316, P317	5-13
Troubleshooting- P316, P317.....	5-13
Specification P316:.....	5-14
Specification P317.....	5-14
Specification P318.....	5-15
CAN Analogue I/O Module (P326)	5-16
Software Interfacing P326	5-19
Troubleshooting- P326	5-19
Specification P326	5-19
SYSTEM SETUP AND DIAGNOSTICS.....	6-3
Preliminary Concepts	6-3
System Setup	6-3
Preliminary checks.....	6-3
Checking Communications and System Configuration	6-4

Setting Servo Gains	6-7
Diagnostic Checklists	6-12
WHAT IS A PROGRAM?	7-3
Controlling the Sequence of Events	7-3
Controller Functions	7-6
Parameters	7-9
Command Line Interface	7-12
Example Programs	7-14
TRIOBASIC COMMANDS	8-3
<i>Motion</i> and Axis Commands	8-13
Input / Output Commands	8-109
Program Loops and Structures	8-141
System Parameters and Commands	8-155
Mathematical Operations and Commands	8-278
Constants	8-307
Axis Parameters	8-310
SUPPORT SOFTWARE.....	9-3
<i>Motion Perfect 2</i>	9-3
System Requirements	9-4
Connecting <i>Motion Perfect</i> to a controller.....	9-4
Running <i>Motion Perfect 2</i> for the First time	9-5
<i>Motion Perfect 2</i> Projects	9-5
Project Check Window	9-6
The <i>Motion Perfect</i> Desktop	9-9
Main Menu.....	9-10
Controller Menu	9-11
Controller Configuration	9-13
CAN I/O Status.....	9-14
Ethernet Configuration	9-14
Feature Enable	9-16
Memory Card Support	9-17
Loading New System Software	9-19
<i>Motion Perfect</i> Tools	9-24
Terminal	9-25
Axis Parameters	9-29
Oscilloscope	9-31
Keypad Emulation.....	9-40
Table / VR Editor.....	9-42
Jog Axes	9-42
Digital IO Status	9-45
Analogue Input Viewer	9-47
Linking to External Tools.....	9-47
Control Panel	9-48
The <i>Motion Perfect</i> Editor	9-52
Editor Menus	9-56
Program Debugger	9-58
Variable Watch Tool	9-60
Running Programs	9-61
Making programs run automatically	9-62

Storing Programs in the Flash EPROM	9-63
Configuring The <i>Motion</i> Perfect 2 Desktop	9-63
Communications.....	9-64
Editor Options	9-66
General Options	9-66
CAN Drive Options	9-67
Diagnostics	9-68
Terminal Font.....	9-68
Program Compare.....	9-68
CX-Drive Configuration	9-69
FINS Configuration	9-69
Saving the Desktop Layout	9-69
Running <i>Motion</i> Perfect 2 Without a Controller	9-70
MC Simulation	9-70
Limitations of MC Simulation.....	9-71
Project Encryptor	9-72
Introduction.....	9-72
Encryption Process.....	9-72
Encrypting a Project.....	9-72
CAD2Motion	9-76
Introduction.....	9-76
Main Screen.....	9-76
Sequence Manipulation Tools	9-79
Import Options	9-79
Preparing A Drawing For CAD2Motion.....	9-81
DocMaker	9-82
AUTOLOADER AND MCLOADER ACTIVEX.....	10-3
Project Autoloader.....	10-3
Using the Autoloader	10-3
Script Commands.....	10-7
Script File	10-15
MC Loader	10-16
Installation of the MC Loader Component.....	10-16
Events	10-18
Methods.....	10-23
TrioPC Motion ActiveX Control	11-3
Requirements.....	11-3
Installation of the ActiveX Component	11-3
Using the Component	11-3
Connection Commands	11-4
Properties	11-7
Motion Commands	11-10
Process Control Commands.....	11-20
Variable Commands.....	11-21
Input / Output Commands.....	11-28
General commands	11-35
Events	11-38
Intelligent Drive Commands	11-40
Program Manipulation Commands	11-41
Data Types	11-43

TrioPC status.....	11-44
INTRODUCTION TO MODBUS	12-3
Modbus RTU.....	12-3
Modbus TCP	12-4
Modbus Technical Reference	12-6
DeviceNet	12-9
DeviceNet Objects Implemented	12-10
Identity Object	12-10
DeviceNet Object	12-11
Assembly Object	12-12
Connection Object.....	12-13
MC Object.....	12-16
Ethernet	12-19
The Subnet Mask.....	12-20
Anybus	12-25
Anybus Configuration	12-26
REFERENCE	13-3
Communications Ports.....	13-3
Error Codes.....	13-3
Data Formats and Floating-Point Operations	13-8
Product Codes	13-9
INDEX.....	III

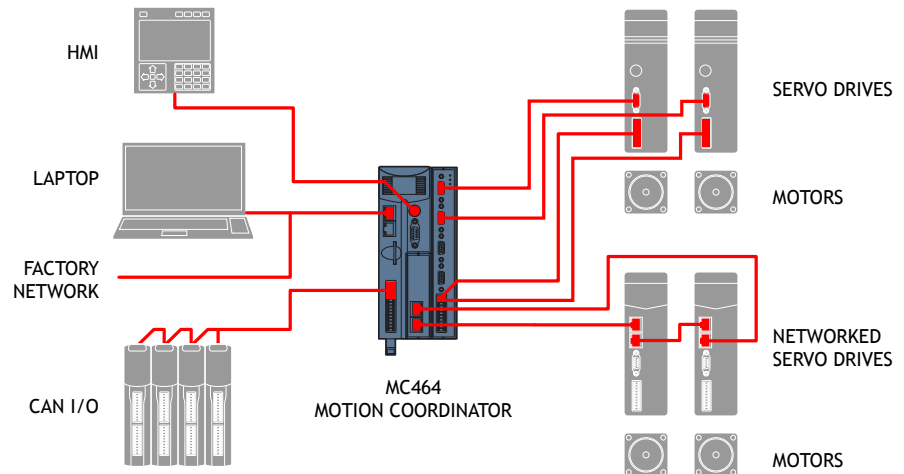
CHAPTER
INTRODUCTION

1

Introduction to the MC464

The MC464 represents a quantum leap in motion control technology. Run your machine faster with this new generation *Motion Coordinator* based on a 64 bit MIPS processor.

Choose the motor and drives to best suit your application without compromise, MC464 provides interface options for traditional servo, stepper and piezo control together with many digital interfaces for current digital servo drives. Increase the flexibility of your equipment with support for up to 64 axes of motion control. Trio's tradition of modular configuration has evolved into convenient clip-on modules allowing the system designer to precisely build the configuration needed for the job.



Typical System Configuration

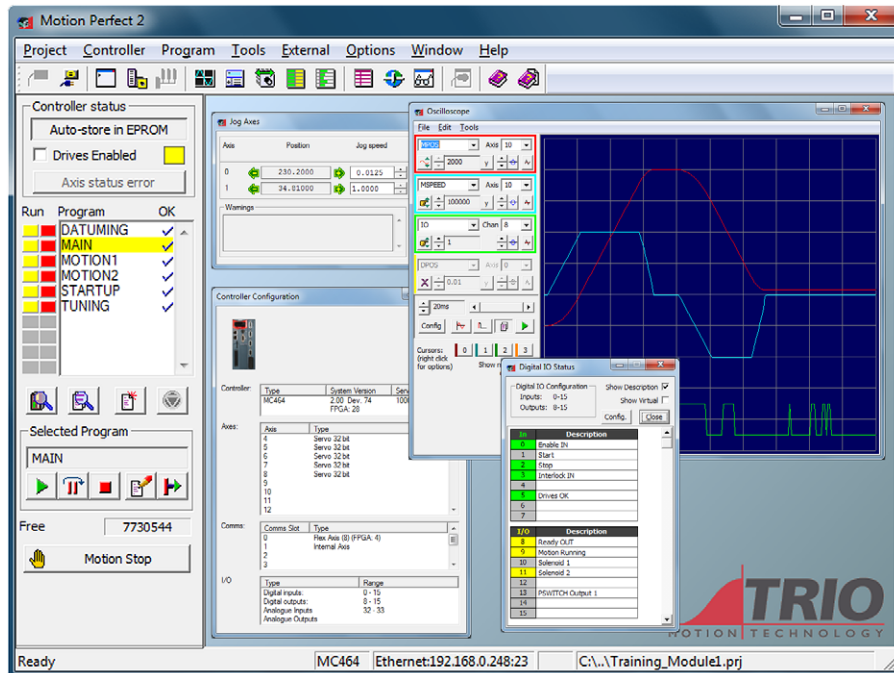
The MC464 supports programs written in TrioBASIC, allowing a smooth upgrade path from earlier series of *Motion Coordinator*. In addition; G-Code, HPGL and the standard IEC 61131-3 languages are supported, with full operation of the IEC 61131-3 language requiring a software license. I/O expansion is provided via a built-in CANbus interface. Further fieldbus networks supporting common factory protocols are supported via the HMS AnyBus® adapter module.

The axis expansion modules feature many options for Drive Network interfaces, analogue servo, pulse/direction, absolute or incremental feedback and accurate hardware registration.

Up to 7 half-height expansion modules or 3 full-height expansion modules can be attached. This modular approach along with Trio's feature enable code system for axis activation allows the whole system to be scaled exactly to need.

Setup and Programming

To program the *Motion Coordinator* a PC is connected via an Ethernet link. The dedicated *Motion Perfect 2* program is normally used to provide a wide range of programming facilities, on a PC running Microsoft Windows XP, Vista or Windows 7 32bit versions.



Motion Perfect 2

Once connected to the *Motion Coordinator*, the user has direct access to TrioBASIC, which provides an easy, rapid way to develop control programs. All the standard program constructs are provided; variables, loops, input/output, maths and conditions. Extensions to this basic instruction set exist to permit a wide variety of motion control facilities, such as single axis moves, synchronised multi axis moves and unsynchronised multi axis moves as well as the control of the digital I/O.

The MC464 controller features multi-tasking TrioBASIC and the standard IEC 61131-3 language. Multiple TrioBASIC programs plus Instruction List (IL), Ladder Diagram (LD), Function Block (FB), Structured Text (ST) and Sequential Function Chart (SFC) can be constructed and run simultaneously to make programming complex applications much easier.

κW-Software's "Multiprog" programming suite is available separately in order to access the full IEC 61131-3 functionality. Multiprog provides a seamless programming, compilation and debug environment that can work in real-time with

the MC464. A motion library is provided which enables the familiar **Trio Motion Technology** commands to be included in IEC 61131-3 programs.

Features

- Supports digital drive systems up to 64 axis
- Based on 64bit 400MHz MIPS processor
- Anybus Module support allowing flexible factory communication options
- 64bit position integers
- High accuracy double floating point resolution
- Multi-tasking BASIC programming
- Backlit LCD display
- Ethernet programming interface
- Expansion flexibility with clip on modules allowing quick interchangeability
- IEC61131-3 programming support
- “Disable zones” for networked drives
- Bi-directional reference encoder port
- I/O expansion up to 272 I/O points

The Trio *Motion* Technology Website

The Trio website contains up to the minute news, information and support for the *Motion Coordinator* product range.



Thursday, March 26th, 2009

Home | Trio News | About Trio | Contact Trio

Product - MC464

Information & Downloads printer friendly

Overview... P860

The *Motion Coordinator MC464* represents a quantum leap in motion control technology. Run your machine faster with this new generation *Motion Coordinator* based on the 64 bit MIPS processor.

Choose the motor and drives to best suit your application without compromise, MC464 provides interface options for traditional servo, stepper and piezo control together with many digital interfaces for current digital servo drives. Increase the flexibility of your equipment with support for up to 64 axes of motion control. Trio's tradition of modular configuration has evolved into convenient clip-on modules allowing the system designer to precisely build the configuration needed for the job.

Redesigned from the ground up the attention to detail is evident. The bright easy to read backlight display makes sure that controller status is easily determined, whilst the single piece metal casting provides an integrated earth chassis to improve noise rejection in the industrial environment.

The MC464 supports programs written in **Trio BASIC**, allowing a smooth upgrade path from earlier series of *Motion Coordinator*. In addition, GCODE, HPGL and the standard IEC 61131 languages are supported, with full operation of the IEC 61131 language requiring a software license. I/O expansion is provided via a built-in CAN Bus interface. Further fieldbus networks supporting common factory protocols are supported via the HMS AnyBus® adapter module.

Every axis can be programmed to move using linear, circular or helical interpolation, electronic cams and gearboxes. Features include support for merging multiple moves that are typically generated by CAD/CAM software and support is provided for continuously rotating machinery.

64 AXES

- Website Features
- Latest News
- Product Information
- Manuals
- Support Software
- System Software Updates
- Technical Support
- User's Forum
- Application Examples
- Employment Opportunities

WWW.TRIOMOTION.COM

CHAPTER
HARDWARE OVERVIEW

2

Hardware Overview

Motion Coordinator MC464

Overview

The *Motion Coordinator MC464* is Trio's new generation modular servo control positioner with the ability to control servo or stepper motors by means of Digital Drive links (e.g. EtherCAT, SERCOS, etc) or via traditional analogue encoder or pulse and direction. A maximum of 7 expansion modules can be fitted to control up to 64 axes which gives the flexibility required in modern system design. The MC464 is housed in a rugged plastic case with integrated earth chassis and incorporates all the isolation circuitry necessary for direct connection to external equipment in an industrial environment. Filtered power supplies are included so that it can be powered from the 24V d.c. logic supply present in most industrial cabinets.

It is designed to be configured and programmed for the application using a PC running the *Motion Perfect 2* application software, and then may be set to run "standalone" if an external computer is not required for the final system.

The Multi-tasking version of TrioBASIC for the MC464 allows up to 20 TrioBASIC programs to be run simultaneously on the controller using pre-emptive multi-tasking. In addition, the operating system software includes a full IEC 61131-3 standard run-time environment (licence key required).

Programming

The Multi-tasking ability of the MC464 allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware. IEC 61131-3 programs can be run at the same time as TrioBASIC allowing the programmer to select the best features of each.



I/O Capability

The MC464 has 8 built in 24V inputs and 8 bi-directional I/O channels. These may be used for system interaction or may be defined to be used by the controller for end of travel limits, registration, datuming and feedhold functions if required. Each of the Input/Output channels has a status indicator to make it easy to check them at a glance. The MC464 can have up to 256 external Input/Output channels connected using DIN rail mounted CAN I/O modules. These units connect to the built-in CAN channel.

Communications

A 10/100 base-T Ethernet port is fitted as standard and this is the primary communications connection to the MC464. Many protocols are supported including Telnet, Modbus TCP, Ethernet IP and TrioPCMotion. Check the Trio website (www.triomotion.com) for a complete list.

The MC464 has one built in RS232 port and one built in duplex RS485 channel for simple factory communication systems. Either the RS232 port or the RS485 port may be configured to run the Modbus or Hostlink protocol for PLC or HMI interfacing.

If the built-in CAN channel is not used for connecting I/O modules, it may optionally be used for CAN communications. e.g. DeviceNet, CANopen etc.

The Anybus CompactCom Carrier Module (P875) can be used to add other fieldbus communications options

Removable Storage

The MC464 has a SD Card slot which allows a simple means of transferring programs, firmware and data without a PC connection. Offering the OEM easy machine replication and servicing.

The memory slot is compatible with a wide range of SD cards up to 2Gbytes using the FAT32 file system.

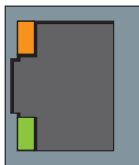
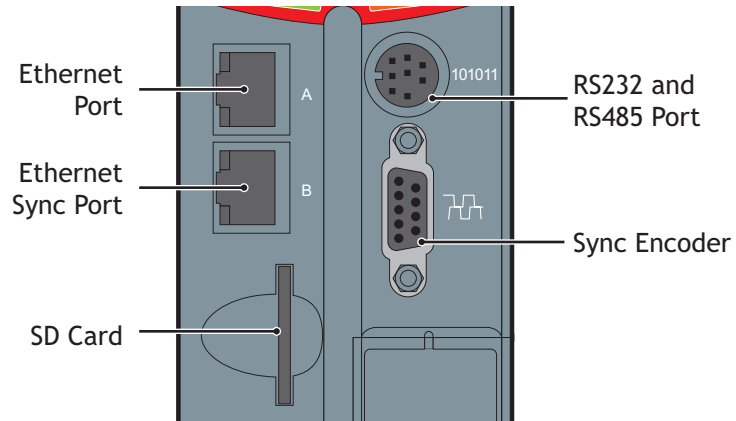


Axis Positioning Functions

The motion control generation software receives instructions to move an axis or axes from the TrioBASIC or IEC 61131-3 language which is running concurrently on the same processor. The motion generation software provides control during operation to ensure smooth, coordinated movements with the velocity profiled as specified by the controlling program. Linear interpolation may be performed on groups of axes, and circular, helical or spherical interpolation in any two/three orthogonal axes. Each axis may run independently or they may be linked in any combination using interpolation, CAM profile or the electronic gearbox facilities.

Consecutive movements may be merged to produce continuous path motion and the user may program the motion using programmable units of measurement (e.g. mm, inches, revs etc.). The module may also be programmed to control only the axis speed. The positioner checks the status of end of travel limit switches which can be used to cancel moves in progress and alter program execution.

Connections to the MC464



Ethernet Port Connection

Physical layer: 10/100 base_T

Connector: RJ45

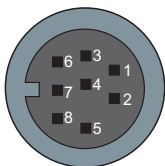
The Ethernet port is the default connection between the *Motion Coordinator* and the host PC running *Motion Perfect 2* programming.

Ethernet Sync Port

This second Ethernet port is provided for inter-connection between *Motion Coordinators* for system and/or motion synchronisation.

MC464 Serial Connections

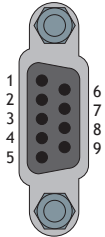
The MC464 features two serial ports. Both ports are accessed through a single 8 pin connector.



Serial Connector

Pin	Function	Note
1	RS485 Data In A Rx+	Serial Port #2
2	RS485 Data In B Rx-	
3	RS232 Transmit	Serial Port #1
4	0V Serial	
5	RS232 Receive	Serial Port #1

Pin	Function	Note
6	Internal 5V	5V supply is limited to 150mA, shared with sync port
7	RS485 Data Out Z Tx-	Serial Port #2
8	RS485 Data Out Y Tx+	Serial Port #2



Sync Encoder

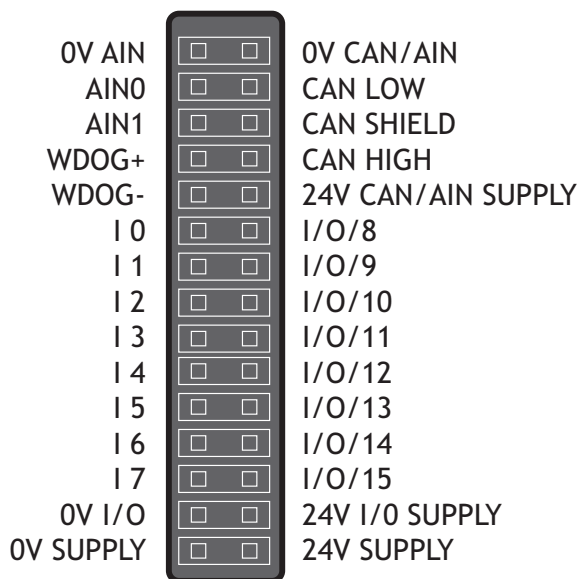
The sync encoder port is bidirectional. It can be used as a reference encoder input or as an encoder simulation output to act as a master reference for other parts of the system.

Pin	Function	Pulse & Direction
1	Enc. A	Step+
2	Enc. /A	Step-
3	Enc. B	Direction+
4	Enc. /B	Direction-
5	0V Encoder	0V Stepper
6	Enc. Z	Enable+
7	Enc. /Z	Enable-
8	5V *	5V*
9	5V Registration input	5V Registration input
*5V supply is limited to 150mA (shared with serial port)		

Registration

The MC464 built in port has 2 available registration events. These can be used with the Z mark, the registration input on the sync port, input 0 or input 1.

24V Power Supply Input



The MC464 is powered entirely via the 24v d.c. supply connections. The unit uses internal DC-DC converters to generate independent 5V logic supply, the encoder/serial 5V supply and other internal power supplies. I/O, analogue and CANbus circuits are isolated from the main 24V power input and must be powered separately. For example; it is often necessary to power the CANbus network remotely via the CANbus cable.



*24V d.c., Class 2 transformer or power source required for **UL** compliance. The MC464 is grounded via the metal chassis. It **MUST** be installed on an unpainted metal plate or DIN rail which is connected to earth.*

Amplifier Enable (Watchdog) Relay Outputs

One internal relay contact is available to enable external amplifiers when the controller has powered up correctly and the system and application software is ready. The amplifier enable is a solid-state relay with an ON resistance of 25 ohms at 100mA. The enable relay will be open circuit if there is no power on the controller **OR** a motion error exists on a servo axis **OR** the user program sets it open with the WDOG=OFF command.

The amplifier enable relay may, for example, be incorporated within a hold-up circuit or chain that must be intact before a 3-phase power input is made live.



ALL STEPPER AND SERVO AMPLIFIERS MUST BE INHIBITED WHEN THE AMPLIFIER ENABLE OUTPUT IS OPEN CIRCUIT

CANbus

The MC464 features a built-in CAN channel. This is primarily intended for Input/Output expansion via Trio's range of CAN digital and analogue I/O modules. It may be used for other purposes when I/O expansion is not required.



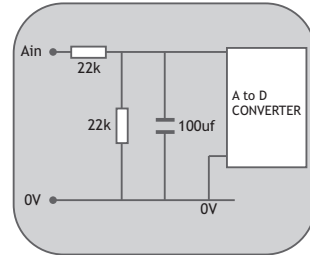
0V CAN/AIN
CAN LOW
CAN SHIELD
CAN HIGH
24V CAN/AIN SUPPLY

The CANbus port is electrically equivalent to a DeviceNet node.

Analogue Inputs

Two built-in 12 bit analogue inputs are provided which are set up with a scale of 0 to 10V. External connection to these inputs is via the 2-part terminal strip on the lower front panel.

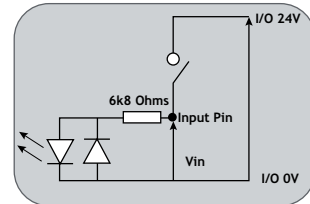
A 24V d.c. supply must be applied to the CANbus port to provide power for the analogue input circuit.



24V Input Channels

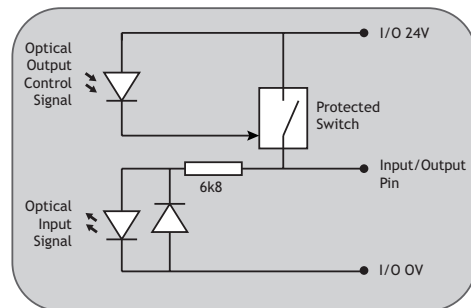
The *Motion Coordinator* has 16 24V Input channels built into the master unit. These may be expanded to 256 Inputs by the addition of CAN-16 I/O modules.

All of the 24V Input channels have the same circuit although 8 on the master unit have 24V Output channels connected to the same pin. These bi-directional channels may be used for Input or Output to suit the application. If the channel is to be used as an Input then the Output should not be switched on in the program.



24V I/O Channels

Input/output channels 8..15 are bi-directional and may be used for Input or Output to suit the application. The inputs have a protected 24V sourcing output connected to the same pin. If the channel is to be used as an Input then the Output should not be switched on in the program. The input circuitry is the same as on the dedicated inputs. The output circuit has electronic over-current protection



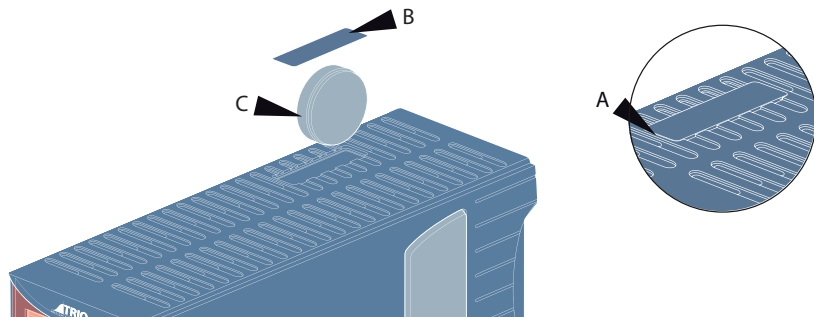
and thermal protection which shuts the output down when the current exceeds 250mA.

Care should be taken to ensure that the 250mA limit for the output circuit is not exceeded, and that the total load for the group of 8 outputs does not exceed 1A

Battery

The MC464 incorporates a user replaceable battery for the battery back-up RAM. For replacement, use battery model CR2450 or equivalent.

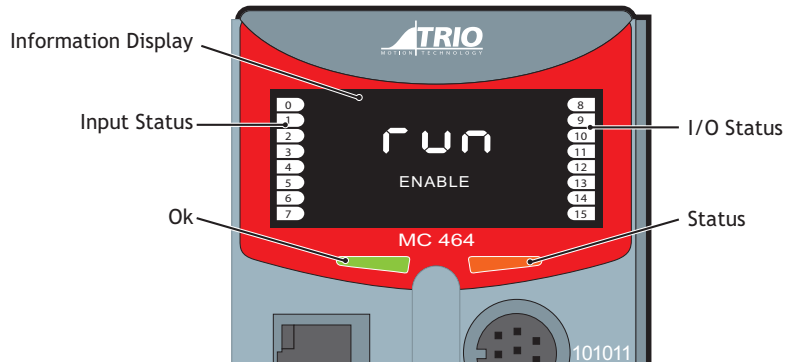
To replace the battery, insert screwdriver under the frontmost ventilation slot (A) and prize off the battery cover (B) and pull the battery ribbon to lift the battery (C) from the MC464. Replacing is the reverse of the procedure.



TO AVOID LOSING THE MEMORY CONTENTS, THE NEW BATTERY SHOULD BE INSERTED WITHIN 30 SECONDS OF THE OLD ONE BEING REMOVED.

Backlit Display

The information display area shows the IP address and subnet mask during power-up and whenever an Ethernet cable is first connected to the MC464. During operation, this display shows run, Off or Err to indicate the MC464 status. Below the main status display are the ERROR, ENABLE and BATTERY LOW indicators.



- ERROR** An error has occurred (see Error Display Codes table below for details).
- ENABLE** When illuminated, WDOG is ON.
- BATTERY LOW** When illuminated the battery needs replacing.

A bank of 8 indicators at the left side shows the Digital Input States and a similar bank on the right shows the state of I/O8 to I/O15. The I/O displayed can be altered using the **DISPLAY** command.

Two LED's are provided to show the processor (OK) and system status.

Error Display Codes		
Unn	Unit error on slot nn	
Ann	Axis error on axis nn	
Caa	Configuration error on unit nn	ie: too many axes
Exx	System error	E00 - RAM error 8bit BB - RAM (VR) E01 - RAM error 16 bit BB - RAM (TABLE) E02 - Not used E03 - Battery Error

MC464 Feature Summary

Size	201 mm x 56 mm x 155 mm (HxWxD).
Weight	750g
Operating Temp.	0 - 45 degrees. C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	RS232 channel: up to 38400 baud. RS485 channel: up to 38400 baud. CANbus port (DeviceNet and CANopen compatible) Ethernet: 10/100 BaseT multiple port connection.
Position Resolution	64 bit position count
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Servo Cycle	125µs minimum, 1ms default, 2ms max.
Programming	Multi-tasking TrioBASIC system, maximum 20 user processes. IEC 61131-3 programming system.
Interpolation modes	Linear 1-64 axes, circular, helical, spherical, CAM Profiles, speed control, electronic gearboxes.
Memory	8 Mbyte user memory. 2 Mbyte TABLE battery-backed memory. Automatic flash EPROM program storage.
Table	512,000 table positions. 196,608 positions in battery backed memory.
VR	65,536 VR positions in battery backed memory.
SD Card	Standard SD Card compatible to 2Gbytes. Used for storing programs and/or data.
Power Input	24V d.c., Class 2 transformer or power source. 18..29V d.c. at 625mA typical.
Amplifier Enable Output	Normally open solid-state relay rated 24V ac/dc nominal. Maximum load 100mA. Maximum voltage 29V.
Analogue Inputs	2 isolated x 12 bit 0 to 10V.
Serial / Encoder Power Output	5V at 150mA.
Digital Inputs	8 Opto-isolated 24V inputs.
Digital I/O	8 Opto-isolated 24V outputs. Current sourcing (PNP) 250 mA. (max. 1A per bank of 8).

CHAPTER
INSTALLATION

3

Installation of the MC464

Packaging

The *Motion Coordinator MC464* is designed to be mounted on a DIN rail or, by use of optional mounting clips, it can be screwed to a backplate.

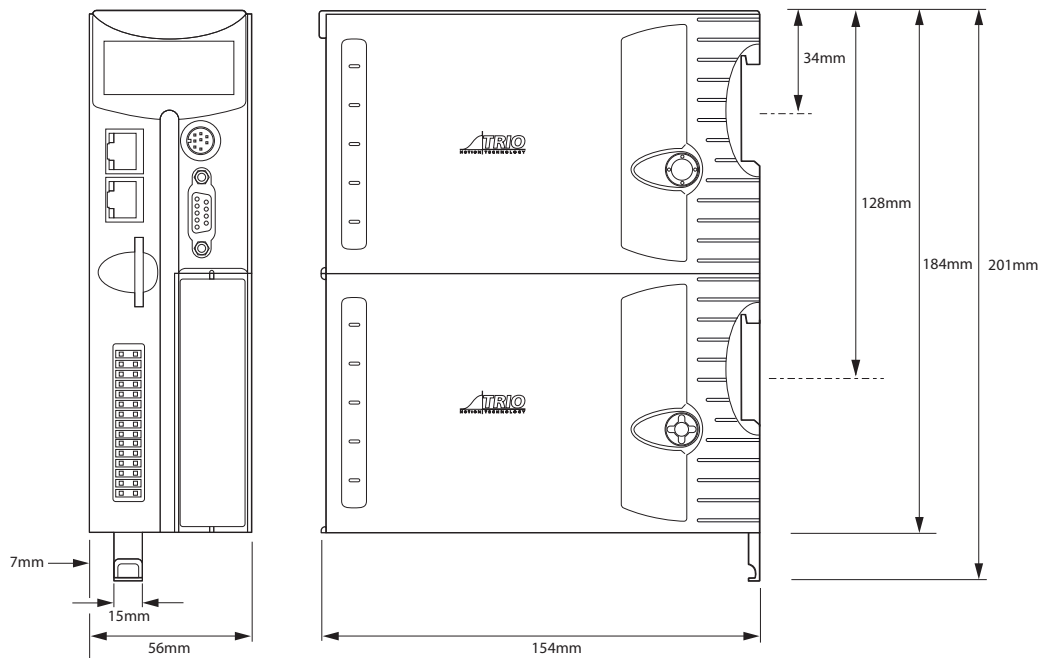
A cast metal chassis provides mechanical stability and a reliable earth connection to aid EMC immunity.

The rugged plastic case includes ventilation holes, top and bottom, and a removable cover to access the memory battery.

Expandable design

System expansion is done by adding either single or double height modules. These are clipped to the MC464 and secured by a bolt which also acts as the earth connection between the MC464 and the module

The dimensions are as shown below.



Items supplied with the MC464

Connectors:

- 9 way D-Type plug
- Quick connect I/O connector (30 way)

Panel mounting set:

- 2 x Mounting bracket
- 1 x M3 x 10mm Countersunk screw
- 1 x M3 x 6mm Countersunk screw
- Quick start guide
- CD ROM with software and documentation

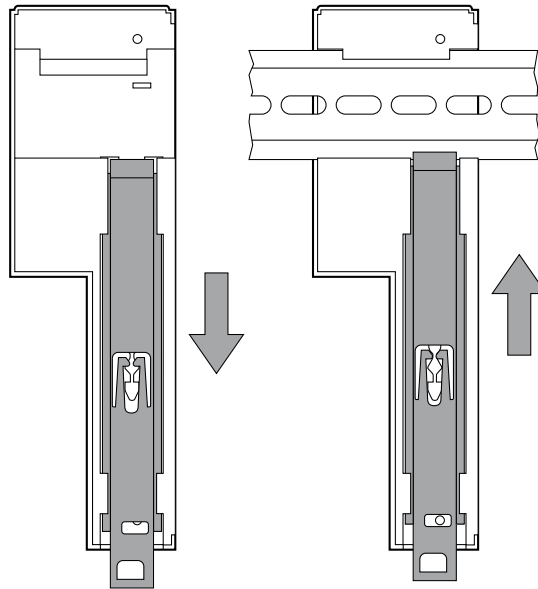
Mounting

General

The MC464 must be mounted vertically and should not be subjected to mechanical loading. Care must be taken to ensure that there is a free flow of air vertically around the MC464.

DIN Rail

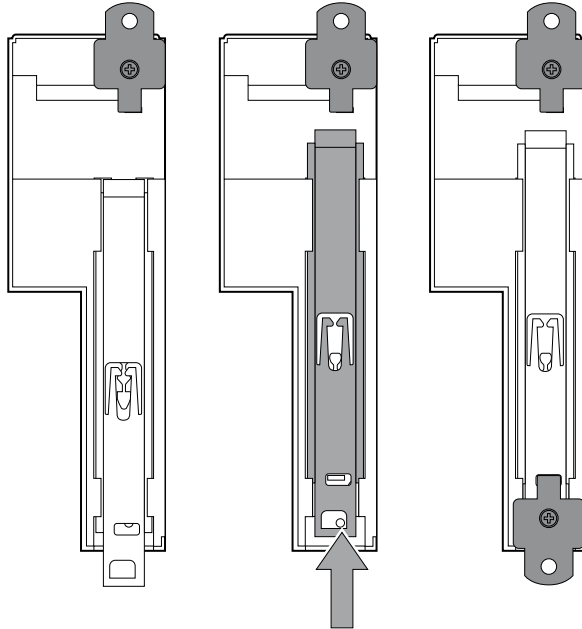
Pull down the clip to allow the MC464 to be mounted on a single DIN rail. Push up the clip to lock it to the rail.



Mounting Clips

Remove the 2 mounting clips from their packaging and insert one at the top rear of the case, by fitting the small tab into the rectangular slot and fix with the M3 x 6mm screw provided.

The second clip fits to the bottom of the case rear. Line up the DIN rail lever with the hole and slot in the metal chassis, fit the clip into the slot and fix it with the M3 x 10mm screw.



Environmental Considerations

The MC464 should not be handled whilst the 24 Volt power is connected.



ENSURE THAT THE AREA AROUND THE VENTILATION HOLES AT THE TOP AND BOTTOM OF THE MC464 AND ANY ADDITIONAL MODULES ARE KEPT CLEAR. AVOID VIOLENT SHOCKS TO, OF VIBRATION OF, THE MC464, SYSTEM AND MODULES WHILST IN USE OR STORAGE.

IP rating: IP 20

The MC464 and add-on modules are protected against solid objects intruding into the case and against humidity levels that do not induce condensation to occur.

EMC considerations

Most pieces of electrical equipment will emit noise either by radiated emissions or conducted emissions along the connecting wires. This noise can cause interference with other equipment near-by which could lead to that equipment malfunctioning. These sort of problems can usually be avoided by careful wiring and following a few basic rules.

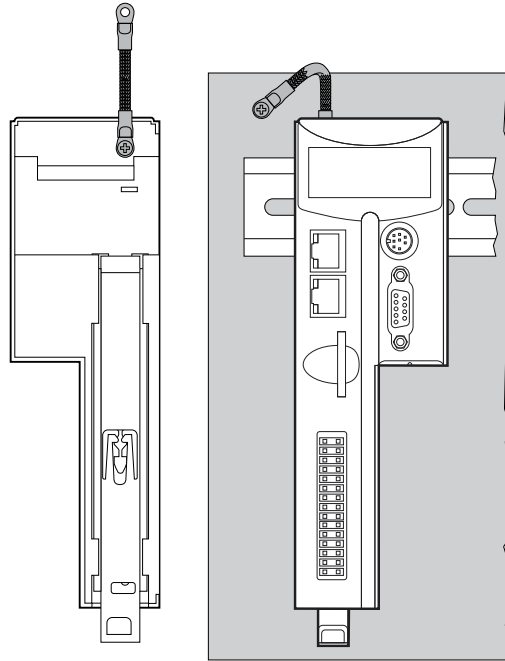
- Mount noise generators such as contactors, solenoid coils and relays as far away as possible from the MC464.
- Where possible use solid-state contactors and relays.
- Fit suppressors across coils and contacts.
- Route heavy current power and motor cables away from signal and data cables.
- Ensure all the modules have a secure earth connection.
- Where screened cables are used terminate the screen with a 360 degree termination rather than a “pig-tail”. Connect both ends of the screen to earth. The screening should be continuous, even where the cable passes through a cabinet wall or connector.

These are just very general guidelines and for more specific advice on specific controllers, see the installation requirements later in this chapter. The consideration of EMC implications is more important than ever since the introduction of the EC EMC directive which makes it a legal requirement for the supplier of a product to the end customer to ensure that it does not cause interference with other equipment and that it is not itself susceptible to interference from other equipment.

EMC Earth

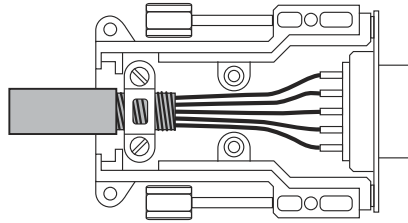
Best EMC performance is obtained when the MC464 is attached to an earthed, unpainted metal panel using the two mounting clips. When screwed directly to the panel, the clips provide the required EMC earth connection.

If the MC464 is mounted on a DIN rail, then an additional EMC earth must be attached as shown below. Use a flat braided conductor, minimum cross-section 4mm x 1mm. Connect to the earthed metal panel as close to the MC464 as possible. Do not use circular cross-section wire. Do not run the conductor to a central star point.



Cable Shields

All encoder cables must be terminated in the correct D-type plug, either 9 way or 15 way as required. For best EMC performance use a metal or metalised plastic cover for the D-type connector. Clamp the screen of the encoder cable where it enters the connector cover. Do not make a “pig-tail” connection from the screen to the plug cover. When plugging the D-type into the MC464, use the jack-screws to firmly attach the D-type plug to the socket on the MC464 or its axis module.

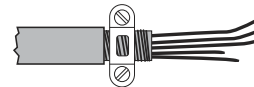


Both ends of the encoder cable's screen must be connected using a 360 degree contact and not a pig-tail connection.

The 0V must be connected separately from the screen. Make sure that encoder cables are specified with one extra wire to carry the 0V.

All serial cables must be terminated in an 8-pin mini-DIN connector. For best EMC performance, clamp the screen of the serial cable where it enters the connector cover. Do not make a “pig-tail” connection from the screen to the pig cover.

SCREEN CONNECTED INTERNALLY
TO METAL SHIELD



SCREEN CLAMPED TO EARTH



Both ends of the serial cable's screen must be connected using a 360 degree contact and not a pig-tail connection.

The 0V must be connected separately from the screen. Make sure that serial cables are specified with one extra wire to carry the 0V. This applies to RS422/RS485 serial connections as well as RS232.

Background to EMC Directive

Since 1st January 1996 all suppliers of electrical equipment to end users must ensure that their product complies with the 89/336/EEC Electromagnetic Compatibility directive. The essential protection requirements of this directive are:

Equipment must be constructed to ensure that any electromagnetic disturbance it generates allows radio and telecommunications equipment and other apparatus to function as intended.

Equipment must be constructed with an inherent level of immunity to externally generated electromagnetic disturbances.

Suppliers of equipment that falls within the scope of this directive must show “due diligence” in ensuring compliance. Trio has achieved this by having products that it considers to be within the scope of the directive tested at an independent test house.

As products comply with the general protection requirements of the directive they can be marked with the CE mark to show compliance with this and any other relevant directives. At the time of writing this manual the only applicable directive is the EMC directive. The low voltage directive (LVD) which took effect from 1st January 1997 does not apply to current Trio products as they are all powered from 24V which is below the voltage range that the LVD applies to.

Just because a system is made up of CE marked products does not necessarily mean that the completed system is compliant. The components in the system must be connected together as specified by the manufacturer and even then it is possible for some interaction between different components to cause problems but obviously it is a step in the right direction if all components are CE marked.

Testing Standards

For the purposes of testing, a typical system configuration was chosen because of the modular nature of the *Motion Coordinator* products. Full details of this and copies of test certificates can be supplied by Trio if required.

For each typical system configuration testing was carried out to the following standards:

Emissions - BS EN61000-6-4 : 2007.

The MC464 products conform to the Class A limits.

Immunity - BS EN61000-6-2 : 2005.

This standard sets limits for immunity in an industrial environment and is a far more rigorous test than part 1 of the standard.

Installation Requirements to Ensure EMC Conformance



WHEN THE TRIO PRODUCTS ARE TESTED THEY ARE WIRED IN A TYPICAL SYSTEM CONFIGURATION. THE WIRING PRACTICES USED IN THIS TEST SYSTEM MUST BE FOLLOWED TO ENSURE THE TRIO PRODUCTS ARE COMPLIANT WITHIN THE COMPLETED SYSTEM.

A summary of the guidelines follows:

- The MC464 modules must be earthed via the main chassis of the MC464 using the lower panel mounting clip or an earth strap. This must be done even if DIN rail mounted.
- If any I/O lines are not to be used they should be left unconnected rather than being taken to a terminal block, for example, as lengths of unterminated cable hanging from an I/O port can act as an antenna for noise.

- Screened cables **MUST** be used for encoder, stepper and registration input feedback signals and for the demand voltage from the controller to the servo amplifier if relevant. The demand voltage wiring must be less than 1m long and preferably as short as possible. The screen must be connected to earth at both ends. Termination of the screen should be made in a 360 degree connection to a metallised connector shell. If the connection is to a screw terminal e.g. demand voltage or registration input the screen can be terminated with a short pig-tail to earth.
- Ethernet cables should be shielded and as a minimum, meet the TIA Cat 5e requirements.
- Connection to the serial ports should be made with a Trio supplied cable.

As well as following these guidelines, any installation instructions for other products in the system must be observed.

CHAPTER
MC464 EXPANSION MODULES

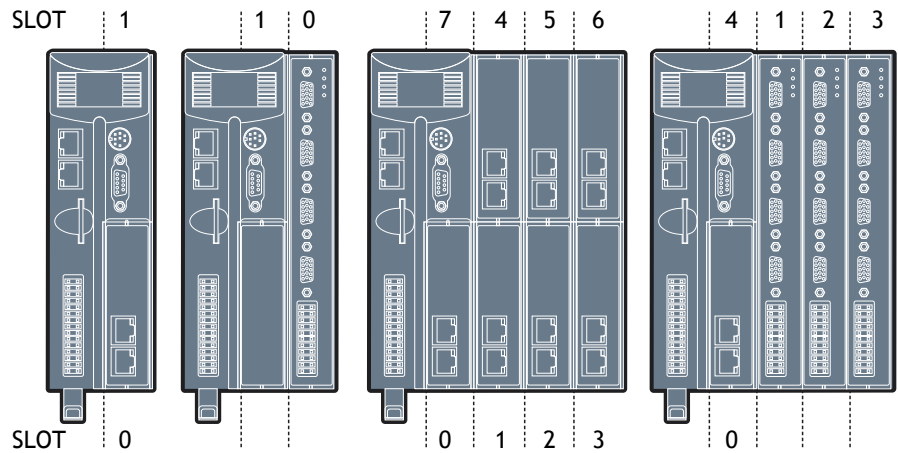
4

Module Assembly

A maximum of 7 half height modules or 3 full height modules may be fitted to the MC464. A system may be made using any combination of half and full height modules providing that the full height modules are the last to be attached.

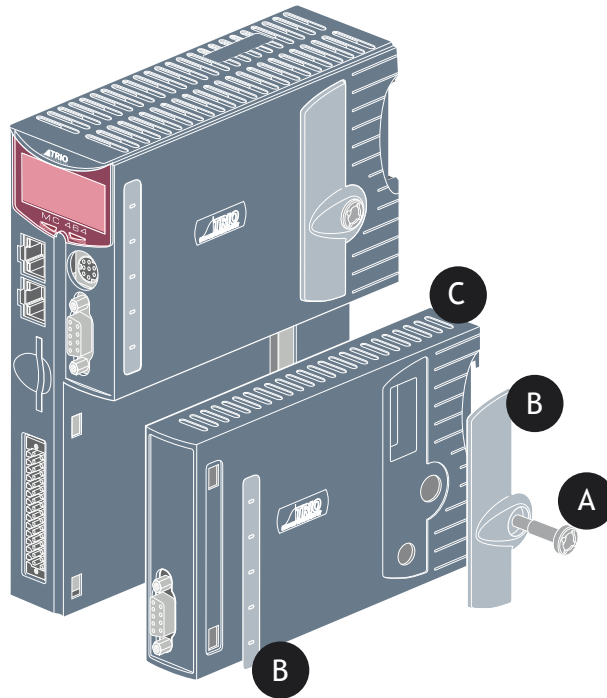
Module SLOT Numbers

SLOT Numbers are allocated by the system software in order, left to right, starting with the lower bus. Lower modules are allocated slots 0 to m, then the upper modules become slots m+1 to n. Finally, the Sync Encoder Port is allocated slot n+1. The Sync Encoder Port has SLOT number -1 in addition to the one allocated (1) in this sequence.



Fitting Expansion Modules

- Remove the 2 covers (B) if fitted to the MC464 or to the previous expansion module (C).
- Locate the 2 hooks at the front of the module, while holding the rear out at an angle
- Push forward to engage the hooks and at the same time swing the rear of the module in so as to locate the connector.
- Press the connector “home” once it is located.
- Tighten the screw (A) using the tool provided or a small coin
- Clip the provided covers (B) in place as shown.



Removing modules is the reversal of the above procedure.

If the system is to be panel mounted, a kit (P8) comprising 2 x panel mounting brackets and 2 x countersunk screws may be purchased separately from your Trio distributor.

RTEX Interface (P871)

For use with Panasonic amplifiers supporting the Panasonic Real Time Express (RTEX) network. Allows Plug & Play interconnection with Shielded twisted pair (TIA/EIA-568B CAT5e or more) Ethernet cables.

A single interface supports up to 32 axes on the RTEX network. The module comes with 2 axes enabled. Further axes can be enabled with Trio's Feature Enable Codes.



Realtime Express

The P871 communicates with up to 32 servo amplifiers using Ethernet Real Time Express. The physical layer is standard Ethernet connected in a ring. Each node has a transmit socket and a receive socket to allow easy connection. The maximum cable length between any 2 nodes is 60 meters and the overall network length is limited to 200 meters.



RJ45 Connector (tx)

(Top connector) 100Mbps Panasonic RTEX transmit - connect to receive of first drive.



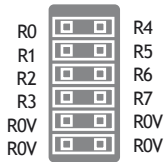
RJ45 Connector(Rx)

(Bottom connector) 100Mbps Panasonic RTEX receive - connect to transmit of last drive.

Time Based Registration

Time based registration uses a 10MHz clock to record the time of a registration event which is then referenced to time stamps on the axis position from the digital drive network. An accurate registration position is then calculated. The 10MHz clock gives a time resolution of 100nsec. The position and speed of the axis are recorded so that the user can compensate for any fixed delays in the registration circuit.

Any time based registration input can be assigned to any Digital or Virtual axis. This makes the registration very flexible and enables multiple registration channels per axis. Each registration channel can be armed independently and assigned to an axis at any time.



Registration connector

R0-R7 registration inputs (24V).

OV common 0V return.

Registration inputs can be allocated to any axis by software.

LED Functions

LED	LED colour	LED function
ok	Green	ON=Module Initialised Okay
0	Red	ON=Module Error
1	Yellow	Status 1
2	Yellow	Status 2

SERCOS II Interface (P872)

The SERCOS interface module is designed to control up to 16 servo amplifiers using the standard SERCOS II fibre-optic ring. Benefits of this system include full isolation from the amplifiers and greatly reduced wiring.



For use with any SERCOS II IEC61491 compliant drive. The module allows control of up to 16 axes via SERCOS with cycle times down to 250usec. Multiple SERCOS interface modules can be used to increase axes count to 64.

2, 4, 8 and 16 Mbit / sec

Software settable intensity

SERCOS Connections

SERCOS is connected by 1mm polymer or glass fibre optic cable terminated with 9mm FSMA connectors. The SERCOS ring is completed by connecting TX to RX in a series loop. The maximum fibre cable length between 2 nodes is 40m for plastic optical fibre (POF) and 200m for hard clad silica (HCS). The total length for POF is 680m and 3,400 for HCS.



Connector (Rx)

(Top connector) SERCOS fibre-optic transmit. 9mm FSMA.



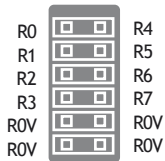
Connector (Tx)

(Bottom connector) SERCOS fibre-optic receive. 9mm FSMA.

Time Based Registration

Time based registration uses a 10MHz clock to record the time of a registration event which is then referenced to time stamps on the axis position from the digital drive network. An accurate registration position is then calculated. The 10MHz clock gives a time resolution of 100nsec. The position and speed of the axis are recorded so that the user can compensate for any fixed delays in the registration circuit.

Any time based registration input can be assigned to any Digital or Virtual axis. This makes the registration very flexible and enables multiple registration channels per axis. Each registration channel can be armed independently and assigned to an axis on the fly.



Registration connector

R0 - R7 registration inputs (24V).

R0V registration common 0V return.

Registration inputs can be allocated to any axis by software.

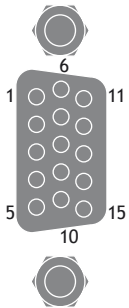
LED Functions

LED	LED colour	LED function
ok	Green	ON=Module Initialised Okay
0	Red	ON=Module Error
1	Yellow	Status 1
2	Yellow	Status 2

SERCOS phase	LED 1	LED 2
0	OFF	FLASH
1	OFF	ON
2	FLASH	OFF 1
3	ON	OFF 2
4	ON	ON

SLM Interface (P873)

For use with drives supporting the Control Techniques SLM protocol. Each module supports 6 axes which can be individual drives or two drives using the CT Multi-ax concept.



SLM Connector

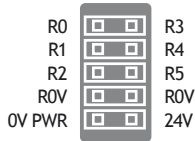
Pin	Upper D-Type	Lower D-Type
1	Com Axis 0	Com Axis 3
2	/Com Axis 0	/Com Axis 3
3	Hardware Enable	Hardware Enable
4	0V Output	0V Output
5	24V Output	24V Output
6	Com Axis 1	Com Axis 4
7	/Com Axis 1	/Com Axis 4
8	No Connection	No Connection
9	No Connection	No Connection
10	No Connection	No Connection

Pin	Upper D-Type	Lower D-Type
11	24V Output	24V Output
12	0V Output	0V Output
13	Com Axis 2	Com Axis 5
14	/Com Axis 2	/Com Axis 5
15	Earth / Shield	Earth / Shield

Time Based Registration

Time based registration uses a 10MHz clock to record the time of a registration event which is then referenced to time stamps on the axis position from the digital drive network. An accurate registration position is then calculated. The 10MHz clock gives a time resolution of 100nsec. The position and speed of the axis are recorded so that the user can compensate for any fixed delays in the registration circuit.

Any time based registration input can be assigned to any Digital or Virtual axis. This makes the registration very flexible and enables multiple registration channels per axis. Each registration channel can be armed independently and assigned to an axis on the fly.



Registration Connector

- R0 - R5 registration inputs (24V).
- 0VR common 0V return.
- 0V PWR Power input for SLM system.
- 24V Power input for SLM system.

LED Functions

LED	LED Colour	LED Function
ok	Green	ON = Module initialised ok
0	Red	ON = Module error
1	Yellow	Status 1
2	Yellow	Status 2

FlexAxis Interface (P874 / P879)

For use with Stepper, Analogue Servo & Piezo motors. The FlexAxis Interface is available in 4 axes (P879) and 8 axes (P874) versions.

Each axis provides a 16 bit analogue output, up to 8 x 24Vdc high speed registration inputs and a 6MHz encoder input. The encoder port can be configured to drive a stepper motor or an encoder simulation port, both at 2MHz.



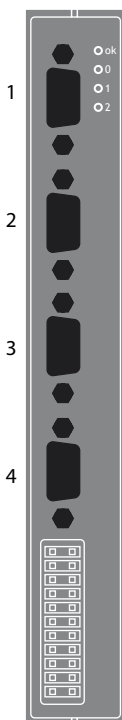
Encoder Connector

Pin	Incremental Encoder	Pulse + Direction	Absolute Encoder
1	Enc. A n	Step+ n	Clock+ n
2	Enc. /A n	Step- n	Clock- n
3	Enc. B n	Direction+ n	n/c
4	Enc. /B n	Direction- n	n/c
5	0V Enc	0V Enc	0V Enc
6	Enc. Z n	Enable+ n	Data+ n
7	Enc. /Z n	Enable- n	Data- n
8	5V*	5V*	5V*
9	Enc A n+4	Step+ n4	Clock+ n+4
10	Enc /A n+4	Step- n4	Clock- n+4
11	Enc B n+4	Direction+ n+4	n/c
12	Enc /B n+4	Direction- n+4	n/c
13	Enc Z n+4	Enable+ n+4	Data+ n+4
14	Enc /Z n+4	Enable- n+4	Data- n+4
15	0V Enc	0V Enc	0V Enc
*5V supply is limited to 150mA per axis.			



Absolute encoder is only available on axes 4-7 on the P874 and on axes 2-3 on P879.

Connector	8 Axes (P874)	4 Axes (P879)
1	0 and 4	0
2	1 and 5	1
3	2 and 6	2
4	3 and 7	3



Multifunction Connector

The 22 pin multifunction connector provides terminals for 8 registration inputs, 8 voltage outputs and 4 hardware PSWITCH outputs.

Analogue Outputs

8 +/-10V 16Bit analogue outputs are available for servo axis control (4 in the P879). Connect V0 as the velocity command signal for the first axis, V1 for the second axis and so on. The maximum load per axis together is 10mA.

Position Based Registration

Position based registration uses the encoder signal. When the registration event occurs the encoder position is latched in hardware. The speed of the axis is also recorded so that the user can compensate for any fixed electronic delays in the registration circuit. Flexible allocation of registration inputs to axes is provided. Each axis can have a number of registration events assigned to it and the source of these events can be from any of the registration channels.

The Flex Axis module has 8 registration inputs in addition to the Z mark for each axis. The first axis has 8 registration events which can be assigned to use any of the registration inputs or its own Z mark. The remaining axes have 2 registration events which can be assigned to use any of the registration inputs or their own Z mark.

PSWITCH Outputs

Inputs R4 to R7 are bi-directional and can be used as outputs for high accuracy PSWITCH operation. When used in this mode, the outputs are controlled by the position value of an axis within the same P874 / P879 module.

DAC 0V		DAC 0V
DAC 0V		DAC 0V
V0		V4
V1		V5
V2		V6
V3		V7
R0		R4/PS4
R1		R5/PS5
R2		R6/PS6
R3		R7/PS7
0V PWR		24V

Multifunction Connector Pin Out

0V	DAC common 0V return
V0 - V7	Voltage outputs
R0 - R3	24V Registration Inputs
R4/PS4 - R7/PS7	Bidirectional 24V registration In/24V: PSWITCH outputs
Inputs / 24V	PSwitch outputs
0V PWR	Power Input
24V	Power Input



4 axis version uses voltage outputs V0 - V3 only.

LED Functions

LED	LED Colour	LED Function
ok	Green	ON = Module initialised ok

LED	LED Colour	LED Function
0	Red	ON = Module error
1	Yellow	Status 1
2	Yellow	Status 2

Anybus-CC Module (P875)

Open communications is an important aspect to any control system. This module adds support for the Anybus CompactCom device modules.

Anybus-CC is a plug-in module supporting all major Fieldbus and Ethernet networks. Its innovative design and versatile functionality offers the Anybus-CC optimal flexibility for OEM manufacturers.

The Anybus modules can be found at:

www.anybus.com



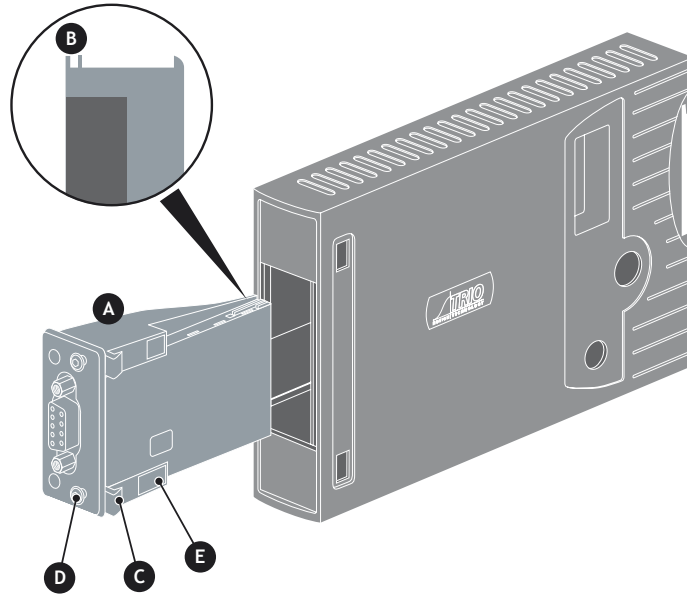
Anybus CompactCom Module shown for illustration only. Anybus CC Modules may be purchased separately.

Anybus Module Fitting

Push the Anybus® module (A) into the Trio Expansion Interface taking care to keep its base in contact with the PCB and align guide slots (B) with the connector rails inside.

Ensure that the moulded hooks (C) on the lower front edge of the Anybus® module locate under the P875 PCB at the front.

When the module is flush with the face of the Trio Expansion Interface, tighten the two “Torx” head screws (D) to locate the two lugs (E) and secure the Anybus® module.



To remove the module, reverse this procedure.

EtherCAT Interface (P876)

For use with EtherCAT compliant drives, this module allows control of up to 64 axes via standard shielded twisted pair (TIA/EIA-568B CAT5e or more) Ethernet cables. Multiple EtherCAT Interface Modules can be used.



EtherCAT is an open, high performance ethernet based fieldbus system, which has been integrated into several IEC standards (IEC 61158, IEC 61784 and IEC61800). It is a high performance, deterministic protocol, with high bandwidth usage, low latency and low communication jitter. Various network topologies are supported, including line, tree or star. The EtherCAT compliant servo amplifiers from any number of vendors may be included in a network.

The module supports both the CANopen and servo drive (SERCOS, IEC 61491) EtherCAT profiles, along with the mailbox transfer protocol to exchange configuration, status and diagnostic information between the master and slave.



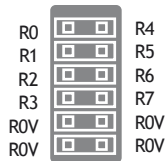
RJ45 Connector

100 base-T Ethernet (EtherCat Master).

Time Based Registration

Time based registration uses a 10MHz clock to record the time of a registration event which is then referenced to time stamps on the axis position from the digital drive network. An accurate registration position is then calculated. The 10MHz clock gives a time resolution of 100nsec. The position and speed of the axis are recorded so that the user can compensate for any fixed delays in the registration circuit.

Any time based registration input can be assigned to any Digital or Virtual axis. This makes the registration very flexible and enables multiple registration channels per axis. Each registration channel can be armed independently and assigned to an axis on the fly.



Registration Connector

R0 - R7: registration inputs (24V).

ROV: registration common 0V return.

Registration inputs can be allocated to any axis by software.

LED Functions

LED	LED colour	LED function
ok	Green	ON=Module Initialised Okay
0	Red	ON=Module Error
1	Yellow	Status 1
2	Yellow	Status 2

CHAPTER
I/O EXPANSION MODULES

5

Input / Output Modules

General Description

Trio can supply a range of Input/Output Modules.

The Motion Coordinator controllers allow for I/O expansion by having a CAN interface. This allows the I/O modules to form a network up to 100m in length.

Operator interface units can communicate with controllers using the serial RS232/RS485 ports or the Ethernet port. Third party operator interface units may connect using either the built-in Modbus protocol or a serial protocol written in BASIC.

Product Code:

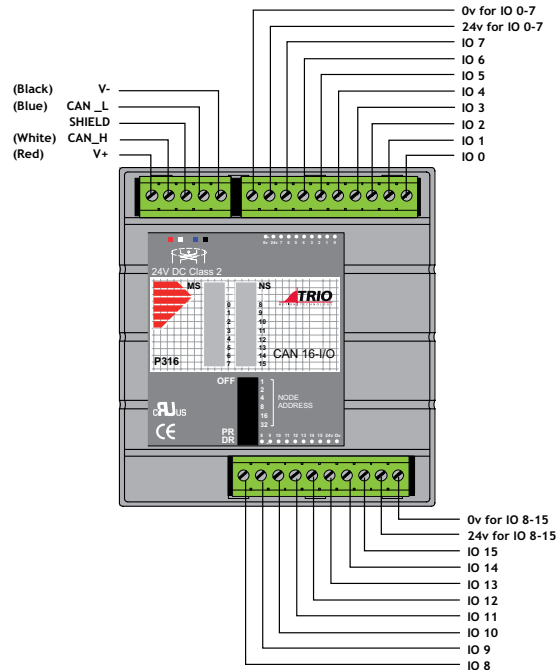
CAN 16-I/O Module	P316
CAN 16-Output Module	P317
CAN 16-Input Module	P318
CAN Analogue I/O Module	P326

CAN 16-I/O Module (P316)

The CAN 16-I/O Module allows the 24 Volt digital inputs and outputs of the *Motion Coordinator* to be expanded in blocks of 16 bi-directional channels.

Up to 16 CAN 16-I/O Modules may be connected allowing up to 256 I/O channels in addition to the internal channels built-in to the *Motion Coordinator*. Each of the 16 channels in each module is bi-directional and can be used either as an input OR as an output.

Convenient disconnect terminals are used for all I/O connections. The P216 CAN 16-I/O Module may also be used as an I/O expander for Lenze drives with an appropriate CAN interface.



I/O Connections

The CAN 16-I/O Module has 3 disconnect terminal connectors:

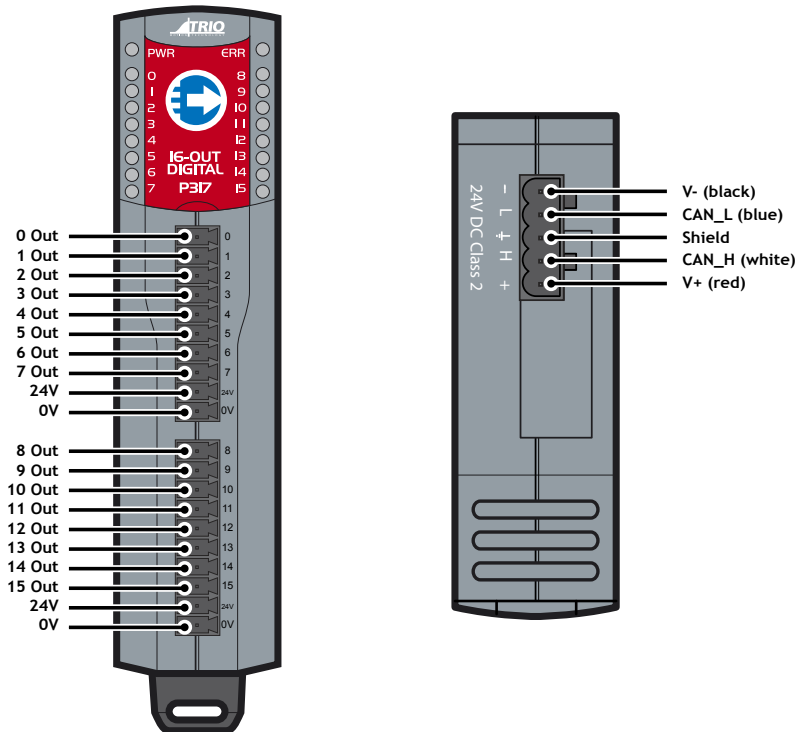
- DeviceNet physical format 5 way CAN connector
- Input/Output Bank 0 - 7 and power supply for bank 0 - 7 on 10 way connector
- Input/Output Bank 8 - 15 and power supply for bank 8 - 15 on 10 way connector.

CAN 16-Output Module (P317)

The CAN 16-Output Module allows the 24 Volt digital outputs of the *Motion Coordinator* to be expanded in blocks of 16 additional output channels.

Up to 16 CAN 16-Output Modules may be connected allowing up to 256 Input channels in addition to the internal channels built-in to the *Motion Coordinator*. CAN 16-Output modules may be mixed with CAN 16-Input and CAN 16-I/O modules on the same network to build the I/O configuration required for the system.

Convenient disconnect terminals are used for all I/O connections.



I/O Connections:

The CAN 16-Output Module has 3 disconnect terminal connectors:

- DeviceNet physical format 5 way CAN connector (on top)
- Output Bank 0 - 7 and power supply for bank 0 - 7 on 10 way connector
- Output Bank 8 - 15 and power supply for bank 8 - 15 on 10 way connector.

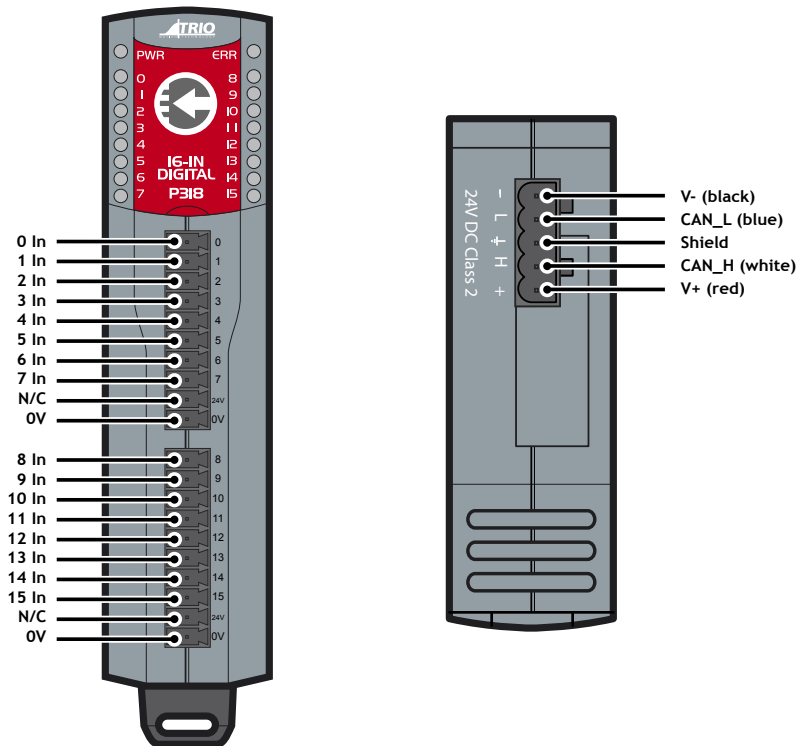
CAN 16-Input Module (P318)

The CAN 16-Input Module allows the 24 Volt digital inputs of the *Motion Coordinator* to be expanded in blocks of 16 additional input channels.

Up to 16 CAN 16-Input Modules may be connected allowing up to 256 Input channels in addition to the internal channels built-in to the *Motion Coordinator*.

CAN 16-Input modules may be mixed with CAN 16-Output and CAN 16-I/O modules on the same network to build the I/O configuration required for the system.

Convenient disconnect terminals are used for all I/O connections.



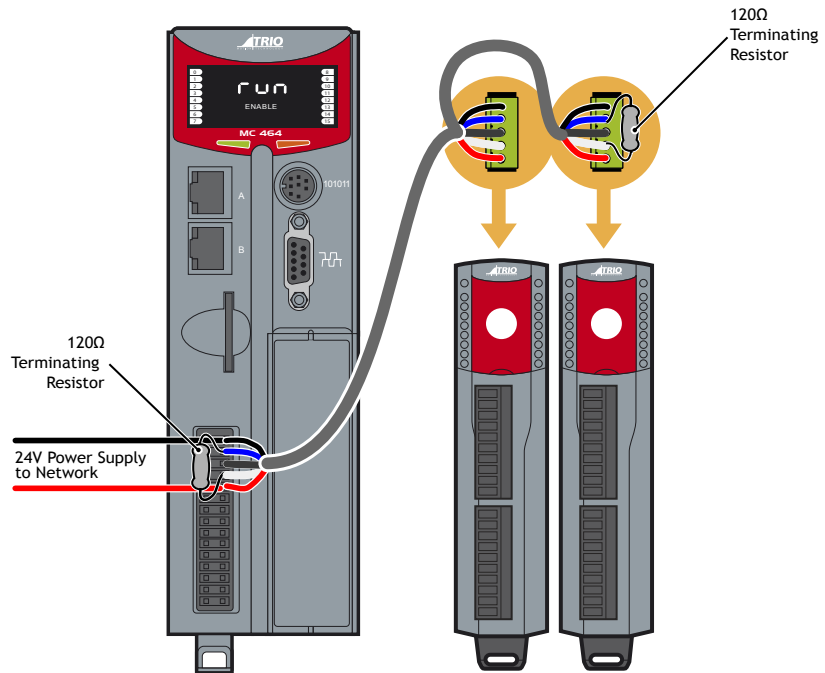
I/O Connections:

The CAN 16-Input Module has 3 disconnect terminal connectors:

- DeviceNet physical format 5 way CAN connector (on top)
- Input Bank 0 - 7 and power supply for bank 0 - 7 on 10 way connector
- Input Bank 8 - 15 and power supply for bank 8 - 15 on 10 way connector.

Bus Wiring

The CAN 16-I/O Modules and the *Motion Coordinator* are connected together on a CAN network running at 500kHz. The network is of a linear bus topology. That is the devices are daisy-chained together with spurs from the chain. The total length is allowed to be up to 100m, with drop lines or spurs of up to 6m in length. At both ends of the network, 120 Ohm terminating resistors are required between the CAN_H and CAN_L connections. The resistor should be 1/4 watt, 1% metal film.



The cable required consists of:

- Blue/White 24AWG data twisted pair
- + Red/Black 22AWG DC power twisted pair
- + Screen

A suitable type is Belden 3084A.

The CAN 16-I/O modules are powered from the network. The 24 Volts supply for the network must be externally connected. The *Motion Coordinator* does NOT provide the network power. In many installations the power supply for the *Motion Coordinator* will also provide the network power.

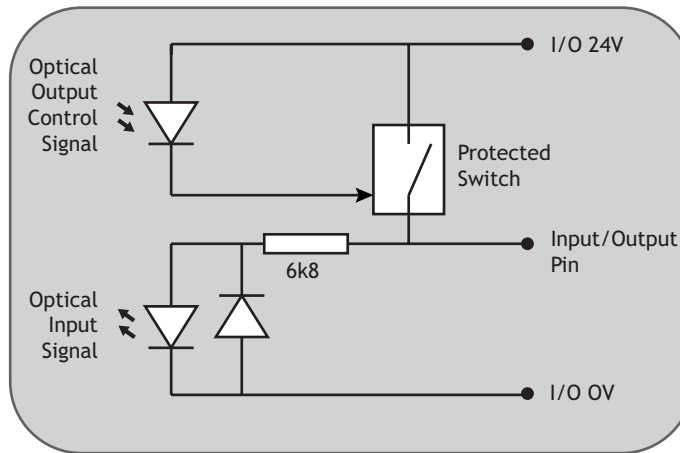


It is recommended that you use a separate power supply from that used to power the I/O to power the network as switching noise from the I/O devices may be carried into the network.

24V I/O Channels

Input/output channels can be bi-directional, input or output. Bi-directional inputs have a protected 24V sourcing output connected to the same pin. If the output is unused, the pin may be used as an input in the program. The output circuit has electronic over-current protection and thermal protection which shuts the output down when the current exceeds 250mA.

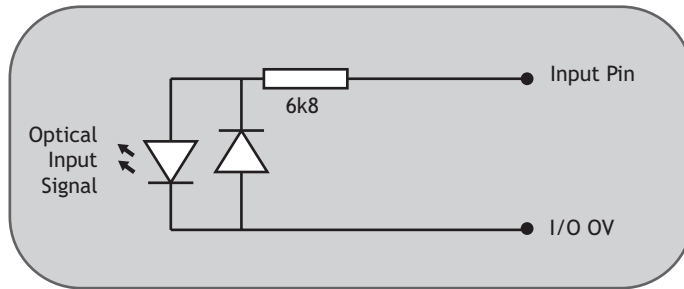
Care should be taken to ensure that the 250mA limit for the output circuit is not exceeded, and that the total load for the group of 8 outputs does not exceed 1 amp.



CAN16-I/O 24V I/O Channel

24V Input Channels

Input channels have an opto-isolated 24V input which is designed to be ON when the input voltage is greater than 18 Volts and OFF when the signal voltage is below 2V. The input has a 6k8 resistor in series and so provides a load of approximately 3.5mA at 24V.

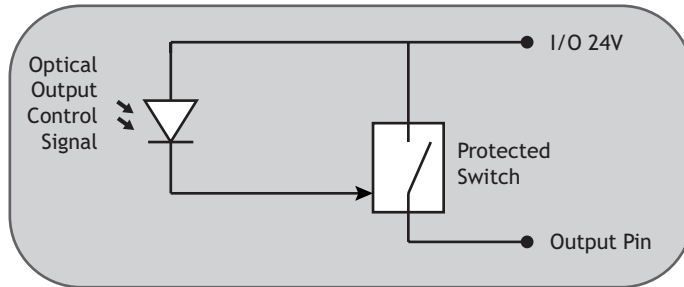


CAN16-Input 24V Input Channel

24V Output Channels

Output channels have a protected 24V sourcing output connected to the output pin. The output circuit has electronic over-current protection and thermal protection which shuts the output down when the current exceeds 250mA.

Care should be taken to ensure that the 250mA limit for the output circuit is not exceeded, and that the total load for the group of 8 outputs does not exceed 1 amp.



CAN16-Output 24V Output Channel

DIP Switch Settings

Address:	Start:	End:
0	16	31
1	32	47
2	48	63
3	64	79
4	80	95
5	96	111
6	112	127
7	128	143
8	144	159
9	160	175
10	176	191
11	192	207
12	208	223
13	224	239
14	240	255
15	256	271

Alternative connection protocols

The DIP switches can be set up to allow for different protocols to be used, enabling the P317 and P318 modules to be used with other manufacturer's devices. The DIP switch marked "PR" selects the protocol to be used. Switched right it selects the TRIO protocol, switched left it selects the module to act as a CANopen DS401 expansion I/O. (Not available on the P316).

TRIO Protocol:

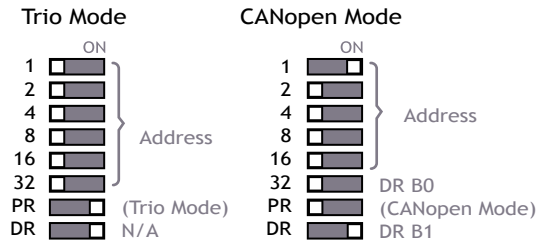
The switch marked PR is set ON to select the standard Trio protocol.


The top 5 DIP switches on the CAN 16-I/O set the module address. Only addresses 0 - 15 are valid for CAN 16-I/O modules.

The switch marked DR sets the CAN Bus communications rate. 500KHz must be selected when using Trio Protocol Mode.

Switch 32 selects the operating mode. Set ON for Trio Mode.

The addresses for I/O modules **MUST** be set in sequence, 0,1,2 etc. Therefore the first two CAN 16-I/O Modules would have switch settings as shown below:



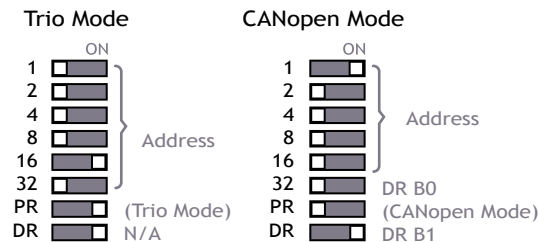
 The I/O Channels referred to above start at 16. This is because the numbering sequence starts with channels 0 - 15, which are on the Motion Coordinator master unit itself.

CANopen Protocol

The switch marked **PR** is set **OFF** to select the CANopen protocol.

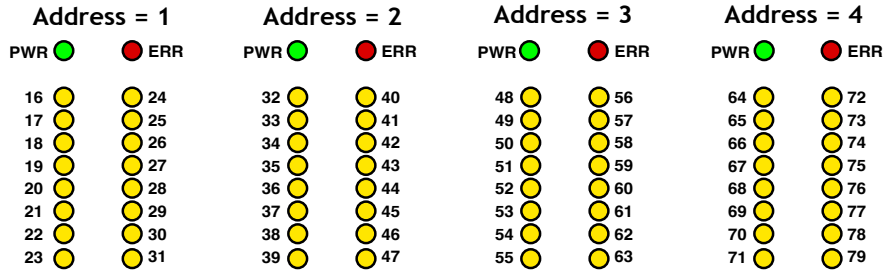
The top 6 **DIP** switches are used to set the node number. This should be set to a number 1..63.

The baud rate is selected by setting the switches marked 32 and **DR**. Four speeds are available.



LED Indicators

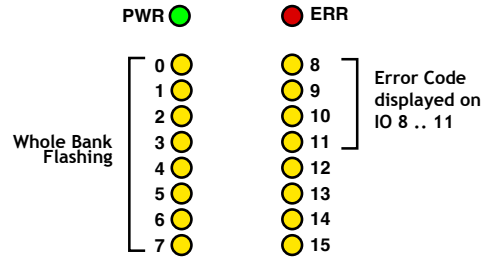
When **ERR** is **OFF** LEDs marked 0 - 15 represent the input or output channels 0 - 15 of the module. The actual input as seen by the *Motion Coordinator* software will depend on the I/O modules' address:



Error Codes:

When an error occurs on a CAN I/O module, the ERR LED will be lit and the fault code is represented by a binary number displayed on the leds.

Code	Error Description
1	Invalid Protocol
2	Invalid Module Address
3	Invalid Data Rate
4	Uninitialised
5	Duplicate Address
6	Start Pending
7	System Shutdown
8	Unknown Poll
9	Poll Not Implemented
10	CAN Error
11	Receive Data Timeout



Software Interfacing P316, P317

The *Motion Coordinator* will automatically detect and allow the use of correctly connected CAN I/O channels. The CAN I/O are accessed with the same **IN** and **OP** commands used to access the built-in I/O on the *Motion Coordinator*. The *Motion Coordinator* sets the system parameter **NIO** which reflects the number of I/O's connected to the system. 3 system parameters are available to facilitate the use of the CAN 16-I/O:

CANIO_STATUS, CANIO_ADDRESS and CANIO_ENABLE

When choosing which I/O devices should be connected to which channels the following points need to be considered:

- Inputs 0 - 63 ONLY are available for use with system parameters which specify an input, such as **FWD _ IN**, **REV _ IN**, **DATUM _ IN** etc.
- The built-in I/O channels have the fastest operation <1mS
- CAN I/O channels 16 - 63 have the next fastest operation up to 2mS
- CAN I/O channels 64 - 271 have the next fastest operation up to 16mS



It is not possible to mix the CAN 16-I/O modules which are running the TRIO I/O protocol with DeviceNet equipment or CANopen devices on the same network.

Troubleshooting- P316, P317

If the network configuration is incorrect 2 indications will be seen: The CAN 16-I/O module will indicate that it is uninitialised and the *Motion Coordinator* will report the wrong number when questioned:

>>? NIO

If this is not as expected check:

- Terminating 120 Ohm Network Resistors fitted?
- 24Volt Power to each IO bank required?
- 24Volt Power to Network?
- DIP switches in sequence starting 0,1,2...?
- Cable used is the correct CAN bus specification?
- *Motion Coordinator* CANIO _ ADDRESS=32?

Specification P316:

Inputs:	16 24 Volt input channels with 2500V isolation
Outputs:	16 24 Volt output channels with 2500V isolation
Configuration:	16 bi-directional channels
Output Capacity:	Outputs are rated at 250mA/channel. (1 Amp total/bank of 8 I/O's)
Protection:	Outputs are overcurrent and over temperature protected
Indicators:	Individual status LED's
Address Setting:	Via DIP switches
Power Supply:	24V dc, Class 2 transformer or power source. 18 ... 29V dc / 1.5W
Mounting:	DIN rail mount
Size:	95mm wide x 45mm deep x 105mm high
Weight:	200g
CAN:	500kHz, Up to 256 expansion I/O channels
EMC:	BSEN50082-2 (1995) Industrial Noise Immunity / BS EN55022 (2001) Industrial Noise Emissions
CAN protocol:	Trio CAN I/O or Lenze CAN.

Specification P317

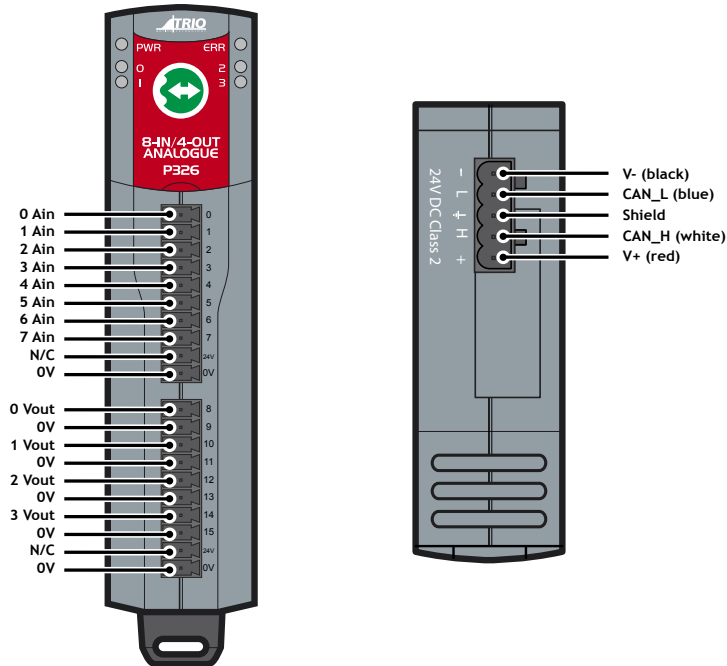
Inputs:	16 24 Volt input channels with 2500V isolation
Configuration:	16 input channels
Protection:	Inputs are reverse polarity protected
Indicators:	Individual status LED's
Address Setting:	Via DIP switches
Power Supply:	24V dc, Class 2 transformer or power source 18 ... 29V dc / 1.5W.
Mounting:	DIN rail mount
Size:	26mm wide 85mm deep 130mm high
Weight:	128g
CAN:	500kHz, Up to 256 expansion I/O channels
EMC:	EN 61000-6-2 : 2005 Industrial Noise Immunity / EN 61000-6-4 : 2007 Industrial Noise Emissions
CAN protocol:	Trio CAN I/O or CANopen DS401.

Specification P318

Outputs:	16 24 Volt output channels with 2500V isolation
Configuration:	16 output channels
Output Capacity:	Outputs are rated at 250mA/channel. (1 Amp total/bank of 8 I/O's)
Protection:	Outputs are overcurrent and over temperature protected
Indicators:	Individual status LED's
Address Setting:	Via DIP switches
Power Supply:	24V dc, Class 2 transformer or power source 18 ... 29V dc / 1.5W.
Mounting:	DIN rail mount
Size:	26mm wide 85mm deep 130mm high
Weight:	128g
CAN:	500kHz, Up to 256 expansion I/O channels
EMC:	EN 61000-6-2 : 2005 Industrial Noise Immunity / EN 61000-6-4 : 2007 Industrial Noise
CAN protocol:	Trio CAN I/O or CANopen DS401.

CAN Analogue I/O Module (P326)

The CAN Analogue I/O Module allows the *Motion Coordinator* to be expanded with banks of 8 analogue input channels and 4 analogue output channels. Up to 4 x P326 Modules may be connected allowing up to 32 x 12 bit analogue inputs and 16 x 12 bit analogue output channels. Convenient disconnect terminals are used for the I/O connections. The input channels are designed for +/-10 Volt operation and the 4 output channels each provide a -10V to +10V signal. Each bank of 8 in / 4 out channels is opto-isolated from the CAN bus.



I/O Connections:

The CAN analogue I/O Module has 3 disconnect terminal connectors:

DeviceNet physical format 5 way CAN connector (on top)

Analogue Input Bank 0 - 7 and 0V ref on 10 way connector

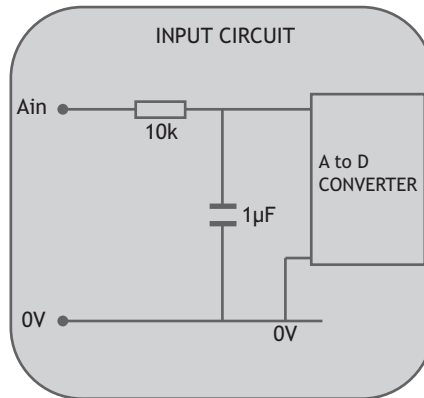
Analogue Output Bank 0 - 3 and 0V ref on 10 way connector.

Bus Wiring

See Can 16-I/O for details

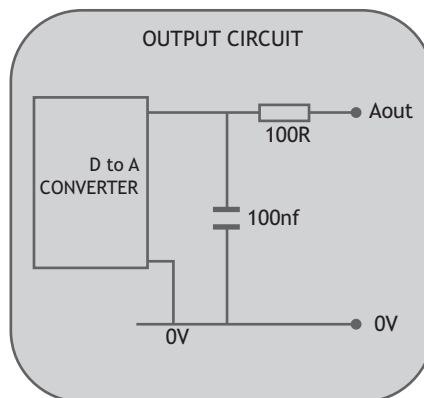
Input Terminals

The 8 analogue inputs are single-ended and have a common 0V. Analogue input nominal impedance = 30kOhm.



Output Terminals

The 4 analogue outputs are single-ended and have a common 0V. Analogue output nominal impedance = 2000hm.

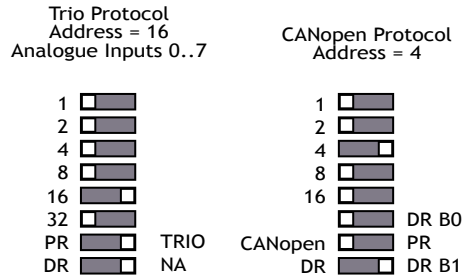


DIP Switch Settings

The switch marked “**PR**” selects the protocol. To the right selects the Trio CAN protocol. To the left selects CANopen protocol, DS401.

The switch marked **DR** sets the baud rate. 500KHz must be selected when using Trio Protocol.

When using the Trio protocol, the addresses for P326 modules **MUST** be set 16,17,18 or 19 in sequence. Therefore the first P326 Module should have the switch setting as shown.



The **AIN** command addresses the analogue inputs as per the following table.

Address:	Start:	End:
16	0	7
17	8	15
18	16	23
19	24	31



P326 modules and P316, P317 and P318 I/O modules may be mixed on the network. The P316, P317 and P318 addresses will be 0 to 15 in sequence and the P326 modules will have addresses 16 to 19 in sequence.

LED Indicators

PWR	ON when module powered on OK
ERR	ON when there is a CAN network error
0	Error code display bit 0
1	Error code display bit 1
2	Error code display bit 2
3	Error code display bit 3



See page 5-12, P317 Error codes for error code table.

Software Interfacing P326

The *Motion Coordinator* will automatically detect and allow the use of correctly connected P326 modules. The number of connected analogue input channels is reported in the startup message and is also available to the programmer via an additional system parameter “**NAIO**”.

In the Trio compatibility mode, the analogue input resolution is fixed at +10Volts to -10Volts single ended and will return values -2047 to 2048 to the function **AIN()**. The first 4 channels are also available as system parameters **AIN0**, **AIN1**, **AIN2**, and **AIN3**. This allows these values to be seen using the **SCOPE** function.

When using extended CAN functions in CANopen, the input scale and offset are programmable. See the P326 CANopen manual for details.

Analogue outputs are not directly accessible in Trio Mode. In CANopen mode, they are settable using standard CANopen objects from TrioBASIC.

Troubleshooting- P326

If the network configuration is incorrect 2 indications will be seen: The P326 module will indicate that it is uninitialised and the *Motion Coordinator* will report the wrong number when questioned:

>>? **NAIO**

If this is not as expected check:

- Terminating 120 Ohm Network Resistors fitted?
- 24Volt Power to Network?
- DIP switches in sequence starting 16,17,18...?
- Cable used is the correct CAN bus specification?
- *Motion Coordinator* CANIO_ADDRESS=32?

Specification P326

Analogue Inputs:	8 +/-10 Volt inputs with 500V isolation from CAN bus.
Resolution:	12 bit.
Protection:	Inputs are protected against 24V over voltage.
Analogue Outputs:	4 -10V to +10V outputs with 500V isolation from CAN bus.
Resolution:	12Bit.
Address Setting:	Via DIP switches.
Power Supply:	24V dc, Class 2 transformer or power source. 18 ... 29V dc / 1.5W.
Mounting:	DIN rail mount.

Size:	26mm wide 85mm deep 130mm high.
Weight:	128g
CAN:	500kHz, Up to 32 analogue input channels and 16 analogue output channels.
EMC:	EN 61000-6-2 : 2005 Industrial Noise Immunity / EN 61000-6-4 : 2007 Industrial Noise Emissions.
CAN Protocol:	Trio CAN I/O or CANopen DS401.

CHAPTER
SYSTEM SETUP AND
DIAGNOSTICS

6

System Setup and Diagnostics

Preliminary Concepts

- Host Computer:** A Windows PC running *Motion Perfect 2*.
- Motor:** A tuned servo drive / motor configuration for a servo axis or a stepper motor and drive combination.
- Prompt:** When the controller is ready to receive a new command, the prompt >> will appear on the left hand side of the current line in the “terminal” under the “tools” menu .
- Axis Parameters:** Can be written to or read from. For example the proportional gain of a servo axis has the name `P _ GAIN`.
It can be written to: `P _ GAIN=0.5`
or read from: `PRINT P _ GAIN`.

For further information see chapter 8.

System Setup



A CONTROL SYSTEM SHOULD BE TREATED WITH RESPECT AS CARELESS OR NEGLIGENT OPERATION MAY RESULT IN DAMAGE TO MACHINERY OR INJURY TO THE OPERATOR. FOR THIS REASON THE SETTING UP OF THE SYSTEM SHOULD NOT BE RUSHED.

This section describes a methodical approach to system configuration and is designed to gradually test each aspect of the system in turn, finally resulting in the connection of the motor. If followed cautiously no unexpected situations should arise.

In cases where the setup procedure for servo and stepper systems differ a separate description is provided for each. In multiple axis systems it is advantageous to set up one axis at a time. The following procedure applies to all *Motion Coordinator* modules.



It is recommended that this section is read in full before attempting to operate the system for the first time.

Preliminary checks

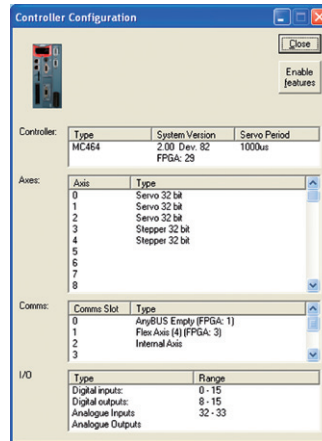


ALL WIRING SHOULD BE CHECKED FOR POSSIBLE MISCONNECTION AND INTEGRITY BEFORE ANY POWER IS APPLIED.

- Disconnect all external connectors from the system, apart from the CANBus.
- Check address DIP switches on any modules fitted.
- Apply power to system and check the 24V power input on the MC464.
- Connect an Ethernet lead between the controller and an unused port on your PC. Either straight or cross over cables will work.

Checking Communications and System Configuration

- Ensure that the Ethernet lead is connected between the *Motion Coordinator* and the PC
- Note: If there is no DHCP server on the Ethernet network, then set the P.C. to use a fixed IP address and subnet mask. For example; IP=192.168.0.001, subnet=255.255.255.0.
- Apply 24V to the *Motion Coordinator*.
- Run *Motion Perfect* on the computer while holding the shift key to stop it searching for a controller. Select 'Options/Communications and add an Ethernet connection. Use the IP address shown on the front of the MC464.
- Select “controller/connect”.
- When *Motion Perfect* detects a controller press the Ox button. If this is the first time you have connected you will need to select the “New Project” option when *Motion Perfect* tries to ensure that your “Project” on the controller matches its copy on disk.
- When the “Project Consistent” message is received in the “Check Project” window you know:
- *Motion Perfect* has made a connection between your PC and the controller.
- *Motion Perfect* has an exact copy of the programs on the controller.
- The controller hardware configuration can now be checked using the “Controller configuration” option under the “Controller” menu. *Motion Perfect* draws a graphical representation of your system, as shown following in the following example.



This message would be produced by a *Motion Coordinator MC464* with the following configuration:

- System Software version 2.00
- An AnyBus-CC Module is in slot 0.
- A FlexAxis (4 axes) Interface is in slot 1
- Slot 2 is for an internal Axis.

Check that the system description corresponds with the modules that are actually present. If this is not so, check any CANbus connections and settings of the address switches on any **CAN** modules if fitted.

Input/Output Connections

- Check each of the 24V input connections with a meter then connect them to the controller.
- Test each of the input channels being used for correct operation in turn. These may be easily viewed in the I/O window. Use “IO Status” under the “Tools” menu.
- Switch each output being used in turn for correct operation. These may be easily set with the IO status window.

Connecting a Servo Motor to a Flexible Axis Module

This description assumes the motor / drive combination has been already tested and is functioning optimally.

Each servo axis should be connected in turn.



FOR THE FOLLOWING TESTS, THE MOTOR MUST BE IN A SAFE DISABLED STATE.

- With the servo drive off or inhibited connect the motor encoder only (or the encoder emulation output from the servo drive).
- Check the encoder counts both up and down by looking at the measured axis position **MPOS** in the Axis parameter window of *Motion Perfect* (“Axis parameters” under the “Tools” menu) whilst turning the axis by hand.
- Ensure the **SERVO** axis parameter is set **OFF** (0) in the Axis parameters window and that the **DAC** axis parameter is set to 0. It may be necessary to use the scroll buttons to view these parameters. This will force 0 volts out of the +/-10V output for the axis. Now connect the servo drive to the V+/V- connections.



BEFORE ENABLING THE DRIVE, MAKE SURE THAT SUDDEN MOVEMENT CANNOT CAUSE HARM OR DAMAGE TO THE MACHINE OR ITS' OPERATORS.

- Enable the servo drive by clicking the “Drives enabled” button on the control panel. If the axis runs away the motor/drive combination must be re-checked. (Note: clicking “Drives enabled” is equivalent to issuing a **WDOG=ON** or **WDOG=OFF** command).
- The servo motor should now be powered and is likely to be creeping in one direction because the position servo is not enabled.
- Set a small positive output voltage by setting **DAC=6000**. The motor should then move slowly forward - Check the encoder is counting up by looking at the **MPOS** axis parameter. If this is correct check that the motor reverses and the encoder counts down when **DAC=-6000**.
- If the encoder counts down when a positive DAC voltage is applied. The motor or position feedback needs to be reversed.

This can be achieved by:

- Swap A and /A connections on the encoder input, or
 - Swap BOTH motor terminals and tacho terminals (DC motors only!) On many digital brushless motors the direction can be reversed by a drive setting, or
 - If the drive has differential inputs, reverse the voltage as it enters the drive. (This can cause problems with some servo-drives. The V- pin of the Flexible Axis Module is a common 0V inside the *Motion Coordinator* so the axis voltage outputs cannot float relative to each other), or
 - set a negative **PP _ STEP** axis parameter. (This is not possible using Absolute encoders)
- or
- Set a negative **DAC _ SCALE** axis parameter.

We are now ready to apply the position servo as described following.

Setting Servo Gains

The servo system controls the motor by constantly adjusting the voltage output which gives a speed demand to the servo drive. The speed demand is worked out by looking at the measured position of the axis from the encoder comparing it with the demand position generated by the *Motion Coordinator*.

The demand position is constantly being changed by the *Motion Coordinator* during a move. The difference between the demand position (Where you want the motor to be) and the measured position (Where it actually is) is called the following error.

The controller checks the following error typically 1000 times per second and updates the speed demand according to the “servo function”. The *Motion Coordinator* has 5 gain values which control how the servo function generates the voltage output from the following error.

Default Settings:

Gain	Parameter Name	Default Value
Proportional Gain	P _ GAIN	1.0
Integral Gain	I _ GAIN	0.0
Derivative Gain	D _ GAIN	0.0
Output Velocity Gain	OV _ GAIN	0.0
Velocity Feedforward Gain	VFF _ GAIN	0.0

A simple test program can be used to generate movement to and fro for examination of the motion profile generated on the oscilloscope. The oscilloscope can be started from the menu “tools” and “oscilloscope”.

```
` Test program for Servo Loop Tuning
axis _ number=1
counts _ per _ rev=4000
max _ motor _ speed=3000 `speed in RPM
BASE(axis _ number)
UNITS=1
DEFPOS(0)
SPEED=max _ motor _ speed*counts _ per _ rev/60
ACCEL=SPEED*1000
DECEL=SPEED*1000
FE _ LIMIT=counts _ per _ rev/2
SERVO=ON
WDOG=ON
` motor will move 1/4 revolution at high speed
stepsize=INT(counts _ per _ rev/4)
WHILE TRUE
TRIGGER
WA(20)
MOVE(stepsize)
WA(500)
MOVE(-stepsize)
WA(480)
WEND
```

The editor built into *Motion Perfect* may be used to enter the test program. Click on Program, New from the pull down menus and enter a program name. Now click on the **EDIT** button and an edit window will be opened where the program shown above may be typed in. See the *Motion Perfect* section for more details on how to use the editor. Once the program is entered, it can be run by clicking on the red button next to its name or the **RUN** button in the Controller Status panel.

Set the oscilloscope to show **MPOS** and **DPOS** for the axis being checked. Set the horizontal timebase to 20msec/division and the trigger mode to trigger from the program.

The servo gain parameters may be set to achieve the desired response from the servo system. The desired response can vary depending on the type of machine.

Different gain settings can be used to obtain:

Smoothest motor running

This can be achieved by using low proportional gain values, adding output velocity gain (**OV _ GAIN**) adds smoothing damping at the expense of higher following errors.

Low following errors during complete motion cycle

This can be achieved by using velocity feed forward (**VFF _ GAIN**) to compensate for following errors together with higher proportional gains.

Exact achievement of end points of moves

This can be achieved by using integral gain in the system together with proportional gain. However overshoot will occur at the end of rapid deceleration.

Typically a combination of the above is required.



The system should be set with proportional gain alone firstly starting with the default value of 1.0. The other gains should then be introduced if necessary according to the descriptions which follow.

Proportional Gain

Description: The proportional gain creates an output value, O_p that is proportional to the following error E .

$$O_p = K_p \times E$$

Axis parameter is called **P _ GAIN**

Example: **P _ GAIN=0.8**

All practical systems use proportional gain, many use this gain parameter alone.

Integral gain

Description: The Integral gain creates an output O_i that is proportional to the sum of the errors that have occurred during the system operation.

$$O_i = K_i \times \int e$$

Integral gain can cause overshoot and so is usually used only on systems working at constant speed or with a slow acceleration.

Axis parameter is called **I _ GAIN**

Example: I _ GAIN=0.0125

Derivative gain

Description: This produces an output O_d that is proportional to the rate of change in the following error and speeds up the response to changes in error whilst maintaining the same relative stability.

$$O_d = K_d \times \Delta E$$

This gain may create a smoother response. High values may lead to oscillation.

Axis parameter is called **D _ GAIN**

Example: D _ GAIN=5

Output Velocity Gain

Description: This increases the system damping, creating an output that is proportional to the change in measured position.

$$O_{ov} = K_{ov} \times \Delta P_m$$

This parameter can be useful for smoothing motions but will generate high following errors. Note that a **NEGATIVE OV _ GAIN** is required for damping.

Axis parameter is called **OV _ GAIN**.

Example: `OV _ GAIN=-5`

Velocity Feed Forward Gain

Description: As movement is created by following errors, at high speed the following error can be quite appreciable. To overcome this the Velocity Feed Forward creates an output proportional to the change in demand position so creating movement without the need for a following error.

$$O_{vff} = K_{vff} \times \Delta P_d$$

Axis parameter is called `VFF _ GAIN`

Example: `VFF _ GAIN=10`

The `VFF _ GAIN` parameter can be set by minimising the following error at a constant machine speed **AFTER** the other gains have been set.

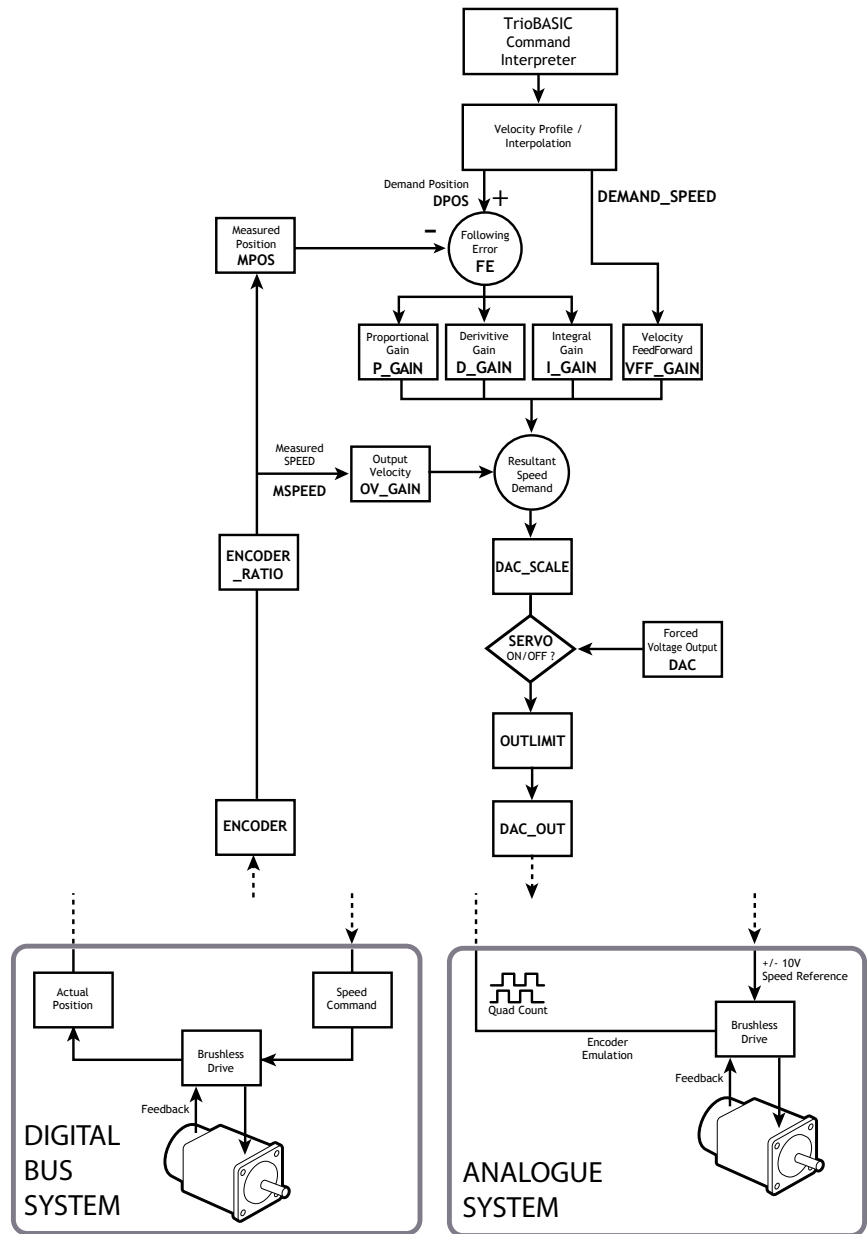
DAC_SCALE

Description: Integer multiplier which is applied to the final velocity command value. The default value is set so that analogue axes are compatible with earlier *Motion Coordinator* and can use the same gain values. For best performance when digital axes are used or a 16 bit DAC is present, the `DAC _ SCALE` should be set to 1.0 and the gains adjusted to suit.

Example: `DAC _ SCALE =1`

ENCODER_RATIO

This function is applied to the encoder counts or feedback counts coming in to the servo loop. The value of `ENCODER _ RATIO` will therefore effect the servo gain values. Set the required `ENCODER _ RATIO` before tuning the position loop.



Diagnostic Checklists

Problem	Potential reasons
No Status LEDs on any module	Power Supply.
LEDs lit on Master but not on other modules	Modules not located correctly.
OK LED ON & Status LED flashing	Following error on at least one axis. The axis demand position and measured position exceed the programmed limit.
Motor runs away without issuing a move command	Motor/drive polarity. Encoder/controller polarity. Gains (drive and/or controller).
Motor runs away upon issuing a move command	Motor feedback. encoder feedback. gains (drive and/or controller).
Motor does not move upon issuing a move command	Wiring (check the enables/inhibits/limits on drive and controller). Check status on all axes. Drive power. Feedhold applied. Speed, acceleration and/or deceleration set to zero. SERVO set off/WDOG set off. Gains (drive and/or controller) Axis is already running a move which has not completed - Check MTYPE and NTYPE .
Axis goes out on following error after a time	Speed being requested exceeds the motor capability - check drive gain and motor/ drive. Drive shutting down on current limit after a time. Drive shutting down on current limit after a time. FE _ LIMIT set too low. VFF _ GAIN needs adjusting.
Axis losing position	Encoder coupling. Encoder signal (wire length, differential/single ended encoder). Mechanics. EMC (check cable screens).

Problem	Potential reasons
<i>Motion Perfect</i> cannot “connect” with the controller	Controller running a program which transmits to “port 0”. If this prevents <i>Motion Perfect</i> connecting to the controller, open Terminal screen in <i>Motion Perfect</i> unconnected mode and type “halt” at the command prompt. Faulty or unconnected Ethernet cable. <i>Motion Perfect</i> IP address is incorrect. PC does not have IP address. Check PC Ethernet settings. PC is on a different subnet to the MC464. Check <i>Motion Perfect</i> version. The latest version can be downloaded from www.triomotion.com .

CHAPTER
PROGRAMMING

7

What is a program?

The traditional description of a program is a task that you want the computer (the *Motion Coordinator*) to perform. The task is described using statements written in the TrioBASIC language which the *Motion Coordinator* can understand.

A program is simply a list of instructions to the *Motion Coordinator*, some of these instructions have a dedicated function to be performed by the controller, others control the program flow, the sequence in which instructions are actually executed.

Statements in your program must be written using a set of rules known as 'Syntax'. You must follow these rules if you are to write TrioBASIC programs. TrioBASIC instructions are divided into the following types:

Instructions

Program Flow

Controller Specific

Identifiers

Labels

Data Storage

Controlling the Sequence of Events

In order to write a program we must break the function of our system down into logical operations which the controller must perform. As we are not able to solve every problem in a purely linear manner, we need more control of the 'flow' of the program instructions, for example to make a decision and decide whether or not certain instructions need to be executed, or to perform a certain task several times. In programming terms we refer to these concepts as **SEQUENCE**, **SELECTION** and **ITERATION**.

Sequence

The ability to process a series of instructions, in a logical order, and to control the flow by branching to another part of the program.

Normally, a program executes statements in sequence starting at the top. In order to branch between different sections of the program we need to be able to identify specific sections of the code. Labels are used as place markers to indicate the start of a routine, or the target for the 'branch' instructions, **GOTO** and **GOSUB**.

It is useful to split your program up into a series of routines, each of which handles a particular function of the machine. The **GOSUB** command will jump to a label

and continue from its new location. When the program encounters a **RETURN** command, the program will jump back to the **GOSUB** from where it originally came.

Take the following example:

```
PRINT "Hello"  
GOSUB a _ subroutine  
STOP  
  
a _ subroutine:  
    PRINT "World"  
RETURN
```

The program will print the “Hello” text to the terminal window, then jump to the line of the program labelled ‘a_subroutine’ and continue execution. The next command it finds will print “World”. The **RETURN** command then returns the program to the point it left, where it then proceeds onto the next command after the **GOSUB** command which in this case is the **STOP** command, which halts the execution of the program.

The **GOTO** command does not remember where it jumped from and will continue running from its new location permanently. This might be used for example, if we have a certain process which needs to be performed when shutting down a machine, we might jump directly to that routine:

i.e. **GOTO** shut_down

TrioBASIC instructions

Labels, **GOTO**, **GOSUB**, **RETURN**, **STOP**

Selection

Commands that enable us to selectively execute instructions depending on certain criteria being met.

Example:

IF we have made a complete batch **THEN** stop the machine.

TrioBASIC Instructions:

```
IF ... THEN ... ELSEIF ... ENDIF  
ON ... GOTO  
ON ... GOSUB
```

Iteration

To repeatedly execute one or more commands automatically, either for a specified number of times, or until a certain condition is met or event occurs.

Example:

```
REPEAT  
  
    GOSUB index _ conveyor  
UNTIL IN(product _ sensor)=ON
```

TrioBASIC instructions:

```
FOR ... TO ... STEP ... NEXT
REPEAT ... UNTIL
WHILE ... WEND
```

FOR..NEXT Statements

The **FOR .. NEXT** commands are used to create a finite loop in which a variable is incremented or decremented from a value to a value.

Example:

```
FOR t=1 TO 5
    PRINT t;" ";
NEXT t
PRINT "Done"
```

The output to the screen would read:

```
1.0000 2.0000 3.0000 4.0000 5.0000
```

The program would set the variable `t` to a value of 1 and then go to the next line to **PRINT**. After the print, the **NEXT** command would return the program to the **FOR** command and increment the value of `T` to make it 2. When the **PRINT** command is used again, the value of `T` has changed and a new value is printed. This continues until `T` has gone from 1 through to 5, then the loop ends and the program is permitted to continue. The next command after the **NEXT** statement prints "Done" to the screen showing the program has left the loop.

You can also use for-next loops to create a loop within a loop, as the following example shows:

```
FOR a=1 TO 5
    PRINT "MAIN A=";a
    FOR b=1 TO 10
        PRINT "LITTLE B=";b
    NEXT b
NEXT a
```

The **FOR..NEXT** statement loops the main `A` variable from 1 to 5, but for every loop of `A` the **FOR..NEXT** statement inside the first loop must also loop its variable `B` from 1 to 10. This is known as a nested loop as the loop in the middle is nested inside an outer loop.

Such loops are especially useful for working on array data by using the variables that increment as position indexes for the arrays. As an example, we could perform a sequence of absolute moves like this:

```
FOR y=12 TO 1 STEP-1
    FOR x=10 to 120 STEP10
        MOVEABS(x,y)
    NEXT x
NEXT y
```

As can be seen, the for-next loop can count down as well as step in value, instead of simply incrementing the loop counter.

Controller Functions

The specific commands, which instruct the processor to perform a predefined function or operation. Each instruction will be assigned its own reserved word in the language.

For example the **PRINT** instruction in TrioBASIC is used to display a message or numeric value on the computer screen or another output device, such as a printer.

Instructions vary in complexity and will take a variety of formats. Some will be a single keyword with a clearly defined function, such as **CANCEL** or **STOP**, whereas others may take one or more parameters which affect the operation of the command.

Example:

```
WA(1000)      wait for a specified time (in milliseconds)
PRINT "Hello"  Display the word "hello" on the terminal screen
GOTO show     redirect the program to the part labelled show
```

Identifiers

Identifiers are the names which the programmer uses to identify things in the program. There are essentially two main types of user-defined identifier, Labels and Variables.

Labels

Labels are used to provide a place-marker in a program. Not only does this make the code more readable, it also enables us to direct the flow of our program to a specific place.

In TrioBASIC, labels are defined by placing a name at the start of the line, followed by a colon (:).

Example:

```
start:
enter _ password:
error _ handler:
```

Variables

Variables are storage locations for numeric values. They are called variables as they can be changed at any time. Just like labels, variables can often be given a user-defined name. Anywhere a number is required a variable can be used. Only the first 32 characters of each variable name are used to identify the unique variable.

Example: `batch_size=10`

would assign a value of 10 to a variable called “batch_size”. Then anywhere in the program that needs to know the value stored can read this value by name.

TrioBASIC has three different variable types:

named variables These are LOCAL variables - i.e. they are only valid within the task they are defined.
Each process can define up to 1024 named variables .

Example:

```
a=123
SPEED=user_speed
PRINT #3,"Length = ";prod_length[2]
```

VR() variables The controller has a global array of variables which are shared between tasks. The MC464 has 65536 VR() variables and most other *Motion Coordinator* have 1024.

Example:

```
VR(2)=123.4567
```

TABLE memory The TABLE memory is a large array of up to 256k entries depending on the controller type. Normally used to store profiles for the CAM/CAMBOX commands.



If the controller features a battery backed memory, VR() variables and TABLE memory will be retained when the power is off.

Expressions

An expression is defined as any calculation or logical function which has to be evaluated. An expression may be used anywhere a number is required, or a logical (**TRUE/FALSE**) decision. In the case of logical expressions, **TRUE** is deemed to be any non-zero result.

In programming, the component parts of an expression are known as operands and operators. The operands are the values, either specific numbers, or variables. The operators are those functions or actions which act on the operands.

Example 1: You can assign the result of an expression to a variable:

```
num_widgets = total_length / widget_length
```

has three operands, num_widgets, total_length and widget_length

and two operators, = (assignment) & / (divide).

Reading the above as simple English would equate to:

Divide the variable total_length by widget_length and assign the result to the variable num_widgets

Example 2: you could use an expression directly:

```
MOVE(widget _ length+10)
```

(MOVE is a TrioBASIC instruction)

Example 3: Sometimes an expression is used to make a decision.

```
IF batch _ count = batch _ size THEN GOTO batch _ done
```


Parameters

Parameters are special purpose variables, used by the system for configuration and feedback.

Axis Parameters

Each of the axes has its own set of axis parameters which are used to achieve many of the *Motion Coordinator* features. The axis parameters may be floating point or integer. The parameters are all set to default values on every power up. Parameters are read from and written to like variables. The TrioBASIC assumes the current **BASE** axis is the required axis unless the **AXIS** modifier is used:

```
>>P _GAIN=2
>>P _GAIN AXIS(8)=0.25
>>? VP _SPEED AXIS(2)
```

A list of all the axis parameters is given in chapter 8

System Parameters

TrioBASIC holds a list of parameters which are common for the whole controller. These parameters can be read from and written to like variables. The system parameters are described in chapter 8. Note that as there is only one system there is no modifier for system parameters.

Process Parameters

TrioBASIC also holds a small number of parameters which are held separately for each **PROCESS**.

Among these are:

TICKS

PROCNUMBER

PMOVE

ERROR _ LINE

INDEVICE / OUTDEVICE

BASE

The process assumed is the current process the command is using, however it is possible to force the controller to read parameters from a specific process with the **PROC()** modifier.

Example: `WAIT UNTIL PMOVE PROC(14)=0`

Forcing priority of program execution

When a user program is running, it is known as a ‘task’, or a ‘process’. The number of simultaneous processes available is dependant on the controller type. When a program is started, the *Motion Coordinator* will allocate it to a process automatically to make the system easier to use. This will normally be sufficient for most applications, especially when there are less than 4 programs in use.

Allocation of Time

For more complex applications it can be useful to allocate execution priorities to programs. In order to do this we need to understand how the *Motion Coordinator* normally allocates the available processing time.

Process Numbers

The processes available for programs are identified by numbers, from 1 to the maximum available on the controller. For example, an MC464 can run 22 simultaneous programs. An additional process is also allocated automatically to the *Motion Coordinator’s* command line interface / *Motion Perfect* connection.

The maximum number of processes available is dependant on the controller type, as shown in the table below.

Controller	Max # Processes	High Priority Processes
MC302X	3	3
Euro205x	7	7,6
MC206x	7	7,6
MC224	14	14,13
MC464	22	21,20

The two highest numbered processes (21 and 20 in our example MC464) are allocated a fixed time slot. These are referred to as the “fast” tasks. They should be used for processes which require:

- Guaranteed processing every servo cycle.
- A large number of calculations or processing.
- Program execution which does not vary in speed as tasks are started or stopped.

Any other processes (including the command line) share the remaining time. Execution speed will therefore reduce as the number of programs running increases. In practice however, a useful execution speed is still obtained.

Programs can be forced to run on a specific process using the commands `RUN` or `RUNTYPE:`

>>RUN "progname",7 'Run the named program immediately on specified task.

If equal time is required to be given to all programs, the high priority processes (21 and 20) should NOT be used. The time available will then be divided evenly between the remaining processes. These programs and the command line use the available time with equal priority.

Command Line Interface

A “Command Line” interface to the controller can be set up by opening a “Terminal” window in *Motion Perfect*. The command line interface always uses channel 0.

```

Terminal: Channel 0
Terminal Edit Options
>>
>>
>>dir
RAM selected for power up
Memory available: 1025829
Selected program: MOTION1
Directory is UNLOCKED
Program      Source Code  Run Type Code Type
-----
STARTUP      834         781 Auto(-1) Normal
HMI          1266        779 Manual   Normal
LOGIC        537         266 Manual   Normal
MOTION1      502         235 Manual   Normal
IN_OUT       52          53 Manual   Normal
OK
>>process
Process Type Status Program      Line
-----
0 Slow Run  Command line/MPE
>>print vr(10)
1.0000
>>

```

Typing Commands for Immediate Execution

When the controller is waiting for a TrioBASIC command to be typed in it prints the prompt `>>`

Example:

```
>>PRINT "HELLO"
```



A line must always be terminated by pressing the **ENTER** key (<CR>)

Limitations of the command line

The command line interface is intended to execute single commands. It is not possible to process multiple-statement lines or those commands which control the sequence or ‘flow’ of a program.

For example, the following type of commands are not available on the command line:

Loop Instructions:

```
FOR..NEXT, WHILE..WEND, REPEAT..UNTIL
```

Wait Instructions:

WA(time), WAIT UNTIL, WAIT IDLE

Named variables:

These are local to a program

Attempting to use any of these commands on the command line may produce unpredictable results!



The command line features a buffer of the last 10 commands used. This can save a lot of typing on the PC. Pressing the up arrow or down arrow cycles through the buffer.

If you find a command you do not recognise it was probably put there by Motion Perfect!

Setting Programs to run on power up

Programs can be set to run automatically on power-up using the “Set power up mode...” facility under the “Program” menu. This sets the **RUNTYPE** automatically.

Example:

Typically only **ONE** program is set to run on power up. This program can then start the others under program control:

```
...body of program
RUN "Prog2"
RUN "Prog3"
...body of program
```

After setting one or more programs to run on power up the project **MUST** be set to “Fixed”. The programs will then be stored in flash Eprom.

The MC464 automatically stores programs to Flash so there is no need to “fix” the project before shutting down *Motion Perfect*.

Example Programs

Example 1:

```
start:
    TICKS=0
    PRINT "Press a key"
    WAIT UNTIL KEY
    GET k
    PRINT "You took ";-TICKS/1000;" seconds"
GOTO start
```

Example 2:

```
`Set speed then move forward then back:
    PRINT "EXAMPLE PROGRAM 2"
    SPEED=100
    ACCEL=1000
    DECEL=1000
    MOVE(250)
    MOVEABS(0)
    STOP
```

Note that the last line stops the program, not the motion. The first line is a comment. It has no effect on the program execution.

Example 3:

```
`Display 16 INPUTS as a row of 1's and 0's
REPEAT
    FOR i=0 TO 15
        IF IN(i)=ON THEN
            PRINT "1";
        ELSE
            PRINT "0";
        ENDIF
    NEXT i
    PRINT CHR (13);
    `Character 13 will do <CR> without linefeed
UNTIL 0
```


CHAPTER
TRIOBASIC COMMANDS

8

TrioBASIC Commands

Contents

MOTION AND AXIS COMMANDS	8-13
ACC	8-13
ADD_DAC	8-14
ADDAX	8-16
AXIS	8-20
BACKLASH	8-22
BASE	8-23
CAM	8-24
CAMBOX	8-29
CANCEL	8-37
CONNECT	8-40
CONNPATH	8-42
DATUM	8-44
DEFPOS	8-49
DISABLE_GROUP	8-51
ENCODER_RATIO	8-54
ENCODER_WRITE	8-56
FLEXLINK	8-57
FORWARD	8-59
MHELICAL	8-61
MHELICALSP	8-64
MOVE	8-67
MOVEABS	8-69
MOVEABSSP	8-72
MOVECIRC	8-73
MOVECIRCSP	8-76
MOVELINK	8-76
MOVEMODIFY	8-81
MOVESP	8-85
MOVETANG	8-86
MSPHERICAL	8-88
MSPHERICALSP	8-92
RAPIDSTOP	8-92
REGIST	8-96
REVERSE	8-104
SERVO_READ	8-107
STEP_RATIO	8-107
INPUT / OUTPUT COMMANDS	8-109
.. (Range)	8-109
AIN	8-109
AINO..3 / AINBIO..3	8-110

CHANNEL_READ	8-110
CHANNEL_WRITE	8-111
CLOSE	8-112
FILE	8-112
FLAG	8-118
FLAGS	8-119
GET	8-120
HW_PSWITCH	8-121
IN	8-122
INPUT	8-123
INPUTS0 / INPUTS1	8-124
INVERT_IN	8-125
KEY	8-126
LINPUT	8-127
MODULE_IO_MODE	8-128
OP	8-129
OPEN	8-131
PRINT	8-133
PSWITCH	8-135
READ_OP	8-137
SETCOM	8-138
TIMER	8-139
PROGRAM LOOPS AND STRUCTURES	8-141
_ (Line Cont)	8-141
BASICERROR	8-141
FOR..TO.. STEP.. NEXT	8-142
GOSUB..RETURN	8-144
GOTO	8-145
IDLE	8-146
IF..THEN..ELSEIF..ELSE..ENDIF	8-146
NEXT	8-148
ON.. GOSUB / GOTO	8-148
REPEAT.. UNTIL	8-150
THEN	8-150
TO	8-151
UNTIL	8-151
WA	8-152
WAIT	8-152
WEND	8-153
WHILE	8-154
SYSTEM PARAMETERS AND COMMANDS	8-155
: (Colon)	8-155
' (Comment)	8-156
# (Hash)	8-157
\$ (Dollar)	8-158
ADDRESS	8-158
ANYBUS	8-159
AOUT	8-164
AUTORUN	8-164

AXIS_OFFSET.....	8-165
BATTERY_LOW.....	8-165
BOOT_LOADER.....	8-166
BREAK_ADD.....	8-166
BREAK_DELETE.....	8-167
BREAK_LIST.....	8-167
BREAK_RESET.....	8-168
CAN.....	8-168
CANIO_ADDRESS.....	8-174
CANIO_ENABLE.....	8-175
CANIO_STATUS.....	8-175
CANOPEN_OP_RATE.....	8-176
CHECKSUM.....	8-176
CLEAR.....	8-176
CLEAR_PARAMS.....	8-177
COMMSERROR.....	8-177
COMMSPOSITION.....	8-178
COMMSTYPE.....	8-178
COMPILE.....	8-180
COMPILE_ALL.....	8-180
CONTROL.....	8-180
COPY.....	8-181
CPU_EXCEPTIONS.....	8-181
DATE.....	8-182
DATES.....	8-183
DAY.....	8-184
DAYS.....	8-185
DEL.....	8-185
DEVICENET.....	8-186
DIR.....	8-187
DISPLAY.....	8-188
DLINK.....	8-189
DUMP.....	8-193
EDPROG.....	8-194
EDPROG1.....	8-196
EPROM.....	8-197
EPROM_STATUS.....	8-197
ERROR_AXIS.....	8-197
ERROR_LINE.....	8-198
ETHERNET.....	8-198
EX.....	8-206
EXECUTE.....	8-207
FEATURE_ENABLE.....	8-207
FLASH_DUMP.....	8-209
FLASHTABLE.....	8-209
FLASHVR.....	8-210
FPGA_VERSION.....	8-211
FPU_EXCEPTIONS.....	8-211
FRAME.....	8-211
FRAME_TRANS.....	8-212

FREE.....	8-212
HALT	8-213
HLM_COMMAND	8-214
HLM_READ	8-216
HLM_STATUS.....	8-217
HLM_TIMEOUT	8-218
HLM_WRITE	8-218
HLS_MODEL	8-220
HLS_NODE.....	8-220
HTTP	8-220
INCLUDE	8-220
INDEVICE	8-221
INITIALISE	8-222
LAST_AXIS.....	8-222
LIST	8-223
LIST_GLOBAL.....	8-223
LOAD_PROJECT.....	8-224
LOADSYSTEM.....	8-224
LOCK	8-225
LOOKUP.....	8-226
MOTION_ERROR	8-226
MPE.....	8-227
N_ANA_IN	8-228
N_ANA_OUT	8-229
NAIO.....	8-229
NEW	8-230
NIO	8-231
OUTDEVICE.....	8-231
PEEK.....	8-232
PLC_ERROR	8-232
PLC_READ	8-233
PLC_STATUS.....	8-234
PMOVE	8-235
PROC	8-236
PROC_LINE	8-236
PROC_STATUS.....	8-236
PROCNUMBER	8-237
RESET	8-237
RUN_ERROR.....	8-238
POKE	8-238
PORT	8-239
POWER_UP	8-239
PRMBLK.....	8-240
PROCESS	8-240
PROJECT_KEY.....	8-241
PROTOCOL.....	8-241
READPACKET	8-242
REMOTE.....	8-244
REMOTE_PROC	8-245
RENAME.....	8-246

RUN.....	8-246
RUNTYPE	8-247
SCHEDULE_TYPE	8-248
SCOPE	8-249
SCOPE_POS.....	8-250
SELECT	8-250
SERCOS.....	8-251
SERCOS_PHASE	8-256
SERIAL_NUMBER.....	8-257
SERVO_PERIOD.....	8-257
SLOT.....	8-258
STEP	8-258
STEPLINE	8-259
STICK_READ.....	8-259
STICK_READVR	8-260
STICK_WRITE.....	8-261
STICK_WRITEVR	8-262
STOP	8-263
STORE	8-264
SYSTEM_VARIABLE	8-264
SYSTEM_ERROR.....	8-264
TABLE	8-265
TABLE_POINTER	8-267
TABLEVALUES	8-268
TICKS	8-269
TIME	8-270
TOKENTABLE	8-270
TRIGGER	8-270
TROFF.....	8-271
TRON.....	8-272
TSIZE	8-273
UNIT_SW_VERSION	8-273
UNLOCK.....	8-274
VERSION	8-274
VIEW	8-275
VR	8-275
VRSTRING	8-277
WDOG	8-277
MATHEMATICAL OPERATIONS AND COMMANDS.....	8-278
+ (Add)	8-278
- (Subtract)	8-278
* (Multiply).....	8-279
/ (Divide)	8-279
^ (Power)	8-280
= (Equals)	8-280
<> (Not Equal).....	8-281
> (Greater Than)	8-282
>= (Greater Than or Equal).....	8-282
< (Less Than)	8-283
<= (Less Than or Equal).....	8-283

ABS	8-284
ACOS	8-285
AND.....	8-285
ASIN	8-287
ATAN	8-287
ATAN2	8-288
B_SPLINE	8-288
CLEAR_BIT	8-291
CONSTANT	8-292
COS	8-293
CRC16	8-293
EXP	8-295
FRAC	8-295
GLOBAL	8-296
IEEE_IN	8-297
IEEE_OUT.....	8-297
INT.....	8-298
INTEGER_READ	8-299
INTEGER_WRITE	8-299
LN.....	8-300
MOD	8-300
NOT	8-301
OR	8-301
READ_BIT.....	8-302
SET_BIT	8-303
SGN.....	8-303
SIN	8-304
SQR.....	8-305
TAN.....	8-305
XOR.....	8-306
CONSTANTS	8-307
FALSE.....	8-307
OFF	8-307
ON	8-308
PI	8-308
TRUE	8-308
AXIS PARAMETERS	8-310
ACCEL	8-310
ADDAX_AXIS	8-310
AFF_GAIN	8-311
ATYPE	8-311
AXIS_ADDRESS	8-313
AXIS_DEBUG_A	8-313
AXIS_DEBUG_B	8-313
AXIS_DISPLAY	8-314
AXIS_ENABLE	8-314
AXIS_ERROR_COUNT.....	8-315
AXIS_MODE.....	8-316
AXISSTATUS	8-316

BACKLASH_DIST	8-317
CHANGE_DIR_LAST	8-318
CLOSE_WIN	8-319
CLUTCH_RATE	8-319
COORDINATOR_DATA	8-320
CORNER_MODE	8-320
CORNER_STATE	8-321
CREEP	8-322
D_GAIN	8-322
D_ZONE_MAX	8-323
D_ZONE_MIN	8-324
DAC	8-324
DAC_OUT	8-325
DAC_SCALE	8-326
DATUM_IN	8-327
DECEL	8-328
DECEL_ANGLE	8-328
DEMAND_EDGES	8-329
DEMAND_SPEED	8-329
DPOS	8-330
ENCODER	8-330
ENCODER_BITS	8-331
ENCODER_CONTROL	8-332
ENCODER_FILTER	8-332
ENCODER_ID	8-333
ENCODER_READ	8-334
ENCODER_STATUS	8-334
ENCODER_TURNS	8-335
END_DIR_LAST	8-335
ENDMOVE	8-336
ENDMOVE_BUFFER	8-337
ENDMOVE_SPEED	8-337
ERRORMASK	8-338
FAST_JOG	8-339
FASTDEC	8-340
FE	8-340
FE_LATCH	8-341
FE_LIMIT	8-342
FE_LIMIT_MODE	8-343
FE_RANGE	8-343
FHOLD_IN	8-344
FHSPEED	8-345
FORCE_SPEED	8-346
FS_LIMIT	8-347
FULL_SP_RADIUS	8-348
FWD_IN	8-349
FWD_JOG	8-349
I_GAIN	8-350
INVERT_STEP	8-350
JOGSPEED	8-351

LIMIT_BUFFERED	8-352
LINK_AXIS	8-352
LOADED	8-353
MARK	8-353
MARKB	8-354
MERGE	8-355
MOVES_BUFFERED	8-356
MPOS	8-356
MSPEED	8-357
MTYPE	8-357
NEG_OFFSET	8-359
NTYPE	8-359
OFFPOS	8-360
OPEN_WIN	8-361
OUTLIMIT	8-362
OV_GAIN	8-362
P_GAIN	8-363
PLM_OFFSET	8-363
POS_OFFSET	8-364
PP_STEP	8-364
PS_ENCODER	8-365
R_MARK	8-365
R_REGISTSPEED	8-366
R_REGPOS	8-367
RAISE_ANGLE	8-368
REG_INPUTS	8-369
REG_POS	8-369
REG_POSB	8-370
REGIST_CONTROL	8-371
REGIST_DELAY	8-371
REGIST_SPEED	8-372
REGIST_SPEEDB	8-372
REMAIN	8-373
REP_DIST	8-374
REP_OPTION	8-375
REV_IN	8-376
REV_JOG	8-376
RS_LIMIT	8-377
SERVO	8-378
SLOT_NUMBER	8-378
SPEED	8-379
SPEED_SIGN	8-379
SPHERE_CENTRE	8-379
SRAMP	8-380
START_DIR_LAST	8-381
STARTMOVE_SPEED	8-381
STOP_ANGLE	8-382
TANG_DIRECTION	8-383
TRANS_DPOS	8-383
TRIOPCTESTVARIAB	8-384

UNITS	8-384
VECTOR_BUFFERED.....	8-385
VERIFY	8-385
VFF_GAIN	8-386
VP_SPEED.....	8-386

Motion and Axis Commands

ACC

Type: Axis Command

Syntax: ACC(rate)

Description: Sets both the acceleration and deceleration rate simultaneously.



This command is provided to aid compatibility with older Trio controllers. Use the ACCEL and DECEL axis parameters in new programs.

Parameters: **rate:** The acceleration rate in UNITS/SEC/SEC.

Example 1: Move an axis at a given speed and using the same rates for both acceleration and deceleration.

```
ACC(120)           `set accel and decel to 120 units/sec/sec
SPEED=14.5        `set programmed speed to 14.5 units/sec
MOVE(200)         `start a relative move with distance of 20
```

Example 2: Changing the ACC whilst motion is in progress.

```
SPEED=100000     `set required target speed (units/sec)
ACC(1000)        `set initial acc rate
FORWARD
```

ADD_DAC

Type: Axis Command

Syntax: `ADD _ DAC(axis)`

Description: Adds the output from the servo control block of a secondary axis to the output of the base axis. The resulting `DAC _ OUT` of the base axis is then the sum of the two control loop outputs.

The `ADD _ DAC` command is provided to allow a secondary encoder to be used on a servo axis to implement dual feedback control.



This would typically be used in applications such as a roll-feed where a secondary encoder to compensate for slippage is required.

Parameters: **axis:** Number of the second axis, who's output will be added to the current axis.

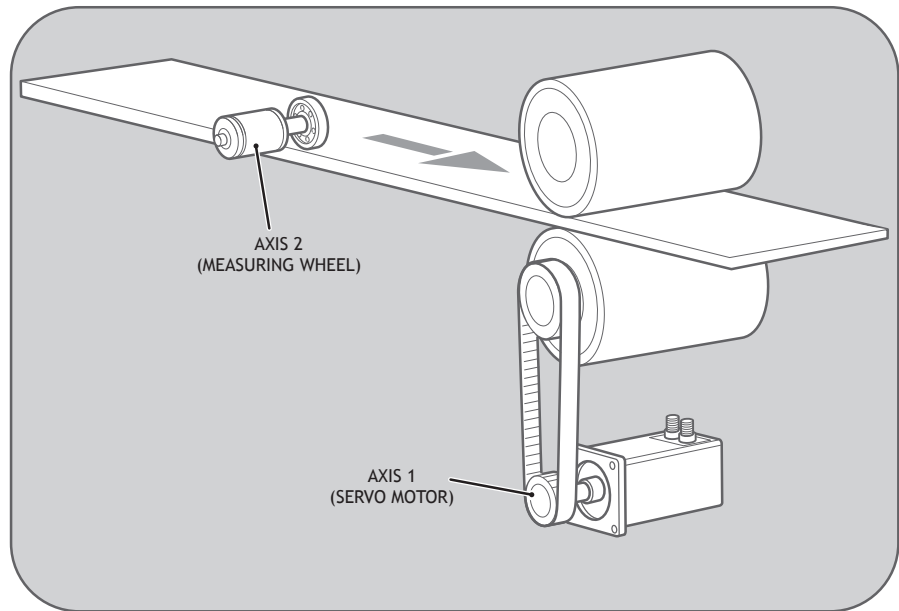
-1 will terminate the `ADD _ DAC` link.

Example 1: Use `ADD _ DAC` to add the output of a measuring wheel to the servo motor axis controlling a roll-feed. Set up the servo motor axis as usual with encoder feedback from the motor drive. The measuring wheel axis must also be set up as a servo. This is so that the software will perform the servo control calculations on that axis.

It is necessary for the two axes to be controlled by a common demand position. Typically this would be achieved by using `ADDAX` to produce a matching `DPOS` on **BOTH** axes. The servo gains are then set up on **BOTH** axes, and the output summed on to one physical output using `ADD _ DAC`.



If the required demand positions on both axes are not identical due to a difference in resolution between the 2 feedback devices, `ENCODER _ RATIO` can be used on one axis to produce matching `UNITS`.



```

BASE(1)
`match the encoder counts per linear distance of the 2 axes
ENCODER_RATIO(counts_per_mm2, counts_per_mm1)
UNITS AXIS(1) = counts_per_mm1
UNITS AXIS(2) = counts_per_mm1    ` units MUST be the same
ADD_DAC(2)                        `Combine axis(2) DAC_OUT
                                   with axis(1)
ADDAX(1) AXIS(2)                  `Superimpose axis 1 demand on axis 2
                                   `the axes are now set up and ready to

move
MOVE(1200)
WAIT IDLE
    
```

ADDAX

Type: Axis Command

Syntax: **ADDAX**(axis)

Description: The **ADDAX** command is used to superimpose 2 or more movements to build up a more complex movement profile:

The **ADDAX** command takes the demand position changes from the specified axis and adds them to any movements running on the base axis.

After the **ADDAX** command has been issued the link between the two axes remains until broken and any further moves on the specified axis will be added to the base axis.



The specified axis can be any axis and does not have to physically exist in the system

The **ADDAX** command therefore allows an axis to perform the moves specified on **TWO** axes added together.



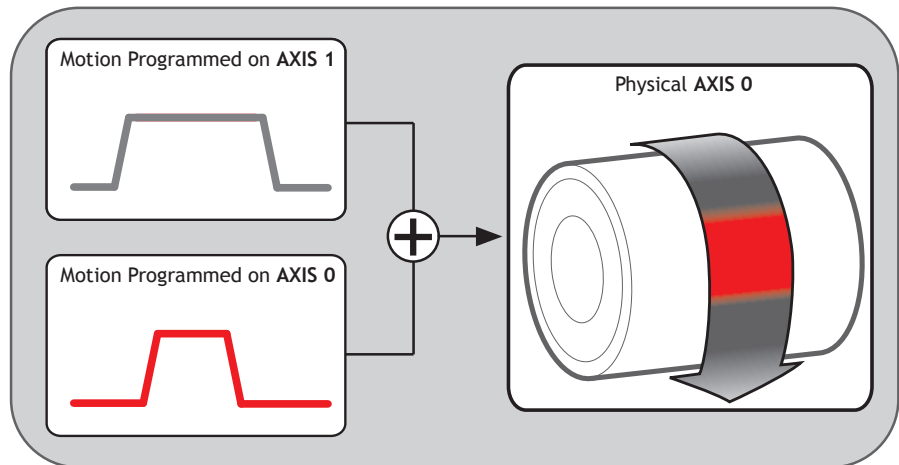
*When using an encoder with **SERVO=OFF** the **MPOS** is copied into the **DPOS**. This allows **ADDAX** to be used to sum encoder inputs.*

Parameter: **axis:** Axis to superimpose.
-1 breaks the link with the other axis.



*The **ADDAX** command sums the movements in encoder edge units.*

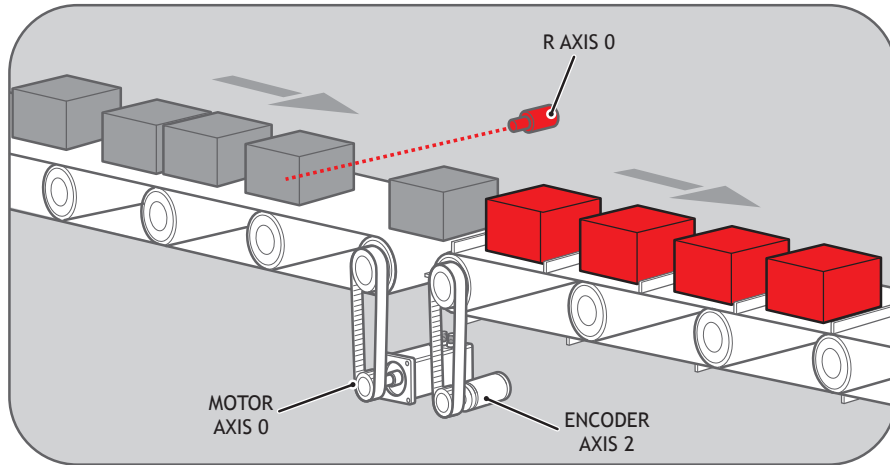
Example 1: Using **ADDAX** on axis with different **UNITS**, Axis 0 will move $1*1000+2*20=1040$ edges.



```
UNITS AXIS(0)=1000
UNITS AXIS(1)=20
  `Superimpose axis 1 on axis 0
ADDAX(1) AXIS(0)
MOVE(1) AXIS(0)
MOVE(2) AXIS(1)
```


Example 2:

Pieces are placed randomly onto a continuously moving belt and further along the line are transferred to a second flighted belt. A detection system gives an indication as to whether a piece is in front of or behind its nominal position, and how far.



```

expected=2000                                'sets expected position
BASE(0)
ADDAX(1)
CONNECT(1,2)                                'continuous geared connection to
                                              flighted belt

REPEAT
  GOSUB getoffset                            'get offset to apply
  MOVE(offset) AXIS(1)                       'make correcting move on virtual
axis
UNTIL IN(2)=OFF                              'repeat until stop signal on input 2
RAPIDSTOP
ADDAX(-1)                                    'clear ADDAX connection
STOP

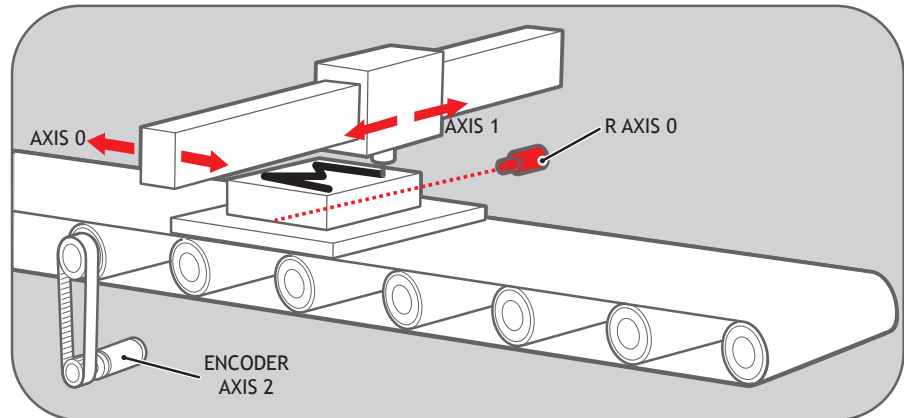
getoffset:                                    'sub routine to register the BASE(0)
REGIST(3)
WAIT UNTIL MARK
seenat=REG_POS
offset=expected-seenat
RETURN

```

Axis 0 in this example is connected to the second conveyor's encoder and a superimposed `MOVE` on axis 1 is used to apply offsets.

Example 3:

An XY marking machine must mark boxes as they move along a conveyor. Using **CONNECT** enables the X marking axis to follow the conveyor. A virtual axis is used to program the marking absolute positions; this is then superimposed onto the X axis using **ADDAX**.



```

ATYPE AXIS(3)=0           'set axis 3 as virtual axis
SERVO AXIS(3)=ON
DEFPOS(0) AXIS(3)
ADDAX (3)AXIS(0)         'connect axis 3 requirement to axis 0
WHILE IN(2)=ON
  REGIST(3)              'registration input detects a box on
                        the conveyor
WAIT UNTIL MARK OR IN(2)=OFF
IF MARK THEN
  CONNECT(1,2) AXIS(0)'connect axis 0 to the moving belt
  BASE(3,1) 'set the drawing motion to axis 3 and 1
  'Draw the M
  MOVEABS(1200,0)'move A > B
  MOVEABS(600,1500)'move B > C
  MOVEABS(1200,3000)' move C > D
  MOVEABS(0,0)'move D > E
  WAIT IDLE
  BASE(0)
  CANCEL                'stop axis 0 from following the belt
  WAIT IDLE
  MOVEABS(0)           'move axis 0 to home position
ENDIF
WEND
CANCEL
    
```

AXIS

Type: Modifier

Syntax: **AXIS(expression)**

Description: Assigns **ONE** command or axis parameter operation to a particular axis.



*If it is required to change the axis used in every subsequent command, the **BASE** command should be used instead.*

Parameters: **Expression:** Any valid TrioBASIC expression. The result of the expression should be a valid integer axis number.

Example 1: The command line has a default base axis of 0. To print the measured position of axis 3 to the terminal in *Motion Perfect*, you must add the axis number after the parameter name.

```
>>PRINT MPOS AXIS(3)
```

Example 2: The base axis is 0, but it is required to start moves on other axes as well as the base axis.

```
MOVE(450)                                `Start a move on the base axis (axis
0)
MOVE(300) AXIS(2)                         `Start a move on axis 2
MOVEABS(120) AXIS(5)                      `Start an absolute move on axis
5
```

Example 3: Set up the repeat distance and repeat option on axis 3, then return to using the base axis for all later commands.

```
REP _ DIST AXIS(3)=100
REP _ OPTION AXIS(3)=1
SPEED=2.30                                `set speed accel and decel on the BASE axis
ACCEL=5.35
DECEL=8.55
```

See Also: **BASE()**

AXISVALUES

Type: Axis Command

Syntax: `AXISVALUES(axis,bank)`

Description: Used by *Motion Perfect* to read a bank of axis parameters.

The data is given in the format:

`<Parameter><type>=<value>`

`<Parameter>` is the name of the parameter.

`<type>` is the type of the value.

i integer.

f float.

s string.

c string of upper and lower case letters, where upper case letters mean an error.

`<value>` an integer, a float or a string depending on the type.

Parameters: **axis:** the axis number where you want to read the parameters.

bank: the bank of parameters that you wish to read.

0: displays the data that is only adjusted through the TrioBASIC

1: displays the data that is changed by the motion generator.

BACKLASH

Type: Axis Command

Syntax: `BACKLASH(enable, distance, speed, acceleration)`

Description: This axis function allows backlash compensation to be loaded. This is achieved by applying an offset move when the motor demand is in one direction, then reversing the offset move when the motor demand is in the opposite direction. These moves are superimposed on the commanded axis movements.



The backlash compensation is applied after a reversal of the direction of change of the DPOS parameter.



The backlash compensation can be seen in the `TRANS _ DPOS` axis parameter. This is effectively `DPOS + backlash compensation`.

Parameters:

Enable:	ON to enable BACKLASH OFF to disable the BACKLASH
distance:	The distance to be offset in user units.
speed:	The speed at which the compensation move is applied in user units.
acceleration:	The accel/decel rate at which compensation move is applied in user units.

Example: `'Apply backlash compensation on axes 0 and 1:`

```
BACKLASH(ON,0.5,10,50) AXIS(0)  
BACKLASH(ON,0.4,8,50) AXIS(1)
```

See Also: `TRANS _ DPOS`

BASE

Type: Process Command

Syntax: `BASE(axis no<,>second axis><,>third axis>...)`

Alternate Format: `BA(...)`

Description: The **BASE** command is used to direct all subsequent motion commands and axis parameter read/writes to a particular axis, or group of axes. The default setting is a sequence: 0, 1, 2...



*Each process has its own **BASE** group of axes and each program can set **BASE** values independently. So the **BASE** array will be different for each of your programs and the command line.*



*The **BASE** array can be printed on the command line by simply entering **BASE***

Parameters: **axis numbers:** The number of the axis or axes to become the new base axis array, i.e. the axis/axes to send the motion commands to or the first axis in a multi axis command.



*The **BASE** array must use ascending values*

Example 1: Setting the base array to non sequential values and printing them back on the command line. This example uses a 16 axis controller.

The controller automatically continues the sequence with 10 and then fills in the missed values at the end of the list.

```
BASE(1,5,9)
BASE(1, 5, 9, 10, 11, 12, 13, 14, 15, 0, 2, 3, 4, 6, 7, 8)
```

Example 2: Set up calibration units, speed and acceleration factors for axes 1 and 2.

```
BASE(1)
UNITS=2000           `unit conversion factor
SPEED=100           `Set speed axis 1 (units/sec)
ACCEL=5000          `acceleration rate (units/sec/sec)
BASE(2)
UNITS=2000           `unit conversion factor
SPEED=125           `Set speed axis 2
```

ACCEL=10000**`acceleration rate**

Example 3: Set up an interpolated move to run on axes; 0 (x), 6 (y) and 9 (z). Axis 0 will move 100 units, axis 6 will move -23.1 and axis 9 will move 1250 units. The axes will move along the resultant path at the speed and acceleration set for axis 0.

```

BASE(0,6,9)
SPEED=120
ACCEL=2000
DECEL=2500
MOVE(100,-23.1,1250)

```

See Also: **AXIS()**

CAM

Type: Axis Command

Syntax: **CAM(start point, end point, table multiplier, distance)**

Description: The **CAM** command is used to generate movement of an axis according to a table of positions which define a movement profile. The table of values is specified with the **TABLE** command. The movement may be defined with any number of points from 3 up to the maximum table size available. The controller performs linear interpolation between the values in the table to allow small numbers of points to define a smooth profile.

The **TABLE** values are translated into positions by offsetting them by the first value and then multiplying them by the multiplier parameter. This means that a non-zero starting profile will be offset so that the first point is zero and then all values are scaled with the multiplier. These are then used as absolute positions from the start position.



*Two or more **CAM** commands executing simultaneously can use the same values in the table.*

The speed of the **CAM** profile is defined through the **SPEED** of the **BASE** axis and the distance parameter. You can use these two values to determine the time taken to execute the **CAM** profile.



*As with any motion command the **SPEED** may be changed at any time to any positive value. The **SPEED** is ramped up to using the current **ACCEL** value. To obtain a **CAM***

shape where ACCEL has no effect the value should be set to at least 1000 times the SPEED value (assuming the default SERVO _ PERIOD of 1ms).

When the CAM command is executing, the ENDMOVE parameter is set to the end of the PREVIOUS move

Parameters:	start point:	The start position of the cam profile in the TABLE.
	end point:	The start position of the cam profile in the TABLE.
	table multiplier:	The table values are multiplied by this value to generate the positions.
	distance:	The distance parameter relates the speed of the axis to the time taken to complete the cam profile. The time taken can be calculated using the current axis speed and this distance parameter (which are in user units).

Example 1: A system is being programmed in mm and the speed is set to 10mm/sec. It is required to take 10 seconds to complete the profile, so a distance of 100mm should be specified.

```
SPEED = 10                                `axis SPEED
time = 10                                  `time to complete profile
distance = SPEED* time                     `distance parameter for CAM
CAM(0, 100, 1, distance)
```

Example 2: Motion is required to follow the POSITION equation:

$$t(x) = x^25 + 10000(1-\cos(x))$$

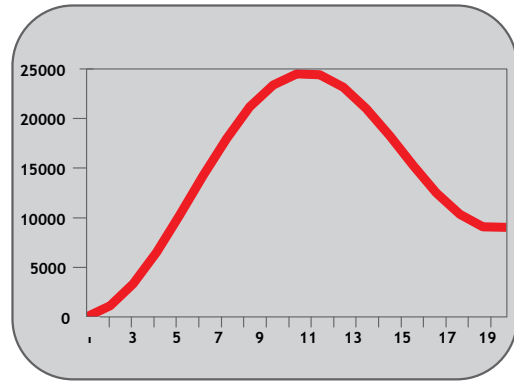
Where x is in degrees. This example table provides a simple oscillation superimposed with a constant speed. To load the table and cycle it continuously the program would be:

```
FOR deg=0 TO 360 STEP 20                   `loop to fill in the table
  rad = deg * 2 * PI/360                   `convert degrees to radians
  x = deg * 25 + 10000 * (1-COS(rad))
  TABLE(deg/20,x)                         `place value of x in table
NEXT deg

WHILE IN(2)=ON                             `repeat cam motion while
input 2 is on
  CAM(0,18,1,200)
  WAIT IDLE
WEND
```

The subroutine camtable loads the data into the cam TABLE, as shown in the graph below.

Table Position	Degrees	Value
1	0	0
2	20	1103
3	40	3340
4	60	6500
5	80	10263
6	100	14236
7	120	18000
8	140	21160
9	160	23396
10	180	24500
11	200	24396
12	220	23160
13	240	21000
14	260	18236
15	280	15263
16	300	12500
17	320	10340
18	340	9103
19	360	9000

**Example 3:**

A masked wheel is used to create a stencil for a laser to shine through for use in a printing system for the ten numerical digits. The required digits are transmitted through port 1 serial port to the controller as ASCII text.

The encoder used has 4000 edges per revolution and so must move 400 between each position. The cam table goes from 0 to 1, which means that the CAM multiplier needs to be a multiple of 400 to move between the positions.

The wheel is required to move to the pre-set positions every 0.25 seconds. The speed is set to 10000 edges/second, and we want the profile to be complete in 0.25 seconds. So multiplying the axis speed by the required completion time

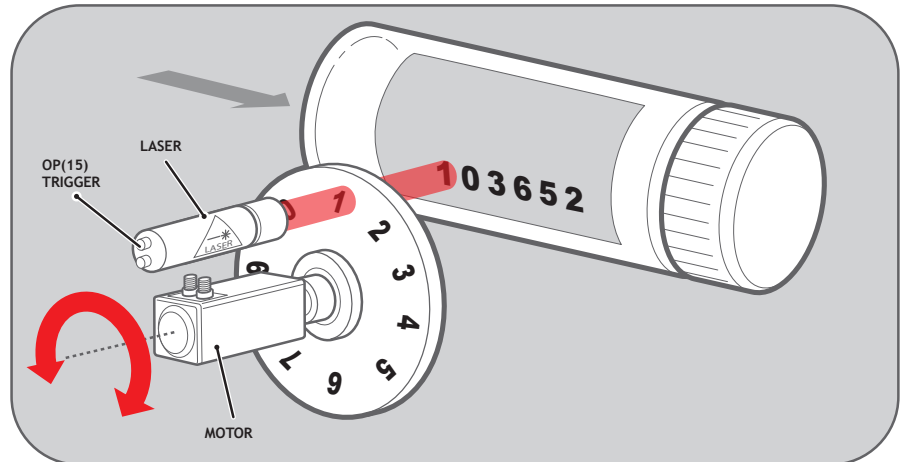
(10000 x 0.25) gives the distance parameter equals 2500.

```
GOSUB profile_gen
  WHILE IN(2)=ON
    WAIT UNTIL KEY#1
    GET#1,k
    'Waits for character on port 1
```

```

        IF k>47 AND k<58 THEN          `check for valid ASCII
character
        position=(k-48)*400           `convert to absolute position
        multiplier=position-offset    `calculate relative movement
        `check if it is shorter to move in reverse direction
        IF multiplier>2000 THEN
            multiplier=multiplier-4000
        ELSEIF multiplier<-2000 THEN
            multiplier=multiplier+4000
        ENDIF
        CAM(0,200,multiplier,2500)    `set the CAM movment
        WAIT IDLE
        OP(15,ON)                      `trigger the laser flash
        WA(20)
        OP(15,OFF)
        offset=(k-48)*400             `calculates current absolute
position
    ENDIF
WEND

profile_gen:
    num_p=201
    scale=1.0
    FOR p=0 TO num_p-1
        TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
    NEXT p
    RETURN
    
```



Example 4:

A suction pick and place system must vary its speed depending on the load carried. The mechanism has a load cell which inputs to the controller on the analogue channel (AIN).

The move profile is fixed, but the time taken to complete this move must be varied depending on the AIN. The AIN value varies from 100 to 800, which has to result in a move time of 1 to 8 seconds. If the speed is set to 10000 units per second and the required time is 1 to 8 seconds, then the distance parameter must range from 10000 to 80000 (distance = speed x time).

The return trip can be completed in 0.5 seconds and so the distance value of 5000 is fixed for the return movement. The Multiplier is set to -1 to reverse the motion.

```
GOSUB profile_gen      'loads the cam profile into the table
SPEED=10000:ACCEL=SPEED*1000:DECEL=SPEED*1000
WHILE IN(2)=ON
  OP(15,ON              'turn on suction
  load=AIN(0)           'capture load value
  distance = 100*load   'calculate the distance parameter
  CAM(0,200,50,distance) 'move 50mm forward in time calculated
  WAIT IDLE
  OP(15,OFF)           'turn off suction
  WA(100)
  CAM(0,200,-50,5000) 'move back to pick up position
WEND
profile_gen:
  num_p=201
  scale=400             'set scale so that multiplier is in mm
  FOR p=0 TO num_p-1
    TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
  NEXT p
  RETURN
```

CAMBOX

Type: Axis Command

Syntax: CAMBOX(start point, end point, table multiplier, link distance , link axis[, link options][, link pos])

Description: The CAMBOX command is used to generate movement of an axis according to a table of POSITIONS which define the movement profile. The motion is linked to the measured motion of another axis to form a continuously variable software gearbox. The table of values is specified with the TABLE command. The movement may be defined with any number of points from 3 up to the maximum table size available. The controller interpolates between the values in the table to allow small numbers of points to define a smooth profile.

The TABLE values are translated into positions by offsetting them by the first value and then multiplying them by the multiplier parameter. This means that a non-zero starting profile will be offset so that the first point is zero and then all values are scaled with the multiplier. These are then used as absolute positions from the start position.



Two or more CAMBOX commands executing simultaneously can use the same values in the table.



When the CAMBOX command is executing the ENDMOVE parameter is set to the end of the PREVIOUS move. The REMAIN axis parameter holds the remainder of the distance on the link axis.

Parameters:

start point:	The start position of the cam profile in the TABLE.
end point:	The end position of the cam profile in the TABLE.
table multiplier:	The table values are multiplied by this value to generate the positions.
link distance:	The distance the link axis must move to complete CAMBOX profile.



The link distance is in the user units of the link axis and should always be specified as a positive distance.

link axis:	The axis to link to.
link options:	Options to customize how your CAMBOX operates.

Bit Values:

1 - link commences exactly when registration event occurs on link axis.

2 - link commences at an absolute position on link axis (see link pos).

4 - **CAMBOX** 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).

8 - **PATTERN** mode. Advanced use of **CAMBOX**: allows multiple scale values to be used. Normally combined with the automatic repeat mode. See example 4.

32 - Link is only active during a positive move on the link axis.

Note: The start options (1 and 2) may be combined with the repeat options (4 and 8).

link pos:

This parameter is the absolute position where the **CAMBOX** link is to be started when parameter 6 is set to 2.

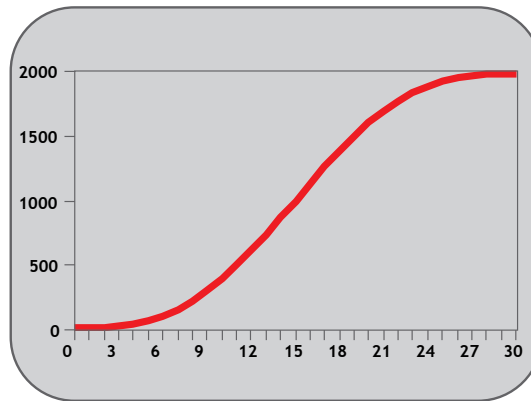
Note: Link pos cannot be at or within one servo period's worth of movement of the **REP _ DIST** position

Example 1: A subroutine can be used to generate a **SIN** shaped speed profile. This profile is used in the other examples.

```

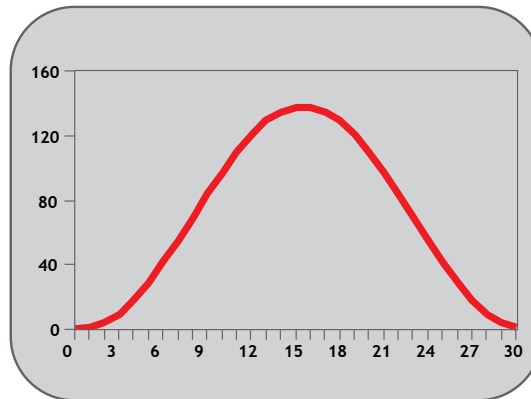
    ` p is loop counter
    ` num_p is number of points stored in tables pos
0..num_p
  ` scale is distance travelled scale factor
profile_gen:
  num_p=30
  scale=2000
  FOR p=0 TO num_p
    TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
  NEXT p
  RETURN

```



This graph plots **TABLE** contents against table array position. This corresponds to motor **POSITION** against link **POSITION** when called using **CAMBOX**. The **SPEED** of the motor will correspond to the derivative of the position curve above:

Speed Curve



Example 2:

A pair of rollers feeds plastic film into a machine. The feed is synchronised to a master encoder and is activated when the master reaches a position held in the variable “start”. This example uses the table points 0...30 generated in Example 1:

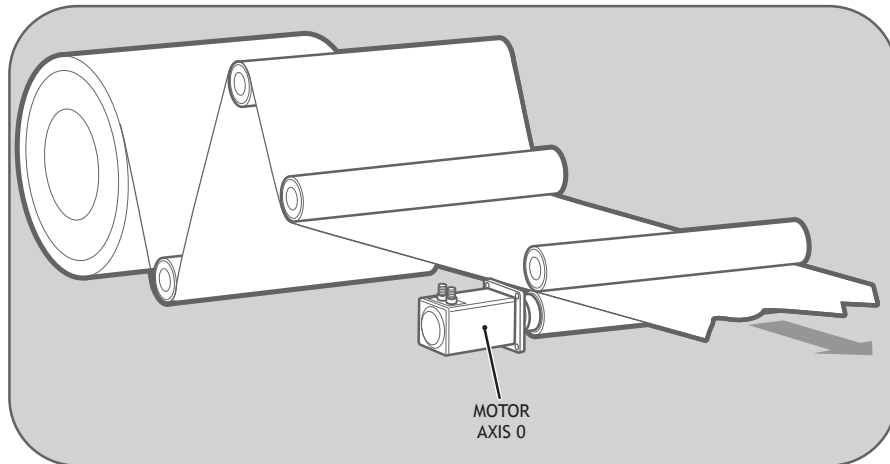
- 0 The start of the profile shape in the **TABLE**.
- 30 The end of the profile shape in the **TABLE**.

- 800 This scales the **TABLE** values. Each **CAMBOX** motion would therefore total 800*2000 encoder edges steps.
- 80 The distance on the product conveyor to link the motion to. The units for this parameter are the programmed distance units on the link axis.
- 15 This specifies the axis to link to.
- 2 This is the link option setting - Start at absolute position on the link axis. “start” variable “start”. The motion will execute when the position “start” is reaches on axis 15.

```

start=1000
FORWARD AXIS(1)
WHILE IN(2)=OFF
  CAMBOX(0,30,800,80,15,2,start)
  WA(10)
  WAIT UNTIL MTYPE=0 OR IN(2)=ON
WEND
CANCEL
CANCEL AXIS(1)
WAIT IDLE

```



Example 3: A motor on Axis 0 is required to emulate a rotating mechanical **CAM**. The position is linked to motion on axis 3. The “shape” of the motion profile is held in **TABLE** values 1000..1035.

The table values represent the mechanical cam but are scaled to range from 0-4000

```

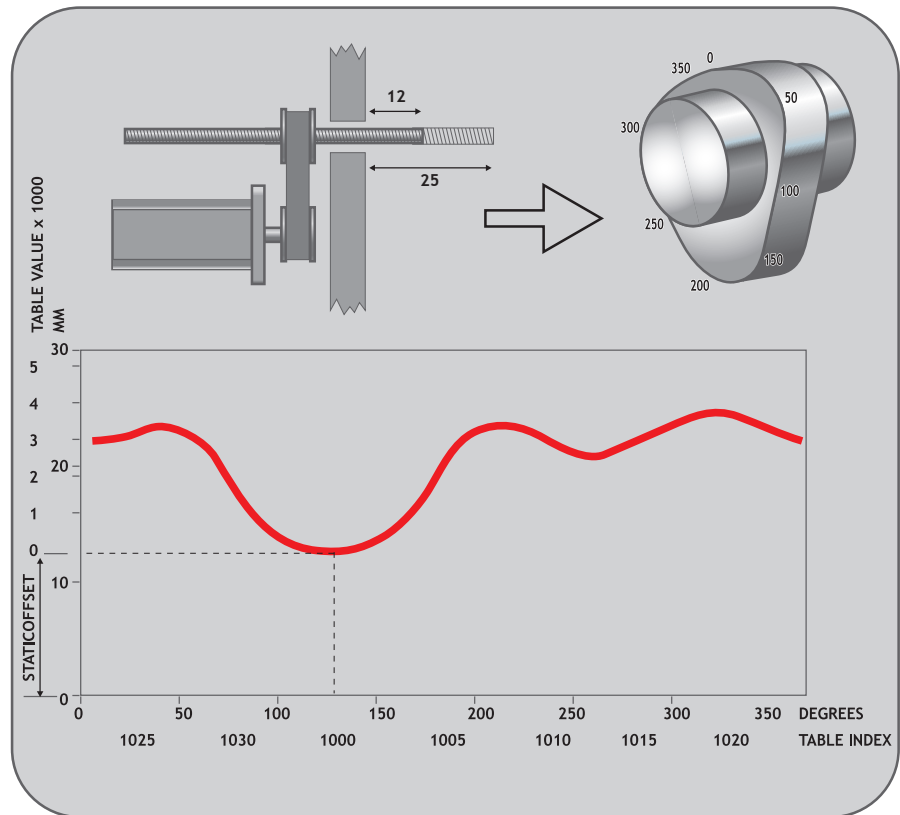
TABLE(1000,0,0,167,500,999,1665,2664,3330,3497,3497)
TABLE(1010,3164,2914,2830,2831,2997,3164,3596,3830,3996,3996)
TABLE(1020,3830,3497,3330,3164,3164,3164,3330,3467,3467,3164)
TABLE(1030,2831,1998,1166,666,333,0)
    
```

```

BASE(3)
MOVEABS(130)
WAIT IDLE
`start the continuously repeating cambox
CAMBOX(1000,1035,1,360,3,4) AXIS(0)
FORWARD `start camshaft axis
WAIT UNTIL IN(2)=OFF
REP_OPTION = 2 `cancel repeating mode by setting bit 1
WAIT IDLE AXIS(0) `waits for cam cycle to finish
CANCEL `stop camshaft axis
WAIT IDLE
    
```



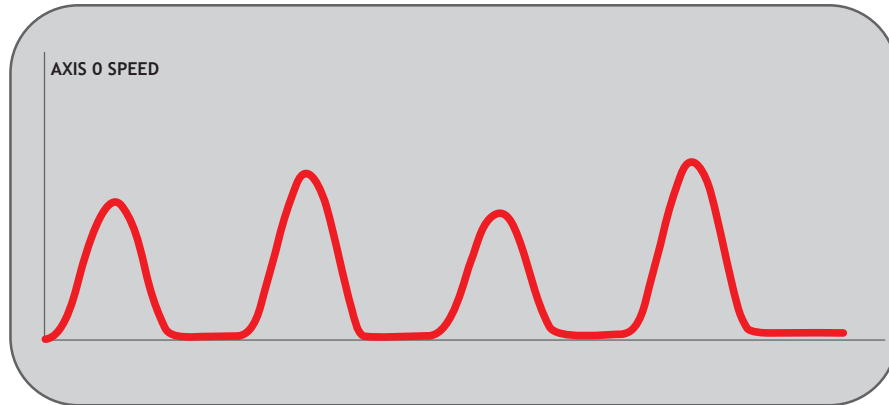
The firmware resets bit 1 of REP_OPTION after the repeating mode has been cancelled.



CAMBOX Pattern Mode:

Syntax: `CAMBOX(start, end, control block pointer, link dist, link axis, options)`

Description: Setting bit 3 (value 8) of the link options parameter enables the **CAMBOX** pattern mode. This mode enables a sequence of scale values to be cycled automatically. This is normally combined with the automatic repeat mode, so the options parameter should be set to 12. This diagram below shows a typical repeating pattern which can be automated with the **CAMBOX** pattern mode:



The start and end parameters specify the basic shape profile **ONLY**. The pattern sequence is specified in a separate section of the **TABLE** memory. There is a new **TABLE** block defined: The “Control Block”. This block of seven **TABLE** values defines the pattern position, repeat controls etc. The block is fixed at 7 values long.

Therefore in this mode only there are 3 independently positioned **TABLE** blocks used to define the required motion:

SHAPE BLOCK	This is directly pointed to by the CAMBOX command as in any CAMBOX .
CONTROL BLOCK	This is pointed to by the third CAMBOX parameter in this options mode only. It is of fixed length (7 table values). It is important to note that the control block is modified during the CAMBOX operation. It must therefore be re-initialised prior to each use.
PATTERN BLOCK	The start and end of this are pointed to by 2 of the CONTROL BLOCK values. The pattern sequence is a sequence of scale factors for the SHAPE .


Negative motion on link axis:

The axis the **CAMBOX** is linked to may be running in a positive or negative direction. In the case of a negative direction link the pattern will execute in reverse. In the case where a certain number of pattern repeats is specified with a negative direction link, the first control block will produce one repeat less than expected. This is because the **CAMBOX** loads a zero link position which immediately goes negative on the next servo cycle triggering a **REPEAT COUNT**. This effect only occurs when the **CAMBOX** is loaded, not on transitions from **CONTROL BLOCK** to **CONTROL BLOCK**. This effect can easily be compensated for either by increasing the required number of repeats, or setting the initial value of **REPEAT POSITION** to 1.

Control Block Parameters

start point:	The start position of the shape block in the TABLE .
end point:	The end position of the shape block in the TABLE .
control block pointer:	The position in the table of the 7 point control block.
link distance:	The distance the link axis must move to complete CAMBOX profile.
link axis:	The axis to link to.
link options:	As CAMBOX , bit 3 must be enabled.

		R/W	Description
0	CURRENT POSITION	R	The current position within the TABLE of the pattern sequence. This value should be initialised to the START PATTERN number.
1	FORCE POSITION	R/W	Normally this value is -1. If at the end of a SHAPE the user program has written a value into this TABLE position the pattern will continue at this position. The system software will then write -1 into this position. The value written should be inside the pattern such that the value: $CB(2) \leq CB(1) \leq CB(3)$.
2	START PATTERN	R	The position in the TABLE of the first pattern value.
3	END PATTERN	R	The position in the TABLE of the final pattern value.
4	REPEAT POSITION	R/W	The current pattern repeat number. Initialise this number to 0. The number will increment when the pattern repeats if the link axis motion is in a positive direction. The number will decrement when the pattern repeats if the link axis motion is in a negative direction. Note that the counter runs starting at zero: 0,1,2,3...

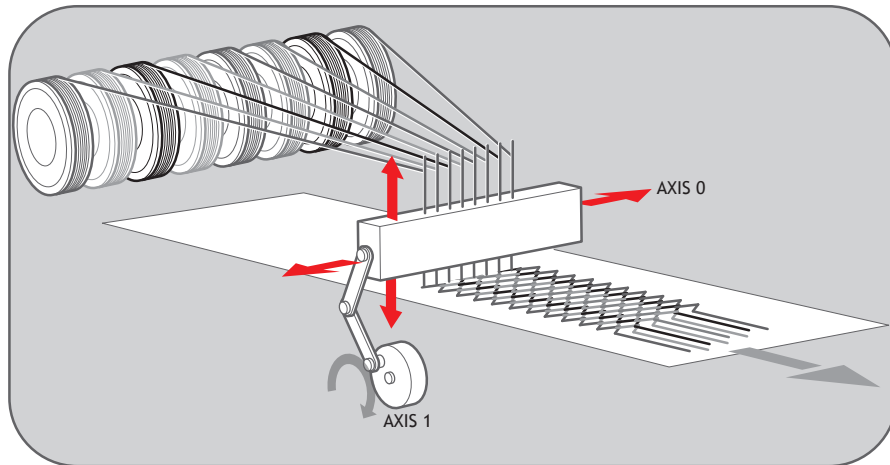
5	REPEAT COUNT	R/W	Required number of pattern repeats. If -1 the pattern repeats endlessly. The number should be positive. When the ABSOLUTE value of CB(4) reaches CB(5) the CAMBOX finishes if CB(6)=-1. The value can be set to 0 to terminate the CAMBOX at the end of the current pattern. See note below, next page, on REPEAT COUNT in the case of negative motion on the link axis.
6	NEXT CONTROL BLOCK	R/W	If set to -1 the pattern will finish when the required number of repeats are done. Alternatively a new control block pointer can be used to point to a further control block.



READ/WRITE values can be written to by the user program during the pattern CAMBOX execution.

Example 4:

A quilt stitching machine runs a feed cycle which stitches a plain pattern before starting a patterned stitch. The plain pattern should run for 1000 cycles prior to running a pattern continuously until requested to stop at the end of the pattern. The cam profile controls the motion of the needle bar between moves and the pattern table controls the distance of the move to make the pattern.



The same shape is used for the initialisation cycles and the pattern. This shape is held in **TABLE** values 100..150

The running pattern sequence is held in **TABLE** values 1000..4999

The initialisation pattern is a single value held in **TABLE**(160)

The initialisation control block is held in **TABLE**(200)..**TABLE**(206)

The running control block is held in **TABLE(300)..TABLE(306)**

```
` Set up Initialisation control block:
TABLE(200,160,-1,160,160,0,1000,300)
```

```
` Set up running control block:
TABLE(300,1000,-1,1000,4999,0,-1,-1)
```

```
` Run whole lot with single CAMBOX:
` Third parameter is pointer to first control block
```

```
CAMBOX(100,150,200,5000,1,20)
WAIT UNTIL IN(7)=OFF
```

```
TABLE(305,0) ` Set zero repeats: This will stop at end of pattern
```

See also: **REP _ OPTION**

CANCEL

Type: Axis Command

Syntax: **CANCEL**([mode])

Alternate Format: **CA**([mode])

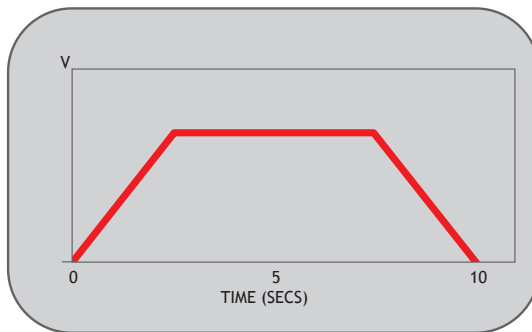
Description: Used to cancel current or buffered axis commands on an axis or an interpolating axis group. Velocity profiled moves, for example; **FORWARD**, **REVERSE**, **MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL**, **MOVEMODIFY**, will be ramped down at the programmed **DECEL** or **FAST _ DEC** rate then terminated. Other move types will be terminated immediately.

Parameters: **Mode:** 0 = Cancels axis commands from the **MTYPE** buffer. Can be used without the parameter
1 = Cancels all buffered moves on the base axis



CANCEL WILL ONLY CANCEL THE PRESENTLY EXECUTING MOVE. IF FURTHER MOVES ARE BUFFERED THEY WILL THEN BE LOADED AND THE AXIS WILL NOT STOP.

Example 1: Move the base axis forward at the programmed **SPEED**, wait for 10 seconds, then slow down and stop the axis at the programmed **DECEL** rate.



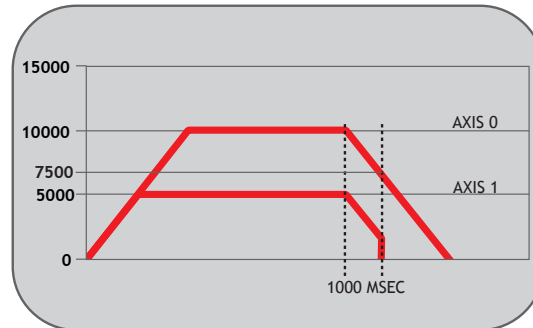
```
FORWARD
WA(10000)
CANCEL' stop movement after 10 seconds
```

Example 2: A flying shear uses a sequence of **MOVELINKs** to make the base axis follow a reference encoder on axis 4. When the shear returns to the top position an input is triggered, this removes the buffered **MOVELINK** and replaces it with a decelerating **MOVELINK** to ramp down the slave (base) axis.

```
ref_axis = 4
REPEAT
  MOVELINK(100,100,0,0,ref_axis)
  WAIT LOADED 'make sure the NTYPE buffer is empty each time
UNTIL IN(5)=ON
CANCEL(1) 'cancel the movelink in the NTYPE buffer
MOVELINK(100,200,0,200,ref_axis) ' deceleration ramp
CANCEL 'cancel the main movelink, this starts the decel
```

Example 3:

Two axes are connected with a ratio of 1:2. Axis 0 is cancelled after 1 second, then axis 1 is cancelled when the speed drops to a specified level. Following the first cancel axis 1 will decelerate at the **DECEL** rate. When axis 1's **CONNECT** is cancelled it will stop instantly.



```

BASE(0)
SPEED=10000
FORWARD
CONNECT(0.5,0) AXIS(1)
WA(1000)
CANCEL
WAIT UNTIL VP_SPEED<=7500
CANCEL AXIS(1)
    
```

See Also:

RAPIDSTOP, FAST_DEC

CONNECT

Type: Axis Command

Syntax: `CONNECT(ratio, driving axis)`

Alternate Format: `CO(...)`

Description: Links the demand position of the base axis to the measured movements of the driving axes to produce an electronic gearbox.

The ratio can be changed at any time by issuing another `CONNECT` command which will automatically update the ratio without the previous `CONNECT` being cancelled. The command can be cancelled with a `CANCEL` or `RAPIDSTOP` command

You can prevent `CONNECT` from being canceled when a hardware or software limit is reached by setting the bit in `AXIS _ MODE`. When this bit is set the ratio is temporarily set to zero while the limit is active so the axis will slow to a stop at the programmed `CLUTCH _ RATE`.

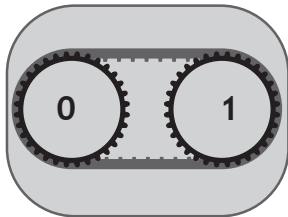
Parameters:

ratio: This parameter holds the number of edges the base axis is required to move per increment of the driving axis. The ratio value can be either positive or negative and has sixteen bit fractional resolution. The ratio is always specified as an encoder edge ratio.

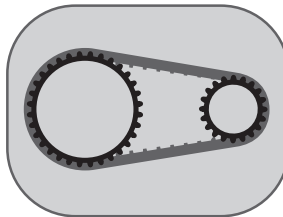
driving axis: This parameter specifies the axis to link to.



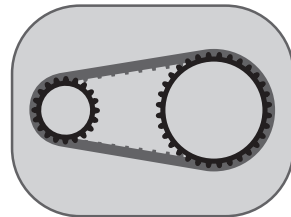
As `CONNECT` uses encoder data it is not affected by `UNITS`, if you need to change the scale of your encoder feedback you should use `ENCODER _ RATIO`.



`CONNECT(1,1)`



`CONNECT(0.5,1)`



`CONNECT(2,1)`

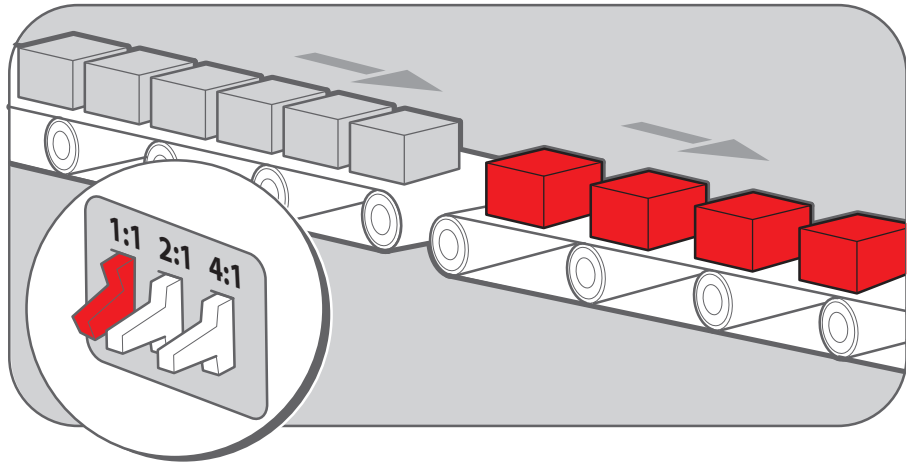


To achieve an exact connection of fractional ratio's of values such as 1024/3072. The `MOVELINK` command can be used with the continuous repeat link option set to `ON`.

Example 1: In a press feed a roller is required to rotate at a speed one quarter of the measured rate from an encoder mounted on the incoming conveyor. The roller is wired to the master axis 0. The reference encoder is connected to axis 1.

```
BASE(0)
SERVO=ON
CONNECT(0.25,1)
```

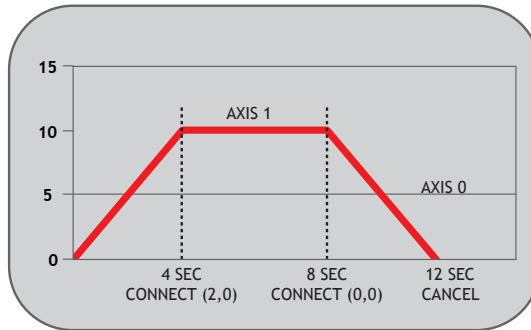
Example 2: A machine has an automatic feed on axis 1 which must move at a set ratio to axis 0. This ratio is selected using inputs 0-2 to select a particular “gear”, this ratio can be updated every 100msec. Combinations of inputs will select intermediate gear ratios. For example 1 ON and 2 ON gives a ratio of 6:1.



```
BASE(1)
FORWARD AXIS(0)
WHILE IN(3)=ON
  WA(100)
  gear = IN(0,2)
  CONNECT(gear,0)
WEND
RAPIDSTOP      `cancel the FORWARD and the CONNECT
```


Example 3:

Axis 0 is required to run a continuous forward, axis 1 must connect to this but without the step change in speed that would be caused by simply calling the `CONNECT`. `CLUTCH_RATE` is used along with an initial and final connect ratio of zero to get the required motion.



```
FORWARD AXIS(0)
BASE(1)
CONNECT(0,0)      `set initial ratio to zero
CLUTCH_RATE=0.5  `set clutch rate
CONNECT(2,0)      `apply the required connect ratio
WA(8000)
CONNECT(0,0)      `apply zero ratio to disconnect
WA(4000)          `wait for deceleration to complete
CANCEL            `cancel connect
```

See Also: `AXIS_MODE`, `CLUTCH_RATE`, `ENCODER_RATIO`

CONNPATH

Type: Axis Command

Syntax: `CONNPATH(ratio , driving axis)`

Description: Enables you to link to the path of an interpolated movement by linking the demand position of the base axis, to the interpolated path distance of the driving axis.

The ratio can be changed at any time by issuing another `CONNPATH` command which will automatically update the ratio without the previous `CONNPATH` being cancelled. The command can be cancelled with a `CANCEL` or `RAPIDSTOP` command



As **CONNPATH** uses encoder data it is not affected by **UNITS**, if you need to change the scale of your encoder feedback you should use **ENCODER _ RATIO**

Parameters:

ratio: This is the ratio between the interpolated distance moved on the driving axis to the distance moved on the base axis.

driving axis: This parameter specifies the axis to link to.

Example 1: A glue laying robot uses a screw feed for the adhesive, this needs to turn a quarter of a revolution for every unit of distance moved.

```
BASE(0)
SERVO=ON
CONNPATH (0.25,1)
```

Example 2: It is required to move 156mm on axis 0 through an interpolated path distance of 100mm on axes 1,2 and 3. This is achieved by using virtual axis 4 as the path distance of the interpolated group and applying a **MOVELINK** from axis 0 to it. **SPEED** is initially set to zero so that the **MOVE** and **MOVLINK** start at the same time.

```
CONNPATH(1,1)AXIS(4)
a=100
b=100
c=100

BASE(1,2,3)
SPEED=0
MERGE=ON

MOVE(a,b,c)
WA(1)
MOVELINK(156,REMAIN AXIS(1),0,0,4)AXIS(0)
SPEED=10
```

See Also: **ENCODER _ RATIO**

DATUM

Type: Axis Command

Syntax: `DATUM(sequence no)`

Description: Performs one of 6 datuming sequences to locate an axis to an absolute position. The creep speed used in the sequences is set using `CREEP`. The programmed speed is set with the `SPEED` command.

`DATUM(0)` is a special case used for resetting the system after an axis critical error. It leaves the positions unchanged.

Parameter:

Seq.	Description
0	<p><code>DATUM(0)</code> clears the following error exceeded <code>FE_LIMIT</code> condition for ALL axes by setting these bits in <code>AXISSTATUS</code> to zero:</p> <ul style="list-style-type: none"> BIT 1 Following Error Warning BIT 2 Remote Drive Comms Error BIT 3 Remote Drive Error BIT 8 Following Error Limit Exceeded BIT 11 Cancelling Move <p>For stepper axes with position verification, the current measured position of ALL axes are set as demand position. <code>FE</code> is therefore set to zero. <code>DATUM(0)</code> must only be used after the <code>WDOG</code> is set to OFF, otherwise there will be unpredictable effects on the motion.</p>
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.

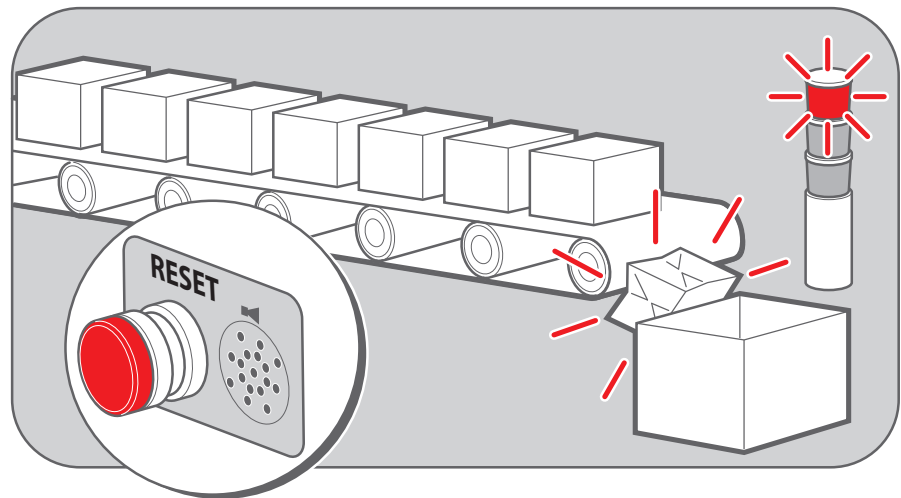
Seq.	Description
5	The axis moves at programmed speed forward until the datum switch is reached. The axis then reverses at creep speed until the datum switch is reset. It then continues in reverse at creep speed looking for the Z marker on the motor. The demand position where the Z input was seen is then set to zero and the measured position corrected so as to maintain the following error.
6	The axis moves at programmed speed forward until the datum switch is reached. The axis then reverses at creep speed until the datum switch is reset. It then continues in reverse at creep speed looking for the Z marker on the motor. The demand position where the Z input was seen is then set to zero and the measured position corrected so as to maintain the following error.
7	Clear AXISSTATUS error bits for the BASE axis only. Otherwise the action is the same as DATUM(0) .



The datuming input set with the **DATUM_IN** which is active low so is set when the input is **OFF**. This is similar to the **FWD**, **REV** and **FHOLD** inputs which are designed to be "fail-safe".

Example 1:

A production line is forced to stop if something jams the product belt, this causes a motion error. The obstacle has to be removed, then a reset switch is pressed to restart the line.



FORWARD

``start production line`

```

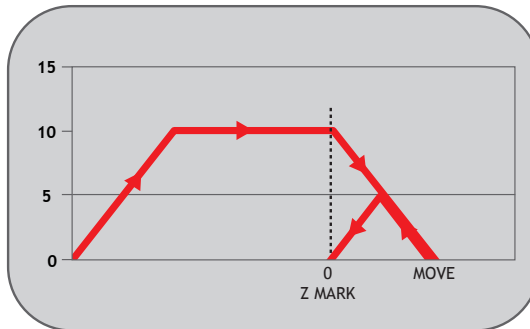
WHILE IN(2)=ON
  IF MOTION_ERROR=0 THEN
    OP(8,ON)          `green light on; line is in motion
  ELSE
    OP(8, OFF)
    GOSUB error _ correct
  ENDIF
WEND
CANCEL
STOP

error _ correct:
  REPEAT
    OP(10,ON)
    WA(250)
    OP(10,OFF)        `flash red light to show crash
    WA(250)
  UNTIL IN(1)=OFF
  DATUM(0)           `reset axis status errors
  SERVO=ON          `turn the servo back on
  WDOG=ON           `turn on the watchdog
  OP(9,ON)          `sound siren that line will restart
  WA(1000)
  OP(9,OFF)
  FORWARD           `restart motion
RETURN

```

Example 2:

An axis requires its position to be defined by the Z marker. This position should be set to zero and then the axis should move to this position. Using the datum 1 the zero point is set on the Z mark, but the axis starts to decelerate at this point so stops after the mark. A move is then used to bring it back to the Z position.



```

SERVO=ON
WDOG=ON
CREEP=1000      `set the search speed
SPEED=5000     `set the return speed
DATUM(1)       `register on Z mark and sets this to datum

```

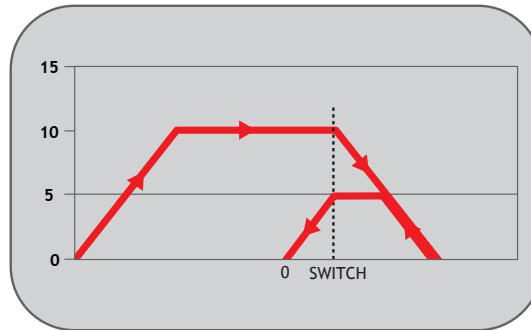
```

WAIT IDLE
MOVEABS (0)    `moves to datum position

```

Example 3:

A machine must home to its limit switch which is found at the rear of the travel before operation. This can be achieved through using `DATUM(4)` which moves in reverse to find the switch.



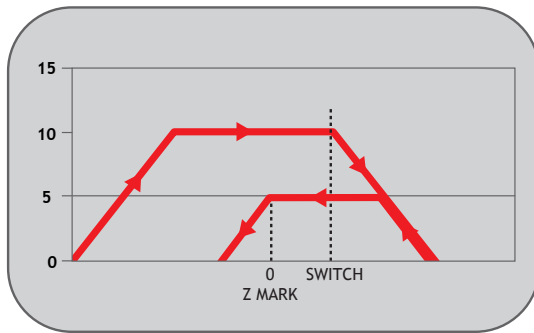
```

SERVO=ON
WDOG=ON
REV_IN=-1    `temporarily turn off the limit switch function
DATUM_IN=5   `sets input 5 for registration
SPEED=5000   `set speed, for quick location of limit switch
CREEP=500    `set creep speed for slow move to find edge of switch
DATUM(4)     `find "edge" at creep speed and stop
WAIT IDLE
DATUM_IN=-1
REV_IN=5     `restore input 5 as a limit switch again

```

Example 4:

A similar machine to Example 3 must locate a home switch, which is at the forward end of travel, and then move backwards to the next Z marker and set this as the datum. This is done using `DATUM(5)` which moves forwards at speed to locate the switch, then reverses at creep to the Z marker. A final move is then needed, if required, as in Example 2 to move to the datum Z marker.



```

SERVO=ON
WDOG=ON
DATUM_IN=7  `sets input 7 as home switch
SPEED=5000  `set speed, for quick location of switch
CREEP=500   `set creep speed for slow move to find edge of switch
DATUM(5)    `start the homing sequence
WAIT IDLE

```

See Also:

`CREEP`, `DATUM_IN`

DEFPOS

Type: Axis Command


Syntax: DEFPOS(pos1 [,pos2[, pos3[, pos4.....]])

Alternate Format: DP(pos1 [,pos2[, pos3[, pos4]])

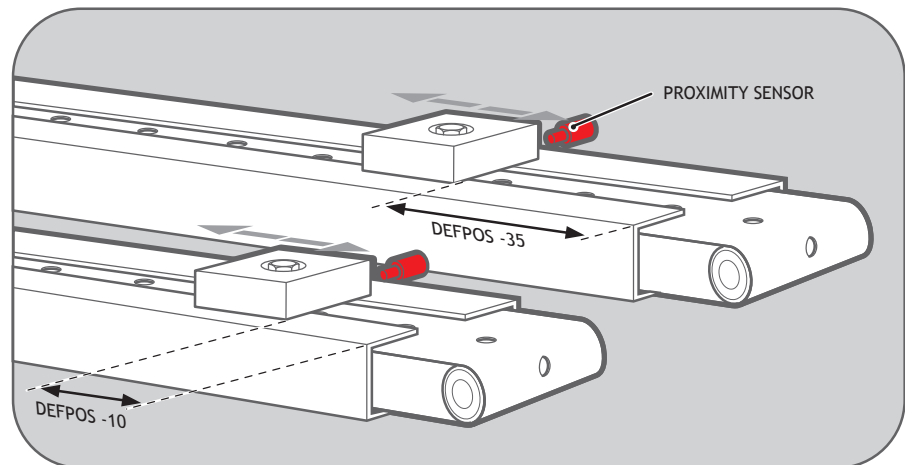
Description: Defines the current position(s) as a new absolute value. The value `pos#` is placed in `DPOS`, while `MPOS` is adjusted to maintain the `FE` value. This function is completed after the next servo-cycle. `DEFPOS` may be used at any time, even whilst a move is in progress, but its normal function is to set the position values of a group of axes which are stationary.

Parameters:

- `pos1`: Absolute position to set on current base axis in user units.
- `pos2`: Abs. position to set on the next axis in `BASE` array in user units.
- `pos3`: Abs. position to set on the next axis in `BASE` array in user units.

 *As many parameters as axes on the system may be specified.*

Example 1: After homing 2 axes, it is required to change the `DPOS` values so that the “home” positions are not zero, but some defined positions instead.

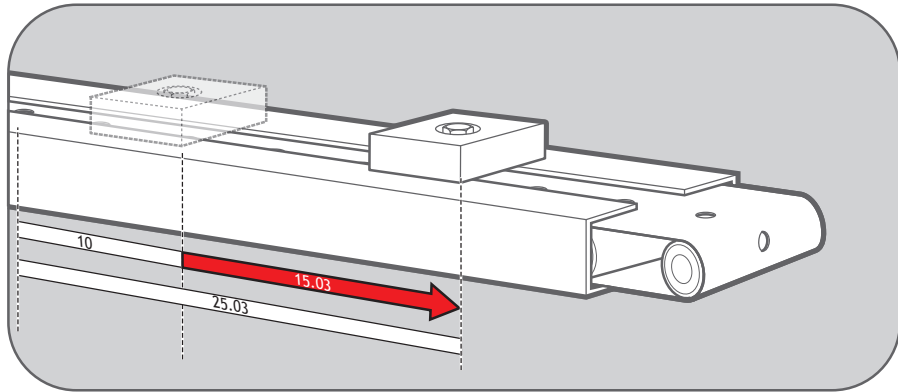



```
DATUM(5) AXIS(1)
end of the DATUM
DATUM(4) AXIS(3)
will be 0,0.
WAIT IDLE AXIS(1)
WAIT IDLE AXIS(3)
BASE(1,3)
DEFPOS(-10,-35)
axes to be -10 and -35
```

```
'home both axes. At the
'procedure, the positions
'set up the BASE array
'define positions of the
```

Example 2:

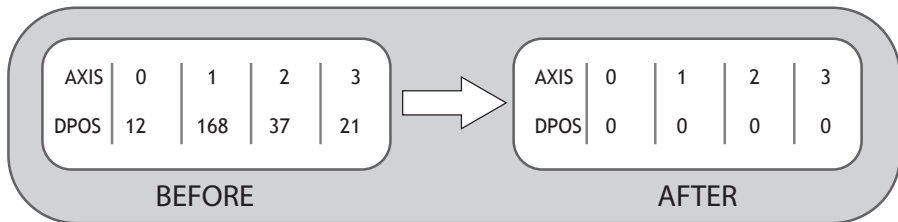
Define the axis position to be 10, then start an absolute move, but make sure the axis has updated the position before loading the **MOVEABS**.



```
DEFPOS(10.0)
WAIT UNTIL OFFPOS=0' Ensures DEFPOS is complete before next line
MOVEABS(25.03)
```

Example 3:

From the *Motion Perfect* terminal, quickly set the **DPOS** values of the first four axes to 0.



```
>>BASE (0)
```

```
>>DP(0,0,0,0)
```

See Also: `OFFPOS`

DISABLE_GROUP

Type: System Command

Syntax: `DISABLE _ GROUP(parameter[,parameters...])`

Description: Used to create a group of axes which will be disabled if there is a motion error in one or more of the group. After the group is created, when an error occurs all the axes in the group will have their `AXIS _ ENABLE` set to `OFF` and `SERVO` set to `OFF`.



Multiple groups can be made, although one axis cannot belong to more than one group.



WARNING: ONLY AXES THAT HAVE INDIVIDUAL ENABLES SHOULD BE USED IN A DISABLE GROUP. SUCH AS DIGITAL DRIVES AND STEPPERS.

Syntax: `DISABLE _ GROUP(-1)`

Description: Clears all groups

Syntax: `DISABLE _ GROUP(axis1 [,axis2[, axis3[, axis4....]]])`

Description: Assigns the listed axis to a group

Parameters: `axis1:` Axis number of first axis in group.
 `axis2:` Axis number of second axis in group.
 `axisN:` Axis number of Nth axis in group.



As many parameters as axes on the system may be specified.

Example 1:

A machine has 2 functionally separate systems, which have their own emergency stop and operator protection guarding. If there is an error on one part of the machine, the other part can safely remain running while the cause of the error is removed and the axis group re-started. We need to set up 2 separate axis groupings

```

DISABLE _ GROUP(-1)                                `remove any
previous axis groupings                            `group
DISABLE _ GROUP(0,1,2,6)                           `group
axes 0 to 2 and 6
DISABLE _ GROUP(3,4,5,7)                           `group
axes 3 to 5 and 7

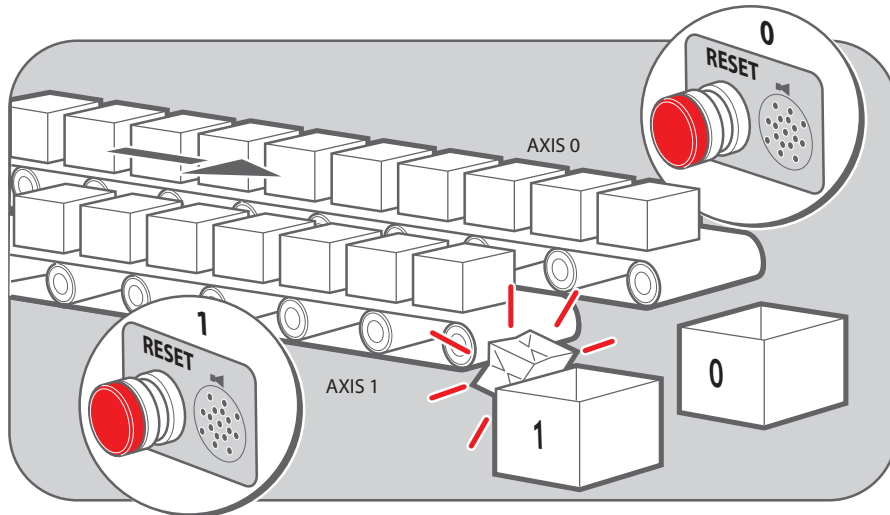
WDOG=ON                                            `turn on the enable relay and the remote
drive enable

FOR ax=0 TO 7
  AXIS _ ENABLE AXIS(ax)=ON `enable the 8 axes
  SERVO AXIS(ax)=ON        `start
position loop for each axis
NEXT ax

```

Example 2:

Two conveyors operated by the same *Motion Coordinator* are required to run independently so that if one has a “jam” it will not stop the second conveyor.



```

DISABLE _ GROUP(0) `put axis 0 in its own group
DISABLE _ GROUP(1) `put axis 1 in another group

GOSUB group_enable0

```

```

GOSUB group_enable1
WDOG=ON

FORWARD AXIS(0)
FORWARD AXIS(1)

WHILE TRUE
  IF AXIS_ENABLE AXIS(0)=0 THEN
    PRINT "motion error axis 0"
    reset_0_flag=1
  ENDIF
  IF AXIS_ENABLE AXIS(1)=0 THEN
    PRINT "motion error axis 1"
    reset_1_flag=1
  ENDIF
  IF reset_0_flag=1 AND IN(0)=ON THEN
    GOSUB group_enable0
    FORWARD AXIS(0)
    reset_0_flag=0
  ENDIF
  IF reset_1_flag=1 AND IN(1)=ON THEN
    GOSUB group_enable1
    FORWARD AXIS(1)
    reset_1_flag=0
  ENDIF
WEND

group_enable0:
  BASE(0)
  DATUM(7) ' clear motion error on axis 0
  WA(10)
  AXIS_ENABLE=ON
  SERVO=ON
RETURN
group_enable1:
  BASE(1)
  DATUM(7) ' clear motion error on axis 0
  WA(10)
  AXIS_ENABLE=ON
  SERVO=ON
RETURN

```

Example 3:

One group of axes in a machine require resetting, without affecting the remaining axes, if a motion error occurs. This should be done manually by clearing the cause of the error, pressing a button to clear the controllers' error flags and re-enabling the motion.

```

DISABLE_GROUP(-1)           'remove any previous axis groupings
DISABLE_GROUP(0,1,2)       'group axes 0 to 2
GOSUB group_enable         'enable the axes and clear errors
WDOG=ON

SPEED=1000
FORWARD

```

```

WHILE IN(2)=ON
  'check axis 0, but all axes in the group will disable
  together
  IF AXIS_ENABLE =0 THEN
    PRINT "Motion error in group 0"
    PRINT "Press input 0 to reset"
    IF IN(0)=0 THEN      'checks if reset button is pressed
      GOSUB group_enable 'clear errors and enable axis
      FORWARD           'restarts the motion
    ENDIF
  ENDIF
WEND
STOP                    'stop program running into sub
routine

group_enable:          'Clear group errors and enable axes
  DATUM(0)             'clear any motion errors
  WA(10)
  FOR axis_no=0 TO 2
    AXIS_ENABLE AXIS(axis_no)=ON 'enable axes
    SERVO AXIS(axis_no)=ON       'start position loop servo
  NEXT axis_no
  RETURN

```

See Also: **AXIS_ENABLE, SERVO**

ENCODER_RATIO

Type: Function

Syntax: **ENCODER_RATIO(mpos_count, input_count)**

Description: This command allows the incoming encoder count to be scaled by a non integer ratio;

MPOS = (mpos_count / input_count) x encoder_edges_input



WHEN USING THE SERVO LOOP YOU WILL NEED TO ADJUST THE GAINS TO MAINTAIN PERFORMANCE AND STABILITY. UNLIKE THE UNIT'S PARAMETER, WHICH ONLY AFFECTS THE SCALING SEEN BY THE USER PROGRAMS, ENCODER_RATIO AFFECTS ALL MOTION COMMANDS.



ENCODER _RATIO does not replace **UNITS**. Only use **ENCODER _RATIO** where absolutely necessary. **PP _STEP** and **ENCODER _RATIO** cannot be used at the same time on the same axis.

Parameters:

mpos _count : A number which defines the numerator.

input _count: A number which defines the denominator.



Large ratios should be avoided as they will lead to either loss of resolution or much reduced smoothness in the motion. The actual physical encoder count is the basic resolution of the axis and use of this command may reduce the ability of the Motion Coordinator to accurately achieve all positions.

Example 1: A rotary table has a servo motor connected directly to its centre of rotation. An encoder is mounted to the rear of the servo motor and returns a value of 8192 counts per rev. The application requires the table to be calibrated in degrees so that each degree is an integer number of counts.

As 8192 cannot be exactly divided into 360 **ENCODER _RATIO** is used to adjust the encoder feedback.

The highest value that is less than 8192 yet divides into 360 should be chosen. This is 7200 (7200 / 20 = 360). This reduces the resolution from 0.044 to 0.055 degrees, but enables you to program easily in degrees.

```
ENCODER _RATIO(7200,8192)
UNITS = 20 ` axis calibrated in degrees
```

Example 2: An X-Y system has 2 different gearboxes on its vertical and horizontal axes. The software needs to use interpolated moves, including **MOVECIRC** and **MUST** therefore have **UNITS** on the 2 axes set the same. Axis 3 (X) is 409 counts per mm and axis 4 (Y) has 560 counts per mm. So as to use the maximum resolution available, set both axes to be 560 counts per mm with the **ENCODER _RATIO** command.

```
ENCODER _RATIO(560,409) AXIS(3) `axis 3 is now 560 counts/mm
UNITS AXIS(3) = 56 `X axis calibrated in mm x 10
UNITS AXIS(4) = 56 `Y axis calibrated in mm x 10
MOVECIRC(200,100,100,0,1) `move axes in a semicircle
```

ENCODER_WRITE

Type: Axis Command

Syntax: Value = ENCODER_WRITE (address, data)

Description: Write an internal register to an Absolute Encoder on an EnDat absolute encoder.

Parameters:

- Value:** Returns **TRUE** if the write was successful and **FALSE** if it fails.
- address:** The address of the EnDat encoder register to be written to.
- data:** Value to be written to the specified register.

Example: Write a value to the EnDat encoder and check it has been written, then set the encoder back to position mode.

```
IF NOT ENCODER_WRITE (endat_address, setvalue) THEN
  PRINT "Fail to write to encoder"
ENDIF
ENCODER_CONTROL=0
```

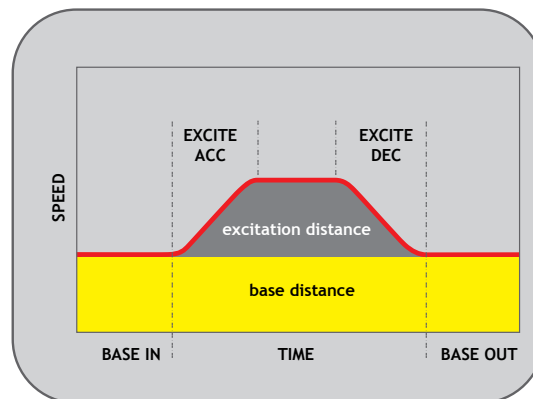
See Also: ENCODER_CONTROL, ENCODER_READ

FLEXLINK

Type: Axis Command

Syntax: `FLEXLINK(base_dist, excite_dist, link_dist, base_in, base_out, excite_acc, excite_dec, link_axis, options, start_pos)`

Description: The **FLEXLINK** command is used to generate movement of an axis according to a defined profile. The motion is linked to the measured motion of another axis. The profile is made up of 2 parts, the base move and the excitation move both of which are specified in the parameters. The base move is a constant speed movement. The excitation movement uses sinusoidal profile and is applied on top of the base movement.



This command allows you to simplify a CAMBOX type movement through not having to use any table data.

Parameters:

<code>base_dist:</code>	The distance the axis should move at a constant speed.
<code>excite_dist:</code>	The distance the axis should perform the profiled move.
<code>link_dist:</code>	The distance the link axis should move while the FLEXLINK profile execute.
<code>base_in:</code>	The percentage of the base move that completes before the excitation move starts.
<code>base_out:</code>	The percentage of the base move that completes after the excitation move completes.
<code>excite_acc:</code>	The percentage of the excitation move used for acceleration.

excite _ dec:	The percentage of the excitation move used for deceleration.
link _ axis:	The axis to link to.
options:	Options to customize how your FLEXLINK operates.
Bit Values:	<p>1 = link commences exactly when registration event occurs on link axis.</p> <p>2 = link commences at an absolute position on link axis</p> <p>4 = FLEXLINK 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).</p> <p>32 = Link is only active during a positive move on the link axis.</p>
start _ pos:	The absolute position on the link axis where the FLEXLINK is to be start. Used with link option 2.



The options (1 and 2) may be combined with the repeat options (4).



START_POS CANNOT BE AT OR WITHIN ONE SERVO PERIOD'S WORTH OF MOVEMENT OF THE REP _ DIST POSITION.

Example 1:

Suppose you want a smooth curve for 40% of a cycle and to remain stationary for the remainder:

```
FLEXLINK(0,10000,20000,60,0,50,50,1)
```

In this example the move length is 10000 and this is linked to 20000 distance on the link axis (1). The axis is stationary for 60% of the cycle and the move is 50% accel/50% decel.

Example 2:

Suppose you want a 1:1 background link but to advance 500 using a smooth curve between 80% and 95% of a cycle:

```
FLEXLINK(10000,500,10000,80,5,50,50,1)
```

In this example the base move length is 10000 and this is linked to 10000 distance on the link axis (1). The excite distance is 500 and this starts after 80% of the cycle, with 5% at the end also clear of excitation. The "excite" move is 50% accel/50% decel.

FORWARD

Type: Axis Command

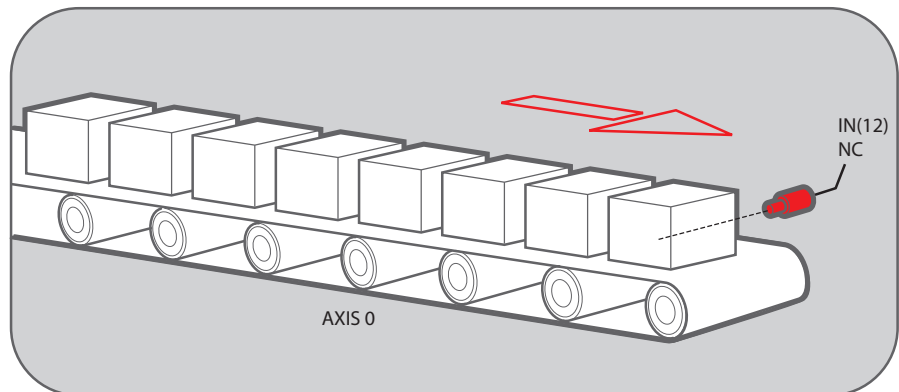
Syntax: FORWARD

Alternate Format: FO

Description: Sets continuous forward movement. The axis accelerates at the programmed **ACCEL** rate and continues moving at the **SPEED** value until either a **CANCEL** or **RAPIDSTOP** command are encountered. It then decelerates to a stop at the programmed **DECEL** rate.

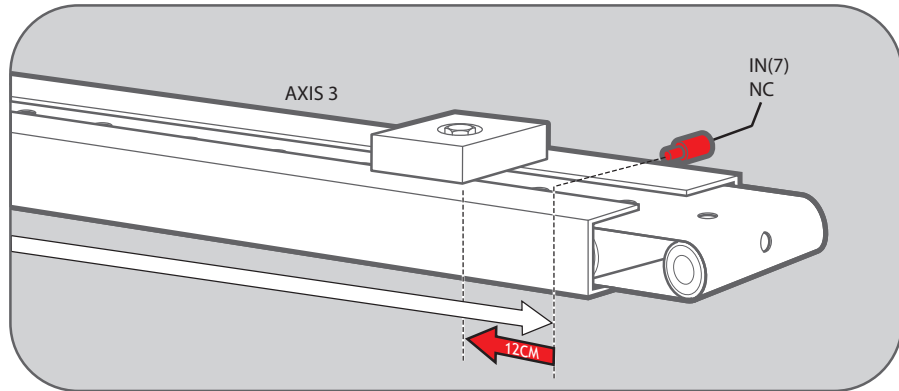
If the axis reaches either the forward limit switch or forward soft limit, the **FORWARD** will be cancelled and the axis will decelerate to a stop.

Example 1: Run an axis forwards. When an input signal is detected on input 12, bring the axis to a stop.



```
FORWARD
` wait for stop signal
WAIT UNTIL IN(12)=ON
CANCEL
WAIT IDLE
```

Example 2: Move an axis forwards until it hits the end limit switch, then move it in the reverse direction for 25 cm.



```

BASE(3)
FWD_IN=7 `limit switch connected to input 7
FORWARD
WAIT IDLE ` wait for motion to stop on the switch
MOVE(-25.0)
WAIT IDLE

```

Example 3: A machine that applies lids to cartons uses a simulated line shaft. This example sets up a virtual axis running forward, this is to simulate the line shaft. Axis 0 is then CONNECTed to this to run the conveyor. Axis 1 controls a vacuum roller that feeds the lids on to the cartons using the MOVELINK control.

```

BASE(4)
ATYPE=0           `Set axis 4 to virtual axis
REP_OPTION=1
SERVO=ON
FORWARD           `starts line shaft
BASE(0)
CONNECT(-1,4)    `Connects base 0 to virtual axis in reverse
WHILE IN(2)=ON
  BASE(1)         `Links axis 1 to the shaft in reverse
direction
  MOVELINK(-4000,2000,0,0,4,2,1000)
  WAIT IDLE
WEND
RAPIDSTOP

```

See Also: REVERSE

MHELICAL

Type: Axis Command.

Syntax: `MHELICAL(end1,end2,centre1,centre2,direction,distance3,[mode])`

Alternate Format: `MH()`

Description: Performs a helical move. Moves 2 orthogonal axes in such a way as to produce a circular arc at the tool point with a simultaneous linear move on a third axis. The first 5 parameters are similar to those of an `MOVECIRC` command. The sixth parameter defines the simultaneous linear move.

Parameters:

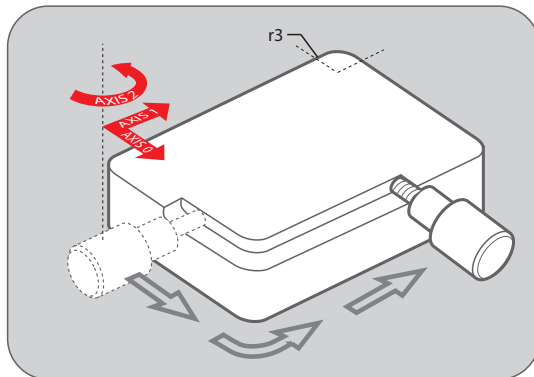
- end1:** position on **BASE** axis to finish at.
- end2:** position on next axis in **BASE** array to finish at.
- centre1:** position on **BASE** axis about which to move.
- centre2:** position on next axis in **BASE** array about which to move.
- direction:** The “direction” is a software switch which determines whether the arc is interpolated in a clockwise or anti- clockwise direction. The parameter is set to 1 or 0. See `MOVECIRC`.
- distance3:** The distance to move on the third axis in the **BASE** array axis in user units.
- mode:** 0 = Interpolate the 3rd axis with the main 2 axes when calculating path speed. (True helical path).
1= Interpolate only the first 2 axes for path speed, but move the 3rd axis in coordination with the other 2 axes. (Circular path with following 3rd axis).



*The first 4 distance parameters are scaled according to the current unit conversion factor for the **BASE** axis. The sixth parameter uses its own axis units.*

Example 1:

The command sequence follows a rounded rectangle path with axis 1 and 2. Axis 3 is the tool rotation so that the tool is always perpendicular to the product. The **UNITS** for axis 3 are set such that the axis is calibrated in degrees.



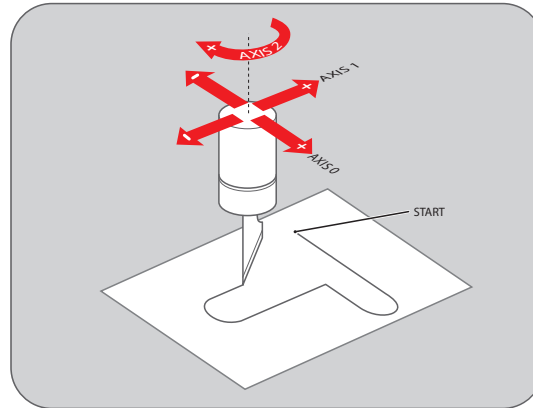
```

REP _ DIST AXIS(3)=360
REP _ OPTION AXIS(3)=ON
  \all 3 axes must be homed before starting
MERGE=ON
MOVEABS(360) AXIS(3) 'point axis 3 in correct starting
direction
WAIT IDLE AXIS(3)
MOVE(0,12)
MHELICAL(3,3,3,0,1,90)
MOVE(16,0)
MHELICAL(3,-3,0,-3,1,90)
MOVE(0,-6)
MHELICAL(-3,-3,-3,0,1,90)
MOVE(-2,0)
MHELICAL(-3,3,0,3,1,90)

```

Example 2:

A PVC cutter uses 2 axis similar to a xy plotter, a third axis is used to control the cutting angle of the knife. To keep the resultant cutting speed for the x and y axis the same when cutting curves, mode 1 is applied to the helical command.



```

BASE(0,1,2) : MERGE=ON `merge moves into one continuous movement
MOVE(50,0)
MHELICAL(0,-6,0,-3,1,180,1)
MOVE(-22,0)
WAIT IDLE
MOVE(-90) AXIS(2)      `rotate the knife after stopping at corner
WAIT IDLE AXIS(2)
MOVE(0,-50)
MHELICAL(-6,0,-3,0,1,180,1)
MOVE(0,50)
WAIT IDLE              `pause again to rotate the knife
MOVE(-90) AXIS(2)
WAIT IDLE AXIS(2)
MOVE(-22,0)
MHELICAL(0,6,0,3,1,180,1)
WAIT IDLE
    
```

MHELICALSP

Type: Axis Command.

Syntax: `MHPHERICAL({parameters}, mode)`

Description: Moves the three axis group defined in **BASE** along a spherical path with a vector speed determined by the **SPEED** set in the first axis of the **BASE** array. There are 2 modes of operation with the option of finishing the move at an endpoint different to the start, or returning to the start point to complete a circle. The path of the movement in 3D space can be defined either by specifying a point somewhere along the path, or by specifying the centre of the sphere.

Parameters:

mode:

- 0 = specify end point and mid point on curve.
- 1 = specify end point and centre of sphere.
- 2 = two mid point are specified and the curve completes a full circle.
- 3 = mid point on curve and centre of sphere are specified and the curve completes a full circle.



*If you specify the parameters for the third axis as 0 and assign it to a virtual, you can use **MSPHERICAL** to perform circular movements. This allows you to specify the arc without knowing the centre point.*

Syntax: `MSPHERICAL(endx, endy, endz, midx, midy, midz, 0)`

Description: Move the three axis, set in the **BASE** array through a section of a sphere by specifying the end point and a mid point on the curve.

Parameters:

- endx:** End position of the first axis.
- endy:** End position of the second axis.
- endz:** End position of the third axis.
- midx:** Mid position of the first axis.
- midy:** Mid position of the second axis.
- midz:** Mid position of the third axis.

Syntax: `MSPHERICAL(endx, endy, endz, centrex, centrey, centrez, 1)`

Description: Move the three axis, set in the **BASE** array through a section of a sphere by specifying the end point and the centre of the sphere. The profile will always go the shortest path to the endpoint, this may be clockwise or counterclockwise.



THE COORDINATES OF THE CENTRE POINT AND END POINT MUST NOT BE CO-LINEAR. SEMI-CIRCLES CANNOT BE DEFINED BY USING MODE 1 BECAUSE THE SPHERE CENTRE WOULD BE CO-LINEAR WITH THE ENDPOINT. IF CO-LINIER POINTS ARE SPECIFIED THE CONTROLLER WILL STOP THE PROGRAM WITH A RUN _ ERROR.

Parameters:

- `endx:` End position of the first axis.
- `endy:` End position of the second axis.
- `endz:` End position of the third axis.
- `centrex:` Centre position of the first axis.
- `centrey:` Centre position of the second axis.
- `centrez:` CentreMid position of the third axis.

Syntax: `MSPHERICAL(midx1, midy1, midz1, midx, midy, midz, 2)`

Description: Move the three axis, set in the **BASE** array through a full circle on a sphere by specifying two mid points of the curve. The profile will move through the first mid position, then the second and finally back to the start point.

Parameters:

- `midx1:` Second mid position of the first axis.
- `midy1:` Second mid position of the second axis.
- `midz1:` Second mid position of the third axis.
- `midx:` First mid position of the first axis.
- `midy:` First mid position of the second axis.
- `midz:` First mid position of the third axis.

Syntax: `MSPHERICAL(midx, midy, midz, centrex, centrey, centrez, 3)`

Description: Move the three axis, set in the **BASE** array through a full circle on a sphere by specifying a mid point and the centre of the sphere. The profile will start by heading in the shortest distance to the mid point, this enables you to define the direction.



THE COORDINATES OF THE CENTRE POINT AND MID POINT MUST NOT BE CO-LINEAR. IF CO-LINIER POINTS ARE SPECIFIED THE CONTROLLER WILL STOP THE PROGRAM WITH A RUN _ ERROR.

Parameters:

- midx:** Mid position of the first axis.
- midy:** Mid position of the second axis.
- midz:** Mid position of the third axis.
- centrex:** Centre position of the first axis.
- centrey:** Centre position of the second axis.
- centrez:** Centre position of the third axis.

Example 1: A move is needed that follows a spherical path which ends 30mm up in the Z direction:

```
BASE(3,4,5)
MSPHERICAL(30,0,30,8.7868,0,21.2132,0)
```

Example 2: A similar move that follows a spherical path but at 45 degrees to the Y axis which ends 30mm above the XY plane:

```
BASE(0,1,2)
MSPHERICAL(21.2132,21.2132,30,6.2132,6.2132,21.2132,0)
```

MOVE

Type: Axis Command

Syntax: `MOVE(distance1 [,distance2 [,distance3 [,distance4...]])`

Alternate Format: `MO()`

Description: Incremental move. One axis or multiple axes move at the programmed speed and acceleration for a distance specified as an increment from the end of the last specified move. The first parameter in the list is sent to the **BASE** axis, the second to the next axis in the **BASE** array, and so on.

In the multi-axis form, the speed and acceleration employed for the movement are taken from the first axis in the **BASE** group. The speeds of each axis are controlled so as to make the resulting vector of the movement run at the **SPEED** setting.

Uninterpolated, unsynchronised multi-axis motion can be achieved by simply placing **MOVE** commands on each axis independently. If needed, the target axis for an individual **MOVE** can be specified using the **AXIS()** command. This overrides the **BASE** axis setting for one **MOVE** only.

The distance values specified are scaled using the unit conversion factor axis parameter; **UNITS**. Therefore if, for example, an axis has 400 encoder edges/mm and **UNITS** for that axis are 400, the command `MOVE(12.5)` would move 12.5 mm. When **MERGE** is set to **ON**, individual moves in the same axis group are merged together to make a continuous path movement.

Parameters:

- `distance1:` distance to move on base axis from current position.
- `distance2:` distance to move on next axis in **BASE** array from current position.]
- [`distance3:` distance to move on next axis in **BASE** array from current position.]
- [`distance4:` distance to move on next axis in **BASE** array from current position.]



The maximum number of parameters is the number of axes on the controller.

Example 1: A system is working with a unit conversion factor of 1 and has a 1000 line encoder. Note that a 1000 line encoder gives 4000 edges/turn.

`MOVE(40000) ` move 10 turns on the motor.`

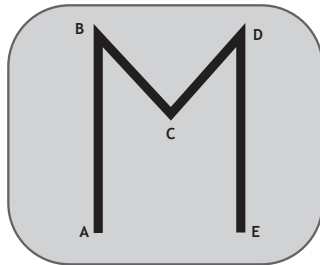
Example 2: Axes 3, 4 and 5 are to move independently (without interpolation). Each axis will move at its own programmed **SPEED**, **ACCEL** and **DECEL** etc.

```

`setup axis speed and enable
BASE(3)
SPEED=5000
ACCEL=100000
DECEL=150000
SERVO=ON
BASE(4)
SPEED=5000
ACCEL=150000
DECEL=560000
SERVO=ON
BASE(5)
SPEED=2000
ACCEL=320000
DECEL=352000
SERVO=ON
WDOG=ON
MOVE(10) AXIS(5)      `start moves
MOVE(10) AXIS(4)
MOVE(10) AXIS(3)
WAIT IDLE AXIS(5)    `wait for moves to finish
WAIT IDLE AXIS(4)
WAIT IDLE AXIS(3)

```

Example 3: An X-Y plotter can write text at any position within its working envelope. Individual characters are defined as a sequence of moves relative to a start point so that the same commands may be used regardless of the plot origin. The command subroutine for the letter 'M' might be:



```

write _m:
  MOVE(0,12)  `move A > B
  MOVE(3,-6)  `move B > C
  MOVE(3,6)   `move C > D
  MOVE(0,-12) `move D > E
RETURN

```

MOVEABS

Type: *Motion Command.*

Syntax: MOVEABS(position1[, position2[, position3[, position4...]])

Alternate Format: MA()

Description: Absolute position move. Move one axis or multiple axes to position(s) referenced with respect to the zero (home) position. The first parameter in the list is sent to the axis specified with the **AXIS** command or to the current **BASE** axis, the second to the next axis, and so on.

In the multi-axis form, the speed, acceleration and deceleration employed for the movement are taken from the first axis in the **BASE** group. The speeds of each axis are controlled so as to make the resulting vector of the movement run at the **SPEED** setting.

Uninterpolated, unsynchronised multi-axis motion can be achieved by simply placing **MOVEABS** commands on each axis independently. If needed, the target axis for an individual **MOVEABS** can be specified using the **AXIS()** command. This overrides the **BASE** axis setting for one **MOVEABS** only.

The values specified are scaled using the unit conversion factor axis parameter; **UNITS**. Therefore if, for example, an axis has 400 encoder edges/mm the **UNITS** for that axis is 400. The command **MOVEABS(6)** would then move to a position 6 mm from the zero position. When **MERGE** is set to **ON**, absolute and relative moves are merged together to make a continuous path movement.

Parameters: **position1:** position to move to on base axis.

position2: position to move to on next axis in **BASE** array.

position3: position to move to on next axis in **BASE** array.

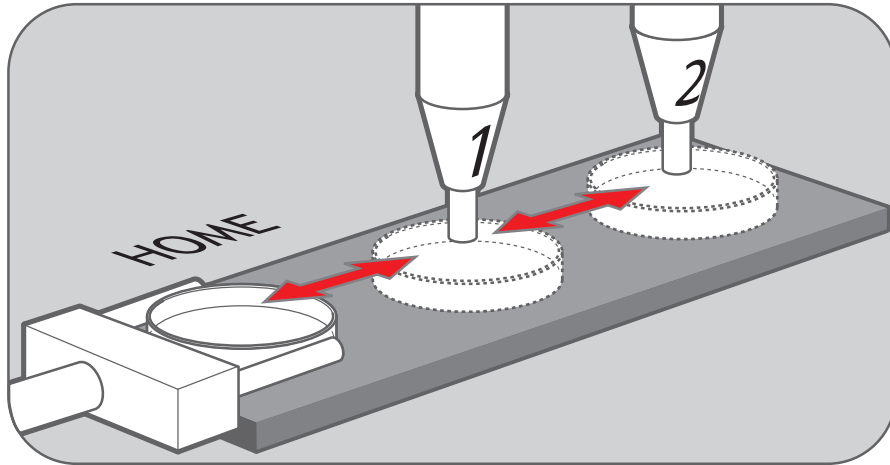
position4: position to move to on next axis in **BASE** array.

The **MOVEABS** command can interpolate up to the full number of axes available on the controller.

The position of the axes' zero (home) positions can be changed by the commands: **OFFPOS**, **DEFPOS**, **REP _ DIST**, **REP _ OPTION**, and **DATUM**

Example 1:

A machine must move to one of 3 positions depending on the selection made by 2 switches. The options are home, position 1 and position 2 where both switches are off, first switch on and second switch on respectively. Position 2 has priority over position 1.



```

`define absolute positions
home=1000
position _ 1=2000
position _ 2=3000

WHILE IN(run_switch)=ON
  IF IN(6)=ON THEN           `switch 6 selects position 2
    MOVEABS(position _ 2)
    WAIT IDLE
  ELSEIF IN(7)=ON THEN      `switch 7 selects position 1
    MOVEABS(position _ 1)
    WAIT IDLE
  ELSE
    MOVEABS(home)
    WAIT IDLE
  ENDIF
WEND

```

Example 2:

An X-Y plotter has a pen carousel whose position is fixed relative to the plotter absolute zero position. To change pen an absolute move to the carousel position will find the target irrespective of the plot position when commanded.

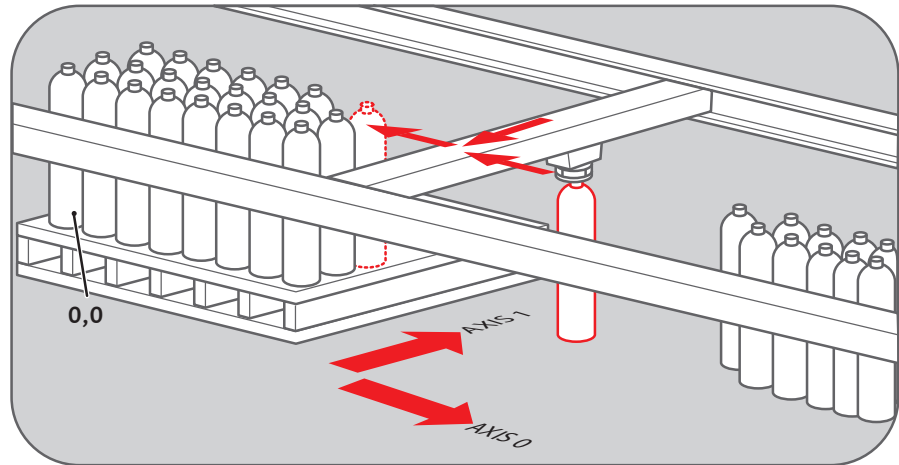
```

MOVEABS(28.5,350) `move to just outside the pen holder area
WAIT IDLE
SPEED = pen_pickup_speed
MOVEABS(20.5,350) `move in to pick up the pen

```

Example 3:

A pallet consists of a 6 by 8 grid in which gas canisters are inserted 185mm apart by a packaging machine. The canisters are picked up from a fixed point. The first position in the pallet is defined as position 0,0 using the `DEFPOS()` command. The part of the program to position the canisters in the pallet is:



```

FOR x=0 TO 5
  FOR y=0 TO 7
    MOVEABS(-340,-516.5)           'move to pick-up point
    WAIT IDLE
    GOSUB pick                     'call pick up subroutine
    PRINT "Move to Position: ";x*6+y+1
    MOVEABS(x*185,y*185)          'move to position in grid
    WAIT IDLE
    GOSUB place                   'call place down subroutine
  NEXT y
NEXT x
    
```

Example 4:

Using `MOVEABS` with `REPDIST` to move to a final position.

```

REPDIST = 360
DEFPOS(0)
MOVEABS(300) 'will move through 300 deg to 300
MOVEABS(200) 'will move back 100 deg to 200
MOVEABS(370) 'will move through 170 deg to 10 crossing repdist
MOVEABS(350) 'will move through 340 deg to 350
    
```



If you want to move in the shortest direction to the absolute position use `MOVETANG`.

See Also:

`MOVETANG`

MOVEABSSP

Type: Axis Command.

Syntax: `MOVEABSSP(position1[, position2[, position3[, position4...]])`

Description: Works as `MOVEABS` and additionally allows vector speed to be changed when using multiple moves in the look ahead buffer when `MERGE=ON`, using additional parameters `FORCE _ SPEED`, `ENDMOVE _ SPEED` and `STARTMOVE _ SPEED`.

Absolute moves are converted to incremental moves as they enter the buffer. This is essential as the vector length is required to calculate the start of deceleration. It should be noted that if any move in the buffer is cancelled by the programmer, the absolute position will not be achieved.

Parameters:

- `position1`: position to move to on base axis.
- `position2`: position to move to on next axis in `BASE` array.
- `position3`: position to move to on next axis in `BASE` array.
- `position4`: position to move to on next axis in `BASE` array.



The maximum number of parameters is the number of axes available on the controller.

Example 1: In a series of buffered moves using the look ahead buffer with `MERGE=ON`, an absolute move is required where the incoming vector speed is 40units/second and the finishing vector speed is 20 units/second.

```
FORCE _ SPEED=40
ENDMOVE _ SPEED=20
MOVEABSSP(100,100)
```

See Also: `MOVEABS`

MOVECIRC

Type: *Motion Command.*

Syntax: MOVECIRC(end1, end2, centre1, centre2, direction)

Alternate Format: MC()

Description: Moves 2 orthogonal axes in such a way as to produce a circular arc at the tool point. The length and radius of the arc are defined by the five parameters in the command line. The move parameters are always relative to the end of the last specified move. This is the start position on the circle circumference. Axis 1 is the current **BASE** axis. Axis 2 is the next axis in the **BASE** array. The first 4 distance parameters are scaled according to the current unit conversion factor for the **BASE** axis.

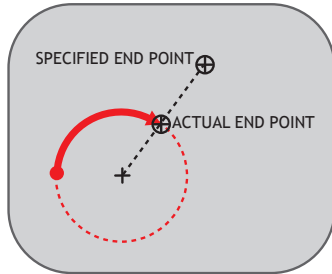
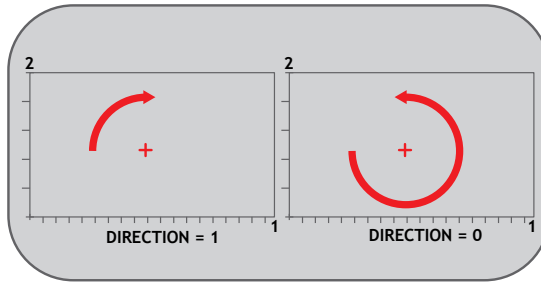
In order for the MOVECIRC() command to be correctly executed, the two axes generating the circular arc must have the same number of encoder pulses/linear axis distance. If this is not the case it is possible to adjust the encoder scales in many cases by using **ENCODER _ RATIO** or **STEP _ RATIO**.

If the end point specified is not on the circular arc. The arc will end at the angle specified by a line between the centre and the end point.

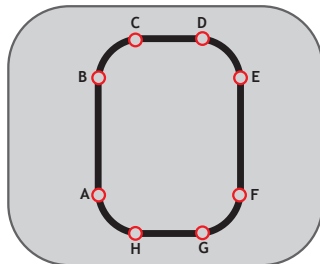
Neither axis may cross the set absolute repeat distance (**REP _ DIST**) during a MOVECIRC. Doing so may cause one or both axes to jump or for their **FE** value to exceed **FE _ LIMIT**.

Parameters:

- end1:** position on **BASE** axis to finish at.
- end2:** position on next axis in **BASE** array to finish at.
- centre1:** position on **BASE** about which to move.
- centre2:** position on next axis in **BASE** array about which to move.
- direction:** The “direction” is a software switch which determines whether the arc is interpolated in a clockwise or anti- clockwise direction.



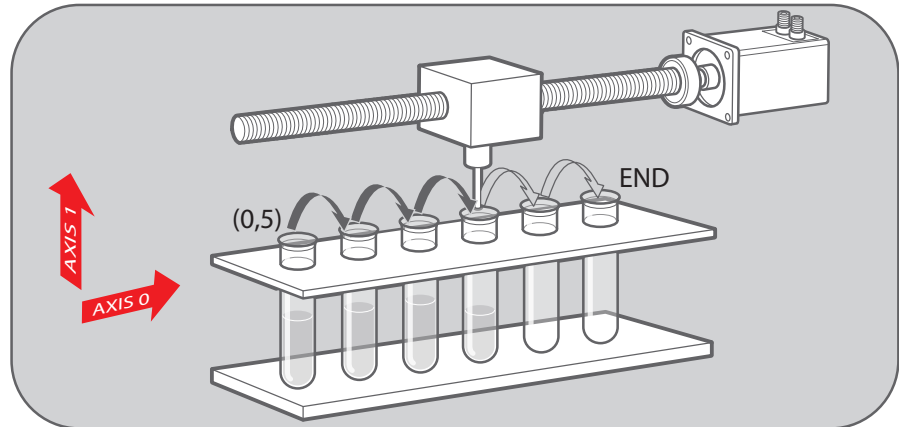
Example 1: The command sequence to plot the letter '0' might be:



<code>MOVE(0,6)</code>	<code>`move A -> B</code>
<code>MOVECIRC(3,3,3,0,1)</code>	<code>`move B -> C</code>
<code>MOVE(2,0)</code>	<code>`move C -> D</code>
<code>MOVECIRC(3,-3,0,-3,1)</code>	<code>`move D -> E</code>
<code>MOVE(0,-6)</code>	<code>`move E -> F</code>
<code>MOVECIRC(-3,-3,-3,0,1)</code>	<code>`move F -> G</code>
<code>MOVE(-2,0)</code>	<code>`move G -> H</code>
<code>MOVECIRC(-3,3,0,3,1)</code>	<code>`move H -> A</code>

Example 2:

A machine is required to drop chemicals into test tubes. The nozzle can move up and down as well as along its rail. The most efficient motion is for the nozzle to move in an arc between the test tubes.



```

BASE(0,1)
MOVEABS(0,5)           'move to position above first tube
MOVEABS(0,0)           'lower for first drop
WAIT IDLE
OP(15,ON)              'apply dropper
WA(20)
OP(15,OFF)
FOR x=0 TO 5
  MOVECIRC(5,0,2.5,0,1) 'arc between the test tubes
  WAIT IDLE
  OP(15,ON)             'Apply dropper
  WA(20)
  OP(15,OFF)
NEXT x
MOVECIRC(5,5,5,0,1)   'move to rest position)
    
```

MOVECIRCSP

Type: Axis Command.

Syntax: MOVECIRCSP(end1, end2, centre1, centre2, direction)

Description: Works as MOVECIRC and additionally allows vector speed to be changed when using multiple moves in the look ahead buffer when MERGE=ON, using additional parameters FORCE _ SPEED and ENDMOVE _ SPEED.

Example 1: In a series of buffered moves using the look ahead buffer with MERGE=ON, a circular move is required where the incoming vector speed is 40units/second and the finishing vector speed is 20 units/second.

```
FORCE _ SPEED=40
ENDMOVE _ SPEED=20
MOVECIRCSP(100,100,0,100,1)
```

See Also: MOVECIRC

MOVELINK

Type: Axis Command.

Syntax: MOVELINK (distance, link dist, link acc, link dec, link axis[, link options][, link pos]).

Alternate Format: ML()

Description: The linked move command is designed for controlling movements such as:

- Synchronization to conveyors
- Flying shears
- Thread chasing, tapping etc.
- Coil winding

The motion consists of a linear movement with separately variable acceleration and deceleration phases linked via a software gearbox to the **MEASURED** position (**MPOS**) of another axis. The command uses the **BASE()** and **AXIS()**, and unit conversion factors in a similar way to other move commands.



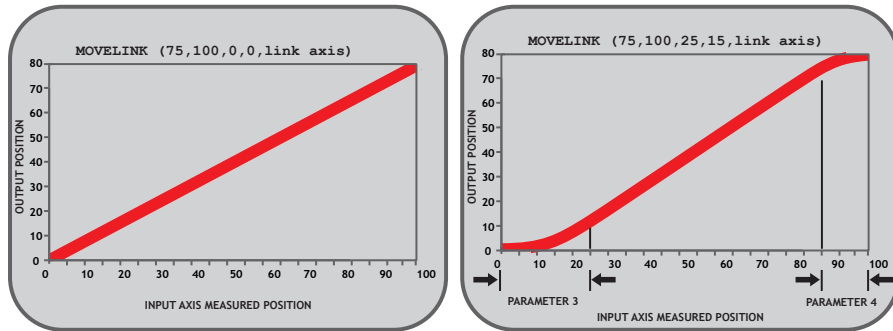
The “link” axis may move in either direction to drive the output motion. The link distances specified are always positive.

Parameters:	distance:	incremental distance in user units to be moved on the current base axis, as a result of the measured movement on the “input” axis which drives the move.
	link dist:	positive incremental distance in user units which is required to be measured on the “link” axis to result in the motion on the base axis.
	link acc:	positive incremental distance in user units on the input axis over which the base axis accelerates.
	link dec:	positive incremental distance in user units on the input axis over which the base axis decelerates.



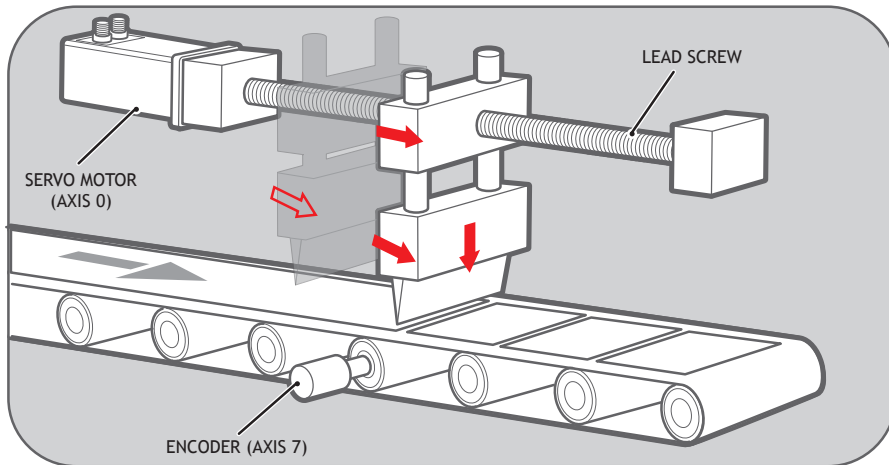
***NOTE:** If the sum of parameter 3 and parameter 4 is greater than parameter 2, they are both reduced in proportion until they equal parameter 2.*

link axis:	Specifies the axis to “link” to. It should be set to a value between 0 and the number of available axes.
link options:	<ul style="list-style-type: none"> 1 link commences exactly when registration event occurs on link axis. 2 link commences at an absolute position on link axis (see link start parameter). 4 MOVELINK repeats automatically and bi-directional when this bit is set. (This mode can be cleared by setting bit 1 of the REP _ OPTION axis parameter). 32 Link is only active during positive moves on the link axis.
link pos:	This parameter is the absolute position where the MOVELINK link is to be started when parameter 6 is set to 2.



Example 1:

A flying shear cuts a long sheet of paper into cards every 160 m whilst moving at the speed of the material. The shear is able to travel up to 1.2 metres of which 1m is used in this example. The paper distance is measured by an encoder, the unit conversion factor being set to give units of metres on both axes: (Note that axis 7 is the link axis)



```

WHILE IN(2)=ON
    MOVELINK(0,150,0,0,7) \ dwell (no movement) for 150m
    MOVELINK(0.3,0.6,0.6,0,7) \ accelerate to paper speed
    MOVELINK(0.7,1.0,0,0.6,7) \ track the paper then
decelerate
WAIT LOADED \ wait until acceleration movelink is
finished
OP(8,ON) \ activate cutter
MOVELINK(-1.0,8.4,0.5,0.5,7) \ retract cutter back to start
    
```

```

                WAIT LOADED
                OP(8,OFF)      ` deactivate cutter at end of outward
stroke
                WEND

```

In this program the controller firstly waits for the roll to feed out 150m in the first line. After this distance the shear accelerates up to match the speed of the paper, moves at the same speed then decelerates to a stop within the 1m stroke. This movement is specified using two separate **MOVELINK** commands. This allows the program to wait for the next move buffer to be clear, **NTYPE=0**, which indicates that the acceleration phase is complete. Note that the distances on the measurement axis (link distance in each **MOVELINK** command): 150, 0.8, 1.0 and 8.2 add up to 160m.

To ensure that speed and positions of the cutter and paper match during the cut process the parameters of the **MOVELINK** command must be correct: It is normally easiest to consider the acceleration, constant speed and deceleration phases separately then combine them as required:

Rule 1: In an acceleration phase to a matching speed the link distance should be twice the movement distance. The acceleration phase could therefore be specified alone as:

```
MOVELINK(0.3,0.6,0.6,0,1)' move is all accel
```

Rule 2: In a constant speed phase with matching speed the two axes travel the same distance so distance to move should equal the link distance. The constant speed phase could therefore be specified as:

```
MOVELINK(0.4,0.4,0,0,1)' all constant speed
```

The deceleration phase is set in this case to match the acceleration:

```
MOVELINK(0.3,0.6,0,0.6,1)' all decel
```

The movements of each phase could now be added to give the total movement.

```
MOVELINK(1,1.6,0.6,0.6,1)' Same as 3 moves above
```

But in the example above, the acceleration phase is kept separate:

```
MOVELINK(0.3,0.6,0.6,0,1)
MOVELINK(0.7,1.0,0,0.6,1)
```

This allows the output to be switched on at the end of the acceleration phase.

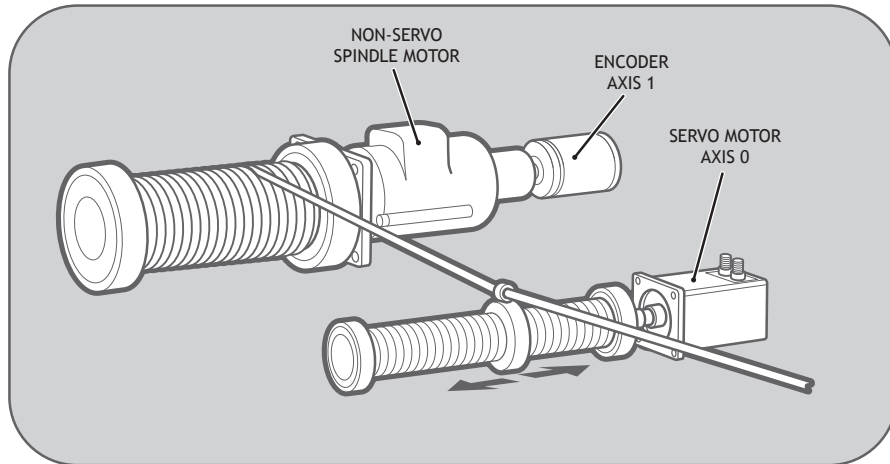
Example 2: Exact Ratio Gearbox

MOVELINK can be used to create an exact ratio gearbox between two axes. Suppose it is required to create gearbox link of 4000/3072. This ratio is inexact (1.30208333) and if entered into a **CONNECT** command the axes will slowly creep out of synchronisation. Setting the “link option” to 4 allows a continuously repeating **MOVELINK** to eliminate this problem:

```
MOVELINK(4000,3072,0,0,linkaxis,4)
```

Example 3: Coil Winding

In this example the unit conversion factors **UNITS** are set so that the payout movements are in mm and the spindle position is measured in revolutions. The payout eye therefore moves 50mm over 25 revolutions of the spindle with the command **MOVELINK**(50,25,0,0,linkax). If it were desired to accelerate up over the first spindle revolution and decelerate over the final 3 the command would be



```
MOVELINK(50,25,1,3,linkax).
OP(motor,ON)    '- Switch spindle motor on
FOR layer=1 TO 10
    MOVELINK(50,25,0,0,1)
    MOVELINK(-50,25,0,0,1)
NEXT layer
WAIT IDLE
OP(motor,OFF)
```

MOVEMODIFY

Type: Axis Command.

Syntax: `MOVEMODIFY(position)`

Alternate Format: `MM()`

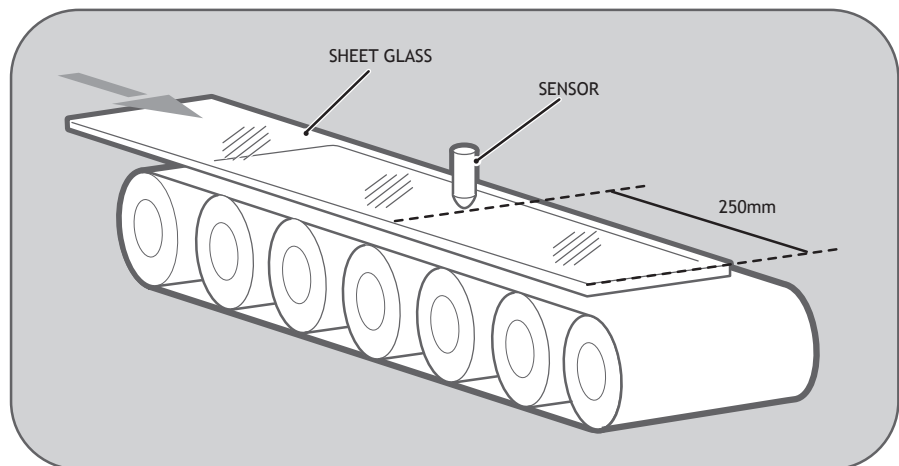
Description: `MOVEMODIFY` will change the absolute end position of the `MOVE`, `MOVEABS`, `MOVESP`, `MOVEABSSP` or `MOVEMODIFY` in the `MTYPE`. If there is no motion command in the `MTYPE` or the `MTYPE` is not a linear move then `MOVEMODIFY` is loaded.

If the change in end position requires a change in direction the move in `MTYPE` is **CANCEL**ed. This will use `DECEL` unless `FASTDEC` has been specified.

If there are multiple buffered moves the `MOVEMODIFY` will only act on the command in front of it in the buffer.

Parameters: `position:` Absolute position for the current move to complete at.

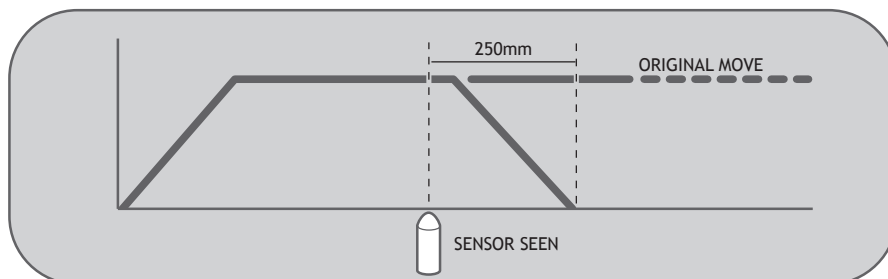
Example 1: A sheet of glass is fed on a conveyor and is required to be stopped 250mm after the leading edge is sensed by a proximity switch. The proximity switch is connected to the registration input:



`MOVE(10000)`

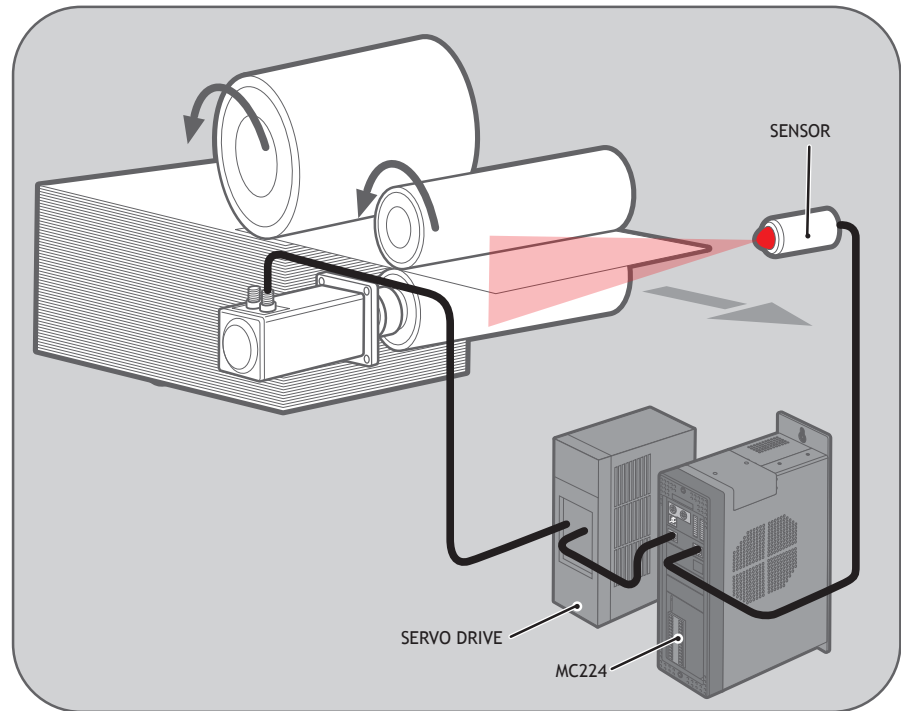
``Start a long move on conveyor`


```
REGIST(3)           `set up registration
WAIT UNTIL MARK     `MARK goes TRUE when sensor detects glass
edge
OFFPOS = -REG_POS   `set position where mark was seen to 0
WAIT UNTIL OFFPOS=0 `wait for OFFPOS to take effect
MOVEMODIFY(250)     `change move to stop at 250mm
```



Example 2:

A paper feed system slips. To counteract this, a proximity sensor is positioned one third of the way into the movement. This detects at which position the paper passes and so how much slip has occurred. The move is then modified to account for this variation.



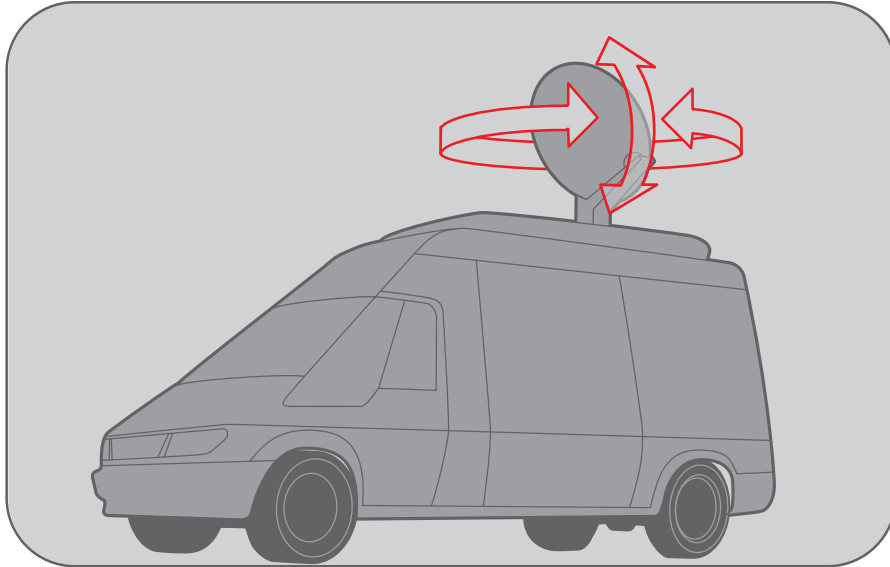
```

paper_length=4000
DEFPOS(0)
REGIST(3)
MOVE(paper_length)
WAIT UNTIL MARK
slip=REG_POS-(paper_length/3)
offset=slip*3
MOVEMODIFY(paper_length+offset)

```

Example 3:

A satellite receiver sits on top of a van; it has to align correctly to the satellite from data processed in a computer. This information is sent to the controller through the serial link and sets VR's 0 and 1. This information is used to control the two axes. **MOVEMODIFY** is used so that the position can be continuously changed even if the previous set position has not been achieved.



```

bearing=0
`set labels for VRs
elevation=1
UNITS AXIS(0)=360/counts_per_rev0
UNITS AXIS(1)=360/counts_per_rev1
WHILE IN(2)=ON
    MOVEMODIFY(VR(bearing))AXIS(0)
                                `adjust bearing to match VR0
    MOVEMODIFY(VR(elevation))AXIS(1)
                                `adjust elev to match VR1
    WA(250)
WEND
RAPIDSTOP
`stop movement
WAIT IDLE AXIS(0)
MOVEABS(0) AXIS(0)
WAIT IDLE AXIS(1)
MOVEABS(0) AXIS (1)
                                `return to transport position

```

See also: **ENDMOVE**

MOVESP

Type: Axis Command

Syntax: `MOVESP(distance1[,distance2[,distance3[,distance4]])`

Description: Works as **MOVE** and additionally allows vector speed to be changed when using multiple moves in the look ahead buffer when **MERGE=ON**, using additional parameters **FORCE _ SPEED**, **ENDMOVE _ SPEED** and **STARTMOVE _ SPEED**.

Parameters:

- distance1:** distance to move on base axis from current position.
- distance2:** distance to move on next axis in **BASE** array from current position.
- distance3:** distance to move on next axis in **BASE** array from current position.
- distance4:** distance to move on next axis in **BASE** array from current position.



The maximum number of parameters is the number of axes available on the controller.

Example: In a series of buffered moves using the look ahead buffer with **MERGE=ON**, an incremental move is required where the incoming vector speed is 40units/second and the finishing vector speed is 20 units/second.

```
FORCE _ SPEED=40
ENDMOVE _ SPEED=20
MOVESP(100,100)
```

See also: **MOVE**

MOVETANG

Type: Axis Command

Syntax: `MOVETANG(absolute _ position, [link _ axis])`

Description: Moves the axis to the required position using the programmed `SPEED`, `ACCEL` and `DECEL` for the axis. The direction of movement is determined by a calculation of the shortest path to the position assuming that the axis is rotating and that `REP _ DIST` has been set to π radians (180 degrees) and that `REP _ OPTION=0`.

The `REP _ DIST` value will depend on the `UNITS` value and the number of steps representing π radians. For example if the rotary axis has 4000 pulses/turn and `UNITS=1` the `REP _ DIST` value would be 2000.

If a `MOVETANG` command is running and another `MOVETANG` is executed for the same axis, the original command will not stop, but the endpoint will become the new absolute position.

Parameters: `absolute _ position:` The absolute position to be set as the endpoint of the move. Value must be within the range $-\pi$ to $+\pi$ in the units of the rotary axis. For example if the rotary axis has 4000 pulses/turn, the `UNITS` value=1 and the angle required is $\pi/2$ (90 deg) the position value would be 1000.

`link _ axis` An optional link axis may be specified. When a link_ axis is specified the system software calculates the absolute position required each servo cycle based on the link axis `TANG _ DIRECTION`. The `TANG _ DIRECTION` is multiplied by the `REP _ DIST/ π` to calculate the required position. Note that when using a link_axis the absolute_position parameter becomes unused. The position is copied every servo cycle until the `MOVETANG` is CANCELLED.

Example 1: An X-Y positioning system has a stylus which must be turned so that it is facing in the same direction as it is travelling at all times. A tangential control routine is run in a separate process.

```
BASE(0,1)
WHILE TRUE
  angle=TANG _ DIRECTION
  MOVETANG(angle) AXIS(2)
WEND
```

Example 2: An X-Y positioning system has a stylus which must be turned so that it is facing in the same direction as it is travelling at all times.

The XY axis pair are axes 4 and 5. The tangential stylus axis is 2:

```
MOVETANG(0,4) AXIS(2)
```

Example 3: An X-Y cutting table has a “pizza wheel” cutter which must be steered so that it is always aligned with the direction of travel. The main X and Y axes are controlled by *Motion Coordinator* axes 0 and 1, and the pizza wheel is turned by axis 2.

Control of the Pizza Wheel is done in a separate program from the main X-Y motion program. In this example the steering program also does the axis initialisation.

```
Program TC_SETUP.BAS:
`Set up 3 axes for Tangential Control

WDOG=OFF
BASE(0)
P_GAIN=0.9
VFF_GAIN=12.85
UNITS=50 `set units for mm
SERVO=ON

BASE(1)
P_GAIN=0.9
VFF_GAIN=12.30
UNITS=50 `units must be the same for both axes
SERVO=ON

BASE(2)
UNITS=1 ` make units 1 for the setting of rep_dist
REP_DIST=2000 `encoder has 4000 edges per rev.
REP_OPTION=0
UNITS=4000/(2*PI) `set units for Radians
SERVO=ON

WDOG=ON
` Home the 3rd axis to its Z mark
DATUM(1) AXIS(2)
WAIT IDLE
WA(10)

`start the tangential control routine
BASE(0,1) `define the pair of axes which are for X and Y
` start the tangential control
BASE(2)
MOVETANG(0, 0) ` use axes 0 and 1 as the linked pair

Program MOTION.BAS:
`program to cut a square shape with rounded corners
MERGE=ON
```

```
SPEED=300

nobuf=FALSE           `when true, the moves are not buffered
size=120              `size of each side of the square
c=30                  `size (radius) of quarter circles on
each corner

DEFPOS(0,0)
WAIT UNTIL OFFPOS=0
WA(10)

MOVEABS(10,10+c)
REPEAT
  MOVE(0,size)
  MOVECIRC(c,c,c,0,1)
  IF nobuf THEN WAIT IDLE:WA(2)
  MOVE(size,0)
  MOVECIRC(c,-c,0,-c,1)
  IF nobuf THEN WAIT IDLE:WA(2)
  MOVE(0,-size)
  MOVECIRC(-c,-c,-c,0,1)
  IF nobuf THEN WAIT IDLE:WA(2)
  MOVE(-size,0)
  MOVECIRC(-c,c,0,c,1)
  IF nobuf THEN WAIT IDLE:WA(2)
UNTIL FALSE
```

MSPHERICAL

Type: Axis Command

Syntax: MSPHERICAL({parameters}, mode)

Description: Moves the three axis group defined in **BASE** along a spherical path with a vector speed determined by the **SPEED** set in the first axis of the **BASE** array. There are 2 modes of operation with the option of finishing the move at an endpoint different to the start, or returning to the start point to complete a circle. The path of the movement in 3D space can be defined either by specifying a point somewhere along the path, or by specifying the centre of the sphere.

Parameters:

- mode:** 0 = specify end point and mid point on curve.
- 1 = specify end point and centre of sphere.
- 2 = two mid point are specified and the curve completes a full circle.
- 3 = mid point on curve and centre of sphere are specified and the curve completes a full circle.



If you specify the parameters for the third axis as 0 and assign it to a virtual, you can use **MSPHERICAL** to perform circular movements. This allows you to specify the arc without knowing the centre point.

Syntax: `MSPHERICAL(endx, endy, endz, midx, midy, midz, 0)`

Description: Move the three axis, set in the **BASE** array through a section of a sphere by specifying the end point and a mid point on the curve.

Parameters:

- endx:** End position of the first axis.
- endy:** End position of the second axis.
- endz:** End position of the third axis.
- midx:** Mid position of the first axis.
- midy:** Mid position of the second axis.
- midz:** Mid position of the third axis.

Syntax: `MSPHERICAL(endx, endy, endz, centrex, centrey, centrez, 1)`

Description: Move the three axis, set in the **BASE** array through a section of a sphere by specifying the end point and the centre of the sphere. The profile will always go the shortest path to the endpoint, this may be clockwise or counter clockwise.



THE COORDINATES OF THE CENTRE POINT AND END POINT MUST NOT BE CO-LINEAR. SEMI-CIRCLES CANNOT BE DEFINED BY USING MODE 1 BECAUSE THE SPHERE CENTRE WOULD BE CO-LINEAR WITH THE ENDPOINT. IF CO-LINEAR POINTS ARE SPECIFIED THE CONTROLLER WILL STOP THE PROGRAM WITH A RUN _ ERROR.

Parameters:

- endx:** End position of the first axis.
- endy:** End position of the second axis.
- endz:** End position of the third axis.
- centrex:** Centre position of the first axis.
- centrey:** Centre position of the second axis.
- centrez:** Centre position of the third axis.

Syntax: `MSPHERICAL(midx1, midy1, midz1, midx, midy, midz, 2)`

Description: Move the three axis, set in the **BASE** array through a full circle on a sphere by specifying two mid points of the curve. The profile will move through the first mid position, then the second and finally back to the start point.

Parameters:

- `midx1:` Second mid position of the first axis.
- `midy1:` Second mid position of the second axis.
- `midz1:` Second mid position of the third axis.
- `midx:` First mid position of the first axis.
- `midy:` First mid position of the second axis.
- `midz:` First mid position of the third axis.

Syntax: `MSPHERICAL(midx, midy, midz, centrex, centrey, centrez, 3)`

Description: Move the three axis, set in the **BASE** array through a full circle on a sphere by specifying a mid point and the centre of the sphere. The profile will start by heading in the shortest distance to the mid point, this enables you to define the direction.

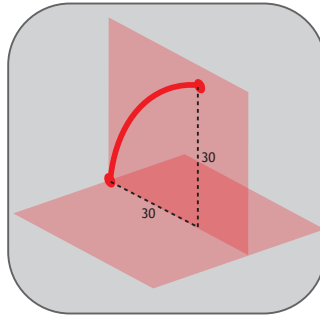


THE COORDINATES OF THE CENTRE POINT AND MID POINT MUST NOT BE CO-LINEAR. IF CO-LINEAR POINTS ARE SPECIFIED THE CONTROLLER WILL STOP THE PROGRAM WITH A RUN _ ERROR.

Parameters:

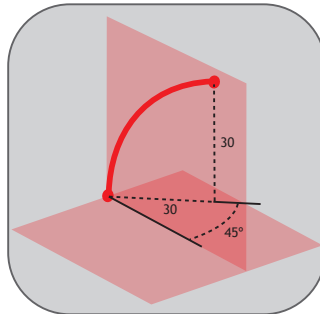
- `midx:` Mid position of the first axis.
- `miy:` Mid position of the second axis.
- `midz:` Mid position of the third axis.
- `centrex:` Centre position of the first axis.
- `centrey:` Centre position of the second axis.
- `centrez:` Centre position of the third axis.

Example 1: A move is needed that follows a spherical path which ends 30mm up in the Z direction:



```
BASE(3,4,5)
MSPHERICAL(30,0,30,8.7868,0,21.2132,0)
```

Example 2: A similar move that follows a spherical path but at 45 degrees to the Y axis which ends 30mm above the XY plane:



```
BASE(0,1,2)
MSPHERICAL(21.2132,21.2132,30,6.2132,6.2132,21.2132,0)
```

MSPHERICALSP

Type: Axis Command

Syntax: `MSPHERICALSP({parameters}, mode)`

Description: Performs a spherical move the same as **MSPHERICAL** and additionally allows vector speed to be changed when using multiple moves in the look ahead buffer when **MERGE=ON**, using additional parameters **FORCE _ SPEED**, **ENDMOVE _ SPEED** and **STARTMOVE _ SPEED**.

Example 1: A move is needed that follows a spherical path which ends 30mm up in the Z direction, the profile should decelerate from the previous move so that it is performed at 30units/second:

```
BASE(3,4,5)
FORCE _ SPEED=30
ENDMOVE _ SPEED=30
MSPHERICALSP(30,0,30,8.7868,0,21.2132,0)
```

See Also: **MSPHERICAL**

RAPIDSTOP

Type: Axis Command

Syntax: `RAPIDSTOP`

Alternate Format: `RS`

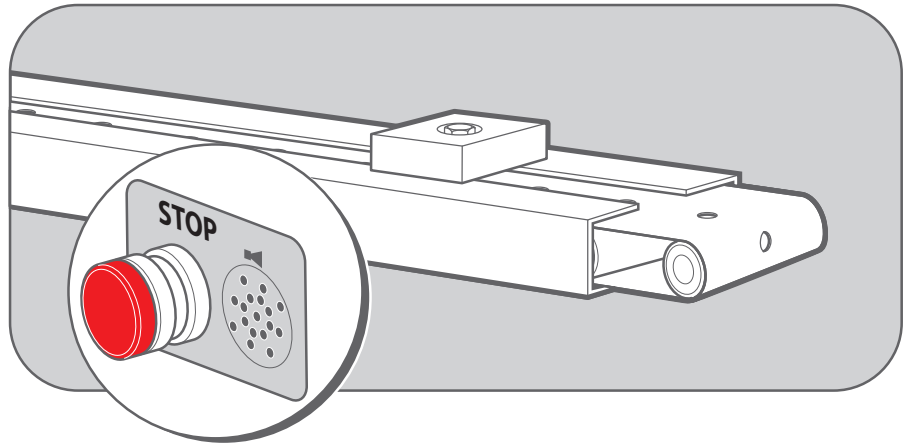
Description: The **RAPIDSTOP** command cancels the currently executing move on **ALL** axes. Velocity profiled moves, for example; **FORWARD**, **REVERSE**, **MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL**, **MOVEMODIFY**, will be ramped down at the programmed **DECEL** or **FAST _ DEC** rate then terminated. Other move types will be terminated immediately.

The next-move buffers and the process buffers are **NOT** cleared.



RAPIDSTOP WILL ONLY CANCEL THE PRESENTLY EXECUTING MOVES. IF FURTHER MOVES ARE BUFFERED THEY WILL THEN BE LOADED AND THE AXIS WILL NOT STOP.

Example 1: Implementing a stop override button that cuts out all motion.



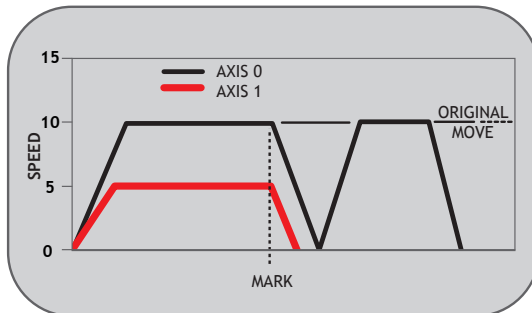
```

CONNECT (1,0) AXIS(1)    `axis 1 follows axis 0
BASE(0)
REPEAT
  MOVE(1000) AXIS (0)
  MOVE(-10000) AXIS (0)
  MOVE(100000) AXIS (0)
UNTIL IN (2)=OFF        `stop button pressed?
RAPIDSTOP
WA(10)                  `wait to allow running move to decel and be
terminated
RAPIDSTOP `cancel the second buffered move
WA(10)
RAPIDSTOP `cancel the third buffered move

```

Example 2:

Using **RAPIDSTOP** to cancel a **MOVE** on the main axis and a **FORWARD** on the second axis. After the axes have stopped, a **MOVEABS** is applied to re-position the main axis.

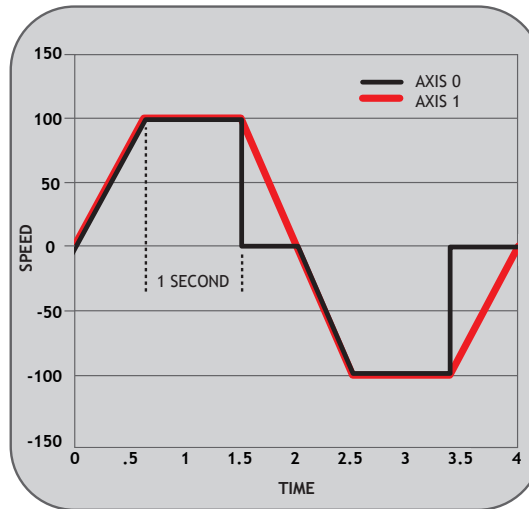


```

BASE(0)
REGIST(3)
FORWARD AXIS(1)
MOVE (100000) `apply a long move
WAIT UNTIL MARK
RAPIDSTOP
WAIT IDLE      `for MOVEABS to be accurate, the axis must stop
MOVEABS(3000)

```

Example 3: Using **RAPIDSTOP** to break a connect, and stop motion. The connected axis stops immediately on the **RAPIDSTOP** command, the forward axis decelerates at the decel value.



```

BASE(0)
CONNECT(1,1)
FORWARD AXIS(1)
WAIT UNTIL VPSPEED=SPEED `let the axis get to full speed
WA(1000)
RAPIDSTOP
WAIT IDLE AXIS(1)           `wait for axis 1 to decel
CONNECT(1,1)               `re-connect axis 0
REVERSE AXIS(1)
WAIT UNTIL VPSPEED=SPEED
WA(1000)
RAPIDSTOP
WAIT IDLE AXIS(1)
    
```

See Also: CANCEL, FAST _ DEC

REGIST

Type: Axis Command

Syntax: `REGIST(mode [,parameters])`

Description: The **REGIST** command initiates a capture of an axis position when it sees a registration input or the Z mark on the encoder. Once a registration event is captured **MARK** is set and the position and speed at the event can be read back.



See the Hardware Chapter to understand which registration mode your hardware supports.

Filtering can be applied to the input as well as defining a window of where to capture.

Hardware registration captures the encoder count against the registration input in hardware

Time based registration captures the time of the registration event and interpolates the position values being sent back from the drive against it.



Although all modes are available it is recommended to use modes 20-22 for new applications. Other modes have been provided for compatibility with older products.


The **REGIST** command must be re-issued for each position capture.

Parameters:

mode:	1..4	= Single channel hardware registration.
	5	= reserved.
	6..13	= Dual channel hardware registration.
	14..17	= Single channel hardware registration.
	20	= Single channel hardware registration.
	21	= Single channel time based registration.
	22	= 8 channel hardware registration.
	23	= Sets 2.4usec minimum pulse width.
	24	= Sets 0.15usec minimum pulse width (default).
	32..39	= Rising edge on time based registration.
	64..71	= Falling edge on time based registration.

Syntax: `REGIST(1..4)`

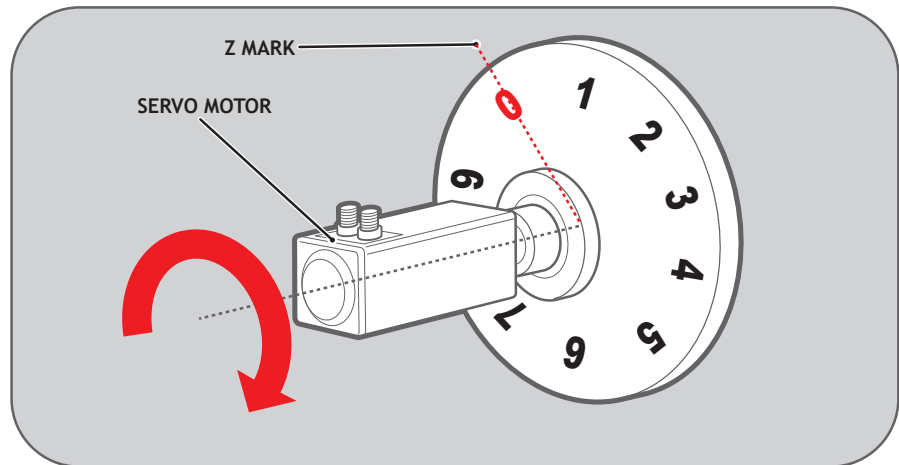
Description: Modes 1 to 4 work with the first channel or Z mark of hardware based registration.

 *You can add values to enable windowing.*

This mode works with `MARK`, `REG _ POS` and `REGIST _ SPEED`.

Parameters: `mode:` 1 = Z Mark rising into `REG _ POS`.
 2 = Z Mark falling into `REG _ POS`.
 3 = RA Input rising into `REG _ POS`.
 4 = RA Input falling into `REG _ POS`.
 +256 = Position must be inside `OPEN _ WIN..CLOSE _ WIN`.
 +768 = Position must be outside `OPEN _ WIN..CLOSE _ WIN`.

Example1: A disc used in a laser printing process requires registration to the Z marker before printing can start. This routine locates to the Z marker, then sets that as the zero position.



```

BASE(0)
REGIST(1)           `Initialise to Z mark
FORWARD            `start movement
WAIT UNTIL MARK
CANCEL             `stops movement after Z mark
WAIT IDLE
MOVEABS (REG _ POS) `relocate to Z mark
WAIT IDLE
    
```


DEFPOS(0)**`set zero position****Syntax:** **REGIST(6..13)****Description:** Modes 6 to 13 work with hardware based registration but enable you to arm 2 registration registers at once.*You can add 256 or 768 to enable windowing.*

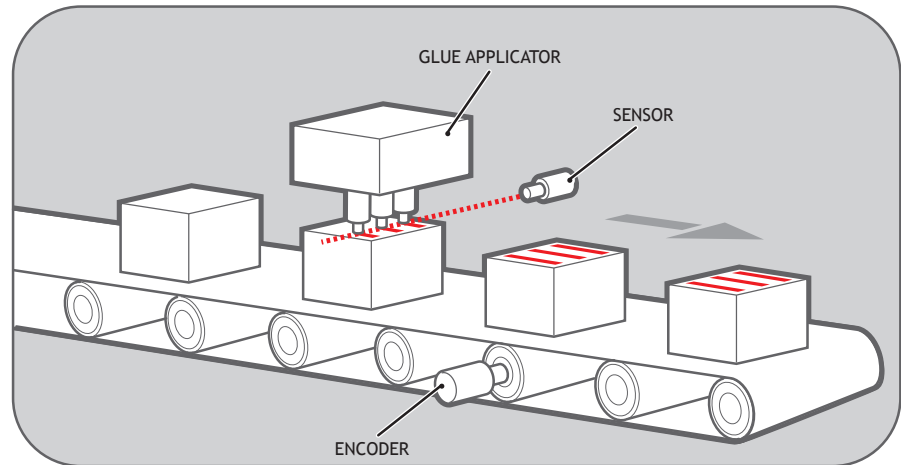
The first channel will use **MARK**, **REG _ POS** and **REGIST _ SPEED** and the second will use **MARKB**, **REG _ POSB** and **REGIST _ SPEEDB**.

Parameters:

mode: 6 = RA Input rising into **REG _ POS** & Z Mark rising into **REG _ POSB**.
7 = RA Input rising into **REG _ POS** & Z Mark falling into **REG _ POSB**.
8 = RA Input falling into **REG _ POS** & Z Mark rising into **REG _ POSB**
9 = RA Input falling into **REG _ POS** & Z Mark falling into **REG _ POSB**
10 = RA Input rising into **REG _ POS** & RB Input rising into **REG _ POSB**.
11 = RA Input rising into **REG _ POS** & RB Input falling into **REG _ POSB**.
12 = RA Input falling into **REG _ POS** & RB Input rising into **REG _ POSB**.
13 = RA Input falling into **REG _ POS** & RB Input falling into **REG _ POSB**.
+256 = Position must be inside **OPEN _ WIN..CLOSE _ WIN**.
+768 = Position must be outside **OPEN _ WIN..CLOSE _ WIN**.

Example 2:

A machine adds glue to the top of a box by switching output 8. It must detect the rising edge (appearance) of and the falling edge (end) of a box. Additionally it is required that the MPOS be reset to zero on the detection of the Z position.



```

    reg=6 `select registration mode 6 (rising edge R, rising
edge Z)
    REGIST(reg)
    FORWARD
    WHILE IN(2)=OFF
        IF MARKB THEN `on a Z mark mpos is reset to zero
            OFFPOS=-REG _ POSB
            REGIST(reg)
        ELSEIF MARK THEN `on R input output 8 is toggled
            IF reg=6 THEN
                `select registration mode 8 (falling edge R, rising
edge Z)
                reg=8
                OP(8,ON)
            ELSE
                reg=6
                OP(8,OFF)
            ENDIF
            REGIST(reg)
        ENDIF
    WEND
    CANCEL
    
```

Syntax: **REGIST(14..17)**

Description: Modes 14 to 17 work with the second channel or Z mark of hardware based registration.

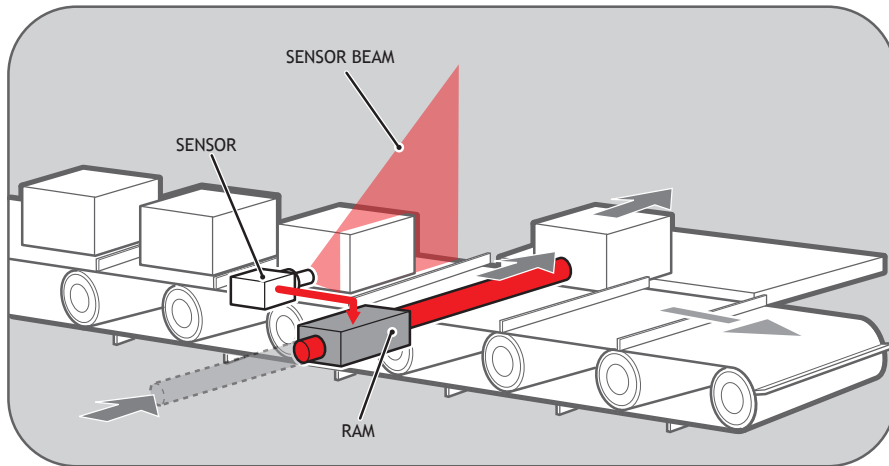
 You can add 256 or 768 to enable windowing.

This mode works with **MARKB**, **REG _ POSB** and **REGIST _ SPEEDB**.

Parameters:

mode: 14 = ZB Mark rising into **REG _ POSB**.
 15 = ZB Mark falling into **REG _ POSB**.
 16 = RB Input rising into **REG _ POSB**.
 17 = RB Input falling into **REG _ POSB**.
 +256 = Position must be inside **OPEN _ WIN..CLOSE _ WIN**.
 +768 = Position must be outside **OPEN _ WIN..CLOSE _ WIN**.

Example 3: It is required to detect if a component is placed on a flighted belt so windowing is used to avoid sensing the flights. The flights are at a pitch of 120 mm and the component will be found between 30 and 90mm. If a component is found then an actuator is fired to push it off the belt.



```
REP_DIST=120      `sets repeat distance to pitch of belt
flights
REP_OPTION=ON
OPEN_WIN=30      `sets window open position
```

```

CLOSE_WIN=90                `sets window close position
REGIST(17+256)              `RB input registration with windowing
FORWARD                      `start the belt
box_seen=0
REPEAT
  WAIT UNTIL MPOS<60        `wait for centre point between flights
  WAIT UNTIL MPOS>60        `so that actuator is fired between
  flights
  IF box_seen=1 THEN        `was a box seen on the previous cycle?
    OP(8,ON)                `fire actuator
    WA(100)
    OP(8,OFF)               `retract actuator
    box_seen=0
  ENDIF
  IF MARKB THEN box_seen=1 `set "box seen" flag
  REGIST(17+256)
  UNTIL IN(2)=OFF
  CANCEL                    `stop the belt
  WAIT IDLE

```

Syntax: `REGIST(20, channel, source, edge, window)`

Description: Mode 20 is used to set the hardware registration inputs A or B. Alternatively A or B can be replaced with the Z mark. A and B are completely independent.



When using a FlexAxis the actual input used for channel A and channel B can be selected with the REG _ INPUTS command.



This mode can be used instead of REGIST modes 1..4 and 14..17.

Parameters:

mode:	0 = Selects channel A. 1 = Selects channel B.
source:	0 = Selects the first 24V input. 1 = Selects the Z mark. 2 = Selects the second 24V input 3 = Selects the 5V registration pin (built-in axis only)
edge:	0 = Rising edge 1 = Falling edge
window:	0 = No windowing. 1 = Position must be inside OPEN _ WIN..CLOSE _ WIN. 2 = Position must be outside OPEN _ WIN..CLOSE _ WIN.



If channel = 0, MARK, REG _ POS and REGIST _ SPEED are used.
If channel = 1, MARKB, REG _ POSB and REGIST _ SPEEDB are used.

Example 4: Configure the windowing which will be used on channel B and then arm both channel B and the Z mark.

```
OPEN _ WIN=200
CLOSE _ WIN=400
REGIST(20,0,1,0,0)
REGIST(20,1,0,1,2)
```

Syntax: REGIST(21, channel, source, edge, window)

Description: REGIST mode 21 is used to arm the time based registration.

This can be used instead of REGIST modes 32..39 and 64..71.



This mode operates with the parameters R _ MARK(channel) , R _ REGPOS(channel) and R _ REG _ SPEED(channel).

Parameters:

- mode:** This is the registration channel to be used (range 0..7).
- source:** Has no function, set to 0.
- edge:** 0 = Rising edge.
1 = Falling edge.
- window:** 0 = No windowing
1 = Position must be inside OPEN _ WIN..CLOSE _ WIN.
2 = Position must be outside OPEN _ WIN..CLOSE _ WIN.

Syntax: REGIST(22, channel, source, edge, window)

Description: This mode allows up to 8 hardware registration inputs to be assigned to one axis.



IF THIS MODE IS USED ALL 8 INPUTS ARE ASSIGNED TO THE ONE AXIS. YOU CANNOT MIX REGIST(22) AND REGIST(20) ON ONE BANK OF INPUTS.

This mode operates with the parameters R _ MARK(channel) , R _ REGPOS(channel) and R _ REG _ SPEED(channel).



To use this mode `REG _ INPUTS` must be set to \$10 before you call the `REGIST` command.

Parameters:

- channel:** This is the registration channel to be used (range 0..7)
- mode:** This is the registration channel to be used (range 0..7).
- source:**
 - 0 = Selects the 24V registration input.
 - 1 = Selects the Z mark.
- edge:**
 - 0 = Rising edge.
 - 1 = Falling edge.
- window:**
 - 0 = No windowing.
 - 1 = Position must be inside `OPEN _ WIN..CLOSE _ WIN`.
 - 2 = Position must be outside `OPEN _ WIN..CLOSE _ WIN`.

Syntax: `REGIST(23)`

Description: This mode assigns a 2.4usec minimum pulse width to the axis. This affects any `REGIST` mode that is used.



The default value is 0.15usec.

Syntax: `REGIST(24)`

Description: This mode assigns a 0.15usec minimum pulse width to the axis. This affects any `REGIST` mode that is used.

This is the default value.

See Also: `MARK`, `MARKB`, `R _ MARK`, `REG _ POS`, `REG _ POSB`, `R _ REGPOS`, `REGIST _ SPEED`, `REGIST _ SPEEDB`, `R _ REGIST _ SPEED`, `REGIST _ DELAY`, `REG _ INPUTS`

REVERSE

Type: Axis Command

Syntax: REVERSE

Alternate Format: RE

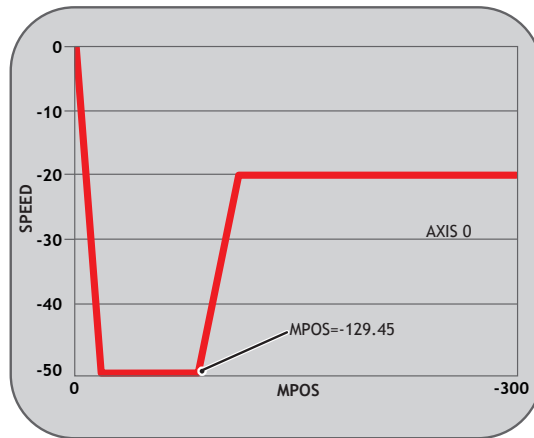
Description: Sets continuous reverse movement on the specified or base axis. The axis accelerates at the programmed **ACCEL** rate and continues moving at the **SPEED** value until either a **CANCEL** or **RAPIDSTOP** command are encountered. It then decelerates to a stop at the programmed **DECEL** rate.

If the axis reaches either the reverse limit switch or reverse soft limit, the **REVERSE** will be cancelled and the axis will decelerate to a stop.

Example 1: Run an axis in reverse. When an input signal is detected on input 5, stop the axis.
back:

```
REVERSE
`Wait for stop signal:
WAIT UNTIL IN(5)=ON
CANCEL
WAIT IDLE
```

Example 2: Run an axis in reverse. When it reaches a certain position, slow down.



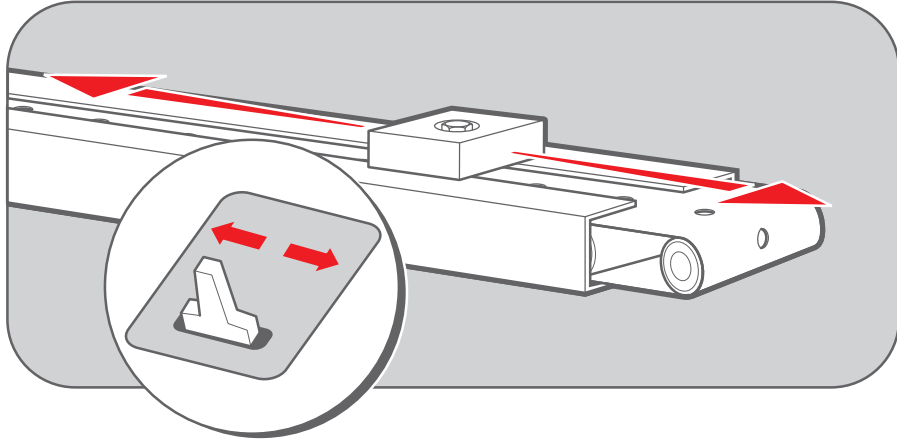
```

DEFPOS(0)      `set starting position to zero
REVERSE
WAIT UNTIL MPOS<-129.45
SPEED=slow_speed
WAIT UNTIL VP_SPEED=slow_speed `wait until the axis
slows
OP(11,ON)     `turn on an output to show that speed is now
slow

```


Example 3:

A joystick is used to control the speed of a platform. A dead-band is required to prevent oscillations from the joystick midpoint. This is achieved through setting reverse, which sets the correct direction relative to the operator, the joystick then adjusts the speed through analogue input 0.



```

REVERSE
WHILE IN(2)=ON
  IF AIN(0)<50 AND AIN(0)>-50 THEN `sets a dead-band in the
input
  SPEED=0
  ELSE
  SPEED=AIN(0)*100           `sets speed to a scale of
AIN
  ENDIF
WEND
CANCEL

```

See Also: [Forward](#)

SERVO_READ

Type:	Axis Command
Syntax:	<code>SERVO_READ(vr start, p0[,p1[,p2[,p3[,p4[,p5[,p6[,p7]]]]]]])</code>
Description:	Provides servo-synchronized access to axis/system parameters. Between 1 and 8 axis/system parameters can be read synchronously on the next servo cycle for consistent data access when required. The data read is stored in successive VR memory locations commencing from 'vr start'.
Parameters:	<p>vr start: base index of VR memory to store data read from parameters.</p> <p>p0..p7: Axis/System parameters to be read.</p>
Example:	<p><code>SERVO_READ(100, MPOS AXIS(0), FE AXIS(0), MPOS AXIS(1), FE AXIS(1))</code></p> <p>Reads MPOS & FE for axes 0 & 1 and stores in VR locations 100,101,102 & 103.</p>

STEP_RATIO

Type:	Axis Command
Syntax:	<code>STEP_RATIO(output _ count, dpos _ count)</code>
Description:	<p>This command sets up an integer ratio for the axis' stepper output. Every servo-period the number of steps is passed through the STEP_RATIO function before it goes to the step pulse output.</p> <p>The STEP_RATIO function operates before the divide by 16 factor in the stepper axis. This maintains the good timing resolution of the stepper output circuit.</p> <p>STEP_RATIO does not replace UNITS. Do not use STEP_RATIO to remove the x16 factor on the stepper axis as this will lead to poor step frequency control. You should use PP_STEP for this.</p>
Parameters:	<p>output _ count: Number of counts to output for the given dpos_count value. Range: 0 to 16777215.</p> <p>dpos _ count: Change in DPOS value for corresponding output count. Range: 0 to 16777215.</p>

Large ratios should be avoided as they will lead to either loss of resolution or much reduced smoothness in the motion. The actual physical step size x 16 is the basic resolution of the axis and use of this command may reduce the ability of the *Motion Coordinator* to accurately achieve all positions.

Example 1:

Two axes are set up as X and Y but the axes' steps per mm are not the same. Interpolated moves require identical **UNITS** values on both axes in order to keep the path speed constant and for **MOVECIRC** to work correctly. The axis with the lower resolution is changed to match the higher step resolution axis so as to maintain the best accuracy for both axes.

```
'Axis 0: 500 counts per mm (31.25 steps per mm)
'Axis 1: 800 counts per mm (50.00 steps per mm)
  BASE(0)
  STEP_RATIO(500,800)
  UNITS = 800
  BASE(1)
  UNITS = 800
```

Example 2:

A stepper motor has 400 steps per revolution and the installation requires that it is controlled in degrees. As there are 360 degrees in one revolution, it would be better from the programmer's point of view if there are 360 counts per revolution.

```
BASE(2)
STEP_RATIO(400, 360)
'Note: this has reduced resolution of the stepper axis
MOVE(360*16) 'move 1 revolution
```

Example 3:

Remove the step ratio from an axis.

```
BASE(0)
STEP_RATIO(1, 1
```

Input / Output Commands

.. (Range)

Type: Reserved Keyword

AIN

Type: System Command

Syntax: `AIN(channel)`

Description Reads a value from an analogue input. Analogue inputs are either built in to the *Motion Coordinator* or available from the `CAN` Analogue modules.

The value returned is the decimal equivalent of the binary number read from the A to D converter.

The built in analogue inputs are updated every servo period. The `CAN` analogue inputs are updated every 10msec.

If no `CAN` Analogue modules are fitted, `AIN(0)` and `AIN(1)` will read the first two built-in channels so as to maintain compatibility with previous versions.

Parameters: `channel:` Analogue input channel number 0...35.
 0 to 31: `CAN` analogue input channel number.
 32 to 35: Built in analogue input channel number.

Example: Material is to be fed off a roll at a constant speed. There is an ultrasonic height sensor that returns 4V when the roll is empty and 0V when the roll is full. A lazy loop is written in the `BASIC` to control the speed of the roll.

```
MOVE(-5000)
REPEAT
  a=AIN(1)
  IF a<0 THEN a=0
  SPEED=a*0.25
UNTIL MTYPE=0
```

The analogue input value is checked to ensure it is above zero even though it always should be positive. This is to allow for any noise on the incoming signal which could make the value negative and cause an error because a negative speed is not valid for any move type except **FORWARD** or **REVERSE**.

AIN0..3 / AINBI0..3

Type: System Parameter

Description: These system parameters duplicate the **AIN()** command.

AIN0..3 is used for single sided analogue inputs.

AINBI0..3 is used for bipolar inputs.

They provide the value of the analogue input channels in system parameter format to allow the **SCOPE** function (Which can only store parameters) to read the analogue inputs.

If no **CAN** Analogue modules are fitted, **AIN0** and **AIN1** will read the first two built-in channels.

CHANNEL_READ

Type: System Command

Syntax: `CHANNEL_READ(channel, buffer_base, size[, delimiter_base, delimiter_size[, escape_character[, crc]])`

Description: **CHANNEL_READ** will read bytes from the channel and store them into the **VR** data starting at **buffer_base**.

CHANNEL_READ will stop when it has read **size** bytes, the channel is empty, or the character read from the channel is specified in the **delimiter** buffer.

If the escape character received then the next character is not interpreted. This allows delimiter characters to be received without stopping the **CHANNEL_READ**.

The calculated **CRC** will be stored in the **VR(crc)**.

Parameters:	channel:	Communication or file channel.
	buffer_base:	Number of the first VR for the buffer.
	size:	Size of the buffer.
	delimiter_base:	Position in the VR data to the start of the delimiter list.
	delimiter_size:	Size of the delimiter list.
	escape_character:	escape_character: When this character is received the following character is not interpreted.
	crc:	Position in the VR data where the CRC will be stored.

CHANNEL_WRITE

Type:	System Command						
Syntax:	<code>CHANNEL_WRITE(channel, buffer_base, buffer_size)</code>						
Description:	<code>CHANNEL_WRITE</code> will send <code>buffer_size</code> bytes from the VR data starting at <code>buffer_base</code> to the channel.						
Parameters:	<table><tr><td>channel:</td><td>Communication or file channel.</td></tr><tr><td>buffer_base:</td><td>Position in the VR data to the start of the buffer.</td></tr><tr><td>buffer_size:</td><td>Size of the buffer.</td></tr></table>	channel:	Communication or file channel.	buffer_base:	Position in the VR data to the start of the buffer.	buffer_size:	Size of the buffer.
channel:	Communication or file channel.						
buffer_base:	Position in the VR data to the start of the buffer.						
buffer_size:	Size of the buffer.						

CLOSE

Type:	Command
Syntax:	<code>CLOSE #<channel></code>
Description:	<code>CLOSE</code> will close the file on the specified channel.
Parameters:	<code><channel></code> The TrioBASIC I/O channel to be associated with the file. It is in the range 40 to 44.
See also:	<code>OPEN</code>

FILE

Type:	System Command																								
Syntax:	<code>value = FILE "function" [parameters]</code>																								
Description:	This command enables the user to manage the data on the <code>SDCARD</code> . When the command prints to the selected channel, this channel can be selected using <code>OUTDEVICE</code>																								
Parameters:	<table><tr><td>Function:</td><td><code>CD</code></td><td>Change directory.</td></tr><tr><td></td><td><code>DEL</code></td><td>Delete file.</td></tr><tr><td></td><td><code>DETECT</code></td><td>Check for <code>SD</code> card.</td></tr><tr><td></td><td><code>DIR</code></td><td>Print the current directory contents.</td></tr><tr><td></td><td><code>FIND _ FIRST</code></td><td>Finds the first entry in the directory structure of the specified file type.</td></tr><tr><td></td><td><code>FIND _ NEXT</code></td><td>Finds the next entry in the directory structure of the specified file type.</td></tr><tr><td></td><td><code>FIND _ PREV</code></td><td>Finds the previous entry in the directory structure of the specified file type.</td></tr><tr><td></td><td><code>LOAD _ PROGRAM</code></td><td>Loads the specified program to the controllers memory.</td></tr></table>	Function:	<code>CD</code>	Change directory.		<code>DEL</code>	Delete file.		<code>DETECT</code>	Check for <code>SD</code> card.		<code>DIR</code>	Print the current directory contents.		<code>FIND _ FIRST</code>	Finds the first entry in the directory structure of the specified file type.		<code>FIND _ NEXT</code>	Finds the next entry in the directory structure of the specified file type.		<code>FIND _ PREV</code>	Finds the previous entry in the directory structure of the specified file type.		<code>LOAD _ PROGRAM</code>	Loads the specified program to the controllers memory.
Function:	<code>CD</code>	Change directory.																							
	<code>DEL</code>	Delete file.																							
	<code>DETECT</code>	Check for <code>SD</code> card.																							
	<code>DIR</code>	Print the current directory contents.																							
	<code>FIND _ FIRST</code>	Finds the first entry in the directory structure of the specified file type.																							
	<code>FIND _ NEXT</code>	Finds the next entry in the directory structure of the specified file type.																							
	<code>FIND _ PREV</code>	Finds the previous entry in the directory structure of the specified file type.																							
	<code>LOAD _ PROGRAM</code>	Loads the specified program to the controllers memory.																							

	LOAD _ PROJECT	Loads the specified project into the controllers memory.
	LOAD _ SYSTEM controller.	Loads the specified firmware into the
	RD	Delete a directory.
	MD	Create a directory.
	PWD	Prints the path of the directory.
	SAVE _ PROGRAM	Saves the specified program to the SD card.
	SAVE _ PROJECT	aves all programs from the controller to the SD card.
	TYPE	Prints the selected file.
value:		returns TRUE if the function was successful otherwise returns FALSE .

Syntax: `value = FILE "CD" "directory"`

Description: Change to the given directory. There is one active directory on the controller all SDCARD commands are relative to this directory.

Parameters: `directory:` The name of the directory to change to.

Example: Use the command line to change to a new directory.

```
>>file "CD" "new_directory"
OK \NEW_DIRECTORY
>>
```

Syntax: `value = FILE "DEL" "file"`

Description: Delete the given file inside the current directory.

Parameters: `file:` The name of the file to be deleted, you must include the file extension.

Example: Delete a BASIC program from the SD card using the command line.

```
>>FILE "DEL" "STARTUP.bas"
```



```
OK
>>
```

Syntax: **value = FILE "DETECT"**

Description: Checks if a SD card is present in the slot

Parameters: **parameters:** Returns TRUE if an SDCARD is detected correctly.

Example: Check if an SD card is present before saving the table data.

```
IF FILE "DETECT" THEN
  STICK _ WRITE(1501, 1000, 2000, 0)
ENDIF
```

Syntax: **value = FILE "DIR"**

Description: Print the contents of the current directory to the current output channel.

Example: Print the contents of the SD card on the command line.

```
>>FILE "DIR"
Volume is NO NAME
Volume Serial Number is 00C8-B79F
Directory of /
07/Aug/2009 15:50      1169978 MC60CC~1.OUT MC464 _ 20055 _ _
BOOT _ 013.out

20/Nov/2009 15:25 <DIR>      MC464 _ ~1      MC464 _ Panasonic _ Home
16/Feb/2009 13:16      1619 TRIOINIT.BAS TRIOINIT.BAS
20/Nov/2009 15:21 <DIR>      SHOW1      Show1
07/Jan/2000 04:54 <DIR>      NEW _ DI~1      NEW _ DIRECTORY
>>
```

Syntax: `value = FILE "FIND _ FIRST", type, vr`

Description: Initialises the internal **FIND** structures and locates the first directory entry of the given type. The found directory entries name is stored in a **VRSTRING**

Parameters: **value:** **TRUE** if a directory entry is found otherwise **FALSE**.
 type: 0 = **FILE** or **DIRECTORY**
 1 = **FILE**
 2 = **DIRECTORY**
vr: The start position in **VR** memory where the **VRSTRING** is stored.
If there is an error initialising the internal **FIND** structures then the function returns **FALSE**.

Syntax: `value = FILE "FIND _ NEXT", vr`

Description: Finds the next directory entry of the type given in the corresponding **FIND _ FIRST** command.

Parameters: **value:** **TRUE** if a directory entry is found otherwise **FALSE**.
vr: The start position in **VR** memory where the **VRSTRING** is stored.
If there is an error initialising the internal **FIND** structures then the function returns **FALSE**.

Syntax: `value = FILE "FIND _ PREV", vr`

Description: Finds the previous directory entry of the type given in the corresponding **FIND _ FIRST** command.

Parameters: **value:** **TRUE** if a directory entry is found otherwise **FALSE**.
vr: The start position in **VR** memory where the **VRSTRING** is stored.
If there is an error initialising the internal **FIND** structures then the function returns **FALSE**.

Syntax: `value = FILE "LOAD_PROGRAM" "file"`

Description: Load the given program into the *Motion Coordinator*. Only .BAS files are handled at the moment.

Parameters: `file:` The name of the file that you wish to load.

Syntax: `value = FILE "LOAD_PROJECT" "name"`

Description: Read the given *Motion Perfect* project file and load all the programs into the *Motion Coordinator*, once loaded any **RUNTYPES** are automatically set.

Parameters: `name:` The name of the project that you wish to load.

Syntax: `value = FILE "LOAD_SYSTEM" "name"`

Description: Read the given *Motion Perfect* project file and load all the programs into the *Motion Coordinator*, once loaded any **RUNTYPES** are automatically set.

Parameters: `name:` The name of the firmware that you wish to load.

Syntax: `value = FILE "RD" "name"`

Loading incorrect firmware can prevent your controller from operating

Description: Delete the given directory inside the current directory.

Parameters: **name:** The name of the directory that you wish to delete.

Syntax: `value = FILE "MD" "name"`

Description: Create the given directory inside the current directory.

Parameters: **name:** The name of the directory that you wish to create.

Example: Using the command line create a new directory.

```
>>FILE "MD" "new_directory"  
OK  
>>
```

Syntax: `value = FILE "PWD"`

Description: Prints the path of the current directory to the current output channel.

Syntax: `value = FILE "SAVE_PROGRAM" "name"`

Description: Save the given program to the corresponding file on the **SDCARD** inside the current directory. Only **.BAS** files are handled at the moment.

Parameters: **name:** The name of the file that you wish to save to the **SD** card.

Syntax: `value = FILE "SAVE _ PROJECT" "name"`

Description: Create a *Motion Perfect* project with the given name inside the current directory. This implies creating the directory and the corresponding project and program files within this directory.

Parameters: **name:** The name of the project that you are creating on the SD card.

Syntax: `value = FILE "TYPE" "name"`

Description: Read the contents of the file inside the current directory and print it to the current output channel.

Parameters: **name:** The name of the file that you wish to print.

See also `OUT _ DEVICE, STICK _ READ, STICK _ WRITE, STICK _ READVR, STICK _ WRITEVR`

FLAG

Type: Logical and Bitwise Command

Syntax: `value = FLAG(flag _ no [,state])`

Description: The **FLAG** command is used to set and read a bank of 24 flag bits.



*The **FLAG** command is provided to aid compatibility with earlier controllers and is not recommended for new programs.*

Parameters: **value:** With one parameter it returns the state of the flag .

flag _ no: The flag number is a value from 0..31.

state: The state to set the given flag to. **ON** or **OFF**.

Example 1: Toggle a flag depending on a VR value.

`IF FLAG(21) and VR(100)=123 THEN`

```

FLAG(21,OFF)
ELSE IF NOT FLAG(21) and VR(100)<>123 THEN
  FLAG(21,ON)
ENDIF

```

FLAGS

Type: Logical and Bitwise Command

Syntax: `value = FLAGS([state])`

Description: Read or Set the 32bit **FLAGS** as a block.



*The **FLAGS** command is provided to aid compatibility with earlier controllers and is not recommended for new programs.*

Parameters:

- value:** no parameters = returns the status of all flag bits
with parameter = returns -1
- stste:** The decimal equivalent of the bit pattern to set the flags to.

Example 1: Set Flags 1,4 and 7 ON, all others OFF

Bit #	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1

`FLAGS(146)' 2 + 16 + 128`

Example 2: Test if **FLAG** 3 is set.

```
IF (FLAGS and 8) <>0 then GOSUB somewhere
```

GET

Type: System Command

Syntax: `GET [#channel,] variable`

Description: Waits for the arrival of a single character on the serial. The `ASCII` value of the character is assigned to the variable specified. The user program will wait until a character is available.



Poll `KEY` to check to if a character has been received before performing a `GET`.

Parameters:

- `#channel:` See `#` for the full channel list (default 0 if omitted).
- `variable:` The variable to store the received character, this may be local variable, `VR` or `TABLE`.

PERFORMING A `GET` OR `GET#0` WILL SUSPEND THE COMMAND LINE UNTIL A CHARACTER IS SENT ON THAT CHANNEL.

Example 1: Ask a user to enter y for yes and n for no on channel 5.

```
start:
PRINT#5, "Press 'y' for YES and 'n' for NO."
GET#5, char
IF char = 121 THEN
    PRINT#5, "YES selected"
ELSEIF char = 110 THEN
    PRINT#5, "NO selected"
ELSE
    PRINT#5, "BAD selection"
    GOTO start
ENDIF
```

Example 2: Clear the serial buffer then request the user to enter a name.

```
WHILE KEY#2
    GET#2, dump
WEND

PRINT#2, "ENTER NAME"
WAIT UNTIL KEY#2
count=0
WHILE char<> $D `carrage return
```

```

GET#2, char
VR(count)=char
count=count+1
WEND

```

See Also: `LINPUT`, `PRINT`, `KEY`

HW_PSWITCH

Type: Axis command

Syntax: `HW_PSWITCH(mode, direction, opstate, table_start, table_end)`

Description: The `HW_PSWITCH` command turns on the `PS` output for the axis when the predefined axis measured position is reached, and turns the output off when another measured position is reached. Positions are defined as a sequence in the `TABLE` memory in range from `table_start` to `table_end`, and on execution of the `HW_PSWITCH` command the positions are stored in a `FIFO` (first in - first out) queue. This command is applicable only to Flexible Axis axes with `ATYPE` values 43, 44 and 45.

The command can be used with either 1 or 5 parameters. Only 1 parameter is needed to disable the switch or clear `FIFO` queue. All five parameters are needed to enable the switch.

After loading the `FIFO` and going through the sequence of positions in it, if the same sequence has to be executed again, the `FIFO` must be cleared before executing another `HW_PSWITCH` command with the same parameters.

Parameters:

<code>mode:</code>	0 = disable switch 1 = on and load <code>FIFO</code> 2 = clear <code>FIFO</code>
<code>direction:</code>	0 = <code>MPOS</code> decreasing 1 = <code>MPOS</code> increasing.
<code>opstate:</code>	Output state to set in the first position in the <code>FIFO</code> ; <code>ON</code> or <code>OFF</code> .
<code>table_start:</code>	Starting <code>TABLE</code> address of the sequence.
<code>table_end:</code>	Ending <code>TABLE</code> address of the sequence.

Example 1: Load the table with 30 ON/OFF positions then run the command to load the FIFO with these positions. When the position stored in TABLE(21) is reached, the PSn output will be set ON and then alternatively OFF and ON on reaching the following positions in the sequence, until the position stored in TABLE(50) is reached.

```
TABLE(21,5,10,15,18,20,24,30,33,45,51,56,57,65,76,79,84,88,90,94)
TABLE(40,99,105,120,140,145,190,235,260,271,280,300)
HW_PSWITCH(1, 1, ON, 21, 50)
```

Example 2: Disable the switch if it was enabled previously. Does not clear the FIFO queue.

```
HW_PSWITCH(0)
```

Example 3: Clear the FIFO queue of a switch not on the BASE axis.

```
HW_PSWITCH(2) AXIS(8)
```



Outputs are assigned to the axes of the FlexAxis module in a fixed way. One output per axis; axis 0 - PS4, axis 1 - PS5, axis 2 - PS6, axis 3 - PS7.)

IN

Type: Function

Syntax: value = IN[(input _ no[,final _ input]]]

Description: IN is used to read the state of the inputs.

If called with no parameters, IN returns the binary sum of the first 32 inputs. If called with one parameter it returns the state (1 or 0) of that particular input channel. If called with 2 parameters IN() returns in binary sum of the group of inputs.



In the 2 parameter case the inputs should be less than 24 apart. IN is equivalent to IN(0,31).

Parameter:

value:	The state of the selected input or range of inputs.
none:	Returns the binary sum of the first 24 inputs.
input _ no:	Input to return the value of/start of input group.

`final _input:` last input of group.

Example 1: In this example a single input is tested:

```
WAIT UNTIL IN(4)=ON
GOSUB place
```

Example 2: Move to the distance set on a thumb wheel multiplied by a factor. The thumb wheel is connected to inputs 4,5,6,7 and gives output in binary coded decimal.

The move command is constructed in the following order:

Step 1: `IN(4,7)` will get a number 0..15

Step 2: multiply by 1.5467 to get required distance

Step 3: absolute `MOVE` to this position

```
WHILE TRUE
  MOVEABS(IN(4,7)*1.5467)
  WAIT IDLE
WEND
```

Example 3: Test if either input 2 or 3 is ON.

```
If (IN and 12) <> 0 THEN GOTO start
'(Bit 2 = 4 + Bit 3 = 8) so mask = 12
```

INPUT

Type: System Command.

Syntax: `INPUT [#channel,] variable [, variable...]`

Description: Waits for an ASCII string to be received on the current input device, terminated with a carriage return <CR>. If the string is valid its numeric value is assigned to the specified variable. If an invalid string is entered it is ignored, an error message displayed and input repeated. Multiple values may be requested on one line, the values are separated by commas, or by carriage returns <CR>.

Poll `KEY` to check to if a character has been received before performing an `INPUT`.

Parameters: **#channel:** See # for the full channel list (default 0 if omitted).
 variable: The variable to store the received character, this may be local variable, VR or TABLE.



PERFORMING A INPUT OR INPUT#0 WILL SUSPEND THE COMMAND LINE UNTIL A CHARACTER IS SENT ON THAT CHANNEL.

Example 1: Receive a single value and store it in a local variable num.

```
INPUT num
PRINT "BATCH COUNT=";num[0]
```

On terminal:
123 <CR>
BATCH COUNT=123

Example 2: Get the length and width variables using one INPUT.

```
PRINT "ENTER LENGTH AND WIDTH?";
INPUT VR(11),VR(12)
```

This will display on terminal:

```
ENTER LENGTH AND WIDTH ? 1200,
1500 <CR>
```

See Also: #, KEY

INPUTS0 / INPUTS1

Type: System Parameter

Description: The **INPUTS0 / INPUTS1** parameters holds the state of the Input channels as a system parameter.

Reading the inputs using these system parameters is not normally required. The **IN(x,y)** command should be used instead. They are made available in this format to make the input channels accessible to the **SCOPE** command which can only store parameters.

Parameters: **value:** **INPUTS0** = the binary sum of **IN(0)..IN(15)**.
 INPUTS1 = the binary sum of **IN(16)..IN(31)**.

See Also: **IN**

INVERT_IN

Type: System Function

Syntax: **INVERT _ IN(input, state)**

Description: The **INVERT _ IN** command allows the input channels to be individually inverted in software.



This is important as these input channels can be assigned to activate functions such as feedhold.

Parameters: **input:** The input to invert.
 state: **ON** = the input is inverted in software.
 OFF = the input is not inverted.

Example: Invert input 7 so that when the input is low the **FWD _ JOG** is off
 INVERT _ IN(7,ON)
 FWD _ JOG=7

KEY

Type: System Function.

Syntax: `value = KEY [#channel]`

Description: Key is used to check if there are characters in a channel buffer. This command does not read the character but allows the program to test if any character has arrived.



A TRUE result will be reset when the character is read with GET.

Parameters: **#channel:** See # for the full channel list (default 0 if omitted).
value: A negative value representing the number of characters in the channel buffer.

Example: Call a subroutine if a character has been received on channel 1.

```
main:
    IF KEY#1 THEN GOSUB read
...
read:
    GET#1 k
RETURN
```

See Also: GET

LINPUT

Type: System Command

Syntax: LINPUT [#channel,] variable

Description: Waits for an input string and stores the ASCII values of the string in an array of variables starting at a specified numbered variable. The string must be terminated with a carriage return <CR> which is also stored. The string is not echoed by the controller.



You can print the string from the VR's using VRSTRING.

Parameters: **#channel:** See # for the full channel list (default 0 if omitted).
variable: The VR variable to store the received character.

Example: LINPUT VR(0)

Now entering: START<CR> will give:

```
VR(0) 83  ASCII 'S'  
VR(1) 84  ASCII 'T'  
VR(2) 65  ASCII 'A'  
VR(3) 82  ASCII 'R'  
VR(4) 84  ASCII 'T'  
VR(5) 13  ASCII carriage return
```

See Also: #, VRSTRING

MODULE_IO_MODE

Type: Slot Parameter

Syntax: `MODULE_IO_MODE = mode`

Description: This parameter sets the start address of any expansion module I/O channels. You can also turn off module I/O for backwards compatibility.



This parameter is stored in Flash EPROM and should only be entered in the command line.

Parameters:

- `mode: 0` = Module I/O disabled
- `1` = Module I/O is after controller I/O and before CANIO (default)
- `2` = Module I/O is after CANIO



IF YOU ARE UPGRADING THE FIRMWARE IN AN EXISTING CONTROLLER, THIS PARAMETER MAY BE SET TO 0. THE DEFAULT OF 1 IS ON A FACTORY INSTALLED SYSTEM.

Example: A system with MC464, a Panasonic module (slot 0), a FlexAxis (slot 1) and a CANIO Module will have the following I/O assignment:

```
MODULE_IO_MODE=1 (default)
0-7   Built in inputs
8-15  Built in bi-directional I/O
16-23 Panasonic inputs
24-27 FlexAxis inputs
28-31 FlexAxis bi-directional I/O
32-47 CANIO bi-directional I/O

MODULE_IO_MODE=0 (off)
0-7   Built in inputs
8-15  Built in bi-directional I/O
16-31 CANIO bi-directional I/O

MODULE_IO_MODE=2 (end)
0-7   Built in inputs
8-15  Built in bi-directional I/O
16-31 CANIO bi-directional I/O
32-39 Panasonic inputs
40-43 FlexAxis inputs
44-47 FlexAxis bi-directional I/O
```

OP

Type:	System Command.
Description:	Sets output(s) and allows the state of the first 32 outputs to be read back. There are four modes of operation for the OP command, using up to three parameters:
Syntax:	<code>value =OP</code>
Description:	Return the state of the first 32 outputs as a binary pattern
Parameters:	<code>value:</code> Binary pattern of the first 32 outputs.

Syntax:	<code>OP(state)</code>
Description:	Simultaneously set the first 32 outputs with the binary pattern of the state.
Parameters:	<code>state:</code> Decimal equivalent of binary number to set on outputs.

Syntax:	<code>OP(output, state)</code>
Description:	Set the state of an individual output
Parameters:	<code>output:</code> Output number to set. <code>state:</code> 0 or OFF 1 or ON

Syntax: `OP(start, end, state)`

Description: Simultaneously set a defined group of outputs with the binary pattern of the state.

Parameters:

- `start:` First output in the group.
- `end:` Last output in the group.
- `state:` Decimal equivalent of binary number to set on the group.

Example 1: Turn on a single output 44

```
OP(44,1)
```

This is equivalent to:

```
OP(44,ON)
```

Example 2: Sets the bit pattern 10010 on the first 5 physical outputs, outputs 13-31 will be cleared. Note how the bit pattern is shifted 8 bits by multiplying by 256 to set the first available outputs as 0 to 7 do not exist.

```
OP (18*256)
```

Example 3: Read the first 32 outputs, clear 0-7 as they are only inputs and 16-32. Then set 16-32 leaving 8-15 in their original state.

```
read _ output:
  VR(0)=OP      `clear 0-7 and 16-32
  VR(0)=VR(0) AND $0000FF00  `set $1A42 in outputs 16-32,
8-15 will remain in their original state
  VR(0)=VR(0) OR $1A420000
  OP(VR(0))
```

Example 4: Simultaneously setting outputs 10 to 13 all on.

```
OP(10,13, $F)
```

See also: `READ _ OP()`

OPEN

Type: Command

Syntax: `OPEN # channel AS "[location:]name" FOR access`

Description: `OPEN` will provide access to a text file on the controller. The text file can be initialised as a file that *Motion Perfect* can synchronise with, a temporary file or as a `FIFO` buffer. All files are in the file list however only a text file can be viewed or edited in *Motion Perfect*.

Once the file has been opened then it can be manipulated by the standard TrioBASIC channel commands. If the file is opened with read access then any TrioBASIC `GET` type commands such as `GET`, `INPUT`, `LINPUT` and `KEY` can be used on the channel. If the file is opened with write access then the `PRINT` type commands can be used on the channel.

Parameters:

The TrioBASIC I/O channel to be associated with the file. It is in the range 40 to 44.

access: The operations permitted on the file. The valid access types are:

INPUT

The file will be opened for reading. When the end of the file is reached `KEY` will return `FALSE`, and the `GET` and `INPUT` functions will fail.

OUTPUT(mode)

The file will be opened for writing. If the file does not exist then it will be created. If the file does exist then it will be cleared.

mode = 0 opens a text file that *Motion Perfect* can read, edit and save into the project.

mode = 1 opens a temporary file that is only accessible by the controller.

FIFO_READ

The file will be opened for reading and will be managed as a circular buffer. This is only valid for files stored in internal RAM.

FIFO_WRITE(size)

The file will be opened for writing and will be managed as a circular buffer. This is only valid for files in internal RAM. If the file does not exist it will be created <size> bytes long. If the file does exist then it must be of type FIFO, the size parameter is ignored and the contents are cleared.

name: Name of the file to be opened. The format is “[RAM|SD:]filename”. If the prefix is omitted or is RAM: then filename refers to an internal RAM directory entry. If the prefix is SD: then filename refers to an SDCARD directory entry.



If you are creating a file on the SD card you will need to append the file extension. A text file stored in RAM will be saved as a .txt file in the project by Motion Perfect. This enables you to generate and read files on the SD card in any text based format.

Example 1: Open a file that can be used to log information to a .txt file on the SD card then print end of shift information to the file.

```
OPEN #40 AS "SD:product_log.txt" FOR OUTPUT (0)
PRINT#40, DATE$ 'Print the date
PRINT#40, products_complete[0]; " products completed"
PRINT#40, product_failures[0]; " products failed"
CLOSE# 40
```

Example 2: A G-Code file is loaded from a serial port into the controller, it is saved into a temp file on the controller for use later on.

```
OPEN #41, AS "gcodeprogram" for OUTPUT (1)
WHILE file_downloading
  IF KEY#1
    GET#1, char
    PRINT#40, char;
  ENDIF
  Length=length + 1
WEND
```

Example 3: The G-Code program has been downloaded to a temp file, it then should be transferred to a FIFO so that it can be interpreted into motion.

```
OPEN #41, AS "gcodeprogram" for INPUT
OPEN#42, AS "gcodefifo" for FIFO_WRITE(length)
WHILE KEY#41
  GET#41, char
  PRINT#42, char;
WEND
```

PRINT

Type: Command.

Alternate format: ?

Syntax: `PRINT [#number,] print_expression`

Description: The `PRINT` command allows the TrioBASIC program to output a series of characters to a channel. A channel may be a serial port or some other type of connection to the *Motion Coordinator*.

A `print_expression` may include parameters, fixed `ASCII` strings, single `ASCII` characters and the returned values from functions. Multiple items to be printed can be put on the same `PRINT` line provided they are separated by a comma or semi-colon. The items can be modified using print formatters including `HEX`, `CHR` and `[w,x]`.



Any value larger than 1e19 and smaller than 1e-18 will be printed in scientific format. You can still use `[w,x]` to format how this is displayed. A value is normally printed to 4 decimal places.

Parameters:

#channel:	See # for the full channel list (default 0 if omitted).
value[w,x]:	Separates items with no space, omits carriage return line feed if used after the last item. w = total number of characters to display, 29 maximum (optional). x = number of decimal places to use, 15 maximum.
"string":	Prints the string.
CHR(value):	Prints the <code>ASCII</code> character referred to by the number.
HEX(expression):	Prints the value in hexadecimal format.
TIME\$	Prints the time from the real time clock in 24hr format.



When using `value[w,x]`, if the number is too big the field will be filled with question marks to signify that there was not sufficient space to display the number. The numbers are right justified in the field with any unused leading characters being filled with spaces.

Example 1: Print a string using quotation marks.

```
PRINT "CAPITALS and lower case CAN BE PRINTED"
```

Example 2: Print a number and a value from a VR, separated by a comma to make the VR value in the next tab space.

```
>>PRINT 123.45,VR(1)
123.4500    1.5000
>>
```

Example 3: Print a VR with 4 characters and 1 decimal place, then in the next tab a local variable with 2 decimal places.

```
VR(1)=6
variable=410.5:
PRINT VR(1)[4,1],variable[2]
print output will be:
6.0    410.50
```

Example 4: Print a string directly followed by a numerical value. Note how in this example the semi-colon separator is used. This does not tab into the next column, allowing the programmer more freedom in where the print items are put.

```
>>PRINT "DISTANCE=";MPOS
DISTANCE=123.0000
>>
```

Example 5: Print a carriage return and no line feed at the end of a message. The semi-colon on the end of the print line suppresses the carriage return normally sent at the end of a print line. ASCII (13) generates CR without a line feed. The string is to output from serial port channel 1.

```
PRINT #1,"ITEM ";total;" OF ";limit;CHR(13);
```

Example 6: Print the status of inputs 8-16 in hexadecimal format to terminal channel 5 in *Motion Perfect*.

```
PRINT #5, HEX(IN(8,16))
```

Example 7: Print `AXISSTATUS` for axis 6 in the hexadecimal format on the command line. (Bits 1 and 8 are set).

```
>>?hex(AXISSTATUS AXIS(6))
```

102
>>

See Also: #

PSWITCH

Type: Command

Syntax: `PSWITCH(switch, enable [,axis, output, state, setpos, resetpos])`

Description: The `PSWITCH` command allows an output to be set when a predefined position is reached, and to be reset when a second position is reached. There are 16 position switches each of which can be assigned to any axis and to any output, virtual or real.

Multiple `PSWITCH`'s can be assigned to a single output.



The actual output is the OR of all position switches on the output OR the OP setting. This means that `OP(output,ON)` can override a `PSWITCH`.

After switching the `PSWITCH OFF`, the output will remain at the current state. You can use the `OP` command to then set it to the state you require.

Parameters:

- switch:** The switch number in the range 0..15.
- enable:** 1 or `ON` = Enable software `PSWITCH` (requires all parameters)
0 or `OFF` = Disable `PSWITCH`
5 = Enable `PSWITCH` on `DPOS`
- axis:** Axis to link the `PSWITCH` to, may be any real or virtual axis.
- output:** Selects the output to set, can be any real or virtual output.
- state:** 1 or `ON` = turn the output `ON` at `setpos`
0 or `OFF` = turn the output `OFF` at `setpos`
- setpos:** The position at which output is set, in user units.
- resetpos:** The position at which output is reset, in user units.

Example:

A rotating shaft has a cam operated switch which has to be changed for different size work pieces. There is also a proximity switch on the shaft to indicate TDC of the machine. With a mechanical cam the change from job to job is time consuming but this can be eased by using the `PSWITCH` as a software 'cam switch'. The proximity switch is wired to input 7 and the output is fired by output 11. The shaft is controlled by axis 0 of a 3 axis system. The motor has a 900ppr encoder. The output must be on from 80° after TDC for a period of 120°. It can be assumed that the machine starts from TDC.

The `PSWITCH` command uses the unit conversion factor to allow the positions to be set in convenient units. So first the unit conversion factor must be calculated and set. Each pulse on an encoder gives four edges which the controller counts, therefore there are 3600 edges/rev or 10 edges/°. If we set the unit conversion factor to 10 we can then work in degrees.

Next we have to determine a value for all the `PSWITCH` parameters.

This can all be put together to form the two lines of TrioBASIC code that set up the position switch:

axis: We are told that the shaft is controlled by axis 0, thus axis is set to 0.

output: We are told that output 11 is the one to fire, so set `opno` to 11.

state: When the output is set it should be `ON`.

setpos: The output is to fire at 80° after TDC hence the set position is 80 as we are working in degrees.

resetpos: The output is to be on for a period of 120° after 80° therefore it goes off at 200°. So the reset position is 200.

This can all be put together to form the two lines of TrioBASIC code that set up the position switch:

switch:

```
UNITS AXIS(0)=10'   Set unit conversion factor (°)
REPDIST=360
REP_OPTION=ON
PSWITCH(0,ON,0,11,ON,80,200)
```

This program uses the repeat distance set to 360 degrees and the repeat option `ON` so that the axis position will be maintained in the range 0..360 degrees.

READ_OP

Type: System Command

Syntax: `value = READ_OP(output [,finaloutput])`

Description: Returns the state of digital output logic.

If called with one parameter, it returns the state (1 or 0) of that particular output channel. If called with 2 parameters `READ_OP()` returns, in binary, the sum of the group of outputs.



READ_OP checks the state of the output logic. The output may be virtual or not powered and you will still see the logic state.

Parameters: **value:** The binary pattern of the selected outputs.

output: Output to return the value of/start of output group.

finaloutput: Last output of group.



The range of output to finaloutput must not exceed 32.

Example 1: In this example a single output is tested:

```
test:
    WAIT UNTIL READ_OP(12)=ON
    GOSUB place
```

Example 2: Check the group of 8 outputs and call a routine if any of them are ON.

```
op_bits = READ_OP(16,23)
IF op_bits<>0 THEN
    GOSUB check_outputs
ENDIF
```


SETCOM

Type: Command

Syntax: `SETCOM(baudrate,databits,stopbits,parity,port[,mode][,variable],timeout)[,linetype]`

Description: Allows the user to configure the serial port parameters and enable communication protocols.



By default the controller sets the serial ports to 38400 baud, 8 data bits, 1 stop bits and even parity.

Parameters:

- baudrate:** 1200, 2400, 4800, 9600, 19200 or 38400.
- databits:** 7 or 8.
- stopbits:** 1 or 2.
- parity:** 0 = none, 1 = odd, 2 = even.
- port:** 1 or 2.
- mode:**
 - 0 = XON/XOFF inactive
 - 1 = XON/XOFF active
 - 4 = MODBUS protocol (16 bit Integer)
 - 5 = Hostlink Slave
 - 6 = Hostlink Master
 - 7 = MODBUS protocol (32 bit IEEE floating point)
 - 8 = Reserved mode
 - 9 = MODBUS protocol (32bit long word integers)
- variable:**
 - 0 = Modbus uses VR
 - 1 = Modbus uses TABLE
- Timeout:** Communications timeout (msec). Default is 3.
- linetype:**
 - 0 = 4 wire RS485,
 - 1 = 2 wire RS485



PCMotion (mode=8) only supports port 1.

- Example 1:** Set port 1 to 19200 baud, 7 data bits, 2 stop bits even parity and **XON/XOFF** enabled.
- ```
SETCOM(19200,7,2,2,1,1)
```
- Example 2:** Set port 2 (RS485) to 9600 baud, 8 data bits, 1 stop bit no parity and no **XON/XOFF** handshake.
- ```
SETCOM(9600,8,1,0,2,0)
```
- Example 3:** The Modbus protocol is initialised by setting the mode parameter of the **SETCOM** instruction to 4. The **ADDRESS** parameter must also be set before the Modbus protocol is activated.
- ```
ADDRESS=1
SETCOM(19200,8,1,2,2,4)
```

---

## TIMER

---

- Type:** Command
- Syntax:** `TIMER(switch, output, pattern, time[,option])`
- Description:** The **TIMER** command allows an output or a selection of outputs to be set or cleared for a predefined period of time. There are 8 timer slots available, each can be assigned to any outputs. The timer can be configured to turn the output **ON** or **OFF**.
- Parameters:**
- switch:** The timer number in the range 0-7.
  - output:** Selects the physical output or first output in a group. Range 0-31.
  - pattern:** 1 = for a single output.  
Number = If set to a number this represents a binary array of outputs to be turned on. Range 0-65535.
  - time:** The period of operation in milliseconds.
  - option:** Inverts the output, set to 1 to turn **OFF** at start and **ON** at end.
- Example 1:** Use the **TIMER** function to flash an output when there is a motion error. The output lamp should flash with a 50% duty cycle at 5Hz.
- ```
WAIT UNTIL MOTION _ ERROR
```

```
WHILE MOTION_ERROR
  TIMER(0,8,1,100) `turns ON output 8 for 100milliseconds
  WA(200) `Waits 200 milliseconds to complete the 5Hz period
WEND
```

Example 2: Setting outputs 10, 12 and 13 **OFF** for 70 milliseconds following a registration event. The first output is set to 10 and the pattern is set to 13 (1 0 1 1 in binary) to enable the three outputs. Output 11 is still available for normal use. The option value is set to 1 to turn **OFF** the outputs for the period, they return to an **ON** state after the 70 milliseconds has elapsed.

```
WHILE running
  REGIST(3)
  WAIT UNTIL MARK
  TIMER(1,10,13,70,1)
WEND
```

Example 3: Firing output 10 for 250 milliseconds during the tracking phase of a **MOVELINK** Profile.

```
WHILE feed=ON
  MOVELINK(30,60,60,0,1)
  MOVELINK(70,100,0,60,1)
  WAIT LOADED `Wait until the tracking phase starts
  TIMER(2,10,1,250) `Fire the output during the tracking phase
  MOVELINK(-100,200,50,50,1)
WEND
```

Program Loops and Structures

_ (Line Cont)

Type: Special Character

Syntax: `expression _ start _ expression _ end`

Description: The line extension allows the user to split a long expression or command over more than one lines in the TrioBASIC program.



The split must be at the end of a parameter or keyword.

Parameters:

`expression _ start:` The start of the command or expression.

`expression _ end:` The end of the command or expression.

Example: Split the `SERVO _ READ` command over 2 lines so you can use all 8 parameters.

```
SERVO _ READ(123, MPOS AXIS(0), MPOS AXIS(1), MPOS AXIS(2), _
MPOS AXIS(3), MPOS AXIS(4), MPOS AXIS(5), MPOS AXIS(6))
```

BASICERROR

Type: System Command

Description: This command is used as part of an `ON... GOSUB` or `ON... GOTO`. This lets the user handle program errors. If the program ends for a reason other than normal stopping then the subroutine is executed, this is when `RUN _ ERROR<>31`.



You should include the `BASICERROR` statement as the first line of the program.

Example: When a program error occurs, print the error to the terminal and record the error number in a VR so that it can be displayed on an HMI through Modbus.

```
ON BASICERROR GOTO error _ routine
....(rest of program)

error _ routine:
  VR(100) = RUN _ ERROR
  PRINT "The error ";RUN _ ERROR[0];
  PRINT " occurred in line ";ERROR _ LINE[0]
STOP
```

See Also: RUN _ ERROR, ERROR _ LINE

FOR..TO.. STEP.. NEXT

Type: Program Structure

```
Syntax:  FOR variable = start TO end [STEP increment]
         commands
         NEXT variable
```

Description: A FOR program structure is used to execute a block of code a number of times. On entering this loop the variable is initialised to the value of start and the block of commands is then executed. Upon reaching the **NEXT** command the variable defined is incremented by the specified **STEP**. If the value of the variable is less than or equal to the end parameter then the block of commands is repeatedly executed. Once the variable is greater than the end value the program drops out of the **FOR..NEXT LOOP**.



FOR..NEXT loops can be nested up to 8 deep in each program.

Parameters:

- commands:** TrioBASIC statements that you wish to execute.
- variable:** A valid TrioBASIC variable. Either a global VR variable, or a local variable may be used.
- start:** Initial value for the variable.
- end:** Final value for the variable.
- increment:** The value that the variable is incremented by , this may be positive or negative.



The **STEP** increment is optional, if this is omitted then the **FOR NEXT** will increment by 1.



The variable can be adjusted or used within the structure.

Example 1: Turn ON outputs 10 to 18, using the variable to change the output.

```
FOR op_num=10 TO 18
    OP(op_num,ON)
NEXT op_num
```

Example 2: Index an axis from 5 to -5 using a negative STEP.

```
FOR dist=5 TO -5 STEP -0.25
    MOVEABS(dist)
    WAIT IDLE
    GOSUB pick_up
NEXT dist
```

Example 3: Using a FOR structure to move through a set of x,y positions. If there is a MOTIONERROR then the variables are set to a large values so the loop no longer repeats.w

```
FOR x=1 TO 8
    FOR y=1 TO 6
        MOVEABS(x*100,y*100)
        WAIT IDLE
        GOSUB operation
        IF MOTIONERROR THEN
            x=10
            y = 10
        ENDIF
    NEXT y
NEXT x
```

GOSUB..RETURN

Type: Program Structure

Syntax: GOSUB label

Description: Stores the position of the line after the GOSUB command and then branches to the label specified. Upon reaching the RETURN statement, control is returned to the stored line.



GOSUB..RETRUN loops can be nested up to 8 deep in each program.

Parameters: **commands:** TrioBASIC statements that you wish to execute.

label: A valid label that occurs in the program.



If the label does not exist an error message will be displayed at run time and the program execution halted.

You must not execute a RETURN without a GOSUB as a runtime error will be displayed and your program will stop.

Example:

```
WHILE machine_active
    GOSUB routine1
    GOSUB routine2
WEND
STOP `prevents running into sub routines when machine stopped.
```

```
routine1:
    PRINT "Measured Position=";MPOS;CHR(13);
RETURN
```

```
routine2:
    PRINT "Demand Position=";DPOS;CHR(13);
RETURN
```

Example 2:
Calculating values in a subroutine.

```
y=1
z=4
GOSUB calc
PRINT "New value = ", x
STOP
```

```
calc:
```

```
    x=y+z/2  
RETURN
```

See Also: GOTO

GOTO

Type: Program Structure

Syntax: GOTO label

Description: Identifies the next line of the program to be executed.



If the label does not exist an error message will be displayed at run time and the program execution halted

Parameters: label: A valid label that occurs in the program.

Example: Use a GOTO to repeat a section of your program after a bad input

```
start:  
PRINT#5, "Press 'y' for YES and 'n' for NO."  
GET#5, char  
IF char = 121 THEN  
    PRINT#5, "YES selected"  
ELSEIF char = 110 THEN  
    PRINT#5, "NO selected"  
ELSE  
    PRINT#5, "BAD selection"  
    GOTO start  
ENDIF
```

See Also: GOSUB

IDLE

Type: Axis Parameter

Description: Checks to see if an axis **MTYPE** is **IDLE**

Parameters: **value: TRUE** = **MTYPE** is empty (**MTYPE=0**).
FALSE = **MTYPE** has a command loaded (**MTYPE<>0**).

Example 1: Start a move and then suspend program execution until the move has finished.
Note: This does not necessarily imply that the axis is stationary in a servo motor system.

```
MOVE(100)
WAIT IDLE
PRINT "Move Done"
```

Example 2: If the axis does not have any moves loaded then load a new sequence.

```
IF IDLE AXIS(1) THEN
    MOVE(100)
    MOVE(50)
    MOVE(-150)
ENDIF
```

IF..THEN..ELSEIF..ELSE..ENDIF

Type: Program Structure

Syntax: **IF** expression **THEN** (commands)
ELSEIF expression **THEN** (commands)
ELSE (commands)
ENDIF

Description:	An IF program structure is used to execute a block of code after a valid expression. The structure will execute only one block of commands depending on the conditions. If multiple expressions are valid then the first will have its commands executed. If no expressions are valid and an ELSE is present the commands under the ELSE will be executed.	
Parameters:	expression:	Any valid TrioBASIC expression.
	commands:	TrioBASIC statements that you wish to execute.
	IF..THEN:	The first condition of an IF statement.
	ELSEIF:	An optional condition, can have multiple ELSEIF s.
	ELSE:	An optional catch condition if no other expressions are valid.
	ENDIF:	The end of the IF statement.

Example 1: Check for the batch to be complete, if it is then tell the user and process the batch.

```
IF count >= batch_size THEN
    PRINT #3,CURSOR(20);"  BATCH COMPLETE  ";
    GOSUB index `Index conveyor to clear batch
    count=0
ENDIF
```

Example 2: Use an **IF** statement to light a warning lamp when machine is running.

```
IF WDOG=ON THEN
    OP(warning, ON)
ELSE
    OP(warning, OFF)
ENDIF
```


Example 3: Use an **IF** structure to report the operating state of a machine.

```
IF operating_state=0 THEN
    PRINT#5, "Machine Running"
ELSEIF operating_state=1 THEN
    PRINT#5, "Machine Idle"
ELSEIF operating_state=2 THEN
    PRINT#5, "Machine Jammed"
ELSE
    PRINT#5, "Machine in unknown state"
ENDIF
```

NEXT

Type:	Program Structure
Description:	Used to mark the end of a FOR..NEXT loop.
See	FOR

ON.. GOSUB / GOTO

Type:	Program Structure
Syntax:	<pre>ON expression GOxxx label[,label1[,...]] ... label: commands RETURN ... label1: commands RETURN</pre>
Description:	<p>The expression is evaluated and then the integer part is used to select a label from the list. If the expression has the value 1 then the first label is used, 2 then the second label is used, and so on. Once a label is selected it is used with either GOSUB or GOTO.</p> <p> <i>If the value of the expression is less than 1 or greater than the number of labels the command is stepped through with no action. Once the label is selected a GOSUB is performed.</i></p>
Parameters:	<p>expression: Any valid TrioBASIC expression, should return a value 1 or greater.</p> <p>commands: TrioBASIC statements that you wish to execute.</p> <p>label: A valid label that occurs in the program.</p> <p>GOXXX: GOSUB or GOTO.</p>



If the label does not exist an error message will be displayed at run time and the program execution halted.

Example 1:

```
REPEAT
    GET #3,char
UNTIL 1<=char AND char<=3
ON char GOSUB mover,stopper,change
```

Example 2:

Use inputs from a PLC to determine which program to run.

```
ON (IN(4,6)+1)GOTO prog0, prog1, prog2, prog3, prog `select prog
GOTO continue `skip progs if unknown input selected
prog0:
RUN "tuning",2
GOTO continue
prog1:
RUN "cutting",2
GOTO continue
prog2:
RUN "packing",2
GOTO continue
prog3:
RUN "moving",2
GOTO continue
Prog4:
RUN "lifting",2
GOTO continue

continue:
...
```

See Also:

GOSUB, GOTO

REPEAT.. UNTIL

Type: Program Structure

Syntax: `REPEAT commands UNTIL expression`

Description: The **REPEAT..UNTIL** construct allows a block of commands to be continuously repeated until an expression becomes **TRUE**. **REPEAT..UNTIL** loops can be nested without limit.



*The commands inside a **REPEAT..UNTIL** structure will always be executed at least once, if you want them to only be executed on the expression you can use a **WHILE..WEND**.*

Parameters: **expression:** Any valid TrioBASIC expression.

commands: TrioBASIC statements that you wish to execute.

Example: A conveyor is to index 100mm at a speed of 1000mm/s wait for 0.5s and then repeat the cycle until an external counter signals to stop by setting input 4 on.

```
SPEED=1000
REPEAT
    MOVE(100)
    WAIT IDLE
    WA(500)
UNTIL IN(4)=ON
```

THEN

Type: Program Structure

Description: Forms part of an **IF** expression. See **IF** for further information.

Example: `IF MARK THEN`

```
    offset=REG _ POS
ELSE
    offset=0
ENDIF
```

TO

Type: Program Structure

See Also: FOR.. TO.. NEXT

UNTIL

Type: Program Structure

Description: Defines the end of a **REPEAT..UNTIL** multi-line loop, or part of a **WAIT UNTIL** structure. After the **UNTIL** statement is a condition which decides if program flow continues on the next line or at the **REPEAT** statement. **REPEAT..UNTIL** loops can be nested without limit.

Example: ‘ This loop loads a **CAMBOX** move each time Input 0 comes on.
‘ It continues until Input 6 is switched **OFF**.

```
REPEAT
    WAIT UNTIL IN(0)=OFF
    WAIT UNTIL IN(0)=ON
    CAMBOX(0,150,1,10000,1)
UNTIL IN(6)=OFF
```

WA

Type: Program Structure

Syntax: `WA(time)`

Description: Holds up program execution for the number of milliseconds specified in the parameter.

Parameters: `time:` The number of milliseconds to wait for.

Example: Turn output 17 off 2 seconds after switching output 11 off.

```
OP(11,OFF)
WA(2000)
OP(17,ON)
```

WAIT

Type: Command

Syntax: `WAIT UNTIL expression`

Description: Suspends program execution until the expression is **TRUE**.



*It is very common to use only **IDLE** and **LOADED** as the expression. In this situation the **UNTIL** is optional. When **IDLE** and **LOADED** are part of an expression **UNTIL** is required.*

Parameters: `condition:` Any valid TrioBASIC expression.

Example 1: The program waits until the measured position on axis 0 exceeds 150 then starts a movement on axis 7.

```
WAIT UNTIL MPOS AXIS(0)>150
MOVE(100) AXIS(7)
```

Example 2: Start a move and then suspend program execution until the move has finished.
Note: This does not necessarily imply that the axis is stationary in a servo motor system.

```
MOVE(100)
WAIT IDLE
PRINT "Move Done"
```

Example 3: Switch output 45 ON at start of MOVE(350) and OFF at the end of that move.

```
MOVE(100)
MOVE(350)
WAIT LOADED
OP(45,ON)
MOVE(200)
WAIT LOADED
OP(45,OFF)
```

Example 4: Force the program to wait until either the current move has finished or an input goes ON.



As the expression contains UNTIL and IN(12) the UNTIL is required.

```
MOVELINK(distance, link_dist, acceldist, deceldist, linkaxis)
WAIT UNTIL IDLE OR IN(12)=ON.
```

WEND

Type: Program Structure

Description: Marks the end of a WHILE..WEND loop.

See also: WHILE



WHILE..WEND loop can be nested without limit other than program size.

WHILE

Type: Program Structure

Syntax: `WHILE condition`

Description: The commands contained in the `WHILE..WEND` loop are continuously executed until the condition becomes `FALSE`. Execution then continues after the `WEND`. If the condition is false when the `WHILE` is first executed then the loop will be skipped.

Parameters: `condition:` Any valid logical TrioBASIC expression.

Example: While input 12 is off, move the base axis and flash an `LED` on output 10.

```
WHILE IN(12)=OFF
  MOVE(200)
  WAIT IDLE
  OP(10,OFF)
  MOVE(-200)
  WAIT IDLE
  OP(10,ON)
WEND
```

System Parameters and Commands

: (Colon)

Type: Special Character

Syntax: label:

Description: The colon character is used to terminate labels used as destinations for `GOTO` and `GOSUB` commands.



Labels can also be used to aid readability of code.

Parameters: **Label:** may be character strings of any length but only the first 32 characters are significant. Labels must be the first item on a line and should have no leading spaces.

Example: Use an `ON GOTO` structure to assign a value into `VR 10` depending on a local variable 'attempts'.

```
ON attempts GOTO label1, label2, label3  
GOTO continue
```

```
label1:  
VR(10)=1  
GOTO continue
```

```
label2:  
VR(10)=5  
GOTO continue
```

```
label3:  
VR(10)=2  
GOTO continue
```

```
continue:
```

Syntax: `statement: statement`

Description: The colon is also used to separate TrioBASIC statements on a multi-statement line.

Parameters: `statement:` any valid TrioBASIC statement. The colon separator must not be used after a **THEN** command in a multi-line **IF..THEN** construct. If a multi-statement line contains a **GOTO** the remaining statements will not be executed:

```
PRINT "Hello":GOTO Routine:PRINT "Goodbye"  
Goodbye will not be printed.
```

Similarly with **GOSUB** because subroutine calls return to the following line.

Example: Set the speed, a position in the table and execute a move all in one line.

```
SPEED=100:TABLE(10,123):MOVE(TABLE(10))
```

' (Comment)

Type: Special Character

Syntax: ``text`

Description: A single ' is used to mark the rest of a line as being a comment only with no execution significance.



Comments use memory space and so should be concise in very long programs. Comments have no effect on execution speed since they are not present in the compiled code.

Parameters: `text:` text any text string.

Example: Adding comment lines and comments after executable sections of code.

```
`PROGRAM TO ROTATE WHEEL  
turns=10  
`turns contains the number of turns required  
MOVE(turns)` the movement occurs here
```

(Hash)

Type: Special Character

Syntax: `command #channel`

Description: The # symbol is used to specify a communications channel to be used for serial input/output commands.

Parameters:

channel:	0	Ethernet port 0 (the command line).
	1	RS232 port 1.
	2	RS485 port 2.
	5	<i>Motion Perfect</i> user channel.
	6	<i>Motion Perfect</i> user channel.
	7	<i>Motion Perfect</i> user channel.
	8	Used for <i>Motion Perfect</i> internal operations.
	9	Used for <i>Motion Perfect</i> internal operations.
	40	Channel configured using the OPEN command.
	41	Channel configured using the OPEN command.
	42	Channel configured using the OPEN command.
	43	Channel configured using the OPEN command.
	44	Channel configured using the OPEN command.



Channels 5 to 9 are logical channels which are superimposed on to Port 0 by Motion Perfect.

Example 1: Printing Ascii strings to different channels.

```
PRINT #1,"Printing data to RS232 Channel"  
PRINT #5,"Printing data to Motion Perfect Terminal 5"
```

Example 2: Checking for and receiving characters on Channel 6.

```
WHILE KEY #6  
  GET #63, VR(123)  
WEND
```

See Also: GET, KEY, LINPUT, OPEN, PRINT

\$ (Dollar)

Type: Special Character

Syntax: \$number

Description: The \$ symbol is used to specify that the following signed 53bit number is in hexadecimal format.

Example 1: Store the hexadecimal value of 38F3B into VR 10 and -A58 into VR 11

```
VR(10)=$38F3B  
VR(11)=-$A58
```

Example 2: Turn on outputs 11,12,15,16

```
OP($CC00)
```

ADDRESS

Type: System Parameter

Syntax: ADDRESS=value

Description: Sets the RS485 or Modbus multi-drop address for the controller

Parameters: Node address: should be in the range of 1..32. If it is set to 255 addressing is not used and all 8 characters from the packet are sent through to the user.

Example: Initialise Modbus as node 5.

```
ADDRESS=5  
SETCOM(19200,8,1,2,1,4)
```

ANYBUS

Type: System Function

Syntax: `ANYBUS(function, slot [, parameters...])`

Description: This function allows the user to configure the active Anybus module and set the network to an operation state. Some networks have limitations on data types and size, please refer the Anybus data sheet for details.



Passive modules require no setup and will appear as a communication channel, they can then be used with PRINT, GET etc.

Parameters:

function:

- 0 = Configure map
- 1 = Configure module and start protocol
- 2 = Stop protocol
- 3 = Read status byte
- 4 = Auto configure mapping

Syntax: `value = ANYBUS(0,slot [, map, source [, index, type, count, direction]])`

Description: Assigns a VR or table point to the memory area that is updated over the network. Individual or all maps can be deleted using the first 4 parameters.

The current mapping can be printed to the terminal using the first 2 parameters.

Parameters:

value: **TRUE** = the command was successful.
FALSE = the command was unsuccessful.

slot: Module slot in which the Anybus is fitted.

map: Map number, use -1 to delete all maps.

source: Location for data on the MC464

- 1 delete map
- 0 VR
- 1 Table

index: Start position in data source.

type: The size and type of data that is sent across the bus

- 0 = boolean
- 1 = signed 8 bit integer
- 2 = signed 16 bit integer
- 3 = signed 32 bit integer
- 4 = unsigned 8 bit integer
- 5 = unsigned 16 bit integer
- 6 = unsigned 32 bit integer
- 7 = character
- 8 = enumeration
- 9-15 = Reserved
- 16 = signed 64 bit integer
- 17 = unsigned 64 bit integer
- 18 = floating point/real number

count: Number of data types mapped.

direction: Data direction

- 0 = data read into the controller
- 1 = data transmitted from the controller

Syntax: `value = ANYBUS(1,slot, address [, baud])`

Description: Resets the Anybus module, loads the mapping and then sets the network to operational mode using the parameters provided.

Parameters:

- value:** `TRUE` = the command was successful.
- slot:** Module slot in which the Anybus is fitted.
- address:** Module address, node number, `MAC` id. etc.
- baud:** Baud rate CC Link - required
 - 0 = 156 kbps
 - 1 = 625 kbps
 - 2 = 2.5 mbps
 - 3 = 5 mbps
 - 4 = 10 mbps

Baud rate Devicenet - optional

0 = 125kbps

1 = 250kbps

2 = 500kbps

3 = autobaud (default)

Baud rate Profibus - automatic, not required.

Syntax: `value = ANYBUS(2,slot)`

Description: Stops the cyclic data transfer.

Parameters: `value:` `TRUE` = the command was successful.
 `FALSE` = the command was unsuccessful.
 `slot:` Module slot in which the Anybus is fitted.

Syntax: `value = ANYBUS(3,slot)`

Description: Reads the status byte from the Anybus module.

Parameters: `value:` Anybus status byte:
 `Bits 0-2` Anybus State
 0 = `SETUP`
 1 = `NW _ INIT`
 2 = `WAIT _ PROCESS`
 3 = `IDLE`
 4 = `PROCESS _ ACTIVE`
 5 = `ERROR`
 6 = (reserved)
 7 = `EXCEPTION`
 `Bit 3` Supervisory bit
 0 = Module is not supervised.

1 = Module is supervised by another network device.
Bits 4-7 Reserved.
slot: Module slot in which the Anybus is fitted.

Syntax: `value = ANYBUS(4,slot, address, type, inoff, outoff)`

Description: Auto-configure and start the cyclic network. The mapping can still be read using function 0.



Currently only available for the Profibus network.

Parameters:

value: **TRUE** = the command was successful.
FALSE = the command was unsuccessful.

slot: Module slot in which the Anybus is fitted.

address: Module address, node number, MAC id. Etc.

type: Data type and location
0 = VR Integer
1 = Table Integer
2 = VR Float
3 = Table Float

inoff: Offset for inputs.

outoff: Offset for outputs.

Example 1: Configure Device Net with 2 16-bit integer inputs and 2 16-bit integer outputs. This data is transmitted cyclically using the 'Polled Connection' method. Ensure to configure the master identically to the slave otherwise the data will not transmit.

```
device _net:
slotnum=0 'Local variable with module slot number
`Map data
  map=FALSE
`Map received data
  map= ANYBUS(0, slotnum, 1, 0, 0, 2, 4, 0) `4*16-bit Int Rx
  IF map=TRUE THEN
```

```

`Map transmit data
  map= ANYBUS(0, slotnum, 2, 0, 4, 2, 4, 1) `4*16-bit Int Tx
ENDIF

IF map=FALSE THEN
  PRINT#term, "Mapping failed"
  STOP
ENDIF

`Print mapped data to the terminal
  ANYBUS(0,slotnum)

`Start Network
map= ANYBUS(1, slotnum, 3, 2) `MAC ID=3, Baud=500k
IF map=FALSE THEN
  PRINT#term, "Failed to start network"
  STOP
ELSE
  PRINT#term, "Network Started"
ENDIF
RETURN

```

Example 2: Configure CC-Link with 2 stations, both with 16 bits in, 16 bits out, 2 SINT16 in and 2 SINT16 out. Ensure that the master is configured identically and that the handshaking bits are implemented.

```

cc_link:
`Function 0 - Set up mapping
`station 1
  map = ANYBUS(0, slotnum, 0, 0, 0, 0, 16, 0) `16*BOOL Rx
  map = ANYBUS(0, slotnum, 1, 0, 1, 0, 16, 1) `16*BOOL Tx
  map = ANYBUS(0, slotnum, 2, 0, 2, 2, 2, 0) `2*16-bit Int Rx
  map = ANYBUS(0, slotnum, 3, 0, 4, 2, 2, 1) `2*16-bit Int Tx
`station 2
  map = ANYBUS(0, slotnum, 4, 0, 6, 0, 16, 0) `16*BOOL Rx
  map = ANYBUS(0, slotnum, 5, 0, 7, 0, 16, 1) `16*BOOL Tx
  map = ANYBUS(0, slotnum, 6, 0, 8, 2, 2, 0) `2*16-bit Int Rx
  map = ANYBUS(0, slotnum, 7, 0, 10, 2, 2, 1) `2*16-bit Int Tx

  ANYBUS(0,slotnum) `print mapping to terminal

`Function 1 - Start Protocol
  IF map = FALSE THEN
    map = ANYBUS(1, slotnum, 1, 2)
  ENDIF

```

Example 3: Configure Profibus using the automated mapping.

```

Profibus:
vrint=0
tableint=1
vrfloat=2

```

```
tablefloat=3
slotnum=0

`Function 4, read network mapping, configure and start.
map= ANYBUS(4, slotnum, 5, vrint, 100, 200)

IF map=FALSE THEN
    PRINT#term, "Failed to start network"
    STOP
ENDIF
ANYBUS(0,slotnum) `print mapping to terminal
```

AOUT

Type: Reserved Keyword

AUTORUN

Type: System Command

Description: Starts running all the programs that have been set to run at power up.



This command should not be used in a TrioBASIC program. You can use it in the command line or a TRIOINIT.bas in a SD card.

Example: Using a TRIOINIT.bas file in a SD card to load and run a new project.

```
FILE "LOAD _ PROJECT" "ROBOT _ ARM"
AUTORUN
```

AXIS_OFFSET

Type: Slot Parameter

Description: This parameter allows the base axis of a hardware group to be defined.

AXIS_OFFSET is set when the programmer wants to arrange the axis order on power up to be different to the system default. After the next power up, the new axis order will take effect. The value is saved in Flash memory.

Example 1: Set the built-in encoder port on a MC464 to be axis 16.

```
>>AXIS_OFFSET SLOT(-1) = 16

` check and set the axis offset for a P874 module in slot 1
IF AXIS_OFFSET SLOT(1)<>32 THEN
  ` change the axis offset and reset the controller.
  AXIS_OFFSET SLOT(1) = 32
  EX
ENDIF
```

Example 2: Set the base axis for module in slot 0 back to the system default

```
>>AXIS_OFFSET SLOT(0) = -1
```

BATTERY_LOW

Type: System Parameter (Read only)

Description: This parameter returns the condition of the non rechargeable battery.

Parameters: **Battery State:** 0: Battery voltage is OK.
1: Battery voltage is low and needs replacing.

BOOT_LOADER

Type: System Command (command line only)

Description: Used by *Motion Perfect* to enter the boot loader software.



DO NOT USE UNLESS INSTRUCTED BY TRIO OR A DISTRIBUTOR.

BREAK_ADD

Type: System Command

Syntax: `BREAK_ADD "program name" line_number`

Description: Used by *Motion Perfect* to insert a break point into the specified program at the specified line number.

If there is no code at the given line number `BREAK_ADD` will add the breakpoint at the next available line of code. I.e. If line 8 is empty but line 9 has "`NEXT x`" and a `BREAK_ADD` is issued for line 8, the break point will be added to line 9.



If a non existent line number is selected (i.e. line 50 when the program only has 40 lines), the controller will return an error.


Parameters: **Program Name:** the name of any program existing on your controller.

Line Number: the line number where to insert the breakpoint.

Example: Will add a break point at line 8 of program "simpletest"

```
BREAK_ADD "simpletest" 8
```

BREAK_DELETE

- Type:** System Command (command line only)
- Syntax:** `BREAK_DELETE "program name" line_number`
- Description:** Used by *Motion Perfect* to remove a break point from the specified program at the specified line number.
-  *If a non-existent line number is selected (i.e. line 50 when the program only has 40 lines), the controller will return an error.*
- Parameters:**
- Program Name:** the name of any program existing on your controller.
 - Line Number:** the line number where to remove the breakpoint.
- Example:** Remove the break point at line 8 of program "simpletest"
- ```
BREAK_DELETE "simpletest" 8
```

---

## BREAK\_LIST

---

- Type:** System Command (command line only)
- Syntax:** `BREAK_LIST "program name"`
- Description:** Used by *Motion Perfect* to returns a list of all the break points in the given program name. The program name, line number and the code associated with that line is displayed.
- Parameters:**
- Program Name:** the name of any program existing on your controller.
- Example:** Show the breakpoints from a program called "simpletest" with break points inserted on lines 8 and 11.
- ```
>>BREAK_LIST "simpletest"

Program: SIMPLETEST
Line 8: SERVO=ON
Line 11: BASE(0)
```

BREAK_RESET

- Type:** System Command (command line only)
- Syntax:** `BREAK_RESET "program name"`
- Description:** Used by *Motion Perfect* to remove all break points from the specified program.
- Parameters:** **Program Name:** the name of any program existing on your controller.
- Example:** Remove all break points from program "simpletest".
`BREAK_RESET "simpletest"`

CAN

- Type:** System Command
- Syntax:** `CAN(slot, function[, parameters]`
- Description:** This function allows the **CAN** communication channels to be controlled from the TrioBASIC. All *Motion Coordinator's* have a single built-in **CAN** channel which is normally
- In addition to using the **CAN** command to control **CAN** channels, there are specific protocol functions into the firmware. These functions are dedicated software modules which interface to particular devices. The built-in **CAN** channel will automatically scan for Trio I/O modules if the system parameter `CANIO_ADDRESS` is set to its default value of 32.

Channel:	Channel Number:	Maximum Baudrate:
Built-in CAN	-1	500 KHz



There are 16 message buffers in the controller.

Parameters:	slot:	Set to -1 for the built in CAN port.
	function:	<ul style="list-style-type: none"> 0 Read Register, do NOT use unless instructed by Trio or a Distributor. 1 Write Register, do NOT use unless instructed by Trio or a Distributor. 2 Initialise baud rate. 3 Check for message received. 4 Set transmit request. 5 Initialise message. 6 Read message. 7 Write message. 8 Read CANOpen Object. 9 Write CANOpen Object. 11 Initialise 29bit message 20 CAN mode 21 Enable CAN driver 22 Reset CAN message buffer 23 Specify CAN VR map

Syntax: `CAN(channel#,2,baudrate)`

Description: Initialise the baud rate of the CANBus

Parameters:	baudrate:	<ul style="list-style-type: none"> 0 1Mhz. 1 500kHz(default value). 2 250kHz.
--------------------	------------------	--

Syntax: `value=CAN(channel, 3, message)`

Description: Check to see if there is a new message in the message buffer

Parameters: `message` message buffer to check.
 `value:` `TRUE` = new message available.
 `FALSE` = no new message.

Syntax: `CAN(channel, 4, message)`

Description: Request to transmit the message in the specified buffer

Parameters: `message` message buffer to transmit.

Syntax: `CAN(channel#, 5, message, identifier, length, rw)`

Description: Initialise a message by configuring its buffers size and if it is transmit or receive.

Parameters: `message:` message buffer to initialise.
 `Identifier:` the identifier which the message buffer appears on the CANBus.
 `length:` the size of the message buffer.
 `rw:` 0 = read buffer
 1 = write buffer

Syntax: `(channel, 6, message, variable)`

Description: Read in the message from the specified buffer to a `VR` array.

 The first `VR` holds the identifier. The subsequent values hold the data bytes from the `CAN` packet.

Parameters: **message** the message buffer to read in.
 variable: the start position in the VR memory for the message to be written.

Syntax: **CAN(channel, 7, message, byte0, byte1..)**

Description: Write a message to a message buffer.

Parameters: **message:** the message buffer to write the message in.
 byte0: the first byte of the message.
 byte1: the second byte of the message.

Syntax: **CAN(channel, 8, transbuf, recbuf, object, subindex, variable)**

Description: Read a CANopen object. The first VR holds the variable data type. The subsequent values hold the data bytes from the CAN packet.

Parameters: **transbuf** the message buffer used to transmit.
 recbuf: the message buffer used to receive.
 object: the CANopen object to read.
 subindex: the sub index of the CANopen object to read.
 subindex: the start position in the VR memory for the message to be written.

Syntax: `CAN(channel, 9, transbuf, recbuf, format, object, subindex, value, {valuems})`

Description: Write a CANopen object. This function automatically requests the send so you do not need to use function 4.

Parameters:

- transbuf:** the message buffer used to transmit.
- recbuf:** the message buffer used to receive.
- format:** data size in bits 8, 16 or 32.
- object:** the CANopen object to write to.
- subindex:** the sub index of the CANopen object to write to.
- value:** the least significant 16 bits of the value to write.
- valuems:** the most significant 16 bit of the value to write.

Syntax: `CAN(channel#, 11, message, identifierms, identifier, length, rw)`

Description: Initialise a message by configuring its buffers size and if it is transmit or receive using 29 bit identifiers.

Parameters:

- message:** message buffer to initialise.
- identifierms:** the most significant 13 bits of the identifier.
- identifier:** the least significant 16 bits if the identifier.
- length:** the size of the message buffer.
- rw:** 0 = read buffer
1 = write buffer

Syntax: `CAN(channel, 20,mode)`

Description: Sets the CAN mode, normally this is done using `CANIO _ ADDRESS`

Parameters:

- mode: 0** Disable all CAN operations.
- 1** CANIO command mode.

- 2 **CANIO** mode (default).
- 3 **CANopenIO** mode (**CANOPEN _ OP _ RATE** controls the cycle period, default = 5ms).



UNLIKE CANIO _ ADDRESS, THIS IS NOT STORED IN FLASH EPROM

Syntax: `CAN(channel, 21,enable)`

Description: Provides the ability to reset the **CAN** driver. Do **NOT** use unless instructed by Trio or a Distributor.

Parameters:

Enable:	0	Disable.
	1	Enable (default).

Syntax: `CAN(channel, 22, message)`

Description: Reset a message buffer

Parameters: **message:** the message buffer to reset.

Syntax: `CAN(channel, 23, [message, map, offset, length, order, variable, direction])`

Description: Specify **CAN VR** map for use with **CANOPENIO** mode. If no parameters provided then current mappings are displayed

Parameters:

Message:	message buffer (0..15).
map:	MAP number (0..7).

offset: CAN buffer byte offset (0..7).
length: CAN buffer byte length (1..8).
order: Endian Byte order (0=Little, 1=Big).
variable: Index of VR to use (0..65535).
direction: Direction (0=Receive, 1=Transmit).

See Also: CANIO _ ADDRESS

CANIO_ADDRESS

Type: System Parameter (Stored in FLASH Eprom)

Description: CANIO_ADDRESS is used to set the operating mode of the CANBus. You can select between Trio CAN, DeviceNet, CANopen and a user configuration when implementing your own can protocol.

The value is held in flash EPROM in the controller and for most systems does not need to be set from the default value of 32.

Parameters:

Value:	Function
32:	Trio CAN I/O Master 64in/64out.
33:	DeviceNet.
34...39:	User range.
40:	CANopen I/O Master 64in/64out.
41:	CANopen I/O Master 128in/128ou
42:	CANopen I/O Master custom mapping.

CANIO_ENABLE

Type: System Parameter

Description: `CANIO_ENABLE` enables the Trio CAN I/O or CANopen protocol.

When using the Trio I/O protocol it is set automatically by firmware. You have to set `CANIO_ENABLE=ON` manually after configuring CANopen IO.

Parameters: `value: ON` = Enable the CAN protocol (default when `CANIO_ADDRESS=32`).
`OFF` = Disable the CAN protocol (default when `CANIO_ADDRESS<>32`).

CANIO_STATUS

Type: System Parameter

Description: Returns the status of the Trio CAN I/O network. You can set bit 4 to reset the network

Parameters:

<code>Bit 1</code>	set indicates an error from the I/O module 0,3,6 or 9
<code>Bit 2</code>	set indicates an error from the I/O module 1,4,7 or 10
<code>Bit 4</code>	set indicates an error from the I/O module 2,5,8 or 11
<code>Bit 8</code>	set indicates an error from the I/O module 12,13,14 or 15
<code>Bit 16</code>	should be set to re-initialise the CANIO network
<code>Bit 32</code>	is set when initialisation is complete

CANOPEN_OP_RATE

Type: System Parameter

Description: Used to adjust the transmission rate of CANopen I/O PDO telegrams.

Parameters: **Value:** Default is 5msec. Adjustable in 1msec steps.

CHECKSUM

Type: System Parameter (Read Only)

Description: The **CHECKSUM** parameter holds the checksum for the programs in battery backed **RAM**. On power up the checksum is recalculated and compared with the previously held value. If the checksum is incorrect the programs will not run.

CLEAR

Type: System Command

Description Sets all global (numbered) variables to 0 and sets local variables on the process on which command is run to 0.



*TrioBASIC does not clear the global variables automatically following a **RUN** command. This allows the global variables, which are all battery-backed to be used to hold information between program runs. Named local variables are always cleared prior to program running. If used in a program **CLEAR** sets local variables in this program only to zero as well as setting the global variables to zero. **CLEAR** does not alter the program in memory.*

Example: Setting and clearing **VR** values.

```
VR(0)=44  
VR(10)=12.3456  
VR(100)=2
```

```
PRINT VR(0),VR(10),VR(100)
CLEAR
PRINT VR(0),VR(10),VR(100)
```

On execution this would give an output such as:

```
44.0000      12.345      62.0000
0.0000      0.0000      0.0000
```

CLEAR_PARAMS

Type: Reserved Keyword.

COMMSError

Type: System Parameter

Description: This parameter returns all the communications errors that have occurred since the last time that it was initialised. It is a bitwise value defined as follows:

Bit	Value
0	RX Buffer overrun on Network channel
1	Re-transmit buffer overrun on Network channel
2	RX structure error on Network channel
3	TX structure error on Network channel
4	Port 0 Rx data ready
5	Port 0 Rx Overrun
6	Port 0 Parity Error
7	Port 0 Rx Frame Error
8	Port 1 Rx data ready
9	Port 1 Rx Overrun
10	Port 1 Parity Error

11	Port 1 Rx Frame Error
12	Port 2 Rx data ready
13	Port 2 Rx Overrun
14	Port 2 Parity Error
15	Port 2 Rx Frame Error
16	Error FO Network port
17	Error FO Network port
18	Error FO Network port
19	Error FO Network port

COMMSPOSITION

Type: Slot Parameter

Description: Returns if the expansion module is on the top or the bottom bus.

Parameters: **value:** -1 = built in controller
 1 = module is on the top bus
 0 = module is on the bottom bus or no module fitted

COMMSTYPE

Type: Slot Parameter (read only)

Description: This parameter returns the type of communications daughter board in a controller slot.

Value	Communication type
0	Empty slot
32	SERCOS
372	Panasonic module

39	Sync encoder port
40	FlexAxis 4
41	FlexAxis 8
42	Ethercat module
43	FlexAxis 8 SSI
62	Anybus module empty/ unrecognised
63	Anybus RS232
64	Anybus RS422
65	Anybus USB
66	Anybus Ethernet
67	Anybus Bluetooth
68	Anybus Zigbee
69	Anybus wireless LAN
70	Anybus RS485
71	Anybus Profibus
72	Anybus CC-Link
73	Anybus DeviceNet

Example:

Check that the correct Anybus module is fitted before starting initialisation.

```

IF COMMSTYPE SLOT(3) = 71
  GOSUB initialise _ profibus
ELSE
  PRINT#5, "No Profibus compact com module detected"
ENDIF

```

COMPILE

Type: System Command

Description: Forces compilation of the currently selected program. Program compilation is performed automatically by the system software prior to program `RUN` or when another program is `SELECTed`. This command is not therefore normally required.

See Also: `SELECT`, `COMPILE _ ALL`

COMPILE_ALL

Type: System Command

Description: Forces compilation of all programs. Program compilation is performed automatically by the system software prior to program `RUN` or when another program is `SELECTed`. This command is not therefore normally required.

See Also: `SELECT`, `COMPILE`

CONTROL

Type: System Parameter (Read Only)

Description: The Control parameter returns the type of *Motion Coordinator* in the system:

Controller `CONTROL`

MC464 `464`



When the Motion Coordinator is `LOCKED`, 1000 is added to the above numbers. e.g. a locked MC464 will return 1464.

Example 1: Checking the control value of a locked controller on the command line.

```
>>PRINT CONTROL
1464
>>
```

Example 2: Checking the controller type in a program, if it fails then stop the programs.

```
IF CONTROL <> 464 THEN
    PRINT#terminal, "This program was designed to run a MC464"
    HALT
ENDIF
```

COPY

Type: System Command (Command line only)

Description: Makes a copy of an existing program in memory under a new name.

Syntax: COPY "program" "newprogram"



Motion Perfect users should use the "Copy program..." function under the "Program" menu.

Parameters: **program:** the name of the program to be copied.
newprogram: the name of the copy.

Example: Make a backup of a program named motion.

```
>>COPY "MOTION" "MOTION _ BACK"
Compiling MOTION
Linking MOTION
Pass=4
OK
>>
```

CPU_EXCEPTIONS

Type: Reserved Keyword.

DATE

Type: System Function

Description: Returns/ Sets the current date held by the real time clock.

Syntax: `DATE=dd:mm:yy`

Description: Sets the date using the two digit year format

Parameters:

- `dd` day.
- `mm:` month.
- `yy:` last two digits of the year using the range 2000-2099.

Syntax: `DATE=DD:MM:YYYY`

Description: Set the date using the four digit year format

- `dd` day.
- `mm:` month.
- `yy:` full four digits of the year using the range 2000-2099.

Syntax: `Value = DATE({mode})`

Description: Read the date value from the real time clock

Parameters:

- `mode:` value.
- `none:` The number of days since 01/01/2000 (with 01/01/2000 = 0).
- `1:` The day of the current month.
- `2:` The month of the current year.
- `3:` The current year.

Example 1: Set the date to the 20th October 2012

```
>>DATE=20:10:12
or
>>DATE=20:10:2012
```

Example 2: Print the number of days since 1st January 2000 (with the 1st being day 0)

```
>>PRINT DATE
4676
>>
```

Example 3: Set a date then print it out using the US format

```
>>DATE=05:08:2008
>>PRINT DATE(1;"/";DATE(0;"/";DATE(2) `Prints the date in US
format.
08/05/2008
>>
```

DATE\$

Type: System Function

Syntax: DATE\$

Description: DATE\$ is used as part of a PRINT statement to write the current date from the real time clock.



The DATE\$ is set through the DATE command.

Parameters: The date is printed in the format DD/MM/YYYY.

The month is displayed in short text form.


Example: This will print the date in format for example: 20/10/15.
`PRINT #5,DATE$`

See Also: `DATE`, `PRINT`

DAY

Type: System Function

Description: Returns the current day as a number.

 *The DAY is set through the DATE command.*

Parameters: `value:` 0..6, Sunday is 0.

Example: Print some text depending on the day.

```
IF DAY=2 THEN
    PRINT#5, "Change filter"
ENDIF
```

See Also: `DATE`, `DAY$`

DAY\$

Type: System Function

Syntax: DAY\$

Description: Used as part of a **PRINT** statement to write the current day as a string.



The DAY\$ is set through the DATE command.

Example: Print the day as part of a welcome message.

```
PRINT#5, "Welcome to Trio on "; DAY$
```

See Also: DATE, DATE\$, DAY

DEL

Type: System Command (command line only)

Alternate Format: RM

Syntax: DEL "program"

Description: Used by *Motion Perfect* to delete a program from the controller memory.



Motion Perfect users should use the "Delete" function under the "Program" menu.

Parameters: **program:** the name of the program to be deleted.

Example: Delete a old program.

```
>>DEL "oldprog"
```


OK
>>

DEVICENET

Type: System Command

Syntax: `DEVICENET(slot, function[,parameters...])`

Description: The command `DEVICENET` is used to start and stop the DeviceNet slave function which is built into the *Motion Coordinator*.

Polled I/O data is transferred periodically:

From PLC to `[TABLE(poll_base) -> TABLE(poll_base + poll_in)]`

To PLC from `[TABLE(poll_base + poll_in + 1) -> TABLE(poll_base + poll_in + poll_out)]`

Parameters:

- `slot`: Set -1 for built-in CAN port .
- `func`: 0 = Start the DeviceNet slave protocol on the given slot.
1 = Stop the DeviceNet protocol.
2 = Put startup baudrate into Flash `EPROM`.

Syntax: `DEVICENET(slot, 0, baud, mac_id, poll_base, poll_in, poll_out)`

Description: Start the DeviceNet protocol using the specified parameters

Parameters:

- `baud`: Set to 125, 250 or 500 to specify the baud rate in kHz.
- `mac_id`: The ID which the *Motion Coordinator* will use to identify itself on the DeviceNet network. Range 0..63.
- `poll_base`: The first `TABLE` location to be transferred as poll data.
- `poll_in`: Number of words to be received during poll. Range 0..4.
- `poll_out`: Number of words to be sent during poll. Range 0..4.

Syntax: `DEVICENET(slot, 1)`

Description: Stop the DeviceNet protocol from running

Syntax: `DEVICENET(slot, 2, baud)`

Description: Store the baud rate in flash EPROM for power up.

Parameters: **baud:** Set to 125, 250 or 500 to specify the baud rate in kHz.

Example 1: Start the DeviceNet protocol on the built-in CAN port;
`DEVICENET(-1,0,500,30,0,4,2)`

Example 2: Stop the DeviceNet protocol on the CAN board in slot 2;
`DEVICENET(2,1)`

Example 3: Set the CAN board in slot 0 to have a baud rate of 125k bps on power-up;
`DEVICENET(0,2,125)`

DIR

Type: System Command (command line only)

Alternate Format: `LS`

Syntax: `DIR [option]`

Description: Prints a list of all programs including their size and `RUNTYPE`.

Parameters:

<code>option:</code>	<code>none</code>	Controller memory.
	<code>d</code>	SD card memory.
	<code>s</code>	Reserved function.
	<code>x</code>	Extended controller memory for <i>Motion Perfect</i> use only.

DISPLAY

Type: System Parameter

Description: Determines which group of the I/O channels are to be displayed on the LCD.

Parameters:

<code>0:</code>	Inputs 0-15 (default value).
<code>1:</code>	Inputs 16-31.
<code>2:</code>	Outputs 0-15 (0-7 unused on existing controllers).
<code>3:</code>	Outputs 16-31.
<code>888:</code>	Reserved value.

Example: Show outputs 16-31

```
>>DISPLAY=3
>>
```

DLINK

Type: System Command

Syntax: `DLINK(function,...)`

Description: This is a specialised command, to allow access to the **SLM**[™] digital drive interface. The axis parameters have to be initialised by the **DLINK** function 2 command before the interface can be used for controlling an external drive.



THE CURRENT SLM[™] SOFTWARE DICTATES THAT THE DRIVE MUST BE POWERED UP AFTER POWER IS APPLIED TO THE MOTION COORDINATOR/ SLM[™].

Parameters:

Function:	Specifies the required function.
0	= Reserved function
1	= Reserved function
2	= Check for presence SLM module
3	= Check for presence of SLM servo drive
4	= Assign a <i>Motion Coordinator</i> axis to a SLM channel
5	= Read an SLM parameter
6	= Write an SLM parameter
7	= Write an SLM command
8	= Read a drive parameter
9	= Returns slot and asic number associated with an axis
10	= Read an EEPROM parameter

Syntax: `value = DLINK(2, slot, com)`

Description: Check for presence **SLM** module on rear of motor.

Parameters:

value:	Returns 1 if the SLM is answering, otherwise it returns 0.
slot:	The communications slot where the module is connected.

com: The communication channel where the axis is connected in the module.

Example: Check for a **SLM** module on slot 0, communication channel 0.

```
>>? DLINK(2,0,0)
1.0000
>>
.0000
>>
```

Syntax: `value = DLINK(3, slot, com)`

Description: Check for presence of **SLM** servo drive, such as MultiAx.

Parameters:

- value:** Returns 1 if the drive is answering, otherwise it returns 0..
- slot:** The communications slot where the module is connected
- com:** The communication channel where the axis is connected in the module.

Example: Check for a **SLM** drive on slot 0, communication channel 0.

```
>>? DLINK(3,0,0)
0.0000
>>
```

Syntax: `value = DLINK(4, slot, co, axis)`

Description: Assign a *Motion Coordinator* axis to a **SLM** channel.

Parameters:

- value:** Returns **TRUE** if successful otherwise returns **FALSE**.
- slot:** The communications slot where the module is connected
- com:** The communication channel where the axis is connected in the module.
- axis:** The axis to be associated with this drive. If this axis is already assigned then it will fail. The **ATYPE** of this axis will be set to 11.

Example: Assign axis 0 to the drive connected to slot 0 and communication channel 0.
>>DLINK(4,0,0,0)

Syntax: value = DLINK(5, axis, parameter)

Description: Read an SLM parameter

Parameters:

- value:** The value returned from SLM, returns -1 if the command fails.
- axis:** The axis number associated with the drive.
- parameter:** The number of the SLM parameter to be read. This is normally in the range 0...127. See the drive documentation for further information.

Example: Print the value of the SLM parameter 5 from axis 0.
>>PRINT DLINK(5,0,1)
463.0000
>>

Syntax: value = DLINK(6, axis, parameter, value)

Description: Write an SLM parameter

Parameters:

- value:** Returns **TRUE** if successful otherwise returns **FALSE**.
- axis:** The axis number associated with the drive.
- parameter:** The number of the SLM parameter to be read. This is normally in the range 0...127. See the drive documentation for further information.
- value:** The value to write to the parameter.

Example: Set SLM parameter 0 to the value 0 on axis 0.
>>DLINK(6,0,0,0)
>>

Syntax: `value = DLINK(7, axis, command)`

Description: Write an *SLM* command.

Parameters: `value:` Returns **TRUE** if successful otherwise returns **FALSE**.
 `axis:` The axis number associated with the drive.
 `command:` The command number. (See drive documentation).

Example: Write *SLM* command 250 to axis 0

```
>>PRINT DLINK(7,0,250)
1.0000
>>
```

Syntax: `value = DLINK(8, axis, parameter)`

Description: Read a drive parameter

Parameters: `value:` The value returned from the drive, returns -1 if the command fails.
 `axis:` The axis number associated with the drive.
 `Parameter:` The number of the drive parameter to be read. This is normally in the range 0...127. See the drive documentation for further information.

Example: Read drive parameter 53248 for axis 0

```
>>PRINT DLINK(8,0,53248)
20504.0000
>>
```

Syntax: `value = DLINK(9, axis)`

Description: Return slot and communication channel associated with an axis.

Parameters: `value:` 10 x slot number + communication channel, returns -1 if the command fails.
 `axis:` The axis number associated with the drive.

Example: Read axis 2 **SLM** information

```
>>PRINT DLINK(9,2)
>>11.0000
```

This example is for slot 1, communication channel 1.

Syntax: value = DLINK(10, axis, parameter)

Description: Read an **EEPROM** parameter

Parameters:

- value:** The value from the **EEPROM** value, returns -1 if the command fails.
- axis:** The axis number associated with the drive.
- parameter:** **EEPROM** parameter number. (See drive documentation).

Example: Return the **EEPROM** parameter 29, the Flux Angle from axis 0

```
>>PRINT DLINK(10,0,29)
>>62128.0000
```

DUMP

Type: Reserved Keyword.

EDPROG

Type: System Command

Alternate Format: &

Syntax: EDPROG mode

Description: This is a special command that may be used to manipulate the SELECTed programs on the controller.



It is not normally used except by Motion Perfect.

Parameters:

mode:	C	Prints the name of the currently selected program.
	D	Delete line.
	I	insert string.
	K	Print checksum.
	L	Print lines.
	N	Print number of lines.
	R	Replace line.
	Z	Print checksum of specified program.

Syntax: EDPROG C

Description: Prints the name of the currently selected program.


Syntax: `EDPROG line _no D`

Description: Deletes the specified line

Parameters: `line _no:` Any valid line number form the **SELECTED** program.

Syntax: `EDPROG line _no I,string`

Description: Insert the text string in the currently selected program at the specified line.

 *you should NOT enclose the string in quotes unless they need to be inserted into the program.*

Parameters: `line _no:` The line to insert the string.
`string:` The text string to insert into the **SELECTED** program.

Syntax: `EDPROG K`

Description: Print the checksum of the system software

Syntax: `EDPROG start, end L`

Description: Print the lines of the currently selected program between start and end

Parameters: `start:` The first line to print from the **SELECTED** program.
`end:` The last line to print from the **SELECTED** program.

Syntax: `EDPROG N`

Description: Print the number of lines in the currently selected program

Syntax: `EDPROG line R, string`

Description: Replace the line <line> in the currently selected program with the text <string>.



you should NOT enclose the string in quotes unless they need to be inserted into the program.

Parameters: `line_no:` The line to replace.

`string:` The text string to replace the line in the **SELEDT**ed program.

Syntax: `EDPROG Z, progname`

Description: Print the CRC checksum of the specified program.

Parameters: Returns the checksum using standard CCITT 16 bit generator polynomial.

See Also: `SELECT`

EDPROG1

Type: Reserved Keyword

EPROM

Type: Reserved Keyword

EPROM_STATUS

Type: Reserved Keyword.

ERROR_AXIS

Type: Returns the number of the axis that caused the `MOTION _ ERROR`.



ERROR _ AXIS should only be read when `MOTION _ ERROR` <> 0.

Parameters: **value:** Number of the axis that caused the `MOTION _ ERROR`.



This default value is 0 and is reset to 0 after `DATUM(0)`.

Example: If there is a motion error print error information to the user.

```
IF MOTION _ ERROR THEN
  PRINT#5, "Axis to cause error = "; ERROR _ AXIS
  PRINT#5, "AXISSTATUS of ERROR _ AXIS = "AXISSTATUS AXIS( ERROR _
  AXIS)
ENDIF
```

See Also: **AXISSTATUS, MOTION _ ERROR, FE _ LATCH**

ERROR_LINE

Type: Process Parameter (Read Only)

Description: Stores the number of the line which caused the last TrioBASIC error. This value is only valid when the **BASICERROR** is **TRUE**.

 *This parameter is held independently for each process.*

Parameters: **value:** The line number on the specified process that caused the error.

Example: Display the **ERROR _ LINE** as part of a sub routine called by 'ON **BASICERROR** GOTO'

```
error _ routine:
  VR(100) = RUN _ ERROR
  PRINT "The error ";RUN _ ERROR[0];
  PRINT " occurred in line ";ERROR _ LINE[0]
STOP
```


See Also: **BASICERROR, RUN _ ERROR**

ETHERNET

Type: System Command

Syntax: **ETHERNET**(rw, slot, function [,parameters...])

Description: The command **ETHERNET** is used to configure the operation of the Ethernet port.

 *Many of the **ETHERNET** functions are command line only; these are stored in flash EPROM and are then used on power up.*

Parameters:	rw:	specifies the required action. 0 = Read 1 = Write
	slot:	Set to -1 for the built in Ethernet port
	function:	Function number must be one of the following values. 0 = IP Address 1 = Reserved function 2 = Subnet Mask 3 = MAC address 4 = Default Port Number 5 = Token Port Number 6 = PRP firmware version (read only) 7 = Modbus TCP mode 8 = Default Gateway 9 = Data configuration 10 = Modbus TCP port number 11 = ARP cache 12 = Reserved function 13 = reserved function 14 = Configure endpoints for Modbus TCP or Ethernet IP

Syntax: `ETHERNET(rw, slot, 0 [,byte1, byte2, byte3])`

Description: Prints or writes the Ethernet **IP** address. This is command line only.



*You must power cycle the controller or perform **EX(1)** to apply the new **IP** address.*

Parameters:

- byte1:** The first byte of the **IP** address.
- byte2:** The second byte of the **IP** address.
- byte3:** The third byte of the **IP** address.



The default address is 192.168.0.250

Example: Read the current IP address and then set a new IP address into the controller and perform an **EX(1)** to activate the address



PERFORMING AN EX(1) AS IN THIS EXAMPLE WILL CLOSE THE COMMUNICATIONS AND YOU WILL ONLY BE ABLE TO COMMUNICATE AGAIN USING THE NEW IP ADDRESS.

```
>>ETHERNET(0, -1, 0)
192.168.0.250
>>ETHERNET(1, -1, 0, 192, 168, 0, 201)
>>EX(1)
>>
```

Syntax: `ETHERNET(rw, slot, 2 [,byte1, byte2, byte3])`

Description: Prints or writes the Subnet Mask. This is command line only.



*You must power cycle the controller or perform **EX(1)** to apply the new IP address.*

Parameters:

- byte1:** The first byte of the Subnet Mask.
- byte2:** The second byte of the Subnet Mask.
- byte3:** The third byte of the Subnet Mask.



The default Subnet Mask is 255.255.255.0

Example: Read the subnet mask and write a new value

```
>>ETHERNET(0, -1, 0)
255.255.255.0
>>ETHERNET(1, -1, 2, 255, 255, 128, 0)
>>
```

Syntax: `ETHERNET(0, slot, 3)`

Description: Prints the **MAC** address. This is command line only.

 *This function is read only.*

Parameters: The **MAC** address is unique to your controller.

Example: Read the **MAC** address of a controller

```
>>ETHERNET(0, -1, 3)
00:06:70:00:00:FA
>>
```

Syntax: `ETHERNET(rw, slot, 4 [, port])`

Description: Prints or writes the default port number. This is command line only.

 **THE DEFAULT VALUE IS USED BY *MOTION PERFECT* AND *TRIO PCMOTION* AND SHOULD NOT BE CHANGED UNLESS ABSOLUTELY NECESSARY.**

Parameters: **port:** The port used for the main command line in the controller (default 23).

Syntax: `ETHERNET(rw, slot, 5 [, port])`

Description: Prints or writes the default port number for token channel which is used by *TrioPCMotion*. This is command line only.

 **THE DEFAULT VALUE IS USED BY *TRIO PCMOTION* AND SHOULD NOT BE CHANGED UNLESS ABSOLUTELY NECESSARY.**

Parameters: **port:** The port used for the token channel in the controller. (default 3240).

Syntax: `Ethernet(0,slot,6)`

Description: Reads the communications processor s firmware version. This is command line only.



This function is read only.

Parameters: `port:` Returns the flash application version and the bootloader version.

Example: Read the communications processor firmware with application version 61 and boot loader version 22.

```
>>ETHERNET(0, -1, 6)
61;22
>>
```

Syntax: `Ethernet(rw, slot, 7 [,mode])`

Description: Sets the Modbus TCP data type. This value is stored in RAM and so must be initialised every time the controller powers up. This can be done in a TrioBASIC program for example **STARTUP**.



This must be configured before the Modbus master opens the port.

Parameters: `mode:` 0 = 16bit integer (default value).
1 = 32bit single precision floating point.

Example 4: Initialise the Modbus TCP port for floating point data.

```
ETHERNET(1,1,7,1)
```

Syntax: `ETHERNET(rw, slot, 8 [,byte1, byte2, byte3])`

Description: Prints or writes the Default Gateway. This is command line only.



You must power cycle the controller or perform EX(1) to apply the new Default Gateway.

Parameters:

<code>byte1:</code>	The first byte of the Default Gateway.
<code>byte2:</code>	The second byte of the Default Gateway.
<code>byte3:</code>	The third byte of the Default Gateway.

Example: Print then change the value of the default gateway.

```
>>ETHERNET(0, -1, 8)
192.168.0.225
>> ETHERNET(0,-1,8, 192, 168, 0, 150)
>>
```

Syntax: `Ethernet(rw, slot, 9 [,mode])`

Description: Sets the Modbus TCP data source. This value is stored in RAM and so must be initialised every time the controller powers up. This can be done in a TrioBASIC program for example STARTUP.



This must be configured before the Modbus master opens the port.

Parameters:

<code>mode:</code>	0 = VR (default value).
	1 = Table.

Example 4: Initialise the Modbus TCP port for table data.

```
ETHERNET(2,1,9,1)
```

Syntax: **ETHERNET**(rw, slot, 10 [, port])

Description: Prints or writes the default port number for token channel which is used by Modbus TCP. This is command line only.



THE DEFAULT VALUE IS USED BY MODBUS AND SHOULD NOT BE CHANGED UNLESS ABSOLUTELY NECESSARY.

Parameters: **port:** The port used for the token channel in the controller. (default 502).

Syntax: **Ethernet**(0, slot, 11)

Description: Reads the ARP cache. This is command line only.



This function is read only.

Syntax: **ETHERNET**(1, slot, 14, endpoint _ id, parameter _ index, parameter _ value)

Description: This function allows the user to configure Ethernet IP and Modbus at a low level. The default values allow a master to connect without any configuration on the Controller side. These settings are stored in RAM and so must be initialised every time the controller powers up. This can be done in a TrioBASIC program for example STARTUP.

Parameters:

endpoint _ id:	This allows you to specify which end point you are reading or writing 0 = Modbus TCP 1 = Ethernet IP Assembly Object, Instance 100 (input) 2 = Ethernet IP Assembly Object, Instance 100 (output)
parameter _ index:	This parameter selects which of the endpoint variables you are reading or writing 0 = Address

`parameter _ value:`

- 1 = Data location
- 2 = Data format
- 3 = Length
- 4 = Class

If the `parameter_index` is address (0), this is the start position of the data location.

If `parameter_index` is data location (1), this is the location of the data on the controller.

0 = Register (reserved use)

1 = IO input

2 = IO output

3 = `VR` (default value)

4 = Table

5 = Digital IO Input

6 = Digital IO Output

7 = Analogue IO Input

8 = Analogue IO Input

If the `parameter_index` is data format (2), this specifies the precision of the data.

0 = Integer 16 bit (default value)

1 = Integer 32 bit

2 = Floating point 32 bit

3 = Floating point 64 bit

If the `parameter_index` is length (3), this is the number of the data locations returned.

If the `parameter_index` is class (4), this returns the class. This function is read only.

4 = Ethernet `IP`

68 = Modbus

If the `parameter_index` is Instance (5), this returns the instance of the endpoint. This function is read only.

0 = Modbus

100 = Ethernet `IP` input

101 = Ethernet IP output

Example 1: Configure Modbus using Function 14 to use Table and floating point 64bit.

```
ETHERNET(1, -1, 14, 0, 1, 4)
ETHERNET(1, -1, 14, 0, 2, 3)
```

Example 2: Configure Ethernet IP for 50 **TABLE** inputs starting at 200 and 50 table outputs starting at 300 all at 32bit float.

```
`Inputs
ETHERNET(1, -1, 14, 1,0,200)
ETHERNET(1, -1, 14, 1, 1, 4)
ETHERNET(1, -1, 14, 1, 2, 2)
ETHERNET(1, -1, 14, 1, 3, 50)
`Outputs
ETHERNET(1, -1, 14, 2,0,300)
ETHERNET(1, -1, 14, 2, 1, 4)
```

EX

Type: System Command

Syntax: **EX**(processor)

Description: Software reset. Resets the controller as if it were being powered up again.



*When performing an **EX** on the command line you will see the controller start up information that provides details of your controller configuration.*

On **EX** the following actions occur:

- The global numbered (**VR**) variables remain in memory.
- The base axis array is reset to 0,1,2... on all processes
- Axis following errors are cleared
- Watchdog is set **OFF**
- Programs may be run depending on **POWER _ UP** and **RUNTYPE** settings
- **ALL** axis parameters are reset.

EX may be included in a program. This can be useful following a run time error. Care must be taken to ensure it is safe to restart the program.



*When running Motion Perfect executing an **EX** command is not allowed. The same effect as an **EX** can be obtained by using “Reset the controller...” under the “Controller” menu in Motion Perfect. To simply re-start the programs, use the **AUTORUN** command.*

Parameters: **0 or None:** Software resets the controller and maintains communications.
 1: Software resets the controller and communications.

EXECUTE

Type: System Command

Description: Used to implement the remote command execution via the Trio *PCMotion* ActiveX. For more details see the section on using the *PCMotion*

FEATURE_ENABLE

Type: System Function

Syntax: **FEATURE _ ENABLE(feature number)**

Description: *Motion Coordinators* have the ability to unlock additional features by entering a “Feature Enable Code”. This function is used to enable protected features, such as additional axes on digital dive networks or other programming languages. This can only be run on the command line.



It is recommended to use Motion Perfect2 to enter and store the feature enable codes.



*You can purchase additional feature codes from the Trio Website or through your distributor, you will need the **SERIAL _ NUMBER** of the controller.*



IF YOU ENTER THE WRONG PASSWORD 3 TIMES THE CONTROLLER WILL ENTER AN ATTACK STATE WHERE IT STOPS COMMUNICATING. YOU CAN RESUME NORMAL OPERATION BY POWER CYCLING THE CONTROLLER.

Parameters:

feature number: None = Prints the security code and currently enabled features.
 0 = 1 additional axis
 1 = 2 additional axes
 2 = 4 additional axes
 3 = 8 additional axes
 4 = 16 additional axes
 5 = 32 additional axes
 6-11 = Reserved use
 12 = 1 additional axis
 13 = 2 additional axes
 14 = 4 additional axes
 15 = 8 additional axes
 16 = 16 additional axes
 17 = 32 additional axes
 18-20 = Reserved use
 21 = IEC runtime
 22-31 = Reserved use

password: If entering a feature a password is requested.



When entering the passwords always enter the characters in upper case. Take care to check that 0 (zero) is not confused with O and 1 (one) is not confused with I.

Example 1:

Check the enabled features on a controller

```
>>FEATURE _ ENABLE
Security code=17980000000028
Enabled features: 0 1
```



Features 0 and 1 are enabled so an additional 3 axes on top of the built in axes included with the module.

Example 2:

Enable an additional 4 axes (feature 2). For this controller and this feature, the password is 5P0APT.

```
>>FEATURE _ ENABLE(2)
Feature 2 Password=5P0APT
>>
```

```
>>FEATURE_ENABLE
Security code=17980000000028
Enabled features: 0 1 2
```

See Also: SERIAL_NUMBER

FLASH_DUMP

Type: Reserved Keyword.

FLASHTABLE

Type: System Function

Syntax: FLASHTABLE(function)

Description: Copies user data in RAM to and from the permanent FLASH memory.

Parameters: **function:** Specifies the required action.

- 1: Write a page of TABLE data into flash EPROM.
- 2: Read a page of flash memory into TABLE data.

flashpage: The index number (0 ... 31) of a 16k page of Flash EPROM where the table data is to be stored to or retrieved from.

tablepage: The index number (0 ... INT(TSIZE/16000)) of the page in table memory where the data is to be copied from or restored to.

Example: Save the TABLE page 2 data in locations TABLE(32000)
 -TABLE(47999) to FLASH memory page 5.
 FLASHTABLE(1,5,2)

See Also: **FLASHVR**

FLASHVR

Type: System Function

Syntax: **FLASHVR(function)**

Description: Copies user data in **RAM** to and from the permanent flash memory.

Parameters: **function:** Specifies the required action.
 -1 = Stores the entire **TABLE** to the Flash **EPROM** and use it to replace the **RAM** table data on power-up.
 -2 = Stop using the **EPROM** copy of table during power-up.



AFTER USING FUNCTION -1, ANY CHANGED TABLE DATA WILL BE OVERWRITTEN ON THE NEXT POWER UP OR RESET.



*In Motion Coordinator with non-volatile VR storage, positive <function> values will be ignored and the **FLASHVR** does not store VR values to **FLASH** memory.*

Example: Save the entire **TABLE** data to **FLASH** memory.

FLASHVR(-1)

See Also: **FLASHTABLE**

FPGA_VERSION

Type: Slot Parameter

Description: Returns the **FPGA** version.

Parameters: **value:** The **FPGA** version of the specified slot

FPU_EXCEPTIONS

Type: Reserved Keyword.

FRAME

Type: System Parameter

Description: Used to specify which “frame” to operate within when employing frame transformations. Frame transformations are used to allow movements to be specified in a multi-axis coordinate frame of reference which do not correspond one-to-one with the axes.



A number of pre-defined FRAMEs are available. Please contact your Trio distributor for details.

Parameters: **value:** 0 - Default
 1 - 2 axis **SCARA** robot
 2 - **XY** single belt
 3 - Double **XY** single belt
 4 - 2 axis pick and place
 5 - 2x2 Matrix transform

- 6 - Polar to Cartesian transformation
- 10 - Cartesian to polar transformation
- 13 - Dual arm robot transformation



See www.triomotion.com or your distributor for more details.

Example: An example is a **SCARA** robot arm with jointed axes. For the end tip of the robot arm to perform straight line movements in X-Y the motors need to move in a pattern determined by the robot's geometry.

Once you set **FRAME** = 1 you can specify x,y positions of the end tip through the axes 0 and 1.

FRAME_TRANS

Type: Mathematical Function

Description: Reserved Keyword

FREE

Type: System Parameter (Read Only)

Description: Returns the amount of program memory available for user programs.



Each line takes a minimum of 4 characters (bytes) in memory. This is for the length of this line, the length of the previous line, number of spaces at the beginning of the line and a single command token. Additional commands need one byte per token, most other data is held as ASCII.

The Motion Coordinator compiles programs before they are run, this means that a little under twice the memory is required to be able to run a program.

Parameters: **value:** The amount of available user memory in byte.

Example 1: Check the available memory on the command line

```
>>PRINT FREE
47104.0000
>>
```

See Also: DIR

HALT

Type: System Command.

Description: Halts execution of all running programs. You can use **HALT** in a program.



HALT DOES NOT STOP ANY MOTION. CURRENTLY EXECUTING, OR BUFFERED MOVES WILL CONTINUE UNLESS THEY ARE TERMINATED WITH A CANCEL OR RAPIDSTOP COMMAND.

Example: Use the command line to stop two running programs:

```
>>HALT%[Process 20:Line 2] (31) - Program is stopped
%[Process 21:Line 1] (31) - Program is stopped
>>
```

See Also: CANCEL, RAPIDSTOP, STOP

HLM_COMMAND

Type: Remote Command

Syntax: `HLM_COMMAND(command, port[, node[, mc_area/mode[, mc_offset]]])`

Description: The `HLM_COMMAND` command performs a specific Host Link command operation to one or to all Host Link Slaves on the selected port. Program execution will be paused until the response string has been received or the timeout time has elapsed. The timeout time is specified by using the `HLM_TIMEOUT` parameter. The status of the transfer can be monitored with the `HLM_STATUS` parameter.

Parameters:

command

The selection of the Host Link operation to perform:

HLM_MREAD (or value 0): This performs the Host Link `PC MODEL READ (MM)` command to read the CPU Unit model code. The result is written to the MC Unit variable specified by `mc_area` and `mc_offset`.

HLM_TEST (or value 1): This performs the Host Link `TEST (TS)` command to check correct communication by sending string “MCxxx TEST STRING” and checking the echoed string. Check the `HLM_STATUS` parameter for the result.

HLM_ABORT (or value 2): This performs the Host Link `ABORT (XZ)` command to abort the Host Link command that is currently being processed. The `ABORT` command does not receive a response.

HLM_INIT (or value 3): This performs the Host Link `INITIALIZE (**)` command to initialize the transmission control procedure of all Slave Units.

HLM_STWR (or value 4): This performs the Host Link `STATUS WRITE (SC)` command to change the operating mode of the CPU Unit.

port: The specified serial port. (See specific controller specification for numbers)

node:(for HLM_MREAD, HLM_TEST, HLM_ABORT and HLM_STWR): The Slave node number to send the Host Link command to. Range: [0, 31].

mode: (for HLM_STWR) The specified CPU Unit operating mode.
0 **PROGRAM** mode

	2 MONITOR mode
	3 RUN mode
<code>mc_area:(for HLM_MREAD)</code>	The MC Unit's memory selection to write the received data to.
<code>mc_offset:(for HLM_MREAD)</code>	The address of the specified MC Unit memory area to read from.

<code>mc_area</code>	Data area
<code>MC_TABLE</code> (or value 8)	Table variable array
<code>MC_VR</code> (or value 9)	Global (VR) variable array



When using `HLM_COMMAND`, be sure to set-up the Host Link Master protocol by using the `SETCOM` command.

The Host Link Master commands are required to be executed from one program task only to avoid any multi-task timing problems.

Example 1: The following command will read the CPU Unit model code of the Host Link Slave with node address 12 connected to the RS232C port. The result is written to `VR(233)`.

```
HLM_COMMAND(HLM_MREAD,1,12,MC_VR,233)
```

If the connected Slave is a C200HX PC, then `VR(233)` will contain value 12 (hex) after successful execution.

Example 2: The following command will check the Host Link communication with the Host Link Slave (node 23) connected to the RS422A port.

```
HLM_COMMAND(HLM_TEST,2,23)
```

```
PRINT HLM_STATUS PORT(2)
```

If the `HLM_STATUS` parameter contains value zero, the communication is functional.

Example 3: The following two commands will perform the Host Link `INITIALIZE` and `ABORT` operations on the RS422A port 2. The Slave has node number 4.

```
HLM_COMMAND(HLM_INIT,2)
```

```
HLM_COMMAND(HLM_ABORT,2,4)
```

Example 4: When data has to be written to a PC using Host Link, the CPU Unit can not be in RUN mode. The HLM_COMMAND command can be used to set it to MONITOR mode. The Slave has node address 0 and is connected to the RS232C port.

```
HLM_COMMAND(HLM_STWR,2,0,2)
```

HLM_READ

Type: Remote Command

Syntax: HLM_READ(port,node,pc_area,pc_offset,length,mc_area,mc_offset)

Description: The HLM_READ command reads data from a Host Link Slave by sending a Host Link command string containing the specified node of the Slave to the serial port. The received response data will be written to either VR or Table variables. Each word of data will be transferred to one variable. The maximum data length is 30 words (single frame transfer). Program execution will be paused until the response string has been received or the timeout time has elapsed. The timeout time is specified by using the HLM_TIMEOUT parameter. The status of the transfer can be monitored with the HLM_STATUS parameter.

Parameters:

- port:** The specified serial port. (See specific controller specification for numbers)
- node:** The Slave node number to send the Host Link command to. Range: [0, 31].
- pc_area:** The PC memory selection for the Host Link command.

pc_area	Data area	Hostlink command
PLC_DM		
(or value 0)	DM	RD
PLC_IR		
(or value 1)	CIO/IR	RR
PLC_LR		
(or value 2)	LR	RL
PLC_HR		
(or value 3)	HR	RH
PLC_AR		

(or value 4)	AR	RJ
PLC_EM		
(or value 6)	EM	RE

pc_offset: The address of the specified **PC** memory area to read from. Range: [0, 9999].

length: The number of words of data to be transferred. Range: [1,30].

mc_area: The **MC** Unit's memory selection to write the received data to.

mc_offset: The address of the specified **MC** Unit memory area to write to.

mc_area	Data area
MC_TABLE (or value 8)	Table variable array
MC_VR (or value 9)	Global (VR) variable array



When using the **HLM_READ**, be sure to set-up the Host Link Master protocol by using the **SETCOM** command.

The Host Link Master commands are required to be executed from one program task only to avoid any multi-task timing problems.

HLM_STATUS

Type: Port Parameter.

Description: Returns the status of the Host Link serial communications.

HLM_TIMEOUT

Type: System Parameter

Description: Sets the timeout value for Hostlink communications.

Parameters: **value:** timeout in msec. Default 500msec.

Example: Set the Hostlink timeout to 600msec.

```
HLM_TIMEOUT = 600
```

HLM_WRITE

Type: Remote Command

Syntax: `HLM_WRITE(port,node,pc_area,pc_offset,length,mc_area,mc_offset)`

Description: The `HLM_WRITE` command writes data from the `MC` Unit to a Host Link Slave by sending a Host Link command string containing the specified node of the Slave to the serial port. The received response data will be written from either `VR` or Table variables. Each variable will define on word of data which will be transferred. The maximum data length is 29 words (single frame transfer).

Program execution will be paused until the response string has been received or the timeout time has elapsed. The timeout time is specified by using the `HLM_TIMEOUT` parameter. The status of the transfer can be monitored with the `HLM_STATUS` parameter.

Parameters:

- port:** The specified serial port. (See specific controller specification for numbers)
- node:** The Slave node number to send the Host Link command to. Range: [0, 31].
- pc_area:** The `PC` memory selection for the Host Link command.

pc_area	Data area	Hostlink command
---------	-----------	------------------

PLC_DM		
(or value 0)	DM	RD
PLC_IR		
(or value 1)	CIO/IR	RR
PLC_LR		
(or value 2)	LR	RL
PLC_HR (or value 3)	HR	RH
PLC_AR (or value 4)	AR	RJ
PLC_EM (or value 6)	EM	RE
PLC_REFRESH (or value 7)		

pc_offset: The address of the specified **PC** memory area to write to. Range: [0,9999].

length: The number of words of data to be transferred. Range: [1, 30].

mc_area: The **MC** Unit's memory selection to read the data from.

mc_offset: The address of the specified **MC** Unit memory area to read from.

mc_area	Data area
MC_TABLE (or value 8)	Table variable array
MC_VR (or value 9)	Global (VR) variable array



*When using the **HLM_WRITE**, be sure to set-up the Host Link Master protocol by using the **SETCOM** command.*

The Host Link Master commands are required to be executed from one program task only to avoid any multi-task timing problems.

Example:

The following example shows how to write 25 words from **MC** Unit's **VR** addresses 200-224 to the **PC** **EM** area addresses 50-74. The **PC** has Slave node address 28 and is connected to the RS232C port.

```
HLM_WRITE(1, 28, PLC_EM, 50, 25, MC_VR, 200)
```

HLS_MODEL

Type: System Parameter

Description: Defines the model number returned to a Hostlink Master.

Parameters: **value:** The model number returned. Default 250.

HLS_NODE

Type: System Parameter

Description: Sets the Hostlink node number for the slave node. Used in multidrop RS485 Hostlink networks or set to 0 for RS232 single master/slave link.

HTTP

Type: Reserved Keyword.

INCLUDE

Type: System Command.

Syntax: `INCLUDE "filename"`
(filename - The program to be included).

Description: The **INCLUDE** command resolves all local variable definitions in the included file at compile time and allows all the local variables to be declared “globally”.



Whenever an included program is modified, all program that depend on it are re-compiled as well, avoiding inconsistency.

- (1) Nested **INCLUDE**s are not allowed.
- (2) The **INCLUDE** command must be the first **BASIC** statement in the program.
- (3) Only variable definitions are allowed in the include file. It cannot be used as a general subroutine with any other **BASIC** commands in it.

Parameters: **filename:** The name of the program to be included.

Example: Initialise all local variables with an include program.

```
PROGRAM "T1":
INCLUDE "GLOBAL_DEFS"
FORWARD AXIS(drive_axis)
CONNECT(1, drive_axis) AXIS(link_axis)

PROGRAM "GLOBAL_DEFS":
drive_axis=4
linked_axis=1
```

INDEVICE

Type: Process Parameter

Description: This parameter specifies the default active input device. Specifying an **INDEVICE** for a process allows the channel number for a program to set for all subsequent **GET**, **KEY**, **INPUT** and **LINPUT** statements.



This command is process specific so other processes will use the default channel.

This command is available for backward compatibility, it is currently recommended to use #channel, instead.

Parameters: **value:** The channel number to use for any inputs.
For a full list of communication channels see **#(HASH)**.

Example: Set up a program to use channel 5 by default for any **GET** commands

```
INDEVICE=5
` Get character on channel 5:
IF KEY THEN
```

```
    GET k
ENDIF
```

See Also: #, GET, INPUT, KEY, LINPUT

INITIALISE

Type: System Command.

Description: Sets all axis, system and process parameters to their default values.



The parameters are also reset each time the controller is powered up, or when an EX (software reset) command is performed.



INITIALISE MAY RESET A PARAMETER RELATING TO A DIGITAL DRIVE COMMUNICATION OR ENCODER CAUSING YOU TO LOOSE THE CONNECTION.

Example: When developing you wish to clear all parameters back to default using the command line.

```
>>INITIALISE
>>
```

LAST_AXIS

Type: System Parameter

Description: The *Motion Coordinator* keeps a list of axes that are currently in use. **LAST _ AXIS** is used to read the number of the highest axis in the list.

LAST _ AXIS is set automatically by the system software when an axis is written to; this can include setting **BASE** for the axis.



*Axes higher than **LAST _ AXIS** are not processed. Not all axis lower than **LAST _ AXIS** are processed.*

Parameters: **value:** The highest axis in the axis list that is processed.

Example: Check `LAST _ AXIS` to ensure that the digital network has configured enough drives.

```
IF LAST _ AXIS <> 26 THEN
  PRINT#user, "Digital Drives not initialised"
ENDIF
```

LIST

Type: System Command (command line only)

Syntax: **LIST** ["program"]

Description: Prints the current `SELECTed` program or a specified program to the current output channel



Usually you will view a program by using Motion Perfect.

Parameters: **value:** Prints the selected program.
program: The name of the program to print.

LIST_GLOBAL

Type: System Command (command line only)

Syntax: **LIST _ GLOBAL**

Description: Prints all the `GLOBAL` and `CONSTANTS` to the current output channel.

Example: In an application where the following `GLOBAL` and `CONSTANT` have been set;

```
CONSTANT "cutter", 23
GLOBAL "conveyor",5

>>LIST _ GLOBAL
Global _            VR
```

```
-----  ---  
conveyor 5  
Constant      Value  
-----  ---  
cutter 23.0000  
>>
```

LOAD_PROJECT

Type: System Command

Description: Used by *Motion Perfect* to load projects to the controller.



If you wish to load projects outside of Motion Perfect use the Autoloader ActiveX.

LOADSYSTEM

Type: System Command

Description: Used by *Motion Perfect* to load Firmware to the controller



*If you wish to load firmware without Motion Perfect you can use the SD card (**FILE** command).*

See Also: **FILE**

LOCK

Type: System Command

Syntax: `LOCK(code)`

Description: `LOCK` is designed to prevent programs from being viewed or modified by personnel unaware of the security code. The lock code number is stored in the flash `EPROM`.

When a *Motion Coordinator* is locked, it is not possible to view, edit or save any programs and command line instructions are limited to those required to execute the program. The `CONTROL` value has 1000 added to it when the controller is `LOCKed`.



You should use Motion Perfect to `LOCK` and `UNLOCK` your controller.

To unlock the *Motion Coordinator*, the `UNLOCK` command should be entered using the same lock code number which was used originally to `LOCK` it.

The lock code number may be any integer and is held in encoded form. Once `LOCKed`, the only way to gain full access to the *Motion Coordinator* is to `UNLOCK` it with the correct code. For best security, the lock number should be 7 digits.



IT IS POSSIBLE TO COMPROMISE THE SECURITY OF THE LOCK SYSTEM. USERS MUST CONSIDER IF THE LEVEL OF SECURITY IS SUFFICIENT TO PROTECT THEIR PROGRAMS. IF YOU WANT BETTER SECURITY CONSIDER ENCRYPTING YOUR PROJECT.



If you forget the security code number, the Motion Coordinator may have to be returned to your supplier to be unlocked.

Parameters: **code:** Any 7 digit integer number.

See Also: **UNLOCK**

LOOKUP

Type: Process Command

Syntax: **LOOKUP(format,entry) <PROC(process#)>**

Description: The **LOOKUP** command allows *Motion Perfect* to access the local variables on an executing process.



You should use the variable watch window in Motion Perfect to access the variables on an executing process.

Parameters: **format:** 0: Prints (in binary) floating point value from an expression
 1: Prints (in binary) integer value from an expression
 2: Prints (in binary) local variable from a process
 3: Returns to **BASIC** local variable from a process
 4: Write

entry: Either an expression string (format=0 or 1) or the offset number of the local variable into the processes local variable list.

MOTION_ERROR

Type: System Parameter

Description: The **MOTION_ERROR** provides a simple single indicator that at least one axis is in error and can indicate multiple axes that have an error.

Parameters: **value:** binary sun of the axis number that are in error.
 Bit 0 = axis 0
 Bit 1 = axis 1
 Bit 2 = axis 2

...

Example: `MOTION_ERROR=11 and ERROR_AXIS=3`

indicates axes 0,1 and 3 have an error and the axis 3 occurred first.

MPE

Type: System Command

Syntax: `MPE(mode)`

Description: Sets the type of channel handshaking to be performed on the command line.

Parameters: `channel type:` Any valid TrioBASIC expression

0	No channel handshaking, <code>XON/XOFF</code> controlled by the port. When the current output channel is changed then nothing is sent to the serial port. When there is not enough space to store any more characters in the current input channel then <code>XOFF</code> is sent even though there may be enough space in a different channel buffer to receive more characters
1	Channel handshaking on, <code>XON/XOFF</code> controlled by the port. When the current output channel is changed, the channel change sequence is sent (<code><ESC><channel number></code>). When there is not enough space to store any more characters in the current input channel then <code>XOFF</code> is sent even though there may be enough space in a different channel buffer to receive more characters
2	Channel handshaking on, <code>XON/XOFF</code> controller by the channel. When the current output channel is changed, the channel change sequence is sent (<code><ESC><channel number></code>). When there is not enough space to store any more characters in the current input buffer, then <code>XOFF</code> is sent for this channel (<code><XOFF><channel number></code>) and characters can still be received into a different channel.

Whatever the **MPE** state, if a channel change sequence is received on serial port A then the current input channel will be changed.

- 3 Channel handshaking on, **XON/XOFF** controller by the channel. In **MPE(3)** mode the system transmits and receives using a protected packet protocol using a 16 bit **CRC**.



Whatever the **MPE** state, if a channel change sequence is received on the command line then the current input channel will be changed.

Example: Use the command line to demonstrate mode 0 and 1.

```
>> PRINT #5,"Hello"
Hello
Example2:
MPE(1)
>> PRINT #5,"Hello"
<ESC>5Hello
<ESC>0
>>
```

N_ANA_IN

Type: System Parameter (read only)

Alternate Format: NAI0

Description: This parameter returns the number of analogue input channels available to the *Motion Coordinator*. This includes all built in and external inputs.

Parameters: **value:** The number of analogue inputs.

Example: Check the system configuration in the command line for the correct number of analogue inputs.

```
>>PRINT N _ ANA _ IN
10
>>
```

N_ANA_OUT

Type: System Parameter (Read Only)

Description: This parameter returns the number of analogue output channels available to the controller

Parameters: `value:` The number of analogue outputs.

Example: Use the command line to check that the system has detected the correct number of analogue outputs:

```
>>PRINT N _ ANA _ OUT  
12  
>>
```

NAIO

Type: System Parameter

Description: This parameter returns the number of analogue input channels available to the *Motion Coordinator*. For example an MC464 will return 10 if there is 1 x P325 **CAN** module connected as it has 2 internal analogue inputs and the 8 inputs from the P325.

If no external I/O is fitted, **NAIO** returns the number of Analogue inputs within the *Motion Coordinator*.

NEW

Type: System Command (command line only)

Syntax: `NEW [item]`

Description Deletes a program or table from the controller memory.



When deleting the table all the values are set to 0.



DO NOT DELETE PROGRAMS WHEN CONNECTED TO MOTION PERFECT AS IT WILL CAUSE A CONTROLLER MISMATCH AND YOU WILL BE DISCONNECTED.

Parameters:

- none:** deletes the currently selected program.
- item:**
 - `"TABLE"` = sets all table values to 0
 - `"name"` = deletes a names program
 - `ALL` = deletes all programs



Quotes (") are required when deleting the table or a named program.

Example 1: Delete a named program on the command line.

```
>>NEW "NAMEDPROGRAM"  
OK  
>>
```

Example 2:
Clear all table values to 0.

```
>>NEW "TABLE"  
OK  
>>
```

NIO

Type: System Parameter

Description: This parameter returns the number of inputs/outputs fitted to the system. The value is normally set by the firmware but you can set the value to enable inputs or outputs as part of CANopen I/O startup.



Inputs / Outputs outside of NIO can be used as virtual.

Parameters: **value:** The highest value of input or output that exists.

OUTDEVICE

Type: Process Parameter

Description: The value in this parameter determines the default active output device. Specifying an **OUTDEVICE** for a process allows the channel number to set for all subsequent **GET**, **KEY**, **INPUT** and **LINPUT** statements.



This command is process specific so other processes will use the default channel.

This command is available for backward compatibility, it is currently recommended to use **#channel** instead.

Parameters: **value:** The channel number to use for any inputs.



For a full list of communication channels see #.

Example: Set up a program to print all data to channel 5.

```
OUTDEVICE = 5

IF error THEN
  PRINT "Error Detected"
ENDIF
See Also:
#, GET, INPUT, KEY, LINPUT
```

PEEK

Type: System Function

Syntax: `value = PEEK(address [,mask])`

Description: The **PEEK** command returns value of a memory location of the controller ANDed with an optional mask value.



PEEK IS ONLY NORMALLY USED FOR DE-BUGGING PURPOSES AND SHOULD ONLY BE USED UNDER THE INSTRUCTION OF TRIO MOTION TECHNOLOGY.

Parameters:

- value:** The value returned from the memory location.
- address:** The memory address to read.
- mask:** A value so you can filter particular bits of the address.

PLC_ERROR

Type: System Parameter. (Read Only)

Description: Fetches the current PLC error status word from the IEC 61131 runtime software.

Parameters: A value is returned which has the following meanings.

- 0 Plc No Error
- 1 Plc Load Error
- 2 Plc Start Error
- 3 Plc DC Realtime Error
- 4 Plc DC Prolog Error
- 5 Plc DC Force List Error
- 6 Plc DC Out Of Memory Error
- 7 Plc DC Internal Error

Example: A TrioBASIC program is checking the PLC error state and sets an output to indicate the error.

```
IF PLC _ ERROR <> 0 THEN
  OP(error _ op, ON)
ENDIF
In the terminal, print the current PLC error status value.
>>?PLC _ ERROR
2.0000
>>
```

PLC_READ

Type: System Command.

Syntax: `value = PLC_READ("IEC_path")`

Description: This command allows the TrioBASIC to access IEC project variables. The supported IEC datatypes must be elementary and can be summarised as follows:

BOOL, SINT, INT, DINT, USINT, UINT, UDINT, REAL, LREAL, TIME, BYTE, WORD, DWORD, LINT.



The PLC_READ command can be used within a SCOPE command request allowing IEC program data to be captured together with other SCOPE data sources.

Parameters: **value:** The value returned from the IEC variable.

IEC_path: This is the variable path as a string in quotes.



*The IEC_path is made up of three parts, each separated by points (periods). For global variables use:
"@GV.variable_name"*

*For program instance variables use the following where the functionblock_instance_name is optional:
"Program_instance_name.[functionblock_instance_name].variable_name"*

You do not specify the task name only the program instance name, MultiProg will not allow you to create the same program instance name in 2 different tasks hence each program instance name is guaranteed to be unique.

Variable name checking is case sensitive so names must match exactly. If the variable cannot be retrieved because the name is invalid or it is not of an elementary datatype then a runtime error will be generated.

Example 1: Read a variable from task1 in the PLC.

```
local _ variable = PLC _ READ("task1.var1")
```

Example 2: In the terminal, print the current value of a function block output. This example uses the optional parameter.

```
>>?PLC _ READ("task2.user _ function1.output")
1500.0000
>>
```

Example 3: Read a Global PLC variable.

```
plc _ global1 = PLC _ READ("@GV.globvar1")
```

Example 4: Fetch a VR value from the PLC. This example shows that individual array elements can be accessed if they themselves are elementary.

```
my _ vr200 _ copy = PLC _ READ("@GV.TC _ VR[200]")
```

PLC_STATUS

Type: System Parameter. (Read Only)

Description: Fetches the current PLC status word from the IEC 61131 runtime software.

Parameters: A value is returned which has the following meanings.

0	Plc On
1	Plc Loading
2	Plc Starting
3	Plc Running
4	Plc Halt Requested
5	Plc Halt
6	Plc Stopping
7	Plc Stop
8	Plc Resetting

Example: A TrioBASIC program is monitoring the **PLC** state and only continues after the **PLC** starts running.

```
WAIT UNTIL PLC _ STATUS = 3
```

In the terminal, print the current **PLC** status value.

```
>>?PLC _ STATUS
5.0000
>>
```

PMOVE

Type: Process Parameter (Read Only)

Description: Returns the state of the process move buffer.

When one of the processes encounters a movement command the process loads the movement requirements into its “process move buffer”. This can hold one movement instruction for any group of axes. When the load into the process move buffer is complete the **PMOVE** parameter is set to 1. When the next servo period occurs the motion generation program will load the movement into the “next move buffer” of the required axes if these are available. When this second transfer is complete the **PMOVE** parameter is cleared to 0.



*Each process has its own **PMOVE** parameter.*

Parameters: **value:** 1 if the process move buffer is occupied
0 if the process move buffer is empty

PROC

Type: Process Modifier

Description: Allows a process parameter from a particular process to be read or set.

Example: `WAIT UNTIL PMOVE PROC(14)=0`

PROC_LINE

Type: Process Parameter (Read Only)

Description: Allows the current line number of another executing program to be obtained.

Example: Find out which line is being executed on the program running in process 2.

```
>>PRINT PROC _ LINE PROC(2)
12
>>
```

PROC_STATUS

Type: Process Parameter (Read Only)

Description: Returns the status of another process, referenced with the `PROC(x)` modifier.

Returns:

0	Process Stopped
1	Process Running
2	Process Stepping
3	Process Paused
4	Process Pausing
5	Process Stopping

Example: Run a program in process 12, check for it to start and then for it to complete.

```
RUN "progname",12
WAIT UNTIL PROC _ STATUS PROC(12)<>0 ` wait for program to start
WAIT UNTIL PROC _ STATUS PROC(12)=0
` Program "progname" has now finished.
```

PROCNUMBER

Type: System Parameter

Description: Returns the process on which a TrioBASIC program is running. This is normally required when multiple copies of a program are running on different processes.

Parameters: `value:` The process number the current program is running on.

Example: Running the same program on processes 0 to 3 to use axes 0-3, `PROCNUMBER` is used to specify which axis the program is using.

```
MOVE(length) AXIS(PROCNUMBER)
```

RESET

Type: Process Command

Description: Sets the value of all the local named variables of a TrioBASIC process to 0.

RUN_ERROR

Type: Process Parameter

Description: Contains the number of the last run time error that stopped the program on the specified process.



`RUN_ERROR = 31` is a normal completion of a program.

Parameters: Please see Error Codes in the appendix for full value listings.

Example: Use the command line to check why a program that was running on process 5 has stopped. The result of 9 indicates a divide by zero error.

```
>>? RUN_ERROR PROC(5)
9.0000
>>
```

POKE

Type: System Command

Syntax: `POKE(address,value)`

Description: The `POKE` command allows a value to be entered into a memory location of the controller.



THE POKE COMMAND CAN PREVENT NORMAL OPERATION OF THE CONTROLLER AND SHOULD ONLY BE USED IF INSTRUCTED BY TRIO MOTION TECHNOLOGY.

Parameters: **address:** The memory address to read.

mask: A value so you can filter particular bits of the address.

PORT

Type:	Modifier
Syntax:	<code>PORT(expression)</code>
Description:	Assigns ONE command, function or port parameter operation to a particular communication PORT .
Parameters:	<p>Expression: Any valid TrioBASIC expression. The result of the expression should be a valid integer PORT number.</p> <ul style="list-style-type: none">0 = Command line1 = RS232 Serial port2 = RS485 Serial port5 = User terminal6 = User terminal7 = User terminal8 = User terminal9 = <i>Motion</i> Perfect channel10-49 = Reserved50 = 1st Anybus module51 = 2nd Anybus module52 = 3rd Anybus module53 = 4th Anybus module54 = 5th Anybus module55 = 6th Anybus module56 = 7th Anybus module

POWER_UP

Type: Reserved keyword.

PRMBLK

Type: Reserved Keyword.

PROCESS

Type: System Command (Command line only)

Description: Displays information about the running processes.



There are some housekeeping process that you cannot stop.

Parameters:

- value:** The process number.
- Type:** The Type of process executing.
- Status:** The execution state of the process.
- Program:** The name of the program running in the process.
- Line:** The line number of a program that is executing.
- Time:** The length of time that the process has been running.
- CPU:** The percentage of CPU time used by the process.

Example: Check the state of the processes in the command line.

```
>>process
Process  Type  Status  Program  Line  hhhh:mm:ss.ms  [CPU%]
-----
21      Fast  Sleep[0]  TEST      1      0000:00:02.634  [ 0.23%]
22      SYS   Run      Command Line  0001:14:05.570  [ 0.16%]
23      SYS   Run      IO Server  0001:14:01.183  [90.46%]
24      SYS   Sleep[8]  MPE       0001:14:05.571  [ 0.00%]
25      SYS   Sleep[6]  CAN Server 0001:14:05.571  [ 0.00%]
KERNEL  SYS   Run      Motion/Housekeeping 0001:14:05.571  [ 9.16%]
```

PROJECT_KEY

TYPE: Reserved Keyword

PROTOCOL

Type: Port Parameter

Description: This parameter allows the user to check which protocol is running on the specified **PORT**.



*You can write to this parameter however it is advisable to initialise the communication protocol through **SETCOM**, **ANTBUS** etc.*



DO NOT WRITE A VALUE TO **PORT(0) AS YOU WILL DISABLE COMMUNICATIONS WITH **MOTION PERFECT**.**

Parameters:

value:	0 = None
	1 = Download
	2 = MPE
	3 = MODBUS
	4 = Transparent
	5 = HostLink

Example: Check that Modbus is running on the RS485 channel (**PORT(2)**).

```
IF PROTOCOL PORT(2) <>3 THEN
  PRINT#user, "MODBUS has stopped"
ENDIF
```

See Also: **ANYBUS**, **SETCOM**

READPACKET

Type: Command

Syntax: READPACKET(port, variable, count [,format])

Description: READPACKET is used to read in data to the VR variables over a serial communications port. The data is transmitted from the PC in binary format with a CRC 16bit checksum. There are four different data formats, all use the same packet structure:

Data						CRC
Byte 0	Byte 1	Byte 2	...	Byte n	Byte 0	Byte 1



The 16bit checksum uses the generator polynomial $x^{16}+x^{15}+x^2+x^0$ or \$8005

Parameters:

- port:** This value should be 0 to 2.
- variable:** This value tells the *Motion Coordinator* where to start setting the variables in the VR() global memory array.
- VR count:** The number of variables to download, maximum 250.
- format:** The number format for the numbers being downloaded
 - 0 = Standard character
 - 1 = Standard integer
 - 2 = Standard long
 - 4 = 7bit long

Depending on the format used the data may be split over multiple bytes. It is up to the user to recombine these to get the final value

Format = 0 (standard character)

Each value is in each Byte:

Value0 = Byte 0

Value1 = Byte 1

...

Format = 1 (standard integer)

Each value is split over 2Bytes

Value0 = Byte1 * 256 + Byte0

Value1 = Byte3 * 256 + Byte2

...

Format = 2 (standard long)

Each value is split over 4Bytes

Value0 = ((Byte3 * 256 + Byte2) * 256 + Byte1) * 256 + Byte0

Value1 = ((Byte7 * 256 + Byte6) * 256 + Byte5) * 256 + Byte4

...

Format = 4 (7bit long)

Each value is split over 4Bytes, but only uses 7 bits of each byte. Only Byte 0 (including the CRC) has bit 7 set. The values sent are therefore 24bits in length.

Bits 15 and Bits 7 of the CRC are not sent and so ignored by the check.

Value0 = ((Byte3 * 128 + Byte2) * 128 + Byte1) * 128 + Byte0

Value1 = ((Byte7 * 128 + Byte6) * 128 + Byte5) * 128 + Byte4

...

Example:

Using Standard Long (format = 2) read in the values to a sequence of VR's starting at 0 from port 1. The bytes from the READPACKET command are stored in VR(100) and onwards.

```

READPACKET(1, 100, 10, 2)
FOR val = 0 to 9
  `Off set the bytes
  VR(val*4+103) = VR(val*4+103) * (2^32)
  VR(val*4+102) = VR(val*4+103) * (2^16)
  VR(val*4+101) = VR(val*4+103) * (2^8)
VR(val)=(val*4+103)+VR(val*4+102))+VR(val*4+101))+VR(val*4+100)
NEXT val

```

REMOTE

Type: System Command

Syntax: `REMOTE(slot)`

Description: Starts up the `REMOTE` communication protocol as a program which communicates with `PCMotion` ActiveX. The `REMOTE` program will take up a user process if it is run automatically or manually. It is recommended that `REMOTE` should run on a high priority process.



The `REMOTE` program is normally started automatically when you open a `PCMotion` connection. You can call it manually if you wish to specify which process it should run on.



IF YOU EXECUTE `REMOTE` MANUALLY THE PROGRAM IT RUNS IN WILL SUSPEND AT THE `REMOTE` LINE. THE `REMOTE` THEREFORE SHOULD BE THE LAST LINE OF THE PROGRAM TO EXECUTE.

Example: A program that will start the `REMOTE` program on process 20 if the project wants to run in debug mode.

```
WHILE(1)
  IF VR(debug)=TRUE THEN
    REMOTE(0)
  ELSE
    WA(100)
  ENDIF
WEND
```

REMOTE_PROC

Type: System Parameter

Description: When the TrioPC ActiveX opens a synchronous connection to the *Motion Coordinator*, the `REMOTE_PROGRAM` is started on the highest available process. Normally this can be process 21. `REMOTE_PROC` can be set so as to specify a different process for the `REMOTE_PROGRAM`. For example if `REMOTE_PROC=p`, the `REMOTE_PROGRAM` will try run on process p if it is available. If process p is in use then the next lower available process will be used.

Example1: Set `remote_program` to start on process 19 or lower (using the command line terminal).

```
>>REMOTE _ PROC=19
>>
```

Example2: Remove the `remote_proc` setting so that `remote_program` starts on default process (using the command line terminal).

```
>>REMOTE _ PROC=-1
>>
```

Example3: In the initialisation program of the project.

```
IF REMOTE _ PROC <> 19 THEN
  REMOTE _ PROC=19
PRINT "Setting remote program startup, please cycle power to
continue "

  STOP
ENDIF
```





`REMOTE _ PROC` is stored in Flash EPROM to be used on all subsequent power-ups or software resets.

RENAME

Type:	System Command
Syntax:	<code>RENAME oldname newname</code>
Description:	Renames a program in the <i>Motion Coordinator</i> directory. It is not normally used except by <i>Motion Perfect</i> .
Parameters:	oldname: The name of the program to rename. newname: The new name of the program.
Example:	<pre>>>RENAME car voiture OK >></pre>

RUN

Type:	System Command
Syntax:	<code>RUN ["program" [, process]]</code>
Description:	Runs a named program on the controller. Programs can be RUN from another program.  <i>A program can be run multiple times in different processes. You can use PROCNUMBER to help assign values in the program.</i>  <i>Programs will continue to execute until there are no more lines to execute, a HALT is typed in the command line, a STOP is issued or there is a run time error.</i>
Parameters:	none: Runs the currently SELECTED programs. program: Name of program to be run. process: Optional process number. (default highest available).

Example 1: **SELECT** the program **STARTUP** and run it on he command line.

```
>>SELECT "STARTUP"
STARTUP selected
>>RUN%[Process 21:Program STARTUP] - Running
>>%[Process 21:Line 238] (31) - Program is stopped
>>
```

Example 2: From the **MAIN** program run the **STARTUP** program on process 2 and wait for its completion.

```
RUN "STARTUP", 2
WAIT UNTIL PROC _ STATUS PROC(2) <> 0  `wait for program to start
WAIT UNTIL PROC _ STATUS PROC(2) = 0 `wait for program to complete
WDOG=ON
```

Example 3: After **STARTUP** has completed the **MAIN** program will start other programs running in the highest available processes.

```
RUN "IO _ CONTROL"
RUN "HMI"
RUN "SAUSAGE _ CHOPPER"
```

See Also: **HALT**, **PROCNUMBER**, **RUN _ ERROR**, **SELECT**, **STOP**

RUNTYPE

Type: System Command

Syntax: **RUNTYPE** "program", mode [,process]

Description: Sets if program is run automatically at power up, and which process it is to run on.



*Usually a programs **RUNTYPE** is set through Motion Perfect. It can be useful to set the **RUNTYPE** when loading programs from a SD card.*



FOR ANY PROGRAM TO RUN AUTOMATICALLY ON POWER-UP ALL THE PROGRAMS ON THE CONTROLLER MUST COMPILE WITHOUT ERRORS. EVEN IF THEY ARE NOT USED.



The current status of each program's **RUNTYPE** is displayed when a **DIR** command is performed.

Parameters:

- program:** The program to set the power up mode.
- mode:**
 - 1 = Run automatically on power up.
 - 0 = Manual running.
- process:** The process number to run the program on.

Example: When loading a sequence of programs from a SD card, **MAIN** must be set to run from power up and **HMI** must be run on process 4 on power up. The following is from the **TRIOINIT.bas** file.

```
FILE "LOAD _ PROGRAM" "MOTION"
FILE "LOAD _ PROGRAM" "HMI"
FILE "LOAD _ PROGRAM" "MAIN"
RUNTYPE "HMI", 1, 4
RUNTYPE "MAIN", 1
AUTORUN
```

SCHEDULE_TYPE

Type: System Parameter

Description: This parameter disables the scheduling algorithm that allows another program to run while the scheduled program is in a sleep state. A sleep state can be started through a pause in the program for example: **WAIT** or **WA**. After the next power up, the new process scheduling will take effect. The value is saved in Flash memory.



This parameter should only be used when upgrading projects from older controllers and the scheduling system causes problems with the program timings.

Parameters:

- value:**
 - 0 = Use new scheduling algorithm to make best use of CPU time eg any program executing a **WA** command will not be available for execution again until the **WA** period is complete (default).
 - 1 = Revert to old style scheduling such that any active process will execute even when executing a **WA** command for example.

SCOPE

Type: System Command

Syntax: `SCOPE(enable, [period, table _ start, table _ stop, p0 [,p1[,p2 [,p3]]]])`

Description: The **SCOPE** command enables capture of up to 4 parameters every sample period. Samples are taken until the table range is filled. **TRIGGER** is used to start the capture.



*The **SCOPE** facility is a “one-shot” and needs to be re-started by the **TRIGGER** command each time an update of the samples is required.*



MAKE SURE TO ASSIGN THE TABLE RANGE OUTSIDE OF ANY TABLE DATA USED BY YOUR PROGRAMS.



*It is normal to use Motion Perfect to assign the **SCOPE** command, but it is sometimes useful to do it manually. The table data can be read back to a PC and displayed on the Motion Perfect Oscilloscope, saved using Motion Perfect or **STICK _ WRITE**.*

Parameters:

enable:	1 or ON = Enable software SCOPE (requires at least 5 parameters). 0 or OFF = Disable SCOPE .
Period:	The number of servo periods between data samples.
table _ start:	Position to start to store the data in the table array.
table _ stop:	End of table range to use.
P2	third parameter to store.
P3	fourth parameter to store.

Example 1: This example arms the **SCOPE** to store the **MPOS** and **DPOS** on axis 5 axis 5 every 10 milliseconds (**SERVO _ PERIOD** = 1000). The **MPOS** will be stored in table values 0..499, the **DPOS** in table values 500 to 999. The sampling does not start until the **TRIGGER** command is executed.

```
SCOPE(ON,10,0,1000,MPOS AXIS(5), DPOS AXIS(5))
```


Example 2: Disable the **SCOPE** to prevent **TRIGGER** from starting a capture.
SCOPE(OFF)

See Also: **TRIGGER**

SCOPE_POS

Type: System Parameter (Read Only)

Description: Returns the current **TABLE** index position where the **SCOPE** function is currently storing its parameters.

Parameters: **value:** The table position that is currently being used.

SELECT

Type: System Command

Syntax: **SELECT "program"**

Description: Makes the named program the currently selected program, if the named program does not exist then it makes a program of that name.



*It is not normally used except by Motion Perfect.
The **SELECT**ed program cannot be changed when programs are running.*

When a program is selected any previously selected program is compiled.

SERCOS

Type: System Function

Syntax: `SERCOS(function#,slot,{parameters})`

Description: This function allows the **SERCOS** ring to be controlled from the TrioBASIC programming system. A **SERCOS** ring consists of a single master and 1 or more slaves daisy-chained together using fibre-optic cable. During initialisation the ring passes through several 'communication phases' before entering the final cyclic deterministic phase in which motion control is possible. In the final phase, the master transmits control information and the slaves transmit status feedback information every cycle time.

Once the **SERCOS** ring is running in CP4, the standard TrioBASIC motion commands can be used.

The *Motion Coordinator* **SERCOS** hardware uses the Sercon 816 **SERCOS** interface chip which allows connection speeds up to 16Mhz. This chip can be programmed at a register level using the **SERCOS** command if necessary. To program in this way it is necessary to obtain a copy of the chip data sheet.

The **SERCOS** command provides access to 10 separate functions:

Function:	0	Read SERCOS Asic:
	1	Write SERCOS Asic:
	2	Initialise command:
	3	Link SERCOS drive to Axis
	4	Read parameter.
	5	Write parameters
	6	Run SERCOS procedure command.
	7	Check for dirve present
	8	Print network parameter
	9	Reserved
	10	SERCOS ring status

Slot: The slot number is in the range 0 to 6 and specifies the master module location.

Parameters:

Function 0	SERCOS(0, slot, ram/reg, address)
Slot	The module slot in which the SERCOS is fitted.
ram/reg	0 = read value from RAM 1 = read value from register.
address	The index address in RAM or register.

Example: >>SERCOS(0, 0, 1, \$0c)

Parameters:

Function 1	SERCOS(1, slot, ram/reg, address, value)
Slot	The module slot in which the SERCOS is fitted.
ram/reg	0 = write value to RAM 1 = write value to register.
address	The index address in RAM or register.
value	Date to be written

Example: Do not use this function without referencing the Sercon 816 data sheet.

Parameters:

Function 2	SERCOS(2, slot [,intensity [,baudrate [, period]])
Slot	The module slot in which the SERCOS is fitted.
intensity	Light transmission intensity (1 to 6). Default value is 3.
baudrate	Communication data rate. Set to 2, 4, 6, 8 or 16.
period	Sercos cycle time in microseconds. Accepted values are 2000, 1000, 500 and 250usec.

Example: >>SERCOS(2, 3, 4, 16, 500)

Parameters:	Function 3	SERCOS(3, slot, slave addr, axis [, slave drive type])
	slot	The module slot in which the SERCOS is fitted.
	slave addr	Slave address of drive to be linked to an axis.
	axis	Axis number which will be used to control this drive.
	slave drive type	Optional parameter to set the slave drive type. All standard SERCOS drives require the GENERIC setting. The other options below are only required when the drive is using non-standard SERCOS functions.
		0 Generic Drive
		1 Sanyo-Denki
		3 Yaskawa + Trio P730
		4 PacSci
		5 Kollmorgen

Example: >>SERCOS(3, 1, 3, 5, 0) `links drive at address 3 to axis 5

Parameters:	Function 4	SERCOS(4, slot, slave address, parameter ID [, parameter size[, element type [, list length offset, [VR start index]])]
	slot	The module slot in which the SERCOS is fitted.
	slave addr	SERCOS address of drive to be read.
	parameter ID	SERCOS parameter IDN
	parameter size	Size of parameter data expected:
		2 = 2 byte parameter (default).
		4 = 4 byte parameter
		6 = list of parameter IDs
		7 = ASCII string
	element type	SERCOS element type in the data block:
		1 ID number
		2 Name
		3 Attribute
		4 Units
		5 Minimum Input value

	6 Maximum Input value
	7 Operational data (default)
List length offset	Optional parameter to offset the list length. For drives that return 2 extra bytes, use -2.
VR start index	Beginning of VR array where list will be stored.



This function returns the value of 2 and 4 byte parameters but prints lists to the terminal in Motion Perfect unless VR start index is defined.

Example:

```
>>SERCOS(4, 0, 5, 140, 7) `request "controller type"
>>SERCOS(4, 0, 5, 129) `request manufacturer class 1
diagnostic
```

Parameters:	Function 5	SERCOS(5, slot , slave address, parameter ID, parameter size, parameter value [, parameter value ...])
	Slot	The module slot in which the SERCOS is fitted.
	slave addr	SERCOS address of drive to be written.
	parameter ID	SERCOS parameter IDN
	parameter size	Size of parameter data to be written. 2, 4, or 6.
	parameter value	Enter one parameter for size 2 and size 4. Enter 2 to 7 parameters for size 6 (list).

Example:

```
>>SERCOS(5, 1, 7, 2, 2, 1000) `set SERCOS cycle time
>>SERCOS(5, 0, 2, 16, 6, 51, 130) `set IDN 16 position feedback
```

Parameters:	Function 6	SERCOS(6, slot , slave address, parameter ID [,time-out,[command type]])
	Slot	The communication slot in which the SERCOS is fitted.
	slave addr	SERCOS address of drive.
	parameter ID	SERCOS procedure command IDN.

<code>time out</code>	Optional time out setting (msec).
<code>command type</code>	Optional parameter to define the operation: -1 Run & cancel operation (default value) 0 Cancel command 1 Run command

Example: `>>SERCOS(6, 0, 2, 99) 'clear drive errors`

Parameters:	<code>Function 7</code>	<code>SERCOS(7 , slot , slave address)</code>
	<code>slot</code>	The module slot in which the <code>SERCOS</code> is fitted.
	<code>slave addr</code>	<code>SERCOS</code> address of drive. Returns 1 if drive detected, -1 if not detected.

Example: `IF SERCOS(7, 2, 3) <0 THEN`
`PRINT#5, "Drive 3 on slot 2 not detected"`
`END IF`

Parameters:	<code>Function 8</code>	<code>SERCOS(8 , slot , required parameter)</code>
	<code>slot</code>	The module slot in which the <code>SERCOS</code> is fitted.
	<code>required parameter</code>	This function will print the required network parameter, where the possible 'required parameter' values are: 0: to print a semi-colon delimited list of 'slave Id, axis number' pairs for the registered network configuration (as defined using function 3). Used in Phase 1: Returns 1 if drive is detected, 0 if no drive detected. 1: to print the baud rate (either 2, 4, 6, or 8), and 2: to print the intensity (a number between 0 and 6).

Example: >>?SERCOS(8,0, 1)

Parameters: **Function 10** **SERCOS(10,<slot>)**
Slot The module slot in which the **SERCOS** is fitted.
 This function checks whether the fibre optic loop is closed in phase 0. Return value is 1 if network is closed, -1 if it is open, and -2 if there is excessive distortion on the network.

Example: >>?SERCOS(10, 1)
 IF SERCOS (10, 0) <> 1 THEN
 PRINT "SERCOS ring is open or distorted"
 END IF



Motion Perfect contains support for commissioning SERCOS rings. This tool simplifies the creation of a TrioBASIC startup program which consists of SERCOS statements to initialise the ring following power-on, and configure the ring in the deterministic cyclic phase.

SERCOS_PHASE

Type: Slot Parameter

Description: Sets the phase for the **SERCOS** ring in the specified slot.

Parameters: **value:** The **SERCOS** phase, range 0-4

Example 1: Set the sercos ring attached to daughter board in slot 0 to phase 3
SERCOS _ PHASE SLOT(0) = 3

Example 2: If the **SERCOS** phase is 4 in slot 2 then turn on the output.
IF SERCOS _ PHASE SLOT(2)<>4 THEN

```

OP(8,ON)
ELSE
    OP(8,OFF)
ENDIF

```

SERIAL_NUMBER

- Type:** System Parameter (Read only)
- Syntax:** SERIAL _ NUMBER
- Description:** Returns the unique Serial Number of the controller.
- Example:** For a controller with serial number 00325:
- ```

>>PRINT SERIAL _ NUMBER
325.0000
>>

```

---

## SERVO\_PERIOD

---

- Type:** System Parameter
- Description:** This parameter allows the controller servo period to be specified. **SERVO \_ PERIOD** is specified in microseconds. Only the values 2000, 1000, 500, 250 or 125 usec may be used and the *Motion Coordinator* must be reset before the new servo period will be applied. The value is saved in Flash memory.
- Example:**
- ```

` check controller servo _ period on startup
IF SERVO _ PERIOD<>250 THEN
    SERVO _ PERIOD=250
EX
ENDIF

```



Axis count will be limited as the SERVO _ PERIOD is reduced. Normally the headline number of axes can be used when SERVO _ PERIOD is set to 1msec.

SLOT

Type: Slot Modifier

Syntax: `SLOT(position)`

Description: Assigns **ONE** command, function or slot parameter operation to a particular slot

Parameters: `position:` 1 = Built in feature
0 to max slot= Expansion module

Example 1: Check for an Anybus cc module in the holder in slot 1.

```
IF COMMSTYPE SLOT(1) = 62 THEN
    PRINT "No Anybus card present"
ENDIF
```

STEP

Type: Program Structure

Description: This optional parameter specifies a step size in a **FOR..NEXT** sequence. See **FOR**.

Example:

```
FOR x=10 TO 100 STEP 10
    MOVEABS(x) AXIS(9)
NEXT x
```

STEPLINE

Type: System Command

Syntax: STEPLINE {Program name}{[,Process number]}

Description: Steps one line in a program. This command is used by *Motion Perfect* to control program stepping. It can also be entered directly from the command line or as a line in a program with the following parameters.



All copies of this named program will step unless the process number is also specified.

If the program is not running it will step to the first executable line on either the specified process or the next available process if the next parameter is omitted.

*If the program name is not supplied, either the **SELECTed** program will step (if command line entry) or the program with the **STEPLINE** in it will stop running and begin stepping.*

Parameters: **Program:** This specifies the program to be stepped.

Process: This specifies the process number.

Example: Start the program conveyor running in the highest available process by stepping into the first executable line.

```
>>STEPLINE "conveyor"
OK
%[Process 21:Line 19] - Paused
>>
```

STICK_READ

Type: System Function

Syntax: value = STICK_READ(flash_file, table_start [, format])

Description: Read table data from the SD card to the controller.



ANY EXISTING TABLE DATA WILL BE OVERWRITTEN

Parameters:	value:	TRUE = the function was successful. FALSE = the function was not successful.
	flash File:	A number which when appended to the characters “SD” will form the data filename.
	table _ start:	The start point in the TABLE where the data values will be transferred to.
	format:	0 = Binary 64bit floating point format (default). 1 = ASCII comma separated values



The binary file is stored in IEEE floating point binary format little-endian, i.e. the least significant byte first.

Example: Read the ASCII file SD1984.csv from the SD card and copies the ‘data to the table starting at TABLE(16500).

```
STICK _ READ (1984, 16500, 1)
```

See Also: STICK _ READVR

STICK_READVR

Type: System Function

Syntax: value = STICK_READVR(flash_file, vr_start [, format])

Description: Read VR data from the SD card to the controller.



ANY EXISTING VR DATA WILL BE OVERWRITTEN.

Parameters:	value:	TRUE = the function was successful. FALSE = the function was not successful
	flash_file:	A number which when appended to the characters “SD” will form the data filename.

vr _ start: The start point in the VRs where the data values will be transferred to.

format: 0 = Binary 64bit floating point format (default).
1 = ASCII comma separated values



The binary file is stored in IEEE floating point binary format little-endian, i.e. the least significant byte first.

Example: ‘Read the ASCII file SD1984.csv from the SD card and copies the ‘data to the VRs starting at VR(16500)

```
STICK _ READVR (1984, 16500, 1)
```

See Also: STICK _ READ

STICK_WRITE

Type: System Function

Syntax: `value = STICK _ WRITE(flash _ file, table _ start [,length [,format]])`

Description: Used to store table data to the SD card in one of two formats



IF THIS FILE ALREADY EXISTS, IT IS OVERWRITTEN.



If you want to store the data without losing any precision use the Binary format.

Parameters:

value: **TRUE** = the function was successful.
FALSE = the function was not successful.

flash _ file: A number which when appended to the characters “SD” will form the data filename.

table _ start: The start point in the **TABLE** where the data values will be transferred from.

length: The number of the table values to be transferred (default 128 values.)

format: 0 = Binary 64bit floating point format, **BIN** file (default).
1 = **ASCII** comma separated values, **CSV** file



*When storing in `format=0` the data is stored in **IEEE** floating point binary format little-endian, i.e. the least significant byte first.*

Example: Transfer 2000 values starting at **TABLE**(1000) to the **SD** Card file ‘called **SD1501.BIN**
`success = STICK _ WRITE (1501, 1000, 2000, 0)`

See Also: **STICK _ WRITEVR**

STICK_WRITEVR

Type: System Function

Syntax: `value = STICK _ WRITEVR(flash _ file, vr _ start [,length [,format]])`

Description: Used to store **VR** data to the **SD** card in one of two formats.



IF THIS FILE ALREADY EXISTS, IT IS OVERWRITTEN.



If you want to store the data without losing any precision use the Binary format.

Parameters:

value:	TRUE = the function was successful. FALSE = the function was not successful.
flash _ file:	A number which when appended to the characters “ SD ” will form the data filename.
vr _ start:	The start point in the TABLE where the data values will be transferred from.
length:	The number of the table values to be transferred (default 128 values.)
format:	0 = Binary 64bit floating point format, BIN file (default).

1 = ASCII comma separated values, CSV file.



When storing in format=0 the data is stored in IEEE floating point binary format little-endian, i.e. the least significant byte first.

Example: Transfer 2000 values starting at VR(1000) to the SD Card file 'called SD1501.BIN
 success = STICK _ WRITEVR (1501, 1000, 2000, 0)

See Also: STICK _ WRITE

STOP

Type: Command

Syntax: STOP "progname",[process _ number]

Description: Stops one program at its current line. A particular program name may be specified and an optional process number. The process number is required if there is more than one instance of the program running. If no name or process number is included then the selected program will be assumed.

Parameters: **progname:** name of program to be stopped.
process _ number: optional process number to be used when multiple instances of the program are running and only one is to be stopped.

Example 1: Stop a program called "axis_init" from the command line. Note that quotes are optional unless the program name is also a BASIC keyword.

```
>>STOP axis _ init
```

Example 2: Stop the named programs when a digital input goes off.

```
IF IN(12)=OFF THEN
  STOP "hmi _ handler"
  STOP "motion1"
ENDIF
```

Example 3: Stop one instance of a named program and leave the other instances running.

```
proc_a = VR(45)
` process to be stopped is put in the VR by an HMI
STOP "test_program",proc_a
` stop the required instance of test_program
```

See also: SELECT, RUN

STORE

Type: System Command

Description: Used by *Motion Perfect* to load Firmware to the controller.



REMOVING THE CONTROLLER POWER DURING A STORE SEQUENCE CAN LEAD TO THE CONTROLLER HAVING TO BE RETURNED TO TRIO FOR RE-INITIALIZATION.

SYSTEM_VARIABLE

Type: Reserved Keyword

SYSTEM_ERROR

Type: System Parameter

Description: The system errors are in blocks based on the following byte masks:

System errors	0x0000ff
Configuration errors	0x00ff00

Unit errors 0xff0000

The following are system errors:

Ram error 0x000001
 Battery error Battery error
 Invalid module error 0x000004

The following are configuration errors:

Unit error 0x000100
 Station error 0x000200

The following are Unit errors:

Unit Lost 0x010000
 Unit Terminator Lost 0x020000
 Unit Station Lost 0x040000
 Invalid Unit error 0x080000
 Unit Station Error 0x100000

TABLE

Type: System Command

Syntax: **TABLE**(address [, data0..data35])

Description: The **TABLE** command can be used to load and read back the internal **TABLE** values. As the table can be written to and read from, it may be used to hold information as an alternative to variables.

The table values are floating point and can therefore be fractional.



*You can clear the **TABLE** using **NEW "TABLE"**.*

Parameters:

- value:** returns the value stored at the address or -1 if used as part of a write.
- address:** The address of the first point of a write, or the address to read.
- data0:** The data written to the address.
- data1:** The data written to the address +1.
- data2:** The data written to the address +2.
- data32:** The data written to the address +35.

Example 1: This loads the **TABLE** with the following values, starting at address 100:

Table Entry:	Value:
100	0
101	120
102	250
103	370
104	470
105	530

TABLE(100,0,120,250,370,470,530)

Example 2: Use the command line to read the value stored in address 1000.

```
>>PRINT TABLE(1000)
1234.0000
>>
```

See also: **FLASHVR, NEW, TSIZE**

TABLE_POINTER

Type: Axis Parameter (Read Only)

Description: Using the `TABLE _ POINTER` command it is possible to determine which `TABLE` memory location is currently being used by the `CAM` or `CAMBOX`.

`TABLE _ POINTER` returns the current table location that the `CAM` function is using. The returned number contains the table location and divides up the interpolated distance between the current and next `TABLE` location to indicate exact location.



The user can load new CAM data into previously processed TABLE location ready for the next CAM cycle. This is ideal for allowing a technician to finely tune a complex process, or changing recipes on the fly whilst running.

Parameters: `value:` The value is returned of type X.Y where X is the current `TABLE` location and Y represents the interpolated distance between the start and end location of the current `TABLE` location.

Example: In this example a `CAM` profile is loaded into `TABLE` location 1000 and is setup on axis 0 and is linked to a master axis 1. A copy of the `CAM` table is added at location 100. The Analogue input is then read and the `CAM TABLE` value is updated when the table pointer is on the next value.

```

` CAM Pointer demo
` store the live table points
TABLE(1000,0,0.8808,6.5485,19.5501,39.001,60.999,80.4499,93.4515)
TABLE(1008,99.1192,100)
` Store another copy of original points
TABLE(100,0,0.8808,6.5485,19.5501,39.001,60.999,80.4499,93.4515)
TABLE(108,99.1192,100)
` Initialise axes
BASE(0)
WDOG=ON
SERVO=ON

` Set up CAM
CAMBOX(1000,1009,10,100,1, 4, 0)

` Start Master axis
BASE(1)
SERVO=ON
SPEED=10
FORWARD

` Read Analog input and scale CAM based on input
pointer=0
WHILE 1
  ` Read Analog Input (Answer 0-10)

```

```

scale=AIN(32)*0.01
` Detects change in table pointer
IF INT(TABLE_POINTER)<>pointer THEN
  pointer=INT(TABLE_POINTER)
  ` First value so update last value
  IF pointer=1000 THEN
    TABLE(1008,(TABLE(108)*scale))
  ` Second Value, so must update First & Last but 1 value
  ELSEIF pointer=1001 THEN
    TABLE(1000,(TABLE(100)*scale))
    TABLE(1009,(TABLE(109)*scale))
  ` Update previous value
  ELSE
    TABLE(pointer-1, (TABLE(pointer-901)*scale))
  ENDIF
ENDIF
WEND
STOP

```

See Also: CAM, CAMBOX, TABLE

TABLEVALUES

Type: System Command

Syntax: TABLEVALUES(first, last [,format])

Description: Returns a list of table values starting at the table address specified. The output is a comma delimited list of values..



TABLEVALUES is provided mainly for Motion Perfect to allow for fast access to banks of TABLE values.

arameters:

first:	First TABLE address to be returned.
last:	Last TABLE address to be returned
format:	Format for the list.

0 = Uncompressed comma delimited text (default)
 1 = Compressed comma delimited text, repeated values are compressed using a repeat count before the value (k7,0.0000 representing 7 successive values of 0.0000).
 Single values do not have the repeat count;

Example: For a controller containing the values 0.0, 0.1, 0.1, 0.1, 0.2, 0.2, 0.0 in addresses 1 to 7:-

```
>>TABLEVALUES(1,7,0)
0.0000,0.1000,0.1000,0.1000,0.2000,0.2000,0.0000
>>
```

```
>>TABLEVALUES(1,7,1)
0.0000,k3,0.1000,k2 0.2000,0.0000
>>
```

TICKS

Type: Process Parameter

Description: The current count of the process clock ticks is stored in this parameter. The process parameter is a 64 bit counter which is **DECREMENTED** on each servo cycle. It can therefore be used to measure cycle times, add time delays, etc. The **TICKS** parameter can be written to and read.



*As **TICKS** is a process parameter each process will have its own counter.*

Parameters: **value:** The value of the 64bit counter.

Example: With **SERVO _ PERIOD** set to 1000 use **TICKS** for a 3 second delay

```
delay:
TICKS=3000
OP(9,ON)
test:
IF TICKS<=0 THEN OP(9,OFF) ELSE GOTO test
```

TIME

Type: System Parameter

Description: Allows the user to set and read the time from the real time clock.

Parameters: **value:** Read = the number of seconds since midnight (24:00 hours)
Write = the time in 24hour format hh:mm:ss

Example 1: Sets the real time clock in 24 hour format; hh:mm:ss

```
    `Set the real time clock  
>>TIME = 13:20:00
```

```
Example 2:  
Calculate elapsed time in seconds.  
time1 = TIME  
`wait for event  
time2 = TIME  
timeelapsed = time1-time2
```

See also: TIME\$

TOKENTABLE

Type: Reserved Keyword

TRIGGER

Type: System Command

Description: Starts a previously set up **SCOPE** command. This allows you to start the scope capture at a specific part of your program.

Example: The *Motion Perfect* oscilloscope is set to record **MPOS** and **DPOS** of axis 0. The settings allow for program trigger and a repeat trigger. This loop can then be used as part of a **PID** tuning routine.

```

WHILE IN(tuning)=ON
DEFPOS(0)
TRIGGER
    WA(5)  'Allow the scope to start
    MOVE(100)
    WAIT IDLE
    WA(100)
    MOVE(-100)
    WA(100)
WEND

```

TROFF

Type: System Command

Syntax: **TROFF** ["program"]

Description: The trace off command resumes execution of the **SELECTed** or specified program. The command can be included in a program to resume the execution of that program.



For de-bugging the Motion Perfect breakpoint tool should be used.

Parameters: **program:** The name of the program which you wish to resume.

Example: Resume execution of a program names **TEST**.

```

>>TROFF "TEST"
OK
>>[%[Process 21:Program TEST] - Released

```

See Also: **HALT, STOP, STEPLINE, TRON**

TRON

Type: System Command

Syntax: TRON ["program"]

Description: The trace on command pauses the **SELECT** ed or specified program. The command can be included in a program to pause the execution of that program. The program can then be stepped through a single line, run or halted.

Parameters: **program:** The name of the program which you wish to step.



Motion Perfect highlights lines containing TRON in its editor and debugger. For de-bugging the Motion Perfect breakpoint tool should be used.

Example 1: Use suspend a program by including TRON. Another program will then use STEPLINE to step through until the TRON.

```
TRON
MOVE(0,10)
MOVE(10,0)
TROFF
MOVE(0,-10)
MOVE(-10,0)
```

Example 2: Start a program by stepping into the first line, then stepping through. The line that is stepped to is displayed.

```
>>SELECT "STARTUP"
STARTUP selected
>>TRON
OK
>>[%[Process 20:Line 3] - Paused
TABLE(0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)

STEPLINE
OK
>>[%[Process 20:Line 4] - Paused
TABLE(10,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)

STEPLINE
OK
>>[%[Process 20:Line 5] - Paused
TABLE(20,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)
```

Example 3: Pause a program called test that is currently running.

```
TRON "TEST"
OK
>>%[Process 21:Line 6] - Paused
WA(4)
```

See Also: HALT, STOP, STEPLINE, TROFF

TSIZE

Type: System Parameter (Read Only)

Description: Returns the size of the **TABLE**.



NOT ALL TABLE POSITIONS ARE BATTERY BACKED, SEE YOUR CONTROLLER INFORMATION FOR EXACT VALUES.

Parameters: **value:** The size of the **TABLE**.

Example: Check the size of the table and write to the last position in the table (remember the table starts at position 0).

```
>>?tsize
500000.0000
>>table(499999,123)
>>
```

UNIT_SW_VERSION

Type: Reserved Keyword

UNLOCK

Type: System Command (command line only)

Syntax: `UNLOCK(code)`

Description: Unlocks a controller than has previously been locked using the `LOCK` command.
To unlock the *Motion Coordinator*, the `UNLOCK` command should be entered using the same security code number which was used originally to `LOCK` it.



You should use Motion Perfect to `LOCK` and `UNLOCK` your controller.

Parameters: `code:` Any 7 digit integer number.

See Also: `LOCK`

VERSION

Type: System Parameter (read only)

Description: Returns the version number of the firmware installed on the *Motion Coordinator*.



You can use Motion Perfect to check the firmware version when looking at the controller configuration.

Parameters: `value:` Controllers firmware version number.


Example: Check the version of the firmware using the command line.

```
>>? VERSION
2.0100
>>
```

VIEW

Type:	System Command
Syntax:	<code>VIEW "program"</code>
Description:	Lists the currently selected or specified program in tokenised and internal compiled format.
Parameters:	program: The program name to <code>VIEW</code> .
Example:	For the following program: <pre>VR(10)=IN AND 255</pre> <p>the view command will give the output:</p> <pre>Source code: from xxx to xxx 10725: 00 15 00 29 92 95 31 30 00 93 88 64 A2 95 32 35 35 00 9B 10746: 15 00 00 00 Object code: from yyy to yyy 10750: 01 00 29 92 95 00 20 03 91 93 9A 64 95 00 00 7F 07 8E 91 9B 10771:</pre>

VR

Type:	Variable
Syntax:	<code>value = VR(expression)</code>
Description:	Recall or assign to a global numbered variable. The variables hold real numbers and can be easily used as an array or as a number of arrays.  <i>The numbered variables are globally shared between programs and can be used for communication between programs. To avoid problems where two processes write unexpectedly to a global variable, the programs should be written so that only one program writes to the global variables.</i>
Parameters:	value: The value written to or read from the <code>VR</code> . expression: Any valid TrioBASIC expression that produces an integer.

Example 1: Put value 1.2555 into VR() variable 15. Note local variable 'val' used to give name to global variable:

```
val=15
VR(val)=1.2555
```

Example 2: A transfer gantry has 10 put down positions in a row. Each position may at any time be FULL or EMPTY. VR(101) to VR(110) are used to hold an array of ten 1's or 0's to signal that the positions are full (1) or EMPTY (0). The gantry puts the load down in the first free position. Part of the program to achieve this would be:

```
movep:
  MOVEABS(115) `MOVE TO FIRST PUT DOWN POSITION:
  FOR VR(0)=101 TO 110
    IF VR(VR(0))=0) THEN
GOSUB load
    ENDIF
    MOVE(200)      ` 200 IS SPACING BETWEEN POSITIONS
  NEXT VR(0)
  PRINT "All Positions Are Full"
  WAIT UNTIL IN(3)=ON
GOTO movep

load:
  `PUT LOAD IN POSITION AND MARK ARRAY
  OP(15,OFF)
  VR(VR(0))=1
```

Example 3: Assign VR(65) with the value VR(0) multiplied by Axis 1 measured position.

```
VR(65)=VR(0)*MPOS AXIS(1)
PRINT VR(65)
```

VRSTRING

Type:	Print Formatter
Syntax:	<code>VRSTRING(variable)</code>
Description:	Combines the contents of an array of <code>VR()</code> variables so that they can be printed as a text string. All printable characters will be output and the string will terminate at the first null character found (i.e. <code>VR(n)</code> contains 0).
Parameters:	variable: Number of first <code>VR()</code> in the character array.
Example:	Print a sequence of characters stored in the <code>VR</code> 's starting at position 100. <code>PRINT #5,VRSTRING(100)</code>

WDOG

Type:	System Parameter
Description:	Controls the <code>WDOG</code> relay contact used for enabling external drives. The <code>WDOG=ON</code> command MUST be issued in a program prior to executing moves. It may then be switched <code>ON</code> and <code>OFF</code> under program control. If however a following error condition exists on any axis the system software will override the <code>WDOG</code> setting and turn watchdog contact <code>OFF</code> . When <code>WDOG=OFF</code> , the relay is opened, the analogue outputs are set to 0V, the step/direction outputs and any digital axis enable functions are disabled.
Example:	<code>WDOG=ON</code>



WDOG=ON / WDOG=OFF is issued automatically by Motion Perfect when the “Drives Enable” button is clicked on the control panel

When the `DISABLE _ GROUP` function is in use, the watchdog relay and `WDOG` remain on if there is an axis error. In this case, the digital enable signal is removed from the drives in that group only.

Mathematical Operations and Commands

+ (Add)

Type: Mathematical operation

Syntax: <expression1> + <expression2>

Description: Adds two expressions.

Parameters: **Expression1:** Any valid TrioBASIC expression.

Expression2: Any valid TrioBASIC expression.

Example: Add 10 onto the expression in the parentheses and store in a local variable. Therefore 'result' holds the value 28.9

```
result=10+(2.1*9)
```

- (Subtract)

Type: Arithmetic operation

Syntax: <expression1> - <expression2>

Description: Subtracts **expression2** from **expression1**.

Parameters: **Expression1:** Any valid TrioBASIC expression.

Expression2: Any valid TrioBASIC expression.

Example: Evaluate 2.1 multiply by 9 and subtract the result from 10, this will then be stored

```
in VR 0. Therefore VR 0 holds the value -8.9.  
VR(0)=10-(2.1*9)
```

* (Multiply)

Type: Mathematical operation.

Syntax: <expression1> * <expression2>

Description: Multiplies expression1 by expression2.

Parameters: **Expression1:** Any valid TrioBASIC expression.
Expression2: Any valid TrioBASIC expression.

Example: Calculate the value of 'factor' by multiplying 10 by the sum of 2.1 and 9. the value stored in 'factor' will be 111.
`factor=10*(2.1+9)`

/ (Divide)

Type: Mathematical operation

Syntax: <expression1> / <expression2>

Description: Divides expression1 by expression2.

Parameters: **Expression1:** Any valid TrioBASIC expression.
Expression2: Any valid TrioBASIC expression.

Example: Raises the first number (2) to the power of the second number (6).and store it in local variable 'x'. Then print the value of 'x' which is 64.
`x=2^6`
`PRINT x`

^ (Power)

Type: Mathematical operation

Syntax: <expression1> ^ <expression2>

Description: Raises `expression1` to the power of `expression2`.

Parameters: **Expression1:** Any valid TrioBASIC expression.

Expression2: Any valid TrioBASIC expression.

Example:
`x=2^6`
`PRINT x`

TrioBASIC raises the first number (2) to the power of the second number (6).

Therefore x has the value of 64

= (Equals)

Type: Comparison Operation

Syntax: <expression1> = <expression2>

Description: Returns `TRUE` if `expression1` is equal to `expression2`, otherwise returns false.



TRUE is defined as -1, and FALSE as 0

Parameters: **Expression1:** Any valid TrioBASIC expression.


Expression2: Any valid TrioBASIC expression.

Example: `IF IN(7)=ON THEN GOTO label`

If input 7 is `ON` then program execution will continue at line starting “label:”

Type:	Mathematical Operator				
Syntax:	<code>Value = expression</code>				
Description:	Assigns a value from the result of the expression.				
Parameters:	<table> <tr> <td>Value:</td> <td>the variable in which to store the value.</td> </tr> <tr> <td>Expression:</td> <td>Any valid TrioBASIC expression.</td> </tr> </table>	Value:	the variable in which to store the value.	Expression:	Any valid TrioBASIC expression.
Value:	the variable in which to store the value.				
Expression:	Any valid TrioBASIC expression.				
Example:	Set the sum of 10 and 9 into local variable named 'result'. <pre>result = 10 + 9</pre>				

<> (Not Equal)

Type:	Comparison Operation				
Syntax:	<code><expression1> <> <expression2></code>				
Description:	Returns TRUE if expression1 is not equal to expression2 , otherwise returns FALSE .				
	 <i>TRUE is defined as -1, and FALSE as 0</i>				
Parameters:	<table> <tr> <td>Expression1:</td> <td>Any valid TrioBASIC expression.</td> </tr> <tr> <td>Expression2:</td> <td>Any valid TrioBASIC expression.</td> </tr> </table>	Expression1:	Any valid TrioBASIC expression.	Expression2:	Any valid TrioBASIC expression.
Expression1:	Any valid TrioBASIC expression.				
Expression2:	Any valid TrioBASIC expression.				
Example:	Run the Scoop subroutine if axis is not idle (MTYPE=0 indicates axis idle). <pre>IF MTYPE<>0 THEN GOTO scoop</pre>				

> (Greater Than)

Type: Comparison Operation

Syntax: <expression1> > <expression2>

Description: Returns **TRUE** if **expression1** is greater than **expression2**, otherwise returns **FALSE**.



TRUE is defined as -1, and FALSE as 0

Parameters: **Expression1:** Any valid TrioBASIC expression

Expression2: Any valid TrioBASIC expression

Example 1: The program will wait until the measured position is greater than 200.

```
WAIT UNTIL MPOS>200
```

Example 2: Set the value of **TRUE** (-1) into VR 0 as 1 is greater than 0.

```
VR(0)=1>0
```

>= (Greater Than or Equal)

Type: Comparison Operation

Syntax: <expression1> >= <expression2>

Description: Returns **TRUE** if **expression1** is greater than or equal to **expression2**, otherwise returns **FALSE**.



TRUE is defined as -1, and FALSE as 0

Parameters: **Expression1:** Any valid TrioBASIC expression.

Expression2: Any valid TrioBASIC expression.

Example: If variable `target` holds a value greater than or equal to 120 then move to the absolute position of 0.

```
IF target>=120 THEN MOVEABS(0)
```

< (Less Than)

Type: Comparison Operation

Syntax: <expression1> < <expression2>

Description: Returns **TRUE** if expression1 is less than expression2, otherwise returns **FALSE**.



TRUE is defined as -1, and FALSE as 0

Parameters:

Expression1: Any valid TrioBASIC expression.

Expression2: Any valid TrioBASIC expression.

Example: Check that the value from analogue input 1 is less than 10, if it is then execute the sub routine 'rollup'.

```
IF AIN(1)<10 THEN GOSUB rollup
```

<= (Less Than or Equal)

Type: Comparison Operation

Syntax: <expression1> = <expression2>

Description: Returns **TRUE** if expression1 is less than or equal to expression2, otherwise returns **FALSE**.



TRUE is defined as -1, and FALSE as 0

Parameters: **Expression1:** Any valid TrioBASIC expression.
 Expression2: Any valid TrioBASIC expression.

Example: 1 is not less than or equal to 0 and therefore variable maybe holds the value 0
 (**FALSE**).
 maybe=1<=0

ABS

Type: Function

Syntax: **ABS(expression)**

Description: The **ABS** function converts a negative number into its positive equal. Positive numbers are unaltered.

Parameters: **Expression:** Any valid TrioBASIC expression.

Example: Check to see if the value from analogue input is outside of the range -100 to 100.
IF ABS(AIN(0))>100 THEN
 PRINT "Analogue Input Outside +/-100"
ENDIF

ACOS

Type:	Mathematical Function
Syntax:	<code>ACOS(expression)</code>
Description:	The <code>ACOS</code> function returns the arc-cosine of a number which should be in the range 1 to -1. The result in radians is in the range <code>0..PI</code>
Parameters:	Expression: Any valid TrioBASIC expression returning a value between -1 and 1.
Example:	Print the arc-cosine of -1 on the command line. <pre>>>PRINT ACOS(-1) 3.1416 >></pre>

AND

Type:	Logical and bitwise operator
Syntax:	<code><expression1> AND <expression2></code>
Description:	This performs an <code>AND</code> function between corresponding bits of the integer part of two valid TrioBASIC expressions. The <code>AND</code> function between two bits is defined as follows:
Parameters:	Expression1: Any valid TrioBASIC expression. Expression2: Any valid TrioBASIC expression.
Example 1:	Using <code>AND</code> to compare two logical expressions, if they are both true then set a local variable. <pre>IF (IN(6)=ON) AND (DPOS>100) THEN tap=ON ENDIF</pre>

Example 2: Use **AND** as a bitwise operator.

```
VR(0)=10 AND (2.1*9)
```

TrioBASIC evaluates the parentheses first giving the value 18.9, but as was specified earlier, only the integer part of the number is used for the operation, therefore this expression is equivalent to:

```
VR(0)=10 AND 18
```

AND is a bitwise operator and so the binary action taking place is:

	0	1
0	0	0
1	0	1

```

          01010
AND      10010
-----
          00010

```

Therefore **VR(0)** holds the value 2

Example 3: If both **MPOS** are set to 0 then run a sub routine 'cycle'

```

IF MPOS AXIS(0)>0 AND MPOS AXIS(1)>0 THEN
  GOSUB cycle
ENDIF

```

ASIN

Type: Mathematical Function

Syntax: `ASIN(expression)`

Alternate Format: `ASN(expression)`

Description: The `ASIN` function returns the arc-sine of a number which should be in the range $+/-1$. The result in radians is in the range $-\pi/2.. +\pi/2$.

Parameters: **Expression:** Any valid TrioBASIC expression returning a value between -1 and 1.

Example: Print the arc-sine of -1 on the command line.

```
>>PRINT ASIN(-1)
-1.5708
```

ATAN

Type: Mathematical Function

Syntax: `ATAN(expression)`

Alternate Format: `ATN(expression)`

Description: The `ATAN` function returns the arc-tangent of a number. The result in radians is in the range $-\pi/2.. +\pi/2$.

Parameters: **Expressions:** Any valid TrioBASIC expression.

Example: Print the arc-tangent of -1 on the command line.

```
>>PRINT ATAN(1)
0.7854
```

ATAN2

Type:	Mathematical Function
Syntax:	<code>ATAN2(expression1,expression 2)</code>
Description:	The ATAN2 function returns the arc-tangent of the ratio expression1 / expression 2 . The result in radians is in the range -PI.. +PI
Parameters:	Expressions: Any valid TrioBASIC expression.
Example:	Print the arc-tangent of 0 divided by 1 on the command line <pre>>>PRINT ATAN2(0,1) 0.0000</pre>

B_SPLINE

Type:	Command
Syntax:	<code>B _ SPLINE(mode, {parameters})</code>
Description:	This function expands data to generate higher resolution motion profiles. It operates in two modes using either B Spline or Non Uniform Rational B Spline (NURBS) mathematical methods.
.....	
Syntax:	<code>B _ SPLINE(1, data _ in, points, data _ out, expansion _ ratio)</code>
Description:	Expands an existing profile stored in the TABLE area using the B Spline mathematical function. The expansion factor is configurable and the B _ SPLINE stores the expanded profile to another area in the TABLE .



This is ideally used where the source CAM profile is too coarse and needs to be extrapolated into a greater number of points.

Parameters:

mode:	1 Standard B-Spline.
data _ in:	Location in the TABLE where the source profile is stored.
points:	Number of points in the source profile.
data _ out:	Location in the TABLE where the expanded profile will be stored.
expansion _ ratio:	The expansion ratio of the B _ SPLINE function. Total output points = (Number of points+1) * expansion (i.e. if the source profile is 100 points and the expansion ratio is set to 10 the resulting profile will be 1010 point ((100+1) * 10).

Example: Expands a 10 point profile in **TABLE** locations 0 to 9 to a larger 110 point profile starting at **TABLE** address 200.

```
B _ SPLINE(1,0,10,200,10)
```

Syntax: `B _ SPLINE(2, dimensions, curve _ type, weight _ op, points, knots, expansion, in _ data, out _ data)`

Description: Non Uniform Rational B-Splines, commonly referred to as **NURBS**, have become the industry standard way of representing geometric surface information designed by a CAD system.

NURBS provide a unified mathematical basis for representing analytic shapes such as conic sections and quadratic surfaces, as well as free form entities, such as car bodies and ship hulls.

NURBS are small for data portability and can be scaled to increase the number of target points along a curve, increasing accuracy. A series of **NURBS** are used to describe a complex shape or surface.

NURBS are represented as a series of **xyz** points with knots + weightings of the knots.

Parameters:

mode:	2 Non Uniform Rational B-Spline.
dimensions:	Defines the number of axes. Reserved for future use must be 3.
Curve _ type:	Classification of the type of NURBS curve.

Reserved for future use must be 3.

Weight _ op: Sets the weighting of the knots
 0=All weighting set to 1.

points: Number of data points.

knots: Number of knots defined.

expansion: Defines the number of points the expanded curve will have in the table.
 Total output points = Number of points * expansion.
 Minimum value = 3.

in _ data: Location of input data.
 Data is stored with X0,Y0,Z0,X1,Y1,Z1...,followed by knots data N0, N1, N2 ...

out _ data: Table start location for output points stored X0, Y0, Z0 etc.

Example:

Starting with 9 sets of X Y Z data point and expanding by 5, resulting with 45 sets of X Y Z data points (135 table points). The profile is then split from the XYZ groups into separate axis so that the profiles can be executed using CAMBOX.

```
weight_op=0          `0 sets all weights to 1.0
points=9             `number of data points
knots=13             `number of knots
expansion=5          `expansion factor
in_data=100          `data points
out_data=1000        `table location to construct output

` Data Points:
TABLE(100,150.709,353.8857,0)
TABLE(103,104.5196,337.7142,0)
TABLE(106,320.1131,499.4647,0)
TABLE(109,449.4824,396.4945,0)
TABLE(112,595.3350,136.4910,0)
TABLE(115,156.816,96.3351,0)
TABLE(118,429.4556,313.7982,0)
TABLE(121,213.3019,375.8004,0)
TABLE(124,150.709,353.8857,0)

` Knots:
TABLE(127,0,0,0,0,146.8154,325.6644,536.0555,763.4151,910.1338,1109.0886)
TABLE(137,1109.0886,1109.0886,1109.0886)

`Expand the curve, generate 5*9=45 XYZ points
`or 135 table locations

B_SPLINE(2, 3, 3, weight_op, points, knots, expansion, in_
data, out_data)
```

```

`Split the profile into X Y Z
FOR p= 0 TO 44
  TABLE(8000+p, TABLE(1000+(p*3)+0))
  TABLE(10000+p, TABLE(1000+(p*3)+1))
  TABLE(12000+p, TABLE(1000+(p*3)+2))
NEXT p

`Execute the profile using CAMBOX, synchronised using axis 4
BASE(0)
DEFPOS(0,0,0,0)
CAMBOX(8000,8044,1,100,4)
BASE(1)
CAMBOX(10000,10044,1,100,4)
BASE(2)
CAMBOX(12000,12044,1,100,4)
BASE(4)
MOVE(100)

```

CLEAR_BIT

Type:	Command
Syntax:	<code>CLEAR_BIT(bit,variable)</code>
Description:	<code>CLEAR_BIT</code> can be used to clear the value of a single bit within a <code>VR()</code> variable.
Parameters:	<p>bit: The bit number to clear, valid range is 0 to 52.</p> <p>variable: The VR which to operate on.</p>
Example:	<p>Set bit 6 in VR 23 to zero.</p> <pre>CLEAR_BIT(6,23)</pre>
See also	<code>READ_BIT</code> , <code>SET_BIT</code>

CONSTANT

Type: System Command

Syntax: `CONSTANT ["name" [, value]]`

Description: Up to 1024 **CONSTANTS** can be declared in the controller, these are then available to all programs. They should be declared on startup and for fast startup the program declaring **CONSTANTS** should also be the **ONLY** process running at power-up.



*Once a **CONSTANT** has been assigned it cannot be changed, even if you change the program that assigns it.*



*While developing you may wish to clear or change a **CONSTANT**. You can clear a single **CONSTANT** by using the first parameter alone. All **CONSTANTS** can be cleared by issuing **CONSTANT**. You can view all **CONSTANTS** using **LIST _ GLOBAL**.*

Parameters:

name:	Any user-defined name containing lower case alpha, numerical or underscore (_) characters.
value	The value assigned to name.

Example 1: Declare 2 **CONSTANT**'s and use them within the program.

```
CONSTANT "nak", $15
CONSTANT "start _ button", 5

IF IN(start _ button)=ON THEN OP(led1,ON)
IF key _ char=nak THEN GOSUB no _ ack _ received
```

Example 2: Use the command line to clear a defined constant.

```
>>CONSTANT "NAK"
>>
```

Example 3: Use the command line to clear all defined constants.

```
>>CONSTANT
>>
```

See Also: GLOBAL, LIST _ GLOBAL

COS

Type: Mathematical Function

Syntax: COS(expression)

Description: Returns the COSINE of an expression. Input values are in radians.

Parameters:

value:	The COSINE of the expression.
expression:	Any valid TrioBASIC expression.

Example: Print the cosine of zero to the command line with 3 decimal places

```
>>PRINT COS(0) [3]
1.000
```

CRC16

Type: Mathematical Command

Syntax: result = CRC16(mode,{parameters})

Description: Calculates a 16 bit Cyclic Redundancy Check (CRC) of data stored in contiguous Table Memory or VR Memory locations.

Parameters:

MODE:	0 = Initialise the polynomial
	1 = Calculate the CRC

Syntax: **result = CRC16(0, poly)**

Description: Initialises the command with the Polynomial

Parameters: **result:** Always returns -1
 poly: Polynomial used as seed for CRC check range 0-65535 (or 0-**\$FFFF**)

Syntax: **result = CRC16(1, source, start, end, initial)**

Description: Calculates the CRC

Parameters: **result:** Returns the result of the CRC calculation. Will be 0 if the calculation fails.
 source: Defines where the data is loaded.
 0 = Table Memory
 1 = VR Memory
 start: start location of first byte.
 end: end Location of last byte.
 initial: initial CRC value. Normally \$0 - **\$FFFF**.

Example 1: Calculate the CRC using Table Memory:

```
poly = $90d9
reginit = $ffff
CRC16(0, poly) 'Initialise internal CRC table memory
TABLE(0,1,2,3,4,5,6,7,8) 'Load data into table memory location 0-7
calc _crc = CRC16(1,0,0,7,reginit) 'Source Data=TABLE(0..7)
```

Example 2: Calculate the CRC using VR Memory:

```
poly = $90d9
reginit = $ffff
CRC16(0, poly) 'Initialise internal CRC table memory
'Load 6 bytes into VR memory location 0-5
for i=0 to 5
  VR(i)=i+1
Next i
calc _crc = CRC16(1,1,0,5,reginit) 'Source Data=VR(0)..VR(5)
```

EXP

Type:	Mathematical Function
Syntax:	<code>EXP(expression)</code>
Description:	Returns the exponential value of the expression.
Parameters:	expression: Any valid TrioBASIC expression.
Example:	Print the exponential value of 1. <pre>>>PRINT EXP(1) 2.7183 >></pre>

FRAC

Type:	Mathematical Function
Syntax:	<code>value = FRAC(expression)</code>
Description:	Returns the fractional part of the expression.
Parameters:	value: The fractional part of the expression. expression: Any valid TrioBASIC expression.
Example:	Print the fractional part of 1.234 on the command line. <pre>>>PRINT FRAC(1.234) 0.2340 >></pre>

GLOBAL

Type: System Command

Syntax: GLOBAL "name", vr _ number

Description: Up to 1024 GLOBALs can be declared in the controller, these are available to all programs. GLOBAL declares the name as a reference to one of the global VR variables. The name can then be used both within the program containing the GLOBAL definition and all other programs in the *Motion Coordinator* project.

They should be declared on startup and for fast startup the program declaring GLOBALs should also be the **ONLY** process running at power-up.



Once a GLOBAL has been assigned it cannot be changed, even if you change the program that assigns it.



While developing you may wish to clear or change a GLOBAL. You can clear a single GLOBAL by using the first parameter alone. All GLOBAL's can be cleared by issuing GLOBAL. You can view all GLOBAL's using LIST _ GLOBAL.

Parameters:

name:	Any user-defined name containing lower case alpha, numerical or underscore (_) characters.
vr _ number	The number of the VR to be associated with name.


Example: Initialise 2 GLOBALs and use then to adjust machine parameters.

```
GLOBAL "screw_pitch",12
GLOBAL "ratio1",534


ratio1 = 3.56
screw_pitch = 23.0
PRINT screw_pitch, ratio1
```

See Also: CONSTANT, LIST _ GLOBAL

IEEE_IN

- Type:** Mathematical Function
- Syntax:** `IEEE_IN(byte0,byte1,byte2,byte3)`
- Description:** The `IEEE_IN` function returns the floating point number represented by 4 bytes which typically have been received over a communications link such as Modbus.
-  *Byte 0 is the high byte of the 32 bit floating point format.*
- Parameters:** `byte0 - 3:` Any combination of 8 bit values that represents a valid `IEEE` floating point number.
- Example:** Take 4 bytes that have been sent over Modbus to `VR`'s and recombine them into a floating point number.
- `VR(200) = IEEE_IN(VR(0),VR(1),VR(2),VR(3))`

IEEE_OUT

- Type:** Mathematical Function
- Syntax:** `byte_n = IEEE_OUT(value, n)`
- Description:** The `IEEE_OUT` function returns a single byte in `IEEE` format extracted from the floating point value for transmission over a bus system. The function will typically be called 4 times to extract each byte in turn.
- Parameters:** `value:` Any TrioBASIC floating point variable or parameter.
`n:` The byte number (0 - 3) to be extracted.
-  *Byte 0 is the high byte of the 32 bit IEEE floating point format.*
- Example:** Extract the 4 bytes from `MPOS` and store them in local variables ready for transmission over a communications bus.
- `a = MPOS AXIS(2)`


```
byte0 = IEEE _ OUT(a, 0)
byte1 = IEEE _ OUT(a, 1)
byte2 = IEEE _ OUT(a, 2)
byte3 = IEEE _ OUT(a, 3)
```

INT

Type: Mathematical Function

Syntax: `value = INT(expression)`

Description: The **INT** function returns the integer part of a number.
To round a positive number to the nearest integer value take the **INT** function of the (number + 0.5).

Parameters:

expression:	Any valid TrioBASIC expression.
value:	The integer part of the expression.

Example 1:

```
>>PRINT INT(1.79)
1.0000
```

Example 2: Round a value to the nearest integer.

```
IF value>0 THEN
rounded = INT(value + 0.5)
ELSE
rounded = INT(value - 0.5)
ENDIF
```

INTEGER_READ

Type:	Mathematical Command
Syntax:	<code>INTEGER_READ(source, least_significant, most_significant)</code>
Description:	<code>INTEGER_READ</code> performs a low level access to the 64 bit register splitting it into two 32 bit segments.



This can be used to read the position from high resolution encoders.

Parameters:	source:	2 bit value that will be read, can be <code>VR</code> , <code>TABLE</code> , or system variable.
	least_significant:	The variable to store the least significant (rightmost) 32 bits, this may be local variable, <code>VR</code> or <code>TABLE</code> .
	most_significant:	The variable to store the most significant (leftmost) 32 bits, this may be local variable, <code>VR</code> or <code>TABLE</code> .

INTEGER_WRITE

Type:	Mathematical Command	
Syntax:	<code>INTEGER_WRITE(destination, least_significant, most_significant)</code>	
Description:	<code>INTEGER_WRITE</code> performs a low level write to a 64 bit register by combining two 32 bit segments.	
Parameters:	destination:	64 bit value that will be written, can be <code>VR</code> , <code>TABLE</code> , or system variable..
	least_significant:	Least significant (rightmost) 16 bits, can be any valid TrioBASIC expression.
	most_significant:	Most significant (leftmost) 16 bits, can be any valid TrioBASIC expression.

LN

Type:	Mathematical Function
Syntax:	<code>value = LN(expression)</code>
Description:	Returns the natural logarithm of the expression.
Parameter:	value: The natural logarithm of the expression. expression: Any valid TrioBASIC expression .
Example:	Storing the natural logarithm of a value in VR(0) <code>VR(0) = LN(a*b)</code>

MOD

Type:	Mathematical Operator
Syntax:	<code>value = expression1 MOD(expression2)</code>
Description:	Returns the integer modulus of an expression, this is the value after the integer has wrapped around the modulus
Parameter:	expression 1: Any valid TrioBASIC expression used as the value to apply the modulus to. expression 2: Any valid TrioBASIC expression used as the modulus.
Example:	Use the MOD(12) to turn a 24 hour value into 12 hour. <pre>>>PRINT 18 MOD(12) 6.0000 >></pre>

NOT

Type: Logical and Bitwise Function

Syntax: NOT *expression*

Description: The NOT function truncates the number and inverts all the bits of the integer remaining.

Parameter: *expression*: Any valid TrioBASIC expression.

Example 1: Bitwise AND 7 with NOT 1.5. This truncates 1.5 to 1 then ANDs it with 7.

```
PRINT 7 AND NOT(1.5)
      6.0000
```

Example 2: If a function fails then print an error message and stop the program.

```
IF NOT CAN(0,9,13,1,8,$6060,0,$02) THEN
  PRINT#user, "Failed to set velocity mode"
  STOP
ENDIF
```

OR

Type: Logical and bitwise operator

Syntax: <*expression1*> OR <*expression2*>

Description: This performs an OR function between corresponding bits of the integer part of two valid TrioBASIC expressions. The OR function between two bits is defined as follows:

OR	0	1
0	0	1
1	1	1

Parameters: **Expression1:** Any valid TrioBASIC expression.
 Expression2: Any valid TrioBASIC expression.

Example 1: Use OR to allow the program to progress if there is a MOTION _ ERROR or an input is pressed.

```
WAIT UNTIL IN(2)=ON OR MOTION _ ERROR
```

Example 2: Calculate the bitwise OR between values

```
result=10 OR (2.1*9)
```

TrioBASIC evaluates the parentheses first giving the value 18.9, but as was specified earlier, only the integer part of the number is used for the operation, therefore this expression is equivalent to:

```
result=10 OR 18
```

The OR is a bitwise operator and so the binary action taking place is:

```
      01010  
OR   10010  
      11010
```

Therefore result holds the value 26.

READ_BIT

Type: Command

Syntax: **READ_BIT**(bit, variable)

Description: **READ_BIT** can be used to test the value of a single bit within a VR() variable.

Parameters: **bit:** The bit number to clear, valid range is 0 to 52.
 variable: The VR which to operate on.

Example: Read bit 4 of VR(13).
Result = `READ_BIT(4,13)`

See also `SET_BIT`, `CLEAR_BIT`

SET_BIT

Type: Logical and Bitwise Command

Syntax: `SET_BIT(bit, variable)`

Description: `SET_BIT` can be used to set the value of a single bit within a `VR()` variable. All other bits are unchanged.

Parameters: **bit:** The bit number to clear, valid range is 0 to 52.
variable: The VR which to operate on.

Example: Set bit 3 of VR(7)
`SET_BIT(3,7)`

See Also `READ_BIT`, `CLEAR_BIT`

SGN

Type: Mathematical Function

Syntax: `value = SGN(expression)`

Description: The `SGN` function returns the `SIGN` of a number.

1	Positive non-zero
0	Zero

-1 Negative

Parameters: **expression:** Any valid TrioBASIC expression.

Example: Detect the sign of the number -1.2 using the command line.

```
>>PRINT SGN(-1.2)
-1.0000
```

SIN

Type: Mathematical Function

Syntax: **value=SIN(expression)**

Description: Returns the **SINE** of an expression. This is valid for any value in expressed in radians.

Parameters: **value:** the **SINE** of the expression in radians.
 expression: Any valid TrioBASIC expression.

Example: Print the **SINE** of 0 on the command line.

```
>>PRINT SIN(0)
          0.0000
>>
```

SQR

Type: Mathematical Function

Syntax: `value=SQR(number)`

Description: Returns the square root of a number.

Parameters: `value:` The square root of the number
`number:` Any valid TrioBASIC number or variable.

Example: Calculate the square root of 4 using the command line.

```
>>PRINT SQR(4)
2.0000
```

TAN

Type: Mathematical Function

Syntax: `value = TAN(expression)`

Description: Returns the **TANGENT** of an expression. This is valid for any value expressed in radians.

Parameters: `value:` The **TANGENT** of the expression.
`Expression:` Any valid TrioBASIC expression.

Example: Print the tangent of 0.5 using the command line.

```
>>PRINT TAN(0.5)
0.5463
>>
```


XOR

Type: Logical and bitwise operator

Description: This performs an exclusive or function between corresponding bits of the integer part of two valid TrioBASIC expressions. It may therefore be used as either a bitwise or logical condition.

The `XOR` function between two bits is defined as follows:

result = expression1 `XOR` expression

Example: `a = 10 XOR (2.1*9)`

TrioBASIC evaluates the parentheses first giving the value 18.9, but as was specified earlier, only the integer part of the number is used for the operation, therefore this expression is equivalent to: `a=10 XOR 18`. The `XOR` is a bitwise operator and so the binary action taking place is:

```
          01010
XOR      10010
-----
          11000
```

The result is therefore 24.

Constants

FALSE

Type: Constant

Description: The constant **FALSE** takes the numerical value of 0.

Example: test:

```
Use FALSE as part of a logical check
res = IN(0) OR IN(2)
IF res = FALSE THEN
    PRINT "Inputs are off"
ENDIF
```

OFF

Type: Constant

Description: **OFF** returns the value 0

Example 1: Run the subroutine "tiger" if input 56 is off.

```
IF IN(56)=OFF THEN GOSUB tiger
```

Example 2: Turn the watchdog relay off.

```
WDOG = OFF
```

ON

Type: Constant

Description: ON returns the value 1.

Example: This sets the output named lever to ON.
`OP(lever,ON)`

PI

Type: Constant

Description: PI is the circumference/diameter constant of approximately 3.14159.

Example 1: To print the radius of a circle of given circumference.

```
circum=100
PRINT "Radius = ";circum / (2*PI)
```

Example 2: Set the axis calibration to work in user units of Radians.

```
Motor has 8192 counts per turn
UNITS = 8192 / (2*PI)
```

TRUE

Type: Constant

Description: The constant TRUE takes the numerical value of -1.

Example: Checks that the logical result of input 0 and 1 is true.

```
t=IN(0)=ON AND IN(2)=ON
IF t=TRUE THEN
    PRINT "Inputs are on"
```

ENDIF

Axis Parameters

ACCEL

Type: Axis Parameter

Description: The `ACCEL` axis parameter may be used to set or read back the acceleration rate of each axis fitted. The acceleration rate is in units/sec/sec.

Example: Set the acceleration rate and print it to the terminal.

```
ACCEL=130
PRINT "Acceleration rate=";ACCEL;" mm/sec/sec"
```

ADDAX_AXIS

Type: Axis Parameter (Read Only)

Description: Returns the axis currently linked to with the `ADDAX` command, if none the parameter returns -1.

Example: Check if an `ADDAX` to axis 2 exists as part of a reset sequence, if it does then cancel it.

```
IF ADDAX_AXIS = 2 then
  ADDAX(-1)
ENDIF
```

AFF_GAIN

Type: Axis Parameter

Description: Sets the acceleration Feed Forward for the axis. This is a multiplying factor which is applied to the rate of change of demand speed. The result is summed to the control loop output to give the `DAC_OUT` value.

`AFF_GAIN` is only effective in systems with very high counts per revolution in the feedback. I.e. 65536 counts per rev or greater.

ATYPE

Type: Axis Parameter

Description: The `ATYPE` axis parameter indicates the type of axis fitted. By default this will be set to match the hardware, but some modules allow configuration of different operation.

If you are setting a non default `ATYPE`, this must be done during initialisation through a TrioBASIC program for example `STARTUP.BAS`.

#	Description
0	No axis daughter board fitted/ virtual axis
43	Step and direction output
44	Incremental encoder Servo
45	Quadrature encoder output
46	Tamagawa absolute Servo
47	Endat absolute Servo
48	SSI absolute Servo
50	RTEX position
51	RTEX speed
52	RTEX torque
53	SERCOS velocity
54	SERCOS position

#	Description
55	SERCOS torque
56	SERCOS open
57	SERCOS velocity with drive registration
58	SERCOS position with drive registration
59	SERCOS spare
60	Step and direction feedback Servo
61	SLM
62	PLM
63	Stepper with Z input
64	Encoder out with Z input
65	EtherCAT position
66	EtherCAT speed



See Chapter 4 to find which **ATYPE** your hardware supports.

Example 1:

Check for a flexible axis on axis 0, then set a stepper on axis 0 and SSI encoder on axis 1. The default for a flexible axis is servo

```

BASE(0)
IF ATYPE = 44 THEN
  ATYPE = 43
  BASE(1)
  ATYPE = 48
ENDIF

```

Example 2:

Set a the **ATYPE** so a **SERCOS** axis uses velocity mode with drive registration.

```
ATYPE AXIS(12)=57
```

AXIS_ADDRESS

Type: Axis Parameter

Description: The **AXIS_ADDRESS** parameter holds the address of the drive or feedback device. For example can be used to specify the **SERCOS** drive address or **AIN** channel that is used for feedback on the base axis.

Parameters: **Drive Address:** node number or analogue input number.



You may require additional Feature Enable Codes before using the remote axis functionality.

Example: Assigning the **SERCOS** drive with the node address 4 to axis 8 in the controller. Then starting it in position mode with drive registration.

```
BASE(8)  
AXIS_ADDRESS = 4  
ATYPE= 58
```

AXIS_DEBUG_A

Type: Reserved Keyword.

Description: Use only when instructed by Trio as part of an operational analysis.

AXIS_DEBUG_B

Type: Reserved Keyword.

Description: Use only when instructed by Trio as part of an operational analysis.

AXIS_DISPLAY

Type: Reserved Keyword.

AXIS_ENABLE

Type: Axis Parameter

Description: Can be used to independently disable an axis. **ON** by default, can be set to **OFF** to disable the axis. The axis is enabled if **AX_ENABLE = ON** and **WDOG=ON**.

On stepper axis **AXIS_ENABLE** will turn on the hardware enable outputs.



*If the axis is part of a **DISABLE_GROUP** and an error occurs **AXIS_ENABLE** is set to **OFF** but the **WDOG** remains **ON**.*

Parameter: Accepts the values **ON** or **OFF**, default is **ON**.

Example: Re-enabling a group of axes after a motion error.

```
DEFPOS(0) 'Clear the error
For axis_number = 4 to 8
BASE(axis_number)
AXIS_ENABLE = ON 'Enable the axis
NEXT axis_number
```

See Also: **DISABLE_GROUP**

AXIS_ERROR_COUNT

Type: Axis Parameter.

Description: Each time there is a communications error on a digital axis, the `AXIS_ERROR_COUNT` parameter is incremented. Where supported, this value can be used as an indication of the error rate on a digital axis. Not all digital axis types have the ability to count the errors. Further information can be found in the description of each type of digital communications bus.

Parameter: The communications error count since last reset.

Example: Initialise the error counter.

```
AXIS_ERROR_COUNT = 0
```

In the terminal, check the latest error count value.

```
>>?AXIS_ERROR_COUNT AXIS(3)
10.0000
>>
```

Keep a record of the overall error rate for an axis.

```
TICKS = 600000
AXIS_ERROR_COUNT = 0
REPEAT
  IF TICKS<0 THEN
    VR(10) = AXIS_ERROR_COUNT
              \ number of errors counted in ten minutes
    TICKS = 600000
    AXIS_ERROR_COUNT = 0
  ENDIF
  ...
  ...
UNTIL FALSE
```

AXIS_MODE

Type: Axis Parameter

Description: This parameter enables various different features that an axis can use.

Parameters: value:

Bit	Description	Value
1	Prevents CONNECT from canceling when a hardware or software limit is reached, the ratio is set to 0.	2
2	Enable 3D direction calculations (default 2D)	4
6	Use non sign-extended analogue feedback	64

Example 1: Enable bit 2 so that you can use 3D direction calculations, the **AND** is used so that only bit 2 is changed.

```
AXIS_MODE = AXIS_MODE AND 4
```

Example: Enable bit 6 so that you can use a 0 to 10V analogue input as axis feedback. The **AND** is used so that only bit 6 is changed.

```
BASE(5)  
AXIS_MODE = AXIS_MODE AND 64
```

See Also: `ERRORMASK`, `DATUM(0)`

AXISSTATUS

Type: Axis Parameter (Read Only)

Description: The **AXISSTATUS** axis parameter may be used to check various status bits held for each axis fitted:

Parameters: **value:** 15bit value, each bit represents a different status bit.

Bit	Description	Value	char
0	Override speed set	1	
1	Following error warning range	2	w
2	Communications error to remote drive	4	a
3	Remote drive error	8	m
4	In forward hardware limit	16	f
5	In reverse hardware limit	32	r
6	Datuming in progress	64	d
7	Feedhold	128	h
8	Following error exceeds limit	256	e
9	In forward software limit	512	x
10	In reverse software limit	1024	y
11	Cancelling move	2048	c
12	Encoder power supply overload	4096	o
13	MOVETANG decelerating	8192	

In the *Motion Perfect* parameter screen the **AXISSTATUS** parameter is displayed as a series of characters, ocyxehdrfmaw, as listed in the table above.

These characters are displayed in green lowercase letters normally, or red uppercase when set.

Example: Check bit 4 to see if the axis is in forward limit.

```
IF (AXISSTATUS AND 16)>0 THEN
    PRINT "In forward limit"
ENDIF
```

See Also: **ERRORMASK, DATUM(0)**

BACKLASH_DIST

Type: Axis Parameter

Description: Amount of backlash compensation that is being applied to the axis when **BACKLASH** is on.

Example: Illuminate a lamp to show that the backlash has been compensated for.

```
IF BACKLASH_DIST>100 THEN
  OP (10, ON)
  'show that backlash compensation has reached this value
ELSE
  OP (10, OFF)
END IF
```

CHANGE_DIR_LAST

Type: Axis Parameter (read only)

Description: Returns the difference between the direction of the end of the previous loaded interpolated motion command and the start direction of the last loaded interpolated motion command. If there is no previous loaded command then **END_DIR_LAST** can be written to set an initial direction.

This parameter is only available when using **SP** motion commands such as **MOVESP**, **MOVEABSSP** etc.

Parameters: **Value:** Change in direction, in radians between 0 and π . Value is always positive.

Example 1: Perform a 90 degree move and print the change.

```
>>MOVESP(0,100)
>>MOVESP(100,0)
>>PRINT CHANGE_DIR_LAST
1.5708
>>
```

See Also: `END _ DIR _ LAST, START _ DIR _ LAST`

CLOSE_WIN

Type: Axis Parameter

Alternate Format: `CW`

Description: By writing to this parameter the end of the window in which a registration mark is expected can be defined. The value is in user units.

Parameters: `Value:` Position of the end of the position window in user units.

Example: Set a position window between 10 and 30

```
OPEN _ WIN = 10
CLOSE _ WIN = 30
```

See Also: `OPEN _ WIN, REGIST`

CLUTCH_RATE

Type: Axis Parameter

Description: This affects operation of `CONNECT` by changing the connection ratio at the specified rate/second.

Default `CLUTCH _ RATE` is set very high to ensure compatibility with earlier versions.

Parameters: **Value:** change in connection ratio per second (default 1000000).

Example: The connection ratio will be changed from 0 to 6 when an input is set. It is required to take 2 second to accelerate the linked axis so the ratio must change at 3 per second.

```
CLUTCH_RATE = 3
CONNECT(0,0)
WAIT UNTIL IN(1)=ON
CONNECT(6,0)
```

COORDINATOR_DATA

Type: Reserved Keyword.

CORNER_MODE

Type: Axis Parameter

Description: Allows the program to control the cornering action.

Automatic corner speed control enables system to reduce the speed depending on **DECEL _ ANGLE** and **STOP _ ANGLE**

The **CORNER _ STATE** machine allows interaction with a TrioBASIC program and the loading of buffered moves depending **RAISE _ ANGLE**

Automatic radius speed control enables the system to reduce the speed depending on **FULL _ SP _ RADIUS**.



You can enable any combination of the speed control bits.

Parameters: **Value:** Bit 0 = Reserved
 Bit 1 = Automatic corner speed control.
 Bit 2 = Enable the **CORNER _ STATE** machine
 Bit 3 = Automatic radius speed control.

Example: Enable the corner state machine and automatic corner speed control.

```
CORNER _ MODE= 2+4
```

See Also: CORNER _ STATE, DECEL _ ANGLE, FULL _ SP _ RADIUS, RAISE _ ANGLE, STOP _ ANGLE

CORNER_STATE

Type: Axis Parameter

Description: Allows a BASIC program to interact with the move loading process.

This can be used to facilitate tool adjustment such as knife rotation at sharp corners.



This parameter is only active when CORNER _ STATE bit 2 is set. It is also required to use bit 1 of CORNER _ STATE with STOP _ ANGLE set to less than or equal to RAISE _ ANGLE to stop the motion.

Parameters: Value: 0 = Load move and ramp up speed
1 = Ready to load move, stopped
3 = Load move

Example: When a transition exceeds RAISE _ ANGLE it is required to lift a cutting knife and rotate it to a new position. The following process is required:

- System sets CORNER _ STATE to 1 to indicate move ready to be loaded with large angle change.
- BASIC program raises knife.
- BASIC program sets CORNER _ STATE to 3.
- System will load following move but with speed overridden to zero. This allows the direction to be obtained from TANG _ DIRECTION.
- BASIC program orients knife possibly using MOVE _ TANG.
- BASIC program clears CORNER _ STATE to 0.
- System will ramp up speed to perform the next move.


```
MOVEABSSP(x,y)
IF CHANGE _DIR _LAST>RAISE _ANGLE THEN
  WAIT UNTIL CORNER _STATE>0
  `Raise Knife
  MOVE(100) AXIS(z)
  CORNER _STATE=3
  WA(10)
  WAIT UNTIL VP _SPEED AXIS(2)=0
  `Rotate Knife
  MOVETANG(0,x) AXIS(r)
  `Lower Knife
  MOVE(-100) AXIS(z)
  `Resume motion
  CORNER _STATE=0
ENDIF
```

See Also: CORNER _MODE, RAISE _ANGLE, STOP _ANGLE

CREEP

Type: Axis Parameter

Description: Sets the **CREEP** speed on the current base axis. The **CREEP** speed is used for the slow part of a **DATUM** sequence.

Parameters: **value:** Any positive value in user **UNITS**.

Example: Set up the **CREEP** speeds on 2 axes and then perform a **DATUM** routine.


```
BASE(2)
CREEP=10
SPEED=500
DATUM(4)
CREEP AXIS(1)=10
SPEED AXIS(1)=500
DATUM(4) AXIS(1)
```

See Also: **DATUM**

D_GAIN

Type:	Axis Parameter
Description:	<p>The derivative gain is a constant which is multiplied by the change in following error.</p> <p>Adding derivative gain to a system is likely to produce a smoother response and allow the use of a higher proportional gain than could otherwise be used.</p> <p>High values may lead to oscillation. For a derivative term K_D and a change in following error $d\epsilon$ the contribution to the output signal is:</p> $\Phi_D = K_D \times d\epsilon$
Parameters:	<p>Value: The derivative gain is a constant which is multiplied by the change in following error. Default value = 0.</p>
Example:	<p>Setting the gain values as part of a STARTUP program</p> <pre>P _ GAIN=1 I _ GAIN=0 D _ GAIN=0.25 OV _ GAIN=0</pre>

D_ZONE_MAX

Type:	Axis Parameter
Description:	<p>This sets works in conjunction with D_ZONE_MIN to clamp the DAC output to zero when the demand movement is complete and the magnitude of the following error is less than the D_ZONE_MIN value. The servo loop will be reactivated when either the following error rises above the D_ZONE_MAX value, or a fresh movement is started.</p> <p> This can be used to prevent oscillations at static positions in Piezo systems.</p>
Parameters:	<p>Value: Above this value the servo loop is reactivated when clamped in the dead band.</p>

Example: The **DAC** output will be clamped at zero when the movement is complete and the following error falls below 3. When a movement is restarted or if the following error rises above a value of 10, the servo loop will be reactivated

```
D_ZONE_MIN = 3
D_ZONE_MAX = 10
```

See Also: D_ZONE_MIN

D_ZONE_MIN

Type: Axis Parameter

Description: Working in conjunction with **D_ZONE_MAX**, **D_ZONE_MIN** defines a **DAC** dead band. This clamps the **DAC** output to zero when the demand movement is complete and the magnitude of the following error is less than the **D_ZONE_MIN** value. The servo loop will be reactivated when either the following error rises above the **D_ZONE_MAX** value, or a fresh movement is started.



This can be used to prevent oscillations at static positions in Piezo systems.

Parameters: **Value:** When the axis is **IDLE** and the magnitude of the following error is less than this value the **DAC** is clamped to zero.

Example: The **DAC** output will be clamped at zero when the movement is complete and the following error falls below 3. When a movement is restarted or if the following error rises above a value of 10, the servo loop will be reactivated

```
D_ZONE_MIN = 3
D_ZONE_MAX = 10
```

See Also: D_ZONE_MAX

DAC

Type: Axis Parameter

Description: Writing to this parameter when `SERVO = OFF` allows the user to force a demand value for that axis. On an analogue axis this will set a voltage on the output. On a digital axis this will be the demand value.



When using a FlexAxis as a stepper or encoder output the voltage outputs are available for user control.

Parameters: `value`: The demand value for the axis.

For a 12 bit DAC on an analogue axis:

`DAC=-2048` corresponds to a voltage of 10V

`DAC=2047` corresponds to a voltage of -10v

For a 16 bit DAC on an analogue axis:

`DAC=32767` corresponds to a voltage of 10V

`DAC=-32768` corresponds to a voltage of -10V

For digital axes check your drive specification for suitable values.

See `DAC _ SCALE` for a list of DAC types.

Example: To force a square wave of amplitude +/-5V and period of approximately 500ms on axis 0.

```
WDOG=ON
SERVO AXIS(0)=OFF
square:
    DAC AXIS(0)=1024
    WA(250)
    DAC AXIS(0)=-1024
    WA(250)
GOTO square
```

See Also: `DAC _ OUT`, `DAC _ SCALE`, `SERVO`

DAC_OUT

Type: Axis Parameter (Read Only)

Description: `DAC _ OUT` reads the demand value for the axis.

In an analogue system this will be the value sent to the voltage output (the `DAC`). If `SERVO = ON` this is the output of the closed loop algorithm. If `SERVO = OFF` it is the value set by the user in `DAC`.

In a digital system it returns the demand value for the axis which could be the actual position, speed or torque depending on the axis `ATYPE`.

Parameters: demand value for the axis.

Example: To check that the controller has set the correct voltage for axis 8 on an analogue system read `DAC _ OUT` in the command line.

```
>>PRINT DAC _ OUT AXIS(8)
288.0000
>>
```

See Also: `DAC`, `DAC _ SCALE`, `ATYPE`

DAC_SCALE

Type: Axis Parameter

Description: `DAC _ SCALE` is an integer that is multiplied to the output of the closed loop algorithm. You can use it to reverse the polarity of the demand value or to scale it so to effectively reduce the resolution of the closed loop algorithm.



As it is applied to the output of the closed loop algorithm it is not applied to position based axis.

Parameters: **Value:** Can be a positive or negative integer.
 EtherCAT default = 1
 SERCOS default = 1
 FlexAxis default = 16
 Panasonic default = 1
 SLM default = 16



To obtain the highest possible resolution of your system `DAC _ SCALE` should be set to 1 or -1.

Example: The FlexAxis uses a 16bit `DAC`, to make it compatible with the gain settings used on older 12 bit `DACS` `DAC _ SCALE` is set to 16.
 The max output from closed loop algorithm is 2048 (for a 12bit system)
 The max output from a 16bit `DAC` is 32768 which is 2048 multiplied by 16

See Also: `DAC`, `DAC _ OUT`

DATUM_IN

Type: Axis Parameter

Alternate Format: `DAT _ IN`

Description: This parameter holds a digital input channel to be used as a datum input.



The input used for `DATUM _ IN` is active low.

Parameters: **Value:** - 1 = disable the input as `DATUM _ IN` (default)/
 0-63 = Input to use as datum input/



Any type of input can be used, built in, Trio `CAN I/O`, `CANopen` or virtual.

Example: Set input 28 as the `DATUM` input for axis 0 then perform a homing routine
`DATUM _ IN AXIS(0)=28`

DATUM(3)**See Also:** **DATUM**

DECEL

Type: Axis Parameter**Syntax:** **DECEL=value****Description:** The **DECEL** axis parameter may be used to set or read back the deceleration rate of each axis fitted.**Parameters:** **value:** The deceleration rate in **UNITS/sec/sec**. Must be a positive value.**Example:** Set the deceleration parameter and print it to the user.

```
DECEL=100' Set deceleration rate
PRINT " Decel is ";DECEL;" mm/sec/sec"
```

See Also: **ACCEL**

DECEL_ANGLE

Type: Axis Parameter**Description:** This parameter is used with **CORNER_MODE**, it defines the maximum change in direction of a 2 axis interpolated move that will be merged at full speed. When the change in direction is greater than this angle the speed will be proportionally reduced so that:
$$VP_SPEED=FORCE_SPEED * (angle - DECEL_ANGLE) / (STOP_ANGLE - DECEL_ANGLE)$$

Where angle is the change in direction of the moves.

Parameters: **Value:** The angle to start to reduce the speed, in radians.

Example1: Decelerate to a slower speed when the transition is between 15 and 45 degrees.

```
CORNER _ MODE=2
DECEL _ ANGLE = 15 * (PI/180)
STOP _ ANGLE = 45 * (PI/180)
```

See Also: CORNER _ MODE, STOP _ ANGLE

DEMAND_EDGES

Type: Axis Parameter (Read Only)

Description: Allows the user to read back the current DPOS in encoder edges.

You can use DEMAND _ EDGES to check that your UNITS or ENCODER _ RATIO values are set correctly.

Parameters: **Value:** demand position in encoder edges.

Example: Print the DEMAND _ EDGES in the command line

```
>>PRINT DEMAND _ EDGES AXIS(4)
523
>>
```


DEMAND_SPEED

Type: Axis Parameter (Read Only)

Description: Returns the speed output of the νPU in edges or counts per servo period. Normally used for low level debug of the motion system.

Parameters: **Value:** νPU speed output in user units per servo period.

Example: Check the νPU speed output using the command line

```
>>?DEMAND _ SPEED  
5.0000  
>>
```

DPOS

Type: Axis Parameter (Read Only)

Description: The demand position **DPOS** is the demanded axis position generated by the motion commands.

DPOS is set to **MPOS** when **SERVO** or **WDOG** are **OFF**.

DPOS can be adjusted without any motion by using **DEFPOS** or **OFFPOS**.

A step change in **DPOS** can be written using **ENDMOVE**.

Parameters: **Value:** Demand position in user units. Default 0 on power up.

Example: Return the demand position for axis 10 in user units.


```
>>? DPOS AXIS(10)  
5432  
>>
```

See Also: **DEFPOS**, **ENDMOVE**, **OFFPOS**, **TRANS _ DPOS**

ENCODER

Type:	Axis Parameter (Read Only)
Description:	The ENCODER axis parameter holds a raw copy of the positional feedback device. The MPOS axis measured position is calculated from the ENCODER value automatically allowing for overflows and offsets.
Parameters:	<p>Incremental encoder: The value latched in the encoder hardware register.</p> <p>Absolute Encoder: The positional value using the number of bits set in ENCODER _ BITS.</p> <p>Digital Axis: Raw position feedback from the drive).</p>
See Also:	ENCODER _ BITS , MPOS

ENCODER _ BITS

Type:	Axis Parameter
Description:	<p>This parameter is only used with an absolute encoder axis. It is used to set the number of data bits to be clocked out of the encoder by the axis hardware. There are 2 types of absolute encoder supported by this parameter; SSI and EnDat.</p> <p> <i>If the number of ENCODER _ BITS is to be changed, the parameter must first be set to zero before entering the new value.</i></p>
Parameters:	<p>off: 0, No data is clocked out of the encoder (default).</p> <p>SSI: Bit 0-5 are the number of bits to be clocked out of the encoder. Range 0-25. Bit 6 set for Binary, clear for Gray code (default).</p> <p>EnDat: Bits 0..7 of the parameter are the total number of encoder bits and bits 8..14 are the number of multi-turn bits.</p>
Example 1:	<p>set up 2 axes of SSI absolute encoder.</p> <pre>ENCODER _ BITS AXIS(3) = 12 ENCODER _ BITS AXIS(7) = 21</pre>

Example 2: re-initialise MPOS using absolute value from encoder.

```
SERVO=OFF
ENCODER _ BITS = 0
ENCODER _ BITS = databits
```

Example 3: A 25 bit EnDat encoder has 12 multi-turn and 13 bits/turn resolution. The total number of bits is 25.

```
ENCODER _ BITS = 25 + (256 * 12)
```

ENCODER_CONTROL

Type: Axis Parameter

Description: Endat encoders can be set to either cyclically return their position, or they can be set to a parameter read/write mode.



Using the ENCODER _ READ or ENCODER _ WRITE functions will set the parameter to 1 automatically.

Parameters: **Value:** 0 = position return mode (default value).

SSI: 1 = sets parameter read/write mode.

Example 1: Reset ENCODER _ CONTROL after an ENCODER _ READ so that the position is returned.

```
value = ENCODER _ READ($A700)
ENCODER _ CONTROL = 0
```

See Also: ENDCODER _ READ, ENDCODER _ WRITE

ENCODER_FILTER

Type: Axis Parameter

Description: This parameter allows filtering to be applied to an encoder feedback to reduce the impact of jitter. The smaller the value the larger the time constant and so the less impact jitter will have on the system.



This parameter can be used to reduce jitter on a master axis which is linked to another axis.

Parameters: **Value:** Filter parameter range 0.001 to 1 (default 1).

Example: Apply a filter to a line encoder so that the connected axis are not affected by any jitter:

```
BASE(0)
ENCODER_FILTER= 0.95
BASE(1)
CONNECT(1,0)
```

ENCODER_ID

Type: Axis Parameter

Description: This parameter returns the **ENID** parameter from the encoder (fixed at 17 decimal). (Tamagawa absolute encoder only)

Parameters: **Value:** Only encoders returning 17 are currently supported.

Example: Initialise a Tamagawa absolute encoder and check it is working by looking at **ENCODER_ID**.

```
ATYPE = 46
IF ENCODER_ID<>17 THEN
  PRINT#term, "Incorrect ENID"
ENDIF
```

ENCODER_READ

Type: Axis Function

Syntax: `value = ENCODER_READ (address)`

Description: Read an internal register from an EnDat absolute encoder.

Parameters: **Value:** Value returned from the specified register. Returns -1 if the encoder has not been initialised.

address: The address of the EnDat encoder register to be read.

Example: Initialise and check an EnDat encoder.

```
ENCODER_BITS=25+256*12
ATYPE=47
IF ENCODER_READ($A700)=-1 then
    PRINT "Failed to initialise EnDat Encoder"
ENDIF
ENCODER_CONTROL=0
```

See Also: `ENCODER_CONTROL`, `ENCODER_WRITE`

ENCODER_STATUS

Type: Axis Parameter

Syntax: `ENCODER_STATUS`

Description: This axis parameter returns both the status field `SF` and the `ALMC` encoder error field from a Tamagawa absolute encoder.

Parameters: **Value:** Bits 0..7 are the `SF` field and 8..15 are the `ALMC` field. Returns 0 if the encoder has not been initialised.

Example: Print the `SF` field and `ALMC` field in hex.

```
PRINT "SF field = 0x"; HEX (ENCODER_STATUS AND $FF)
```

```
PRINT "ALMC field = 0x"; HEX ((ENCODER _ STATUS AND $FF00)/$FF)
```

ENCODER_TURNS

Type: Axis Parameter

Description: Returns the number of multi-turn counts from Endat or Tamagawa absolute encoders.



The multi-turn data is not automatically applied to the axis MPOS after initialisation of a Tamagawa absolute encoder. The application programmer must apply this from BASIC using OFFPOS or DEFPOS as required.

Parameters: **value:** The number of multi-turn counts from the encoder.

Example: Initialise a Tamagawa encoder and apply the number of turns to MPOS. The encoder returns 17bits for the position and 16bits for the number of turns.

```
ATYPE=46
OFFPOS= ENCODER _ TURNS*2^17
WAIT UNTIL OFFPOS = 0
```

END_DIR_LAST

Type: Axis Parameter

Description: Returns the direction of the end of the last loaded interpolated motion command. You can use the parameter to set an initial direction before loading a SP command. **END_DIR_LAST** will be the same as **START_DIR_LAST** except in the case of circular moves.



*Write to **END_DIR_LAST** when initialising a system or after a sequence of moves which are not SP commands.*



*This parameter is only available when using SP motion commands such as **MOVESP**, **MOVEABSSP** etc.*

Parameters: **Value:** End direction, in radians between $-\pi$ and π . Value is always positive.

Example 1: Return the end direction of a move.

```
>>MOVESP(10000,-10000)
>>PRINT END _ DIR _ LAST
2.3562
>>
```

Example 2: Write to the end direction to set the direction of the **MOVE** before calculating the change.

```
MOVE(10000,-10000)
END _ DIR _ LAST = 2.3562
MOVESP(10000,1324)
VR(10)=CHANGE _ DIR _ LAST
```

See Also: **CHANGE _ DIR _ LAST, START _ DIR _ LAST**

ENDMOVE

Type: Axis Parameter

Description: This parameter holds the absolute position of the end of the current move in user units. It is normally only read back although may be written to if required provided that **SERVO=ON** and no move is in progress.



WRITING TO DPOS WILL MAKE A STEP CHANGE. THIS CAN EASILY LEAD TO "FOLLOWING ERROR EXCEEDS LIMIT" ERRORS UNLESS THE STEPS ARE SMALL OR THE FE _ LIMIT IS HIGH.



*As it is an absolute value **ENDMOVE** is adjusted by **OFFPOS/DEFPOS**. The individual moves in the buffer are incremental and are not adjusted by **OFFPOS**.*

Parameters: **Value:** The absolute position of the end of the current move in user units.

Example: Check the value of **ENDMOVE** to confirm you calculated move is correct.

```
MOVE(distance*pitch)
```

```

IF ENDMOVE>200 THEN
  CANCEL
  PRINT#5, "Calculated distance to large"
ENDIF

```

ENDMOVE_BUFFER

Type: Axis Parameter (Read only)

Description: This holds the absolute position of end of the buffered sequence of moves.



As it is an absolute value ENDMOVE_BUFFER is adjusted by OFFPOS/DEFPOS. The individual moves in the buffer are incremental are not adjusted by OFFPOS.

Parameters: **Value:** Returns the length of all remaining moves for an axis.

Example: Add some moves to the buffer, then check the value of ENDMOVE_BUFFER.

```

>>MOVE(100)
>>MOVE(150)
>>MOVE(25)
>>PRINT ENDMOVE_BUFFER
275.000
>>

```

ENDMOVE_SPEED

Type: Axis Parameter

Description: This parameter sets the end speed for a motion command that support the advanced speed control (commands ending in SP). The VP_SPEED will decelerate until ENDMOVE_SPEED is reached at the end of the profile.



The lowest value of ENDMOVE_SPEED, FORCE_SPEED or STARTMOVE_SPEED will take priority.

Parameters: **Value:** The speed at which the **SP** motion command will end, in user **UNITS**. (default 0).

ENDMOVE _ SPEED is loaded into the buffer at the same time as the move so you can set different speeds for subsequent moves. If there is no further motion commands in the buffer the current move will decelerate to a stop.

Example 1: In this example the controller will start ramping down the speed (at the specified rate of **DECEL**) so at the end of the **MOVESP(20)** the **VPSPEED=10**. The next move continues with a **FORCE _ SPEED** of 10. The final **ENDMOVE _ SPEED** is overwritten to zero as there are no more buffered moves.

```
FORCE _ SPEED=15
ENDMOVE _ SPEED=10
MOVESP(20)
FORCE _ SPEED=10
ENDMOVE _ SPEED=5
MOVESP(5)
```

Example 2: A machine can merge interpolated moves however it must slow down to 50% of the speed for the transition.

```
FORCE _ SPEED=1000
ENDMOVE _ SPEED=500 '50% of FORCE _ SPEED
MOVE(100,10)
MOVE(70,-10)
MOVE(120,15)
```

ERRORMASK

Type: Axis Parameter

Description: The value held in this parameter is bitwise ANDed with the **AXISSTATUS** parameter by every axis on every servo cycle to determine if a runtime error should switch off the enable (**WDOG**) relay. If the result of the **AND** operation is not zero the enable relay is switched **OFF**.



*After a critical error has tripped the enable relay, the Motion Coordinator must either be reset, or a **DATUM(0)** command must be executed to reset the error flags.*

Parameters: **Value:** The mask to be **AND** ed with the **AXISSTATUS**.



For the MC464, the default value is 268 which will trap critical errors with digital drive communications as well as exceeding the following error limit.

Example: Configure the **ERRORMASK** so that the **WDOG** is turned off when there are communication failures (4), remote drive errors (8), the following error exceeds the limit (256) or the limit switches have been hit(16 + 32).

ERRORMASK= 4+8+16+32+256

See Also: **AXISSTATUS**, **DATUM(0)**

FAST_JOG

Type: Axis Parameter

Description: This parameter holds the input number to be used as the fast jog input. If the **FAST_JOG** is active then the jog inputs use the axis **SPEED** for the jog functions, otherwise the **JOGSPEED** will be used.



*The input used for **FAST_JOG** is active low.*

Parameters: **Value:** -1 = disable the input as **FAST_JOG** (default).
0-63 = Input to use as datum input.



*Any type of input can be used, built in, Trio **CAN I/O**, **CANopen** or virtual.*

Example: Configure input 12 and 13 as jog inputs

FWD_JOG = 12
FAST_JOG = 13
JOGSPEED = 200

See Also: **FWD_JOG**, **JOGSPEED**, **REV_JOG**

FASTDEC

Type: Axis Parameter

Description: The **FASTDEC** axis parameter may be used to set or read back the fast deceleration rate of each axis fitted. Fast deceleration is used when a **CANCEL** is issued, for example; from the user, a program, or from a software or hardware limit. If the motion finishes normally or **FASTDEC** = 0 then the **DECEL** value is used.

Parameters: **Value:** The deceleration rate in **UNITS/sec/sec**. Must be a positive value.

Example:

```
DECEL=100                `set normal deceleration rate
FASTDEC=1000            `set fast deceleration rate
MOVEABS(10000)          `start a move
WAIT UNTIL MPOS= 5000  `wait until the move is half finished
CANCEL                  `stop move at fast deceleration rate
```

See Also: **DECEL**

FE

Type: Axis Parameter (Read Only)

Description: This parameter returns the position error, which is equal to the demand position (**DPOS**)-measured position (**MPOS**).

Parameters: **Value:** The following error returned in user units.

Example: Wait for the position error to be below a value for 5 servo periods then pulse an output.

```
MOVEABS(200)
WAIT IDLE
FOR x=0 to 4
    WAIT UNTIL FE<5
NEXT x
OP(5,ON)
WA(2)
OP(5,OFF)
```

See Also: `FE _ LATCH`, `FE _ LIMIT`, `FE _ RANGE`

FE_LATCH

Type: Axis Parameter (Read Only)

Description: Contains the `FE` value which caused the axis to put the controller into `MOTION _ ERROR`. This value is only set when the `FE` exceeds the `FE _ LIMIT` and the `SERVO = OFF`.

Parameters: **Value:** Returns the `FE` value that caused a `MOTION _ ERROR`.



`FE _ LATCH` is reset to 0 when the axis `SERVO = ON`.

Example: Read the `LE _ LATCH` when there is a `MOTION _ ERROR`.

```
IF MOTION _ ERROR THEN
  VR(10) = FE _ LATCH AXIS (ERROR _ AXIS)
ENDIF
```

See Also: `FE`, `FE _ LIMIT`

FE_LIMIT

Type: Axis Parameter

Alternate Format: FELIMIT

Syntax: FE_LIMIT = value

Description: This is the maximum allowable following error. When exceeded the controller will generate an **AXISSTATUS** error, by default this will also generate a **MOTIONERROR**. The **MOTIONERROR** will disable the **WDOG** relay thus stopping further motor operation.



This limit may be used to guard against fault conditions such as mechanical lock-up, loss of encoder feedback, etc.

Parameters: **Value:** The maximum allowable following error in user units. The default value is 2000 encoder edges.

Example: Initialise the axis as part of a **STARTUP** routine

```
FOR x = 0 to 4
  BASE(x)
  UNITS = 100
  FE_LIMIT = 10
  SPEED = 100
  ACCEL=1000
  DECEL=ACCEL
NEXT x
```

See Also: FE, FE_LATCH

FE_LIMIT_MODE

Type: Axis Parameter

Description: This parameter determines if an **AXISSTATUS** error is produced immediately when the **FE** exceeds the **FE_LIMIT** or if it exceeds for 2 consecutive servo periods. This means that if **FE_LIMIT** is exceeded for one servo period only, it will be ignored.



THIS WILL INCREASE THE TIME TO DISABLE YOUR DRIVES IN AN ERROR. YOU SHOULD ONLY CHANGE FROM THE DEFAULT VALUES UNDER ADVICE FROM TRIO OR YOUR DISTRIBUTOR.

Parameters: **Value:** 0 = **AXISSTATUS** error generated immediately (default).
1 = **AXISSTATUS** error generated when **FE_LIMIT** is exceeded for 2 consecutive servo periods.

See Also: **FE**, **FE_LIMIT**

FE_RANGE

Type: Axis Parameter

Description: Following error report range. When the **FE** exceeds this value the axis has bit 1 in the **AXISSTATUS** axis parameter set.

Parameters: **value:** The value in user **UNITS** above which bit 1 is set in **AXISSTATUS**.

Example: Using **FE_RANGE** to slow a machine down when the **FE** is too large.

```
'initialise the axis
FE_RANGE = 10
FE_LIMIT = 15
SPEED=100
```

```
...  
`loop to check if FE _ RANGE has been exceeded  
WHILE NOT IDLE  
VR(10) = AXISSTATUS  
IF READBIT(1, 10) THEN  
  `slow down by 1%  
  SPEED = SPEED * 0.99  
ENDIF  
WEND  
SPEED = 100
```

See Also: FE, FE _ LIMIT

FHOLD_IN

Type: Axis Parameter

Alternate Format: FH _ IN

Syntax: FHOLD _ IN=value

Description: This parameter holds the input number to be used as a feedhold input.

When the feedhold input is active motion on the specified axis has its speed overridden to the feedhold speed (**FHSPEED**) without canceling the move in progress. The change in speed uses **ACCEL** and **DECEL**. When the input is reset any move in progress when the input was set will go back to the programmed speed.



Set FHSPEED to zero to pause the motion on that axis.

Moves which are not speed controlled e.g. **CONNECT**, **CAMBOX**, **MOVELINK** are not affected.



The input used for FHOLD _ IN is active low.

Parameters: Value: 1 = disable the input as feedhold (default).
0-63 = Input to use as feedhold.



Any type of input can be used, built in, Trio CAN I/O, CANopen or virtual.

Example: Configure inputs 21 as feedhold inputs for axis 2. The default **FHSPEED** = 0 so the motion can be paused using the feedhold input.

```
BASE(2)
FHOLD _ IN = 21
```

See Also: **FHSPEED**

FHSPEED

Type: Axis Parameter

Description: When the feedhold input is active motion is ramped down to **FHSPEED**.

Parameters: **Value:** The speed in user units to use when the **FHOLD _ IN** is active (default 0).

Example: Set **FHSPEED** to a value so that a slower speed is selected when the **FHOLD _ IN** is active.

```
BASE(3)
SPEED=1000
FHSPEED=SPEED*0.1
```

See Also: **FHOLD _ IN**

FORCE_SPEED

Type: Axis Parameter

Description: This parameter sets the main speed for a motion command that supports the advanced speed control (commands ending in **SP**). The **VP_SPEED** will accelerate or decelerate so that the profile is completed at **FORCE_SPEED**



*The lowest value of **SPEED**, **ENDMOVE_SPEED**, **FORCE_SPEED** or **STARTMOVE_SPEED** will take priority.*

FORCE_SPEED is loaded into the buffer at the same time as the move so you can set different speeds for subsequent moves.

Parameters: **Value:** The speed at which the **SP** motion command will execute, in user **UNITS**. (default 0).

Example 1: In this example the controller will ramp the speed down to a speed of 10 at the end of the **MOVE**. Then for the duration of the **MOVESP(20)** the speed will be 10, after which it will ramp back to a speed of 15.

```
SPEED = 15
MOVE(100)
FORCE_SPEED = 10
MOVESP(20)
MOVE(100)
```

Example 2: Use **FORCE_SPEED** to slow the profile speed down during a corner move.

```
FORCE_SPEED=100
MOVESP(100,0)
FORCE_SPEED=50
MOVECIRC(100,100,100,0,1)
FORCE_SPEED=100
MOVESP(0,100)
```

See Also: **ENDMOVE_SPEED**, **STARTMOVE_SPEED**

FS_LIMIT

Type: Axis Parameter

Alternate Format: `FSLIMIT`

Description: An end of travel limit may be set up in software thus allowing the program control of the working envelope of the machine. This parameter holds the absolute position of the forward travel limit in user units.

Bit 9 of the `AXISSTATUS` register is set when the axis position is greater than the `FS_LIMIT`.



When DPOS reaches `FS_LIMIT` the controller will cancel the move, so the axis will decelerate at `DECEL` or `FAST_DEC`.



`FS_LIMIT` is disabled when it has a value greater than `REP_DIST`.

Parameters: **Value:** The absolute position of the software forward travel limit in user units. (default = 200,000,000,000).

Example 1: Datum axis 1, then define a forward limit from this point.

```
BASE(1)
DATUM(3)
WAIT IDLE
FS_LIMIT=200
```

Example 2: Disable the `FS_LIMIT` by setting it greater than repdist.

```
FS_LIMIT = REPDIST+10
```

See Also: `RS_LIMIT`, `FWD_IN`, `REV_IN`

FULL_SP_RADIUS

Type: Controller Parameter

Description: This parameter is used with `CORNER_MODE`, it defines the minimum radius that will be executed at full speed. When a radius is smaller than `FULL_SP_RADIUS` the speed will be proportionally reduces so that:

$$VP_SPEED = FORCE_SPEED * radius / FULL_SP_RADIUS$$

Where radius is the radius of the corner that is executing.

Parameters: **Value:** The full speed radius in user UNITS (default = 0).

Example: In the following program, when the first `MOVECIRCSP` is reached the speed remains at 10 because the radius (8) is greater than that set in `FULL_SP_RADIUS`. For the second `MOVECIRCSP` the speed is reduced by 50% to a value of 5, because the radius is 50% of that stored in `FULL_SP_RADIUS`.

```
CORNER_MODE=8
MERGE=ON
SPEED=10
FULL_SP_RADIUS=6
DEFPOS(0,0)

MOVESP(10,10)
MOVESP(10,5)
MOVESP(5,5)
MOVECIRCSP(8,8,0,8,1)
MOVECIRCSP(3,3,0,3,1)
MOVESP(5,5)
MOVESP(10,5)
```

See Also: `CORNER_MODE`

FWD_IN

Type: Axis Parameter

Description: This parameter holds the input number to be used as a forward limit input. When the forward limit input is active any motion on that axis is CANCELED.



When FWD _ IN is active AXISSTATUS bit 4 is set.

Parameters: **Value:** 1 = disable the input as FWD _ IN (default).
0-63 = Input to use as forward input switch.



Any type of input can be used, built in, TrioCAN I/O, CANopen or virtual.

Example: Initialise input 19 for the forward limit switch.

`FWD _ IN AXIS(9)=19`

See Also: REV _ IN, FS _ LIMIT, RS _ LIMIT

FWD_JOG

Type: Axis Parameter

Description: This parameter holds the input number to be used as a jog forward input. When the FWD _ JOG input is active the axis moves forward at JOG _ SPEED.

Example: `FWD _ JOG=7`



*The input used for FWD _ IN is active low.
It is advisable to use INVERT _ IN on the input for FWD _ JOG so that 0V at the input disables the jog.
FWD _ JOG overrides REV _ JOG if both are active.*

Parameters: **Value:** 1 = disable the input as `FWD _ JOG` (default).
 0-63 = Input to use as datum input.

Example: Initialise the `FWD _ JOG` so that it is active high on input 7.

```
INVERT _ IN(7,ON)  
FWD _ JOG=7
```

I_GAIN

Type: Axis Parameter

Description: Used as part of the closed loop control, adding integral gain to a system reduces position error when at rest or moving steadily. It will produce or increase overshoot and may lead to oscillation.

For an integral gain K_i and a sum of position errors $\int \epsilon$, the contribution to the output signal is:

$$\Phi_i = K_i \times \int \epsilon$$

Parameters: **Value:** The integral gain is a constant which is multiplied by the sum of following errors. Default value = 0.

Example: Setting the gain values as part of a `STARTUP` program.

```
P _ GAIN=1
```

INVERT_STEP

Type: Axis Parameter

Description: `INVERT _ STEP` is used to switch a hardware inverter into the stepper pulse output circuit. This can be necessary for connecting to some stepper drives. The electronic logic inside the *Motion Coordinator* stepper pulse generation assumes that the **FALLING** edge of the step output is the active edge which results in motor movement. This is suitable for the majority of stepper drives.



`INVERT _ STEP` should be set with `WDOG=OFF`.



IF THE SETTING IS INCORRECT, A STEPPER MOTOR MAY LOSE POSITION BY ONE STEP WHEN CHANGING DIRECTION.

Parameters: **Value:** **ON** = RISING edge of the step signal the active edge.

Example: Set **INVERT** step for axis 2 as part of a startup routine.

```
BASE(2)
INVERT _ STEP = ON
```

JOGSPEED

Type: Axis Parameter

Description: Sets the jog speed in user units for an axis to run at when performing a jog.



*You can set a faster jog speed using **SPEED** and the **FAST _ JOG** input.*

Parameters: **Value:** The speed in user units/ second which an axis will use when being jogged.

Example: Configure an input to be the jog input at 20mm/sec on axis 12.

```
BASE(12)
SPEED=3000
FWD _ JOG = 12
JOGSPEED = 20
```

See Also: **FAST _ JOG, FWD _ JOG, REV _ JOG**

LIMIT_BUFFERED

Type: System Parameter

Description: This sets the maximum number of move buffers available in the controller.



*You can increase the machine speed when using **MERGE** or **CORNER _ MODE** by increasing the number of buffers.*

Parameters: **Value:** 1..64 = The number of move buffers (default = 1).

Example: Configure axis 3 to have 10 move buffers so a large sequence of small moves can be merged together.

```
LIMIT _ BUFFERED AXIS(3) = 10
```

LINK_AXIS

Type: Axis Parameter (Read Only)

Alternate format: **LINKAX**

Description: Returns the axis number that the axis is linked to during any linked moves.



*Linked moves are where the demand position is a function of another axis. E.G. **CONNECT**, **CAMBOX**, **MOVELINK**.*

Parameters: **Value:** -1 = Axis is not linked.
number = Axis number the **BASE** axis is linked to.

Example: **CONNECT** an axis, then check that it is linked.

```
>>BASE(0)
>>CONNECT(12,4)
>>PRINT LINK _ AXIS
4.0000
>>
```

LOADED

Type: Axis Parameter

Description: Checks to see if a move is being loaded into the **MTYPE** buffer.



ALTHOUGH IT IS POSSIBLE TO USE LOADED AS PART OF ANY EXPRESSION IT IS ADVISABLE TO ONLY USE IT WITH A WAIT. THIS IS BECAUSE THE IF LOOP MAY MISS A TRUE VALUE WHILE A MOVE IS BEING LOADED.

Parameters: **Value:** **TRUE** = when there are no buffered moves or when a move is being loaded into the **MTYPE**.
FALSE = when the loading of a move is complete and there are buffered moves.

Example: Continue to load a sequence of moves when the **NTYPE** buffer is free.

```

WHILE machine_on =TRUE
  WAIT UNTIL LOADED or machine_off=FALSE
  IF machine_on=TRUE THEN
    MOVE(TABLE(position)
    position=position+1
  ENDIF
WEND

```

MARK

Type: Axis Parameter (Read Only)

Description: This parameter can be polled to determine if the registration event has occurred. **MARK** is reset when **REGIST** is executed.

Parameters: **Value:** **TRUE** = The registration event has occurred (default).
FALSE = The registration event has not occurred.



When TRUE the REG _ POS is valid.

Example: Apply an offset to the position of the axis depending on the registration position.

```
loop:
  WAIT UNTIL IN(punch _ clr)=ON
  MOVE(index _ length)
  REGIST(3)           `rising edge of R
  WAIT UNTIL MARK
  MOVEMODIFY(REG _ POS + offset)
  WAIT IDLE
GOTO loop
```

See Also: REGIST, REG _ POS

MARKB

Type: Axis Parameter (Read Only)

Description: This parameter can be polled to determine if the registration event has occurred on the second registration channel.

Parameters: **Value:** TRUE = The registration event has occurred (default).
FALSE = The registration event has not occurred.



When TRUE the REG _ POS is valid.

See Also REGIST, REG _ POSB

MERGE

Type: Axis Parameter

Description: Velocity profiled moves can be **MERGED** together so that the speed will not ramp down to zero between the current move and the buffered move.



IT IS UP TO THE PROGRAMMER TO ENSURE THAT THE MERGING IS SENSIBLE. FOR EXAMPLE MERGING A FORWARD MOVE WITH A REVERSE MOVE WILL CAUSE AN ATTEMPTED INSTANTANEOUS CHANGE OF DIRECTION.

MERGE will only function if:

- The next move is loaded into the buffer.
- The axis group does not change on multi-axis moves.
- Velocity profiled moves (**MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL**, **REVERSE**, **FORWARD**) cannot be merged with linked moves (**CONNECT**, **MOVELINK**, **CAMBOX**)



*When merging multi-axis moves only the base axis **MERGE** flag needs to be set.*



*If you are merging short moves you may need to increase the number of buffered moves by increasing **LIMIT _ BUFFERED**.*

Parameters: **Value:** **ON** = motion commands are merged.
OFF = motion commands decelerate to zero speed.

Example: Turn on **MERGE** before a sequence of moves, then disable at the end.

```

BASE(0,1) `set base array
MERGE=ON `set MERGE state
MOVEABS(0,50) `run a sequence of moves
MOVE(0,100)
MOVECIRC(50,50,50,0,1)
MOVE(100,0)
MOVECIRC(50,-50,0,-50,1)
MOVE(0,-100)
MOVECIRC(-50,-50,-50,0,1)
MOVE(-100,0)
MOVECIRC(-50,50,0,50,1)
WAIT IDLE
MERGE=OFF

```

MOVES_BUFFERED

Type: Axis Parameter (Read only)

Description: This returns the number of moves being buffered by the axis.



*The value does not include the move in the **MTYPE** buffer.*

Parameters: **Value:** Number of commands in the move buffers.

Example: Check if there is room in the move buffer before adding in another command.

```
IF MOVES_BUFFERED < 64 THEN
  xpos = TABLE(count+x)
  ypos = TABLE(count+y)
  MOVEABS(xpos, ypos)
  count=count + 1
ENDIF
```

MPOS

Type: Axis Parameter (Read Only)

Description: This parameter is the position of the axis as measured by the encoder or resolver.



*Unless using an absolute encoder **MPOS** is reset to 0 on power up or software reset.*

The value is adjusted using the **DEFPOS()** command or **OFFPOS** axis parameter to shift the datum position or when the **REP_DIST** is in operation. The position is reported in user units.

Parameters: **Value:** actual axis position in user units.

Example:

```
WAIT UNTIL MPOS>=1250
SPEED=2.5
```

MSPEED

Type: Axis Parameter (Read Only)

Description: **MSPEED** can be used to represent the speed measured as it represents the change in measured position in user units (per second) in the last servo period.



This value represents a snapshot of the speed and significant fluctuations can occur, particularly at low speeds. It can be worthwhile to average several readings if a stable value is required at low speeds.

Parameters: **Value:** Change in measured position per second in user units.

Example: Average **MSPEED** using a filter algorithm.

```
` VR(10) filter output
c = 0.005 `filter coefficient (0<c<1)
VR(10)=MSPEED `initialise filter output to MSPEED

WHILE TRUE
  WA(1)
  VR(10)=(1-c)*VR(10)+c*MSPEED
WEND
```

MTYPE

Type: Axis Parameter (Read Only)

Description: This parameter holds the type of move currently being executed.

This parameter may be interrogated to determine whether a move has finished or if a transition from one move type to another has taken place.



A non-idle move type does not necessarily mean that the axis is actually moving. It may be at zero speed part way along a move or interpolating with another axis without moving itself.

*It takes a servo period before a motion command is loaded into the buffer, so checking **MTYPE** immediately after a motion command will probably fail. You should use **WAIT LOADED** or **WAIT IDLE** to check that a command is loaded or complete*

Parameters:	Value:	Motion command in progress
	0	idle (No move)
	1	MOVE
	2	MOVEABS
	3	MHELICAL
	4	MOVECIRC
	5	MOVEMODIFY
	6	MOVESP
	7	MOVEABSSP
	8	MOVECIRCSP
	9	MHELICALSP
	10	FORWARD
	11	REVERSE
	12	DATUM
	13	CAM
	14	FWD _ JOG
	15	REV _ JOG
	20	CAMBOX
	21	CONNECT
	22	MOVELINK
	23	CONNPATH
	24	FLEXLINK
	30	MOVETANG
	31	MSPHERICAL

Example: Load another move if the existing move has finished.

```
IF MTYPE AXIS(2) = 0 THEN
  MOVE (TABLE(count)) AXIS(2)
  count = count + 1
ENDIF
```

NEG_OFFSET

Type: Axis Parameter

Description: For Piezo Motor Control. This sets an offset to the DAC output when the position loop is demanding a negative voltage output. `NEG_OFFSET` is applied after `DAC_SCALE` so is always a value appropriate to the D to A converter resolution. The negative offset must be a negative value.

Example: An offset of -0.1V is required on an axis with a 16 bit D to A converter. With a 16 bit DAC, -10V is commanded with the value -32768 so for -0.1V need $-32768 / 100$.

```
NEG_OFFSET = -328
```

`POS_OFFSET` and `NEG_OFFSET` are normally used together. It is suggested that the offset is 65% to 70% of the value required to make the stage move in an open loop situation.

```
POS_OFFSET = 450
NEG_OFFSET = -395
```

NTYPE

Type: Axis Parameter (Read Only)

Description: This parameter holds the type of the first buffered move.



The NTYPE buffer can be cleared using `CANCEL(1)`.

Parameters: **value:** The numerical value of the move type.



See `MTYPE` for a list of return values.

Example: If the first move buffer (`NTYPE`) is empty apply another move from a table

```
IF MTYPE = 0 THEN
  MOVE( TABLE(count)
  count = count +1
```

ENDIF**See Also:** **MTYPE**

OFFPOS

Type: Axis Parameter

Description: The **OFFPOS** parameter allows the axis position value to be offset by any amount without affecting the motion which is in progress. **OFFPOS** can therefore be used to effectively datum a system at full speed. Values loaded into the **OFFPOS** axis parameter are reset to 0 by the system software after the axis position is changed.

Parameters: **Value:** the distance to offset the current position.

Example 1: Change the current position by 125, using the command line terminal:

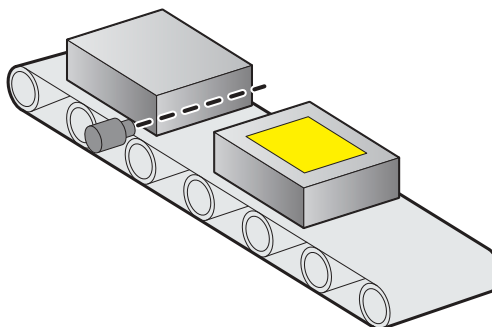
```
>>PRINT DPOS
300.0000
>>OFFPOS=125
>>PRINT DPOS
425.0000
>>
```

Example 2: Define the current demand position as zero:

```
OFFPOS=-DPOS 'This is equivalent to DEFPOS(0
```

Example 3: A conveyor is used to transport boxes onto which labels must be applied.

Using the **REGIST()** function, we can capture the position at which the leading edge of the box is seen, then by using **OFFPOS** we can adjust the measured position of the axis to be zero at that point. Therefore, after the registration event has occurred, the measured position (seen in **MPOS**) will actually reflect the absolute distance from the start of the box, the mechanism which



applies the label can take advantage of the absolute position start mode of the **MOVELINK** or **CAMBOX** commands to apply the label.

```
BASE(conv)
REGIST(3)
WAIT UNTIL MARK
OFFPOS = -REG _ POS ` Leading edge of box is now zero
```

OPEN_WIN

Type: Axis Parameter

Alternate Format: OW

Description: This parameter defines the first position of the window which will be used for registration marks if windowing is specified by the **REGIST()** command.

Parameters: **value:** Absolute position of the first registration window.

Example: Enable registration but only look for registration marks between 170 and 230mm.

```
OPEN _ WIN=170.00
CLOSE _ WIN=230.0
REGIST(256+3)
WAIT UNTIL MARK
```

See Also: **CLOSE _ WIN, REGIST**

OUTLIMIT

Type: Axis Parameter

Description: The output limit restricts the **DAC** output to a lower value than the maximum. This can be used to limit the analogue outputs or demand value to a digital drive. **OUTLIMIT** will always limit the **DAC** output if you are using a servo control or just manually setting **DAC**.



As it is applied to the output of the closed loop algorithm it is not applied to position based axis.

Parameters: **value:** The range that the **DAC** is limited to.



*The value required varies depending on whether the axis has a 12 bit or 16 bit **DAC**. If the voltage output is generated by a 12 bit **DAC** values an **OUTLIMIT** of 2047 will produce the full +/-10V range. If the voltage output is generated by a 16 bit **DAC** values an **OUTLIMIT** of 32767 will produce the full +/-10V range.*

Example: Limit a 12bit **DAC** to ±5V (±1023).

OUTLIMIT **AXIS**(0)=1023

OV_GAIN

Type: Axis Parameter

Description: The output velocity gain is a gain constant which is multiplied by the change in measured position. The result is summed with all the other gain terms and applied to the servo **DAC**. Default value is 0. Adding **NEGATIVE** output velocity gain to a system is mechanically equivalent to adding damping. It is likely to produce a smoother response and allow the use of a higher proportional gain than could otherwise be used, but at the expense of higher following errors. High values may lead to oscillation and produce high following errors. For an output velocity term K_{ov} and change in position ΔP_m , the contribution to the output signal is:

$$\Phi_{OV} = K_{OV} \times \Delta P_m$$

Parameters: **value:** Output velocity gain constant (default = 0).



Negative values are normally required.

P_GAIN

Type: Axis Parameter

Description: The proportional gain sets the ‘stiffness’ of the servo response. Values that are too high will produce oscillation. Values that are too low will produce large following errors.

For a proportional gain K_p and position error E , its contribution to the output signal is:

$$\Phi_p = K_p \times E$$

Parameters: **value:** Proportional gain constant (default =1).

Example: Set the P_GAIN on axis 11 to be a value smaller than the default.

P_GAIN AXIS(11)=0.25

PLM_OFFSET

Type: Axis Parameter

Description: This axis parameter is used exclusively for the **SLM** interface module and only in **PLM** (position mode). The parameter allows for an offset between the absolute position within one turn held by the **SLM/PLM** motor encoder and the zero position in the controller.



UNITS and ENCODER_RATIO should be used in preference to PP_STEPS.

Parameters: **value:** The offset between the absolute position and the controller zero position.

POS_OFFSET

Type: Axis parameter

Description: For Piezo Motor Control. This sets an offset to the DAC output when the position loop is demanding a positive voltage output. POS_OFFSET is applied after DAC_SCALE so is always a value appropriate to the D to A converter resolution.

Example: An offset of 0.1V is required on an axis with a 16 bit D to A converter. With a 16 bit DAC, +10V is commanded with the value 32767 so for 0.1V need $32767 / 100$.

`POS_OFFSET = 328`

POS_OFFSET and NEG_OFFSET are normally used together. It is suggested that the offset is 65% to 70% of the value required to make the stage move in an open loop situation.

`POS_OFFSET = 300`
`NEG_OFFSET = -270`

PP_STEP

Type: Axis parameter

Description: PP_STEP is an integer multiplier on the encoder value.



UNITS and ENCODER_RATIO should be used in preference to PP_STEPS.

Parameters: **value:** Integer multiplier range (default = 1).



IT IS RECOMMENDED TO ONLY USE VALUES BETWEEN -1024 AND 1023.



If used in a Servo axis, increasing PP_STEP will require a proportionate decrease of all loop gain parameters.

PS_ENCODER

Type:	Axis Parameter (Read Only)
Description:	The <code>PS_ENCODER</code> axis parameter holds a raw copy of the positional feedback device used for the hardware p-switch.
Parameters:	<code>value:</code> The 30bit value used for hardware p-switch encoder.
See Also:	<code>HW_PSWITCH</code>

R_MARK

Type:	Axis Parameter (Read Only)
Description:	This parameter can be polled to determine if the registration event has occurred. <code>R_MARK</code> is reset when <code>REGIST</code> is executed
Parameters:	<code>value:</code> <code>TRUE</code> = The registration event has occurred (default). <code>FALSE</code> = The registration event has not occurred.



When `TRUE` the `R_REGPOS` is valid.

Example: Apply an offset to the position of the axis depending on the registration position.

loop:

```

WAIT UNTIL IN(punch_clr)=ON
MOVE(index_length)
REGIST(32+1)           ` rising edge input channel 1
WAIT UNTIL R_MARK
MOVEMODIFY(R_REGPOS + offset)
WAIT IDLE
GOTO loop

```

See Also: `REGIST`, `R_REGPOS`

R_REGISTSPEED

Type: Axis Parameter (Read Only)

Description: Stores the speed of the axis when a registration mark was seen. Value is in user units per millisecond. This parameter is used with the time based registration channel set with the **REGIST** command.



In most real-world systems there are delays built into the registration circuit; the external sensor and the input opto-isolator will have some fixed response time. As machine speed increases, the fixed electrical delays will have an effect on the captured registration position.

R_REGISTSPEED returns the value of axis speed captured at the same time as R_REGPOS. The captured speed and position values can be used to calculate a registration position that does not vary with speed because of the fixed delays.

Parameters: **value:** The speed of the axis in user units per millisecond at which the registration event occurred.



This parameter has the units of user_units/msec at all SERVO_PERIOD settings.

Example: Compensate for fixed delays in the registration circuit using R_REGISTSPEED.
fixed_delays=0.012 ' circuit delays in milliseconds

```
REGIST(32+3) ' registration on time based channel 3
WAIT UNTIL R_MARK
captured_position = R_REGPOS-(R_REGISTSPEED*fixed_delays)
```

See Also: **REGIST**, **REGIST_SPEED**, **REGIST_SPEEDB**

R_REGPOS

- Type:** Axis Parameter (Read Only)
- Description:** Stores the position at which a registration mark was seen on the axis in user units. This parameter is used with the time based registration channel that was set by the `REGIST` command.
- Parameters:** `value`: The absolute position in user `UNITS` at which the registration event occurred.
- Example:** A paper cutting machine uses a `CAM` profile shape to quickly draw paper through servo driven rollers then stop it whilst it is cut. The paper is printed with a registration mark. This mark is detected and the length of the next sheet is adjusted by scaling the `CAM` profile with the third parameter of the `CAM` command:

```

` Example Registration Program using CAM stretching:
` Set window open and close:
  length=200
  OPEN _WIN=100
  CLOSE _WIN=130
  GOSUB _Initial
Loop:
  TICKS=0          `Set millisecond counter to 0
  IF R_MARK THEN
    offset=R_REGPOS
    ` This next line makes offset -ve if at end of sheet:
    IF ABS(offset-length)<offset THEN offset=offset-length
    PRINT "Mark seen at:"offset[5.1]
  ELSE
    offset=0
    PRINT "Mark not seen"
  ENDIF

  ` Reset registration prior to each move:
  DEFPOS(0)
  REGIST(32+0+256) ` Allow mark to be seen between 100 and
130      CAM(0,50,(length+offset*0.5)*cf,1000)
  WAIT UNTIL TICKS<=-500
  GOTO Loop

```

(variable "cf" is a constant which would be calculated depending on the machine draw length per encoder edge).

See Also: **REGIST, REG _ POS, REG _ POSB**

RAISE_ANGLE

Type: Axis Parameter

Description: This parameter is used with **CORNER _ MODE**, it defines the maximum change in direction of a 2 axis interpolated move before **CORNER _ STATE** is triggered. When the change in direction is greater than this angle **CORNER _ STATE** will change state so the system can interact with a program.

This can be used to change the angle of a cutting knife.



RAISE _ ANGLE does not control the speed so it should be set equal or greater than STOP _ ANGLE.

Parameters: **value:** The angle to start to interact with a program through **CORNER _ STATE**.

Example: Decelerate to a slower speed when the transition is between 15 and 45 degrees. If the transition is greater than 45degrees sop so that a **CORNER _ STATE** routine can run.

```
CORNER _ MODE=2 + 4
DECEL _ ANGLE = 15 * (PI/180)
STOP _ ANGLE = 45 * (PI/180)
RAISE _ ANGLE= STOP _ ANGLE
```

See Also: **CORNER _ MODE, CORNER _ STATE, DECEL _ ANGLE, STOP _ ANGLE**

REG_INPUTS

Type: Axis Parameter

Syntax: Selects which of the time based registration inputs to use for the A and B channel of registration.

Parameters:

Bits	function
3:0	Input select for registration channel A
0000	Flex Axis Input 0
0001	Flex Axis Input 1
0010	Flex Axis Input 2
0011	Flex Axis Input 3
0100	Flex Axis Input 4
0101	Flex Axis Input 5
0110	Flex Axis Input 6
0111	Flex Axis Input 7
7:4	Input select for registration channel B
0000	Flex Axis Input 0
0001	Flex Axis Input 1
0010	Flex Axis Input 2
0011	Flex Axis Input 3
0100	Flex Axis Input 4
0101	Flex Axis Input 5
0110	Flex Axis Input 6
0111	Flex Axis Input 7

Example: Set registration input 2 as A and 7 as B

```
REG _ INPUTS=$72
```


REG_POS

Type: Axis Parameter (Read Only)

Alternate Format: RPOS

Description: Stores the position at which a registration mark was seen on each axis in user units. This parameter is used with the first (A) hardware registration channel, or Z mark only.

Parameters: **value:** The absolute position in user UNITS at which the registration event occurred.

Example: A paper cutting machine uses a CAM profile shape to quickly draw paper through servo driven rollers then stop it whilst it is cut. The paper is printed with a registration mark. This mark is detected and the length of the next sheet is adjusted by scaling the CAM profile with the third parameter of the CAM command:

```

` Example Registration Program using CAM stretching:
` Set window open and close:
  length=200
  OPEN _WIN=10
  CLOSE _WIN=length-10
  GOSUB Initial
Loop:
  TICKS=0           `Set millisecond counter to 0
  IF MARK THEN
    offset=REG_POS
    ` This next line makes offset -ve if at end of sheet:
    IF ABS(offset-length)<offset THEN offset=offset-length
    PRINT "Mark seen at:"offset[5.1]
  ELSE
    offset=0
    PRINT "Mark not seen"
  ENDIF

  ` Reset registration prior to each move:
  DEFPOS(0)
  REGIST(3+768)' Allow mark at first 10mm/last 10mm of
sheet
  CAM(0,50,(length+offset*0.5)*cf,1000)
  WAIT UNTIL TICKS<-500
  GOTO Loop

```

(variable "cf" is a constant which would be calculated depending on the machine draw length per encoder edge).

See Also: `REGIST, REG _ POSB, R _ REGPOS`

REG_POSB

Type: Axis Parameter (Read Only)

Description: Stores the position at which a registration mark was seen on each axis in user units. This parameter is used with the second (B) hardware registration channel, or Z mark only.

Parameters: `value:` The absolute position in user `UNITS` at which the registration event occurred.

Example: Detect the front and rear edges of an object on a conveyor and measure its length.

```
` Registration on rising edge R0 and falling edge R1
REGIST(11)
WAIT UNTIL MARK
position1 = REG _ POS
WAIT UNTIL MARKB
position2 = REG _ POSB

length = position2 - position1
```

See Also: `REGIST, REG _ POS, R _ REGPOS`

REGIST_CONTROL

Type: Reserved Keyword.

Description: Read or set the low level bit pattern in the control register.

REGIST_DELAY

Type: Reserved Keyword.

Description: The value, in milliseconds, of the total system delays between a signal appearing on the registration input and the position being available to the time-based registration algorithm. A digital system will usually transfer the actual position information with a one servo period delay. Therefore the **REGIST_DELAY** must be adjusted when the **SERVO_PERIOD** parameter is not at the default value.



*In most real-world systems there are delays built into the registration circuit; the external sensor and the input opto-isolator will have some fixed response time. As machine speed increases, the fixed electrical delays will have an effect on the captured registration position. **REGIST_DELAY** can be adjusted to take account of the total delays due to the servo period and input.*

Parameters: **value:** The total registration delay in milliseconds.

Example: Compensate for fixed delay of one servo period plus 10 microseconds sensor input delay when **SERVO_PERIOD** is 1000.

REGIST_DELAY = -1.01

Compensate for fixed delay of one servo period plus 15 microseconds sensor input delay when **SERVO_PERIOD** is 500.

REGIST_DELAY = -0.51

Compensate for fixed delay of one servo period plus 10 microseconds sensor input delay plus one additional **SLM** cycle of 125 microseconds.

REGIST_DELAY = -1.135

REGIST_SPEED

Type: Axis Parameter (Read Only)

Description: Stores the speed of the axis when a registration mark was seen user units per milli-second. This parameter is used with the first (A) hardware registration channel, or Z mark only.



In most real-world systems there are delays built into the registration circuit; the external sensor and the input opto-isolator will have some fixed response time. As machine speed increases, the fixed electrical delays will have an effect on the captured registration position.

Parameters: **value:** The speed of the axis in user units per milli-second at which the registration event occurred.



This parameter has the units of user_units/msec at all SERVO _ PERIOD settings.

Example: Compensate for fixed delays in the registration circuit using REGIST_SPEED.

```
fixed_delays=0.020 ` circuit delays in milliseconds
REGIST(3)
WAIT UNTIL MARK
captured_position = REG_POS-(REGIST_SPEED*fixed_delays)
```

See Also: REGIST, REGIST_SPEEDB, R_REGIST_SPEED

REGIST_SPEEDB

Type: Axis Parameter (Read Only)

Description: Stores the speed of the axis when a registration mark was seen user units per milli-second. This parameter is used with the second (B) hardware registration channel, or Z mark only.



In most real-world systems there are delays built into the registration circuit; the external sensor and the input opto-isolator will have some fixed response time. As machine speed increases, the fixed electrical delays will have an effect on the captured registration position.

`REGIST _SPEEDB` returns the value of axis speed captured at the same time as `REG _POSB`. The captured speed and position values can be used to calculate a registration position that does not vary with speed because of the fixed delays.

Parameters: **value:** The speed of the axis in user units per milli-second at which the registration event occurred.



This parameter has the units of user_units/msec at all `SERVO _PERIOD` settings.

See Also: `REGIST`, `REGIST _SPEED`, `R _REGIST _SPEED`

REMAIN

Type: Axis Parameter (Read Only)

Description: This is the distance remaining to the end of the current move. It may be tested to see what amount of the move has been completed. The units are user distance units.

Parameters: **value:** The distance remaining in user units of the current move.

Example: To change the speed to a slower value 5mm from the end of a move.

```
start:
  SPEED=10
  MOVE(45)
  WAIT UNTIL REMAIN<5
  SPEED=1
  WAIT IDLE
```

REP_DIST

Type: Axis Parameter

Description: The repeat distance contains the allowable range of movement for an axis before the position count overflows or underflows.

When MPOS and DPOS reach REP_DIST they will wrap to either 0 or -REP_DIST depending on REP_OPTION. The same applies in reverse so when MPOS and DPOS reach either 0 or -REP_DIST they wrap to REP_DIST.



BY DEFAULT REP_DIST IS LESS THAN THE SOFTWARE LIMITS. IF YOU INCREASE REP_DIST FROM THE DEFAULT VALUE YOU MAY ACCIDENTLY ACTIVATE FS_LIMIT OR RS_LIMIT.

Parameters: value: The position in user units where the axis position wraps.

Example 1: Units are set so that an axis units is degrees. The programmer wants to work in the range 1-360, which requires REP_OPTION=1.

```
REP_OPTION=1
REP_DIST=360
```

Example 2: MOVETANG requires the axis to be configured so it pi radians of the full revolution. For a 4000 count per rev encoder this means between -2000 and 2000. This can be configured as follows

```
BASE(0)
UNITS=1
REP_OPTION=0
REP_DIST=2000
MOVETANG(0,1)
```

See Also: FS_LIMIT, RS_LIMIT

REP_OPTION

Type: Axis Parameter

Description: `REP_OPTION` allows different repeat options for the axis. It can be used to affect the way the position of an axis wraps or the repeating mode of `CAMBOX` and `MOVELINK`.

Parameters:

value:	Operation.
bit 0:	0 = Axis position range is <code>-REP_DIST</code> to <code>+REP_DIST</code> . 1 = Axis position range is 0 to <code>+REP_DIST</code> .
bit 1:	0 = Automatic repeat option is disabled. 1 = Disable the automatic repeat option of <code>CAMBOX</code> and <code>MOVELINK</code> .
bit 2:	0 = <code>REP_DIST</code> , <code>DEFPOS</code> and <code>OFFPOS</code> will affect <code>MPOS</code> and <code>DPOS</code> . 1 = <code>REP_DIST</code> , <code>DEFPOS</code> and <code>OFFPOS</code> will affect <code>MPOS</code> only.



Bit 2 has been included for backward compatibility, it is not recommended to use this on new applications.

Example 1: An axis has 400 counts per revolution, configure `REP_DIST` and `REP_OPTION` so that it wraps from 0 to 4000.

```
REP_OPTION = 1
```

```
REP_DIST = 4000
```

Example 2: A program is running a continuous `MOVELINK`, when an input is triggered the link must end at the end of the next cycle. Set bit is used so not to clear any other bits that may be active.

```
MOVELINK((1, 1.6, 0.6, 0.6, 1, 4)
WAIT UNTIL IN(1) = ON
REP_OPTION = REP_OPTION AND 2
```

See Also: `CAMBOX`, `MOVELINK`, `REP_DIST`

REV_IN

Type: Axis Parameter

Description: This parameter holds the input number to be used as a reverse limit input. When the reverse limit input is active any motion on that axis is **CANCELED**. When **REV_IN** is active **AXISSTATUS** bit 5 is set.



*The input used for **REV_IN** is active low.*

Parameters: **Value:** 1 = disable the input as **REV_IN** (default).
0-63 = Input to use as the reverse input switch.



Any type of input can be used, built in, TrioCAN I/O, CANopen or virtual.

Example: Set up inputs 8 and 9 as forward and reverse limit switches for axis 4.

```
BASE(4)
FWD_IN = 8
REV_IN = 9
```

See Also: **FWD_IN**, **FS_LIMIT**, **RS_LIMIT**

REV_JOG

Type: Axis Parameter

Description: This parameter holds the input number to be used as a jog reverse input. When the **REV_JOG** input is active the axis moves in reverse at **JOG_SPEED**.



*The input used for **REV_IN** is active low.*

*It is advisable to use **INVERT_IN** on the input for **REV_JOG** so that 0V at the input disables the jog.*

***FWD_JOG** overrides **REV_JOG** if both are active.*

Parameters: **Value:** 1 = disable the input as REV _ JOG (default).
 0-63 = Input to use as datum input.

Example: Initialise the REV _ JOG so that it is active high on input 12.

 INVERT _ IN(12,ON)

RS_LIMIT

Type: Axis Parameter

Alternate Format: RSLIMIT

Description: An end of travel limit may be set up in software thus allowing the program control of the working envelope of the machine. This parameter holds the absolute position of the forward travel limit in user units.

Bit 10 of the **AXISSTATUS** register is set when the axis position is greater than the **RS _ LIMIT**.



When DPOS reaches **RS _ LIMIT** the controller will cancel the move, so the axis will decelerate at **DECEL** or **FAST _ DEC**.



RS _ LIMIT is disabled when it has a value greater than **REP _ DIST**.

Parameters: **Value:** The absolute position of the software forward travel limit in user units. (default = 20000000000).

Example 1: After homing a machine set up the reverse software limit so that the axis will stop 10mm away from the hard stop. So if the hard limit is at -200, with a maximum speed of 400 and a **FASTDEC** of 1000 the reverse limit will be -189.6.

```
hard_limit_position = -200
max_speed = 400
FASTDEC = 1000
```

```
DATUM(3)
WAIT IDLE
```

```
RS_LIMIT= hard_limit_position + ( max_speed/FASTDEC +10 )
```

See Also: FS _ LIMIT, FWD _ IN, REV _ IN

SERVO

Type: Axis Parameter

Description: On a servo axis this parameter determines whether the axis runs under servo control or open loop. When **SERVO=OFF** the axis hardware will output demand value dependent on the **DAC** parameter. When **SERVO=ON** the axis hardware will output a demand value dependant on the gain settings and the following error.

Parameters: Value: ON = closed loop servo control enabled.
OFF = closed loop servo control disabled.

Example: Enable axis 1 to run under closed loop control and axis 1 as open loop.

```
SERVO AXIS(0)=ON'   Axis 0 is under servo control
SERVO AXIS(1)=OFF' Axis 1 is run open loop
```

SLOT_NUMBER

Type: Axis Parameter (Read Only)

Description: Returns the **SLOT** number where the axis is located. Axis numbers can be allocated to hardware in a flexible way, so the physical location of the axis cannot be found by the **AXIS** number alone. **SLOT_NUMBER** returns the value from the **BASE** axis or if the **AXIS(number)** modifier is used, it returns the **SLOT** associated with that axis.

Example:

```
PRINT SLOT _ NUMBER AXIS(12)

BASE(2)
axis2 _ slot = SLOT _ NUMBER

IF SLOT _ NUMBER AXIS(0)<>-1 THEN
  PRINT "Warning - Built-in axis configuration incorrect"
  PRINT "Axis 0 expected for this application."
ENDIF
```

See also: `SLOT, AXIS _ OFFSET`

SPEED

Type: Axis Command

Description: The `SPEED` axis parameter can be used to set/read back the demand speed axis parameter.

Parameters: `value:` The axis speed in user units.

Example: Set the speed and then print it to the user.

```
SPEED=1000
PRINT "Speed Set=";SPEED
```

SPEED_SIGN

Type: Reserved Keyword

SPHERE_CENTRE

Type: Axis Command

Syntax: `SPHERE _ CENTRE(tablex, tabley, tablez)`

Description: Returns the co-ordinates of the centre point (x, y, z) of the most recent `MSPHERICAL`. X, Y and Z are returned in the `TABLE` memory area and can be printed to the terminal as required.

Parameters:

- tablex:** Position in table to store the X coordinate.
- tabley:** Position in table to store the Y coordinate.
- tablez:** Position in table to store the Z coordinate.

Example: After a **MSPHERICAL** completes on axis 0 find the coordinates of the centre.

```
SPHERE _CENTRE(10, 11, 30) AXIS(0)
PRINT TABLE(10);", ";TABLE(11);", ";TABLE(12)
```

SRAMP

Type: Axis Parameter

Description: This parameter stores the s-ramp factor. It controls the amount of rounding applied to trapezoidal profiles. **SRAMP** should be set, when a move is not in progress, to a maximum of half the **ACCEL/DECEL** time. The setting takes a short while to be applied after changes.

Parameters: **value:** Time between 0..250 milliseconds.



SRAMP MUST BE SET BEFORE A MOVE STARTS. IF FOR EXAMPLE YOU CHANGE THE SRAMP FROM 0 TO 200, THEN START A MOVE WITHIN 200 MILLISEC THE FULL SRAMP SETTING WILL NOT BE APPLIED.

Example: To provide smooth transition into the acceleration, an S-ramp is applied with a time of 50msec.

```
SPEED = 160000
ACCEL = 1600000
DECEL = 1600000
SRAMP = 50
```

```
WA(50)
```

```
MOVEABS(100000)
```

Without the S-ramp factor, the acceleration takes 100 msec to reach the set speed. With **SRAMP=50**, the acceleration takes 150 msec but the rate of change of force (torque) is controlled. i.e. Jerk is limited.

START_DIR_LAST

Type: Axis Parameter (Read Only)

Description: Returns the direction of the start of the last loaded interpolated motion command. **START_DIR_LAST** will be the same as **END_DIR_LAST** except in the case of circular moves.



This parameter is only available when using SP motion commands such as MOVESP, MOVEABSSP etc.

Parameters: **value:** End direction, in radians between $-\pi$ and π . Value is always positive.

Example1: Run two moves the first starting at a direction of 45 degrees and the second 0 degrees.

```
>>MOVESP(10000,10000)
>>? START_DIR_LAST
0.7854
>>MOVESP(0,10000)
>>? START_DIR_LAST
0.0000
>>
```

See Also: **CHANGE_DIR_LAST, END_DIR_LAST**

STARTMOVE_SPEED

Type: Axis Parameter

Description: This parameter sets the start speed for a motion command that support the advanced speed control (commands ending in **SP**). The **VP_SPEED** will decelerate until **STARTMOVE_SPEED** is reached for the start of the motion command.



*The lowest value of **SPEED**, **ENDMOVE_SPEED**, **FORCE_SPEED** or **STARTMOVE_SPEED** will take priority.*

`STARTMOVE _ SPEED` is loaded into the buffer at the same time as the move so you can set different speeds for subsequent moves.

In general `START _ MOVE` speed is only used by the `CORNER _ MODE` methods. The user can program all profiles using only `FORCE _ SPEED` and `ENDMOVE _ SPEED`.

Parameters: `value:` The speed at which the `SP` motion command will start, in user `UNITS`. (default 0).

See Also: `FORCE _ SPEED`, `ENDMOVE _ SPEED`, `CORNER _ MODE`

STOP_ANGLE

Type: Axis Parameter

Description: This parameter is used with `CORNER _ MODE`, it defines the maximum change in direction of a 2 axis interpolated move that will be merged at speed. When the change in direction is greater than this angle the reduced to 0.

Parameters: `value:` The angle to reduce the speed to 0, in radians).

Example1: Reduce the speed to zero on a transition greater than 25 degrees. `DECEL _ ANGLE` is set to 25 degrees as well so that there is no reduction of speed below 25 degrees.

```
CORNER _ MODE=2
STOP _ ANGLE=25 * (PI/180)
DECEL _ ANGLE=STOP _ ANGLE
```

See Also: `CORNER _ MODE`, `DECEL _ ANGLE`

TANG _DIRECTION

Type: Axis Parameter

Description: When used with a 2 axis X-Y system, this parameter returns the angle in radians that represents the vector direction of the interpolated axes.

Parameters: The value returned is between $-\pi$ and $+\pi$ and is determined by the directions of the interpolated axes.

value:	X	Y
0	0	1
$\pi/2$	1	0
$\pi/2(+\pi$ or $\pi)$	0	-1
$-\pi/2$	-1	0

Example: Note `scale_factor_x` **MUST** be the same as `scale_factor_y`

```
UNITS AXIS(4)=scale_factor_x
UNITS AXIS(5)=scale_factor_y

BASE(4,5)
MOVE(100,50)
angle = TANG _DIRECTION
```

Example2: `BASE(0,1)`

```
angle_deg = 180 * TANG _DIRECTION / PI
```

TRANS_DPOS

Type: Axis Parameter (Read Only)

Description: `TRANS _DPOS` is the axis demand position at output of frame transformation.

`TRANS _DPOS` is normally equal to `DPOS` on each axis. The frame transformation is therefore equivalent to 1:1 for each axis (**FRAME** = 0). For some machinery configurations it can be useful to install a frame transformation which is not 1:1,

these are typically machines such as robotic arms or machines with parasitic motions on the axes. In this situation when **FRAME** is not zero **TRANS _ DPOS** returns the demand position for the actual motor.

Parameters: **value:** The axis demand position at the output of the **FRAME** transformation.

See also: **FRAME**

TRIOPTTESTVARIAB

Type: Reserved Keyword

UNITS

Type: Axis Parameter

Description: **UNITS** is a conversion factor that allows the user to scale the edges/ stepper pulses to a more convenient scale. The motion commands to set speeds, acceleration and moves use the **UNITS** scalar to allow values to be entered in more convenient units e.g.: mm for a move or mm/sec for a speed.



*Units may be any positive value but it is recommended to design systems with an integer number of encoder pulses/user unit. If you need to use a non integer number you should use **ENCODER _ RATIO**. **STEP _ RATIO** can be used for non integer conversion on a stepper axis.*

Parameters: **value:** The number of counts per required units.

Example: A leadscrew arrangement has a 5mm pitch and a 1000 pulse/rev encoder. The units should be set to allow moves to be specified in mm.

The 1000 pulses/rev will generate $1000 \times 4 = 4000$ edges/rev in the controller. One rev is equal to 5mm therefore there are $4000/5 = 800$ edges/mm.

`>>UNITS=1000*4/5`

Example 2: A stepper motor has 180 pulses/rev. There is a built in 16 multiplier so the controller will use 180*16 counts per revolution.

To program in revolutions the unit conversion factor will be:

```
>>UNITS=180*16
```

See Also: ENCODER _ RATIO, STEP _ RATIO

VECTOR_BUFFERED

Type: Axis Parameter (Read only)

Description: This holds the total vector length of the buffered moves. It is effectively the amount the $\nu P U$ can assume is available for deceleration. It should be executed with respect to the first axis in the group.

Parameters: `value:` The vector length of buffered moves on the axis group.

Example: Return the total vector length for the current buffered moves whose axis group begins with axis(0).

```
>>BASE(0,1,2)
>>? VECTOR _ BUFFERED AXIS(0)
1245.0000
>>
```

VERIFY

Type: Reserved Keyword

VFF_GAIN

Type: Axis Parameter

Description: The velocity feed forward gain is a constant which is multiplied by the change in demand position. Velocity feed forward gain can be used to decrease the following error during constant speed by increasing the output proportionally with the speed. For a velocity feed forward K_{Vff} and change in position ΔP_d , the contribution to the output signal is:

$$O_{Vff} = K_{Vff} \times \Delta P_d$$

Parameters: **value:** Velocity feed forward constant (default =0).

Example: Set the `VFF_GAIN` on axis 15 to 12

```
BASE(15)  
VFF_GAIN=12
```

VP_SPEED

Type: Axis Parameter (Read Only)

Alternate Format: `VPSPEED`

Description: The velocity profile speed is an internal speed which is ramped up and down as the movement is velocity profiled.

Parameters: **value:** The velocity profile speed in user units/second.

Example: Wait until command speed is achieved:

```
MOVE(100)  
WAIT UNTIL SPEED=VP_SPEED
```


CHAPTER
SUPPORT SOFTWARE

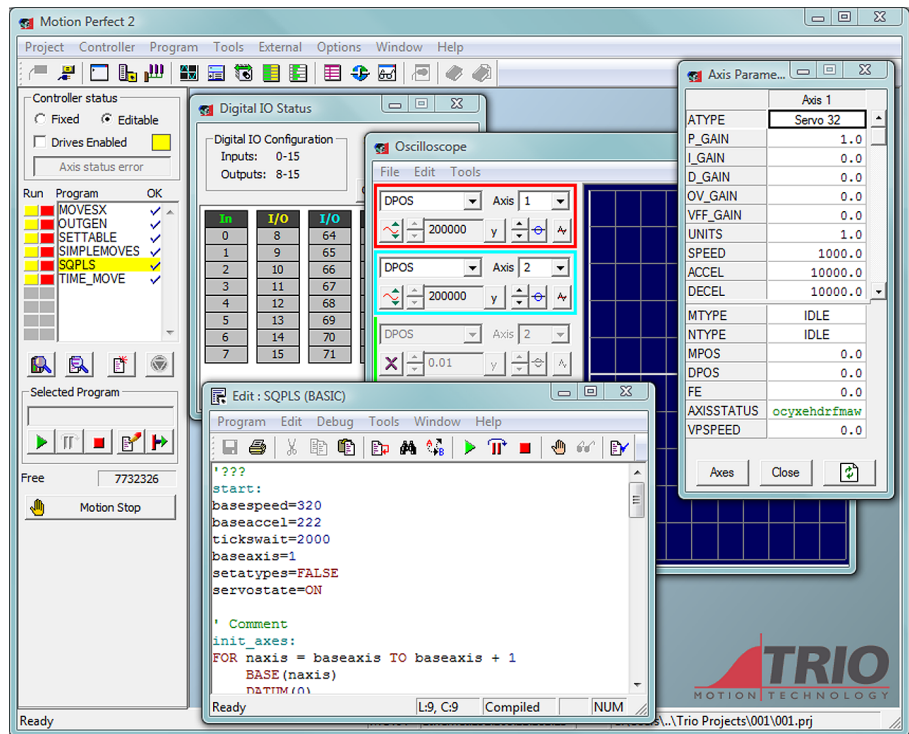
9

Support Software

Motion Perfect 2

Motion Perfect 2 is an application for the PC, designed to be used in conjunction with the *Motion Coordinator* range of multi-tasking motion controllers.

Motion Perfect provides the user with an easy to use Windows based interface for controller configuration, rapid application development, and run-time diagnostics of processes running on the *Motion Coordinator*.



System Requirements

The following equipment is required to use *Motion Perfect 2*.

PC	Minimum Specification	Recommended
CPU	Pentium class processor, operating at 1GHz	Pentium class processor, operating at 2GHz
RAM	256MB (XP), 512 MB (Vista)	512MB (XP), 1GB (Vista)
Hard disk space	20 Mb	20 Mb
Operating System	Windows XP or Vista .	Windows XP or Vista.
Display	1024 x 768, 24-bit colour.	1024 x 768, 24-bit colour.
Communications	Single RS232 Serial Port	RS232 serial port, USB port



Motion Perfect may work on Windows 2000 but it is no longer supported on this platform. It will not work on Windows 95, 98 or ME.

Motion Coordinator controller or compatible controllers

Compatible controllers include:

MC2, MC202, MC204, MC402e, Euro205, Euro205x, MC206, MC206X, PCI208, MC216, MC224, MC302, MC464 etc.

In order to use the serial link Packet Communications mode, system software version 1.49 or higher is required.



You should always try to use the most recent version of Motion Perfect. Updates are available from your local distributor or you can download the latest version from the Trio Web site: WWW.TRIOMOTION.COM

Connecting *Motion Perfect* to a controller

Motion Perfect can be connected to the *Motion Coordinator* using a serial, USB, Ethernet or PCI connection depending on the interface(s) fitted to the *Motion Coordinator*.

It is possible to edit a project without having a controller connected to your PC by using the MC Simulator program.

Running *Motion Perfect 2* for the First time

Make sure the *Motion Coordinator* is connected to the PC and turned on then, with Windows running, select “TrioMotion / Motion Perfect 2” from “All Programs” on the Windows start menu to Launch *Motion Perfect*. As *Motion Perfect* starts up you will see a splash screen such as the one below.



The splash screen features a small messages window (bottom left) which is used to display the status of the connection process. In this example *Motion Perfect* is connected to an MC464 controller via an Ethernet link to IP address 192.168.11.181 on Ethernet port 23.

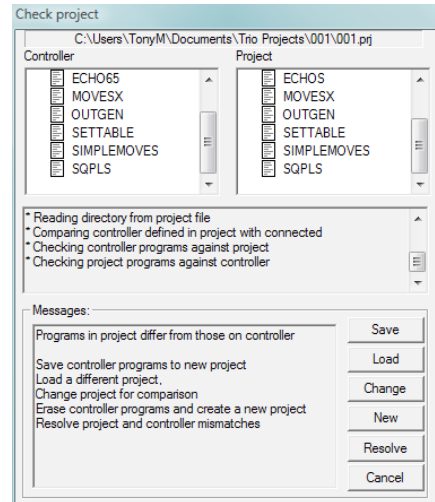
Motion Perfect 2 Projects

One of the keys to using *Motion Perfect* is to understand its concept of a “Project”. The project facilitates the application design and development process, by providing a disk based copy of the multiple controller programs, parameters and data which may be used for a single motion application. Once the user has defined a project, *Motion Perfect* works behind the scenes automatically maintaining consistency between the programs on the controller and the files on the PC. When creating or editing programs on the controller they are automatically duplicated on the PC which means you do not have to worry about loading or saving programs and you can be confident that next time you connect to the controller you will have the correct information on your PC.

Project Check Window

Whenever you connect to the controller, *Motion Perfect* will perform a project check to compare the programs on the controller with those defined in the current project on the PC. During the project check a window similar to the one below will be displayed. If the projects match then you will see a “project checked ok” message and an OK button to continue. If however there is any inconsistency between the controller and the PC, the display will feature a number of addition options, shown below.

You can force *Motion Perfect* to perform a project check at any time with the “Check Project” option from the project menu. (Ctrl+Alt+P)



Project Check Options

Save: Save the controller contents to disk.

If you have never connected with this controller before, and therefore do not have the project on your PC, or if there is an inconsistency in the project check and you are sure that the project on the controller is the correct version, then select **SAVE** to copy the programs on the controller to disk.



This will of course overwrite any programs already in the PC copy of the project. If you are unsure which is the correct version, you should save the project with a new name to avoid overwriting any existing project programs on the PC.

Load: Load the PC files onto the controller

If you are uploading a complete project from the PC to the controller, or the project check fails and you are sure the version on your PC is correct, then you should use this option to upload the entire project from the PC to the controller.

Note: The entire contents of the programs on the controller will be erased. If you are unsure, **SAVE** the controller contents first!

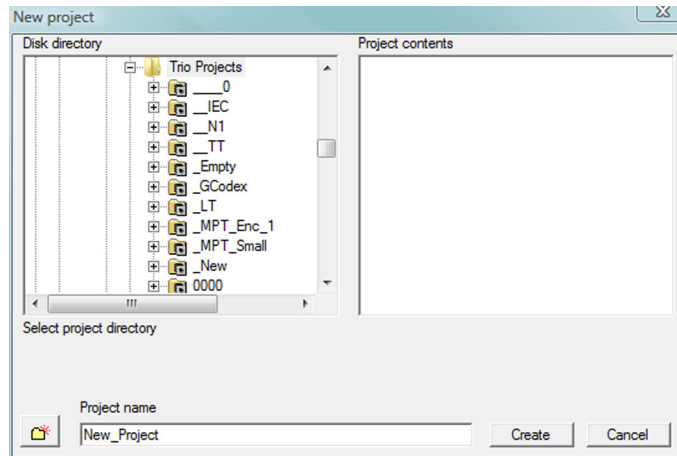
Change: Change the project on the PC to compare with.

If you have been working on more than one project, the project on the controller may not match the ‘last project’ remembered by *Motion Perfect*. If this is the case you can use this option to select another project on the PC. Once you select an alternative, *Motion Perfect* will perform a fresh project check and the above process will be repeated.

New: Create a new project

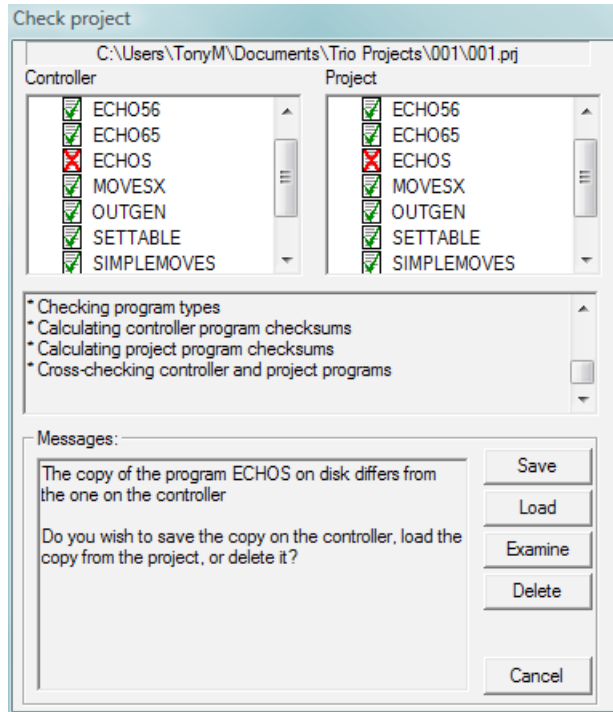
The controller contents will be erased and a new project created on the PC. You will be prompted to select a directory and project name.

When you create a new project, *Motion Perfect* will make a new directory with the project name, and within that directory a project file with the same name (the `.PRJ` extension is added to the filename).



Resolve:

This option should be used when you have the correct project selected, but one or more of the files differ between the controller and PC version, or do not exist in one of the copies.



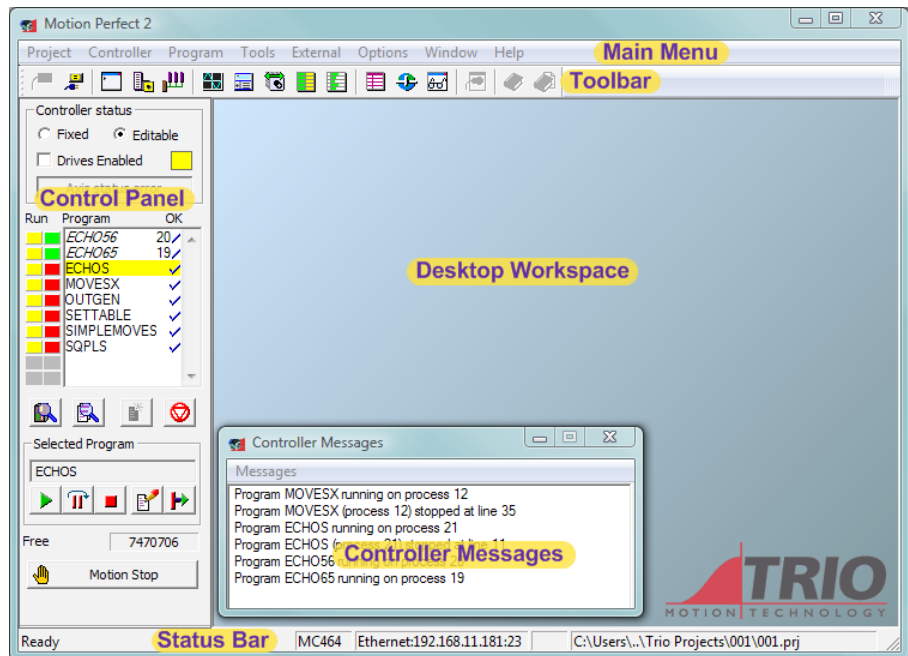
You will need to use your judgment to decide whether the disk or controller version is correct. Typically, if you are recovering the project after a comms failure or PC crash then the version on the controller should be saved. If you have modified the disk based copy of the program then you will need to load this version onto the controller. The examine button starts an external compare program to allow you to visually compare the version on the controller to the one on the PC.

Cancel:

Cancels the connection process and starts *Motion Perfect* in disconnected mode.

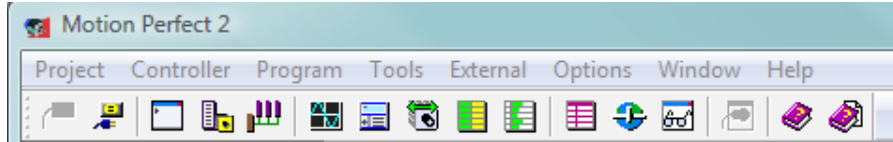
Once the project has been checked and is consistent then a backup copy of the PC project will be created.

The *Motion* Perfect Desktop



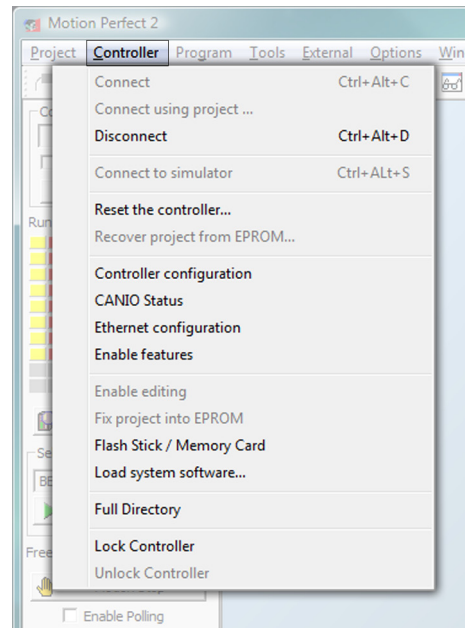
- Main Menu:** Standard Windows menu to access all features of the *Motion* Perfect application.
- Toolbar:** Shortcut buttons to access the *Motion* Perfect tools
- Control Panel:** Displays the current controller contents and provides controls for interrogating the controller status, running / editing programs
- Desktop Workspace:** This area is used to display the user windows and tools
- Controller Messages:** Status and error messages reported by the controller
- Status Bar:** Information about the current project and controller connection.

Main Menu



Project:	Options for Creating, Loading & Saving <i>Motion Perfect</i> Projects, Loading/Saving program files and Table data
Controller:	Options relating to the controller hardware, including connecting/disconnecting and checking configuration information.
Program:	Program specific options, including creating, editing and running controller tasks.
Tools:	Access to the main <i>Motion Perfect</i> tools. These options are also available from the Toolbar
Options:	Configure the <i>Motion Perfect</i> Environment. Includes options to setup the communications ports and to customise the editor display.
Window:	Control the appearance of the <i>Motion Perfect</i> desktop.
Help:	Access the help files and version information.

Controller Menu



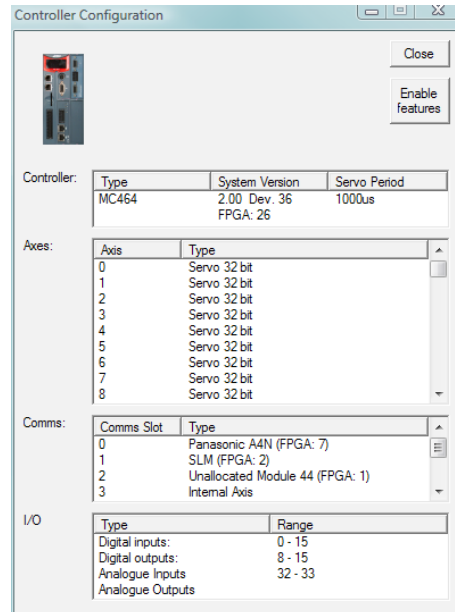
The controller menu contains the following items:

- | | |
|------------------------------------|--|
| Connect: | Connect to the controller and start the project manager. This is only available if <i>Motion Perfect</i> is currently disconnected from the controller. |
| Connect using project | Connect to the controller and start the project manager, displaying a “Select Project” dialogue to allow the user to specify a project. This is only available if <i>Motion Perfect</i> is currently disconnected from the controller. |
| Disconnect: | Disconnect from the controller, and stop using the project tools. Only available if <i>Motion Perfect 2</i> is currently connected to the controller. |
| Connect to Simulator: | Connect to the controller simulator and start the project manager. The controller simulator is started if it is not already running. This is only available if <i>Motion Perfect</i> is currently disconnected from the controller. |
| Reset Controller: | Perform a software-reset (EX) on the controller. This will cause <i>Motion Perfect 2</i> to disconnect from the controller. |
| Recover Project from EPROM: | Reset the controller and restore the programs which were previously stored in the EPROM. |

Controller Configuration:	Display hardware and system software configuration data for the controller.
CANIO Status:	Display the status of any CAN I/O modules connected to the controller.
Ethernet Configuration:	Configure the parameters of any ethernet interfaces on the controller.
Enable Features:	Enable or disable any features which can be enabled using feature codes.
Enable Editing:	Restore the power-up state of a controller currently starting from EPROM to run from RAM and allow editing.
Fix Project into EPROM:	Store the programs in RAM into the controllers flash-EPROM memory. The startup state for each program will not be changed.
Flash Stick/ Memory Card:	Store the current project on a flash stick or load a project from a flash stick/memory card (for controllers with a flash stick interface).
Load System Software:	Update the controller system software.
Full Directory:	Display a complete listing of all files on the controller, details of memory used and the run status of each program.
Lock Controller:	Lock the controller to prevent modification of the programs.
Unlock Controller:	Unlock a previously locked controller to allow programs to be edited.

Controller Configuration

This screen interrogates the hardware and displays the configuration information reported back by the controller.



Looking at the example screen shown here from top to bottom:

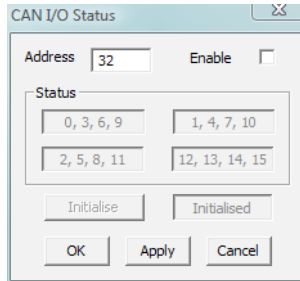
- Controller:** We are connected to a *Motion Coordinator MC464*
- Software Version:** The controller is running version 2.00 (development version 36) of the system software. The main FPGA on the controller is at version 26
- Servo Period:** The controller is running with a control servo period of 1000µs.
- Axis:** A list of the types of all the axes on the controller.
- Comms:** If the controller is fitted with any of the extended / communications daughter boards or modules, they will be shown here. Certain internal communication or axes can also be shown here.
- I/O:** The channel range available for each type of I/O both digital and analogue. Remember that on many *Motion Coordinator* the digital channels are shared, e.g. if Output 15 is available, then it implies that Input 15 is also available and shares the same connection.



FPGA versions are not shown for all Motion Coordinators.

CAN I/O Status

This shows the status of the built-in CAN port on a *Motion Coordinator* and any CAN I/O modules connected to it.



- Address:** This is the CAN address of the built-in CAN port. The address can be set in the range 32 to 47. If the address is 32 the controller can automatically poll CAN I/O modules connected to it.
- Enable:** If this is checked (and the CAN address is set to 32) automatic polling of I/O modules is active.
- Status:** This shows the status of groups of I/O modules by CAN address, green for OK, red for error.
- Initialise:** Clicking on this initialises the built-in CAN port on the controller.
- Initialised:** This shows the state of the built-in CAN port, green for OK, red for error.

Ethernet Configuration

This shows the configuration for an ethernet interface on the controller. It allows the user to set up ethernet addressing parameters for built-in or daughterboard ethernet interfaces.

Ethernet

Slot: -1

Data

IP Address: 192 168 11 181

Subnet Mask: 255 255 255 0

Default Gateway: 192 168 11 225

MAC Address: 00:06:70:00:00:B5

Normal Communications Port Number: 23 (Default 23)

Token Communications Port Number: 3240 (Default 3240)

Ethernet Firmware Version: 51.22

Modbus TCP Mode: Float

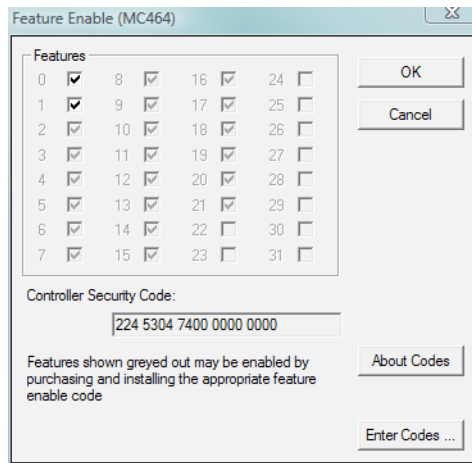
OK Cancel

- Slot:** This is the expansion module slot (-1 for built-in) of the ethernet interface being viewed.
- IP Address:** This is the ethernet IP address of this ethernet interface.
- Subnet Mask:** This is the ethernet subnet mast for the network to which this ethernet interface is connected.
- Default Gateway:** This is the default gateway for this ethernet interface. It is only needed if the controller is required to communicate with a device on a different ethernet subnet to its own.
- MAC Address:** This is the hardware MAC address for the current interface.
- Normal Communications Port Number:** This is the IP port number on which normal communications will take place. This is the port used by *Motion Perfect* for communications. The default value is 23, which is the reserved port for telnet communications.
- Normal Communications Port Number:** This is the IP port used for token based communications. This port is used by the Trio PC *Motion ActiveX* control. The default value is 3240, which is the reserved port for Trio *Motion Control*.
- Ethernet firmware version:** This shows the version number of the ethernet firmware for the current interface.
- MODBUS tcp mode:** This sets the type of numerical representation used by MODBUS tcp over this interface. The value can be float or integer.

Feature Enable

The MC464 *Motion Coordinator* has the ability to unlock additional axes by entering a “Feature Enable Code”.

When you access the Feature Enable dialogue, you will be presented with a display similar to one of the following:



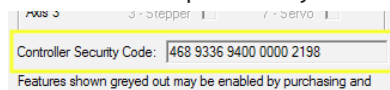
This display shows the features which are currently available. If the codes for additional features have been purchased and stored on the PC, the relevant boxes will be available for checking, otherwise the check boxes will be greyed out. If a feature has been enabled on the controller but the code has not been stored on the PC its box will be checked but also greyed out.

Enabling Additional Features

To enable a feature you must enter a Feature Enable Code, which is unique to each controller and feature. To obtain a Feature Enable Code, you will need to specify the feature required and the security code for the specific controller to be updated. The order for the required codes should be FAXed to Trio or an authorised Trio distributor.

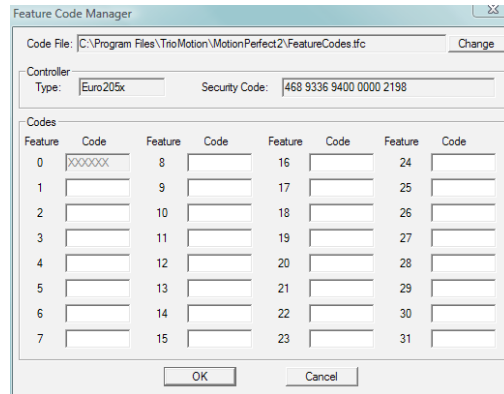
Security Code

Controllers with features which can be enabled each have a unique security code number which is implanted when the unit is manufactured. This security code number is displayed on the above screen (as highlighted right).



Once you have the required codes, select the button.

A dialogue similar to the following example will appear.



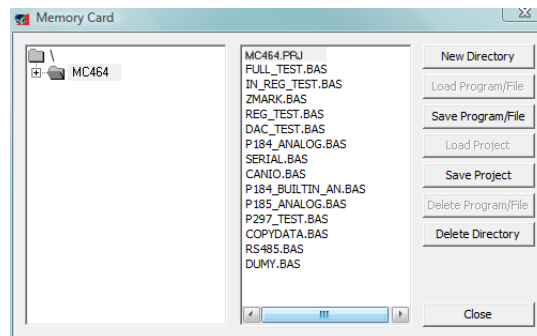
Each feature requested has a feature number. Enter the relevant code for each feature number, being careful to enter the characters in upper case. Take care to check that 0 (zero) is not confused with the letter “O” and 1 (one) is not confused with the letter “l”.


Feature Code File

Motion Perfect stores all of the Feature Enable Codes of which it is aware in a file called “FeatureCodes.TFC”. By default this file is located in the same directory as the *Motion Perfect 2* executable file.

Memory Card Support

When a controller with SD Card support is powered on with a memory card inserted, then the controller will automatically run the program **TRIOINIT.BAS** in the root directory of the SD Card. Full details of this can be found in Trio Technical Note TN20-99



When the Memory Card dialogue is first displayed it shows only the root directory. Double clicking on its icon  will expand the tree one level. Double clicking on a directory icon at any time will toggle its state from collapsed to expanded and vice versa.

The function buttons on the right of the dialogue are enabled and disabled according to the type of item selected in the directory tree or the directory listing. The functions are as follows:

- | | |
|-------------------------------|---|
| New Directory: | Creates a new subdirectory on the card in the directory selected. |
| Load Program / File: | Loads the selected program from the card onto the controller. |
| Save Program / File: | Saves a program file from the controller into the directory currently selected on the card. |
| Load Project: | Loads the project selected on the card onto the controller. |
| Save Project: | Saves the project on the controller into the directory currently selected on the card. |
| Delete Program / File: | Delete the program file currently selected on the card. |
| Delete Directory: | Deletes the selected directory from the card. |

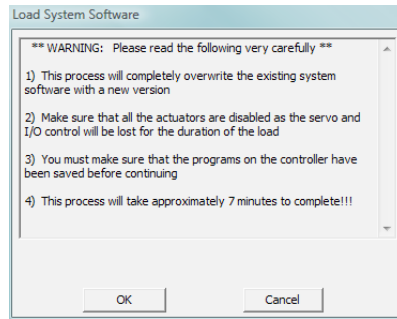
Loading New System Software

Motion Coordinators feature a flash EPROM for storage of both user programs and the system software. From *Motion Perfect 2* it is possible to upgrade the software to a newer version using a system file supplied by Trio.

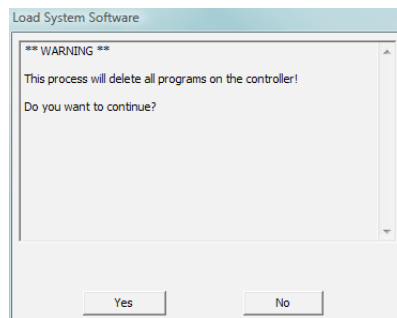


We do not advise that you load a new version of the system software unless you are specifically advised to do so by your distributor or by Trio.

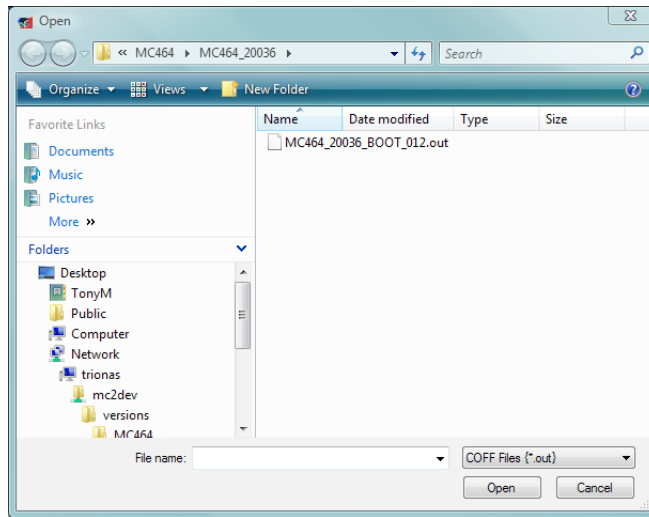
When you select the ‘Load System Software’ option from the controller menu, you will first be presented with a warning dialogue to ensure you have saved your project and are sure you wish to continue.



if you press **OK** you will then be warned that the operation will delete all programs on the controller. This must be done because the programs are stored on the controller in a tokenized form and loading new system code may change the token list, consequently changing the commands in the programs.



When you press Yes you will be presented with the standard Windows file selector to choose the file you wish to load.



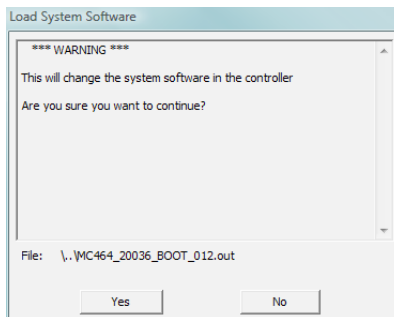
Each *Motion Coordinator* controller has its own system file, identified by the first letter (or letters) of the file name.

System Software File Prefix Codes:

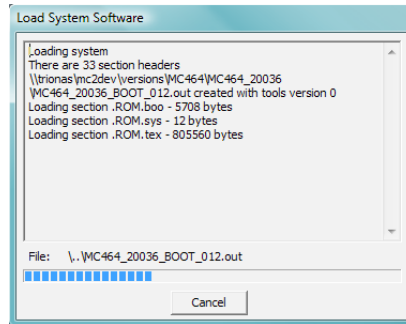
Filename	Controller Type
MC464	MC464

You must ensure that you load only software designed for this specific controller, other versions will not work and will probably make the controller unusable.

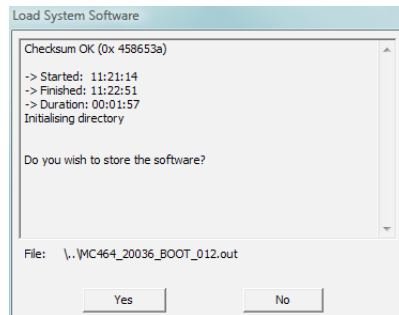
When you have chosen the appropriate file you will be prompted once again to check that you wish to continue. Press **OK** to start the download process.



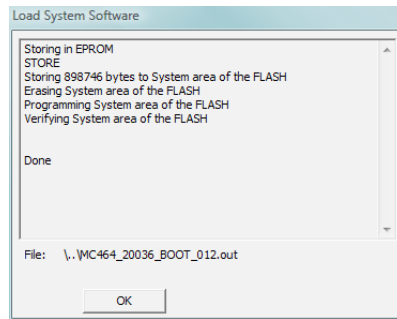
Downloading may take several minutes, depending on the speed of your PC and the controller. During the download, you should see the progress of each section updated as follows:-



When the download is complete, a checksum is performed to ensure that the download process was successful. If it was you will be presented with a confirmation screen and asked if you wish to store the software into EPROM.



When you press Yes, the controller will take a few moments to fix the project into the EPROM and you can then continue as normal.

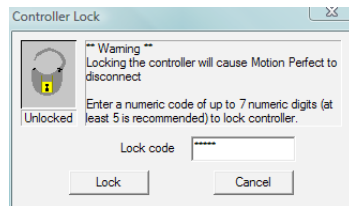


At this point you can check the controller configuration to confirm the new software version.

Lock / Unlock

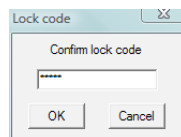
Lock Controller:

Locking the controller will prevent any unauthorised user from viewing or modifying the programs in memory.



You simply need to enter a numeric code (up to 7 digits). This value will be encoded by the system and used to lock the directory structure. The lock code is held in encrypted form in the flash memory of the *Motion Coordinator*.

Once you have entered a code and clicked on Lock another dialogue will appear asking you to confirm the lock code.



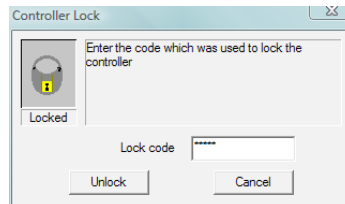
Once the *Motion Coordinator* is locked it is not possible to list, edit or save any of the controller programs. You cannot connect to the controller with *Motion Perfect 2*, although the terminal screen and unlock dialogue will still be available.



If you forget the lock code there is no way to unlock the controller. You will need to return it to Trio or a distributor to have the lock removed.

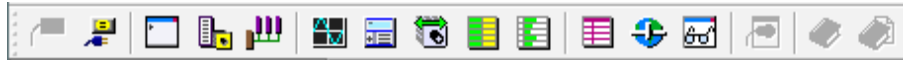
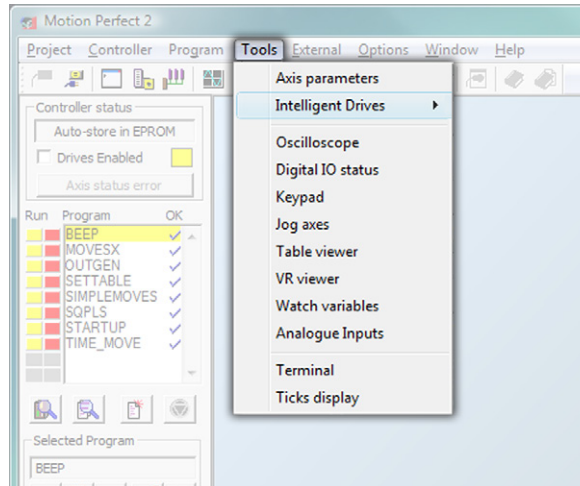
Unlock Controller:














In order to unlock the controller you need to enter the same numeric code which was used to lock it. Once the unlock code is entered it will be possible to gain full access to the programs in memory.






Motion Perfect Tools

The *Motion Perfect* tools can be accessed from either the Tools Menu or the Toolbar buttons.



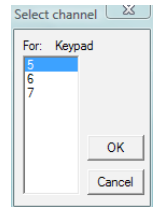
-  Connect to Controller
-  Disconnect from Controller
-  Launch Terminal Tool
-  Launch Axis Parameter Tool
-  Launch Intelligent Drives Configuration Tool
-  Launch Oscilloscope
-  Launch Keypad Emulator
-  Launch Jog Axes Tool
-  Launch Digital I/O Tool
-  Launch Analogue I/O Tool
-  Launch Table Viewer
-  Launch VR Variable Viewer
-  Launch Variable Watch Tool

-  Connect to Simulator
-  Motion Perfect Help
-  TrioBASIC Help

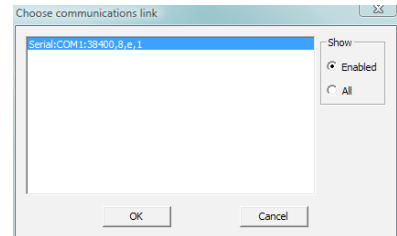
Terminal

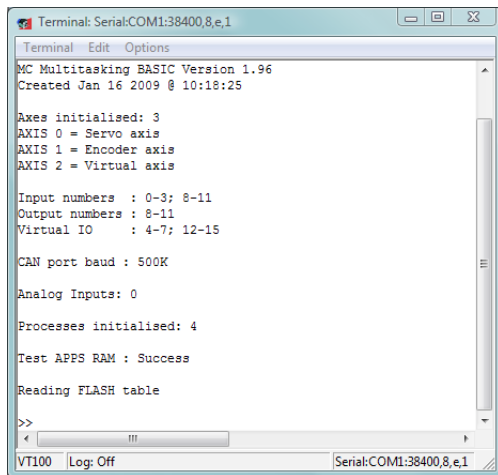
The terminal window provides a direct connection to the *Motion Coordinator*. Most of the functions that must be performed during the installation, programming and commissioning of a system with a *Motion Coordinator* have been automated by the options available in the *Motion Perfect* menu options. However, if direct intervention is required the terminal window may be used.

When *Motion Perfect* is in connected mode then, on starting the terminal tool you will be presented with a dialogue to select the communications channel. Channel 0 is used for the controller command line and channels 5, 6 and 7 are used for communication with programs running on the controller. Selecting the required channel then pressing “OK” will start a terminal tool on the selected channel. Only one terminal tool (or keypad tool) can be connected to a channel at one time.



When *Motion Perfect* is in disconnected mode then, on starting the terminal tool you will be presented with a dialogue to select the communications port for connection. The available ports will be those previously configured in the communications options tool. Selecting an interface (probably COM1) and pressing “OK” will start a terminal tool. Only one terminal tool can be used at any one time when operating in disconnected mode.





```
Terminal: SerialCOM1:38400,8,e,1
Terminal Edit Options
MC Multitasking BASIC Version 1.96
Created Jan 16 2009 @ 10:18:25

Axes initialised: 3
AXIS 0 = Servo axis
AXIS 1 = Encoder axis
AXIS 2 = Virtual axis

Input numbers : 0-3; 8-11
Output numbers : 8-11
Virtual IO : 4-7; 12-15

CAN port baud : 500K

Analog Inputs: 0

Processes initialised: 4
Test APPS RAM : Success
Reading FLASH table

>>
VT100 Log: Off SerialCOM1:38400,8,e,1
```

Terminal Menus

- Terminal This controls terminal logging and scripting.
- Edit This has cut and paste as well as clear screen operations.
- Options This controls the terminal emulation (ASCII or VT100) and the line length and number of lines buffered for display.

Terminal Logging

When logging is active all the data displayed on the terminal is also written to a file. The name of the log file is displayed in the status bar at the bottom of the terminal window.

Terminal Scripting

Introduction:

Motion Perfect 2 has built in support for simple terminal scripting. This allows the user to write files of commands and then send the file contents to the controller in a single operation. In addition to the commands to be sent to the controller there are some extra commands which are used by *Motion* Perfect to control the running of the script.

Interaction with the controller:

Command lines are sent to the controller one at a time in sequence. *Motion Perfect* sends a command then waits to receive a prompt (>>) before sending the next one.

To not wait for a prompt put the two character sequence \& on the end of the line. These extra characters are not sent to the controller.

Script commands:

Script commands control the running of the script. All script commands start with two colons. The following commands are valid:

Command	Parameter	Description
<code>::Timeout</code>	timeout in seconds	Changes the time <i>Motion Perfect</i> waits for a prompt to be returned. The default value is 10 seconds.
<code>::Wait</code>	wait time in seconds	Wait and do nothing for the given time

Example:

```
::Timeout 55
```

sets the timeout to 55 seconds

Tests

Special support has been added in order to enable the use of scripts for testing purposes. The response from a command can be tested by *Motion Perfect* and the results written to a log file. A test is written on the line after the one whose response is to be tested and consists of a single ^ character followed by a list of alternative responses separated by single | characters. The comparison is done as a string comparison after all leading and trailing spaces have been removed.

Example:

```
^12.0000|13.0000
```

gives a **PASS** if the returned string is “12.0000” or “13.0000”, otherwise a **FAIL**.

The **PASS** or **FAIL** state of each test is logged in the log file and a summary of passes and failures is given at the end.

Editing Scripts:

To edit or write a new script, select Terminal/Script/Edit from the terminal window menu.

Running Scripts:

To run a script normally, select Terminal/Script/Run from the terminal window menu. This does not produce a log of what has happened.

To run a script with full logging, select Terminal/Script/Run logged from the terminal window menu. The log will contain a full log of what has happened including test results.

To run a script in test mode, select Terminal/Script/Run Test from the terminal window menu. This will produce a log containing only test failures and a **PASS/FAIL** summary.

Axis Parameters

The Axis Parameters window enables you to monitor and change the motion parameters for any axis on the controller. The window is made up of a number of cells, separated into two banks, bank 1 at the top and bank 2 at the bottom:

Bank 1 contains the values of parameters that may be changed by the user.

Bank 2 contains the values of parameters that cannot be changed by the user, as these values are set by the system software of the *Motion Coordinator* as it processes the TrioBASIC motion commands and monitors the status of the external inputs.

The black dividing bar that separates the two banks may be repositioned using the mouse to redistribute the space occupied by the different banks, for example to allow the user to shrink the window and view other windows whilst still watching the bank 2 information.

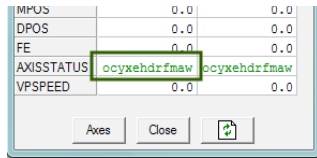
When there are more parameters in a bank that can be shown in the window a scroll bar will appear beside that bank so that the user can scroll up and down the parameter list to see the required values.

The user can select different parameters using the cursor keys or using the mouse. Multiple items may be selected by pressing the shift key and then using the cursor keys or the clicking the mouse to select a different cell, or by pressing the left mouse button in the start cell and the moving the mouse to select the last cell in the selection. Functions may be implemented in the future that work on a selection of multiples cells.

When the user changes the **UNITS** parameter for any axis, all the data for this axis is re-read as many of the parameters, such as the **SPEED**, **ACCEL**, **MPOS**, etc., are adjusted by this factor to be shown in user units.

In the *Motion Perfect* axis parameter screen the **AXISSTATUS** parameter is displayed as a series of characters, ocyxehdrfmaw.

	Axis 1	Axis 2
ATYPE	Servo 32	Servo 32
P_GAIN	1.0	1.0
I_GAIN	0.0	0.0
D_GAIN	0.0	0.0
OV_GAIN	0.0	0.0
VFF_GAIN	0.0	0.0
UNITS	1.0	1.0
SPEED	1000.0	1000.0
ACCEL	10000.0	10000.0
DECEL	10000.0	10000.0
CREEP	100.0	100.0
JOGSPEED	100.0	100.0
FELIMIT	20000.0	20000.0
DAC	0	0
SERVO	0	0
REPDIST	200000000000	200000000000
FWD_IN	-1	-1
REV_IN	-1	-1
DAT_IN	-1	-1
FH_IN	-1	-1
FSLIMIT	200000000000	200000000000
RSLIMIT	-2.000000e11	-2.000000e11
MTYPE	IDLE	IDLE
NTYPE	IDLE	IDLE
MPOS	0.0	0.0
DPOS	0.0	0.0
FE	0.0	0.0
AXISSTATUS	ocyxehdrfmaw	ocyxehdrfmaw
VPSPEED	0.0	0.0



These characters represent **AXISSTATUS** bits in order, as follows:-

char	status bit
w	Warning FE Range
a	Drive Comms Error
m	Remote Drive Error
f	Forward Limit
r	Reverse Limit
d	Datum Input
h	Feed Hold Input
e	Following Error
x	Forward Soft Limit
y	Reverse Soft Limit
c	Cancelling Move
o	Encoder Overcurrent

Parameter Screen Options

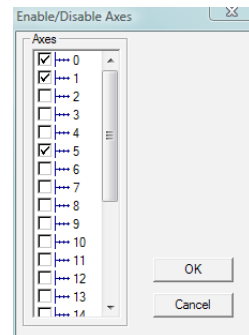
Select Axes


This shows a dialogue that allows the user to select the axes for which the data will be displayed.

The axes set by the last Create Startup, Jog Axes window or Axes Parameters window will be displayed by default.

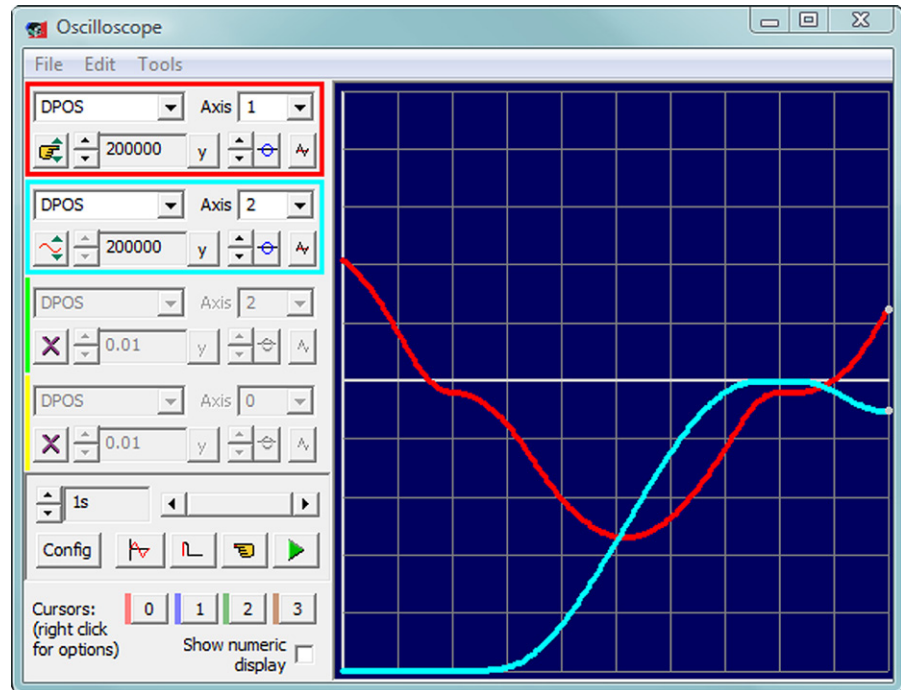
Refresh Display

In order to minimise the load placed upon the controller communications, the parameters in the bank 1 section are only read when the screen is first displayed or the parameter is edited by the user. It is possible that if a parameter is changed in a user program then value displayed may be incorrect. The refresh button will force *Motion Perfect* to read the whole selection again.



 If there is any possibility that a program has changed any of the parameters then you should ensure that you refresh the display before making changes.

Oscilloscope



The software oscilloscope can be used to trace axis and motion parameters, aiding program development and machine commissioning.

There are four channels, each capable of recording at up to 1000 samples/sec, with manual cycling or program linked triggering.

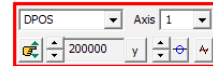
The controller records the data at the selected frequency, and then uploads the information to the oscilloscope to be displayed. If a larger time base value is used, the data is retrieved in sections, and the trace is seen to be plotted in sections across the display. Exactly when the controller starts to record the required data depends upon whether it is in manual or program trigger mode. In program mode, it starts to record data when it encounters a **TRIGGER** instruction in a program running on the controller. However, in manual mode it starts recording data immediately.

Controls

There are four groups of controls, one for each of the oscilloscope's four channels, a group of horizontal function controls and a group to control up to four cursors.

Oscilloscope Channel Controls

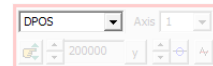
The controls for each of the four channels are grouped together and are surrounded by a coloured rectangle if the channel is **ON**, or a coloured bar to the left of the group if the channel is **OFF**. The colour is the same as the trace for that channel.



The group contains controls for channel operating mode, parameter selection and scaling.

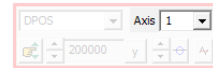
Parameter:

The parameters which the oscilloscope can record and display are selected using the pull-down list box in the upper left hand corner of each channel control block. Depending upon the parameter chosen, the next label switches between 'axis' or 'ch' (channel). This leads to the second pull-down list box which enables the user to select the required axis for a motion parameter, or channel for a digital input/output or analogue input parameter. It is also possible to plot the points held in the controller table directly, by selecting the 'TABLE' parameter, followed by the number of a channel whose first/last points have been configured using the advanced options dialogue. If the channel is not required then 'NONE' should be selected in the parameter list box.



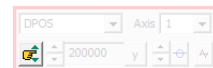
Axis / Channel Number:





A pull-down list box which enables the user to select the required axis for a motion parameter, or channel for a digital input/output or analogue input parameter. The list box label switches between being blank if the oscilloscope channel is not in use, 'axis' if an axis parameter has been selected, or 'ch' if a channel parameter has been selected.



Operating Mode:

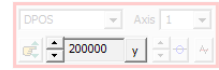
The channel operating mode controls how the trace is displayed and scaled



-  Trace off - no data gathered, trace not displayed
-  Automatic Scaling - data gathered - trace automatically scaled to fit display
-  Manual Scaling - data gathered - trace manually scaled
-  Frozen - no data gathered - trace displayed as it was when frozen

Vertical Scaling:

In automatic mode the oscilloscope calculates the most appropriate scale when it has finished recording, prior to displaying the trace. The value shown is the value calculated by the oscilloscope.



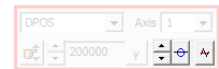
In manual mode the user selects the scale per grid division.

The vertical scale is changed by pressing the up/down scale buttons on the left side of the current scale text box.

Some parameters can be scaled by units. This is controlled by the button to the right of the scale box. In the up position the trace is scaled in raw units as gathered internally in the *Motion Coordinator*. In the down position the raw value is divided by the value of **UNITS** before being displayed. If a parameter cannot be scaled by **UNITS** this control is greyed out.

Channel Trace Vertical Offset:

There are three controls which control the vertical offset of the trace:





The vertical offset buttons are used to move a trace vertically on the display. This control is of particular use when two or more traces are identical, in which case they overlay each other and only the uppermost trace will be seen on the display.



This clears the vertical offset.



When in the up position , only manual offset is applied.

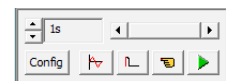
When in the down position  the trace is offset so that the average value of the trace is in the centre of the screen. This is equivalent to AC coupling on a conventional oscilloscope.

Oscilloscope Horizontal Controls

The oscilloscope horizontal controls appear in the lower left of the oscilloscope window. From here you can control such aspect as the timebase, triggering modes and memory used for the captured data.

Timebase:

The required time base is selected using the up/down scale buttons on the left side of the current time base scale text box. The value selected is the time per grid division on the display.



If the time base is greater than a predefined value, then the data is retrieved from the controller in sections (as opposed to retrieving a complete trace of data at one time.) These sections of data are plotted on the display as they are received, and the last point plotted is seen as a white spot.

After the oscilloscope has finished running and a trace has been displayed, the time base scale may be changed to view the trace with respect to different horizontal time scales. If the time base scale is reduced, a section of the trace can be viewed in greater detail, with access provided to the complete trace by moving the horizontal scrollbar.

Horizontal scrollbar:

Once the oscilloscope has finished running and displayed the trace of the recorded data, if the time base is changed to a faster value, only part of the trace is displayed. The remainder can be viewed by moving the thumb box on the horizontal scrollbar.



Additionally, if the oscilloscope is configured to record both motion parameters and plot table data, then the number of points plotted across the display can be determined by the motion parameter. If there are additional table points not visible, these can be brought into view by scrolling the table trace using the horizontal scrollbar. The motion parameter trace does not move.

Horizontal Display Mode:

Button up  = One Shot Trigger Mode.

In one-shot mode, the oscilloscope runs until it has been triggered and one set of data recorded by the controller, retrieved and displayed.

Button down  = Continuous (Auto-repeat) Trigger Mode.

In continuous mode the oscilloscope continues running and retrieving data from the controller each time it is re-triggered and new data is recorded. The oscilloscope continues to run until the trigger button is pressed for a second time.

One Shot / Continuous Trigger Mode:

Button up  = One Shot Trigger Mode.

In one-shot mode, the oscilloscope runs until it has been triggered and one set of data recorded by the controller, retrieved and displayed.

Button down  = Continuous (Auto-repeat) Trigger Mode.

In continuous mode the oscilloscope continues running and retrieving data from the controller each time it is re-triggered and new data is recorded. The oscilloscope continues to run until the trigger button is pressed for a second time.

Manual/Program Trigger Mode:

The manual/program trigger mode button toggles between these two modes. When pressed, the oscilloscope is set to trigger in the program mode, and two program listings can be seen on the button. When raised, the oscilloscope is set to the manual trigger mode, and a pointing hand can be seen on the button.

Button up = Manual Trigger Mode:


In manual mode, the controller is triggered, and starts to record data immediately the oscilloscope trigger button is pressed.




Button down = Program Trigger Mode:

In program mode the oscilloscope starts running when the trigger button is pressed, but the controller does not start to record data until a **TRIGGER** instruction is executed by a program running on the controller. After the trigger instruction is executed by the program, and the controller has recorded the required data. The required data is retrieved by the oscilloscope and displayed.

The oscilloscope stops running if in one-shot mode, or it waits for the next trigger on the controller if in continuous mode

Trigger Button:

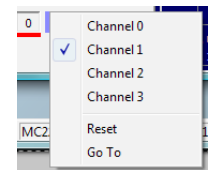
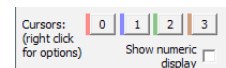
When the trigger button  is pressed the oscilloscope is enabled. If it is manual mode the controller immediately commences recording data. If it is in program mode then it waits until it encounters a trigger command in a running program.

After the trigger button has been pressed, it changes to  (stop) whilst the oscilloscope is running. If the oscilloscope is in the one-shot mode, then after the data has been recorded and plotted on the display, the trigger button returns to  indicating that the operation has been completed. The oscilloscope can be halted at any time when it is running by pressing the  button.

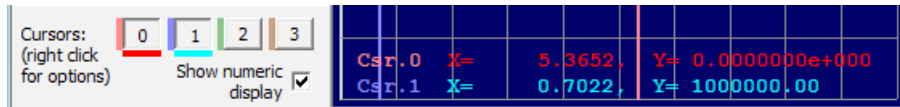
Oscilloscope Cursors

The cursor bars are enabled/disabled by clicking on one of the the cursor buttons which shows/hides the corresponding cursor. A cursor can be moved by positioning the mouse cursor over the required bar, holding down the left mouse button, and dragging the bar to the required position. Cursors are automatically allocated to the first channel currently enabled. To allocate a cursor to a different channel, right click on its button and choose the desired channel from the pop-up menu. When a cursor is active a coloured bar representing the channel to which the cursor has been allocated is displayed under the cursor's button.

The cursor (right click) menu allows the user to assign the cursor to a channel and also contains Reset which resets the cursor position to a position close to the start of the display and Go To which scrolls the display so that the cursor is visible (only if zoomed in).

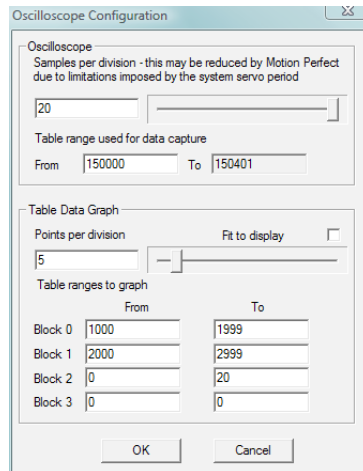


If the Show numeric display box is checked then the numeric display is enabled, this shows maximum and minimum values for all enabled traces if there are no cursors active or the positions of the active cursors if there are cursors active.



Capture Configuration

When the options button is pressed the advanced oscilloscope configuration settings dialogue is displayed, as shown below. Click the mouse button over the various controls to reveal further information.



Samples per division:

The oscilloscope defaults to recording five points per horizontal (time base) grid division. This value can be adjusted using the adjacent scrollbar.

To achieve the fastest possible sample rate it is necessary to reduce the number of samples per grid division to 1, and increase the time base scale to its fastest value (1 servo period per grid division).

It should be noted that the trace might not be plotted completely to the right hand side of the display, depending upon the time base scale and number of samples per grid division.

Oscilloscope Table Values:

The controller records the required parameter data values in the controller as table data prior to uploading these values to the scope. By default, the lowest oscilloscope table value used is zero. However, if this conflicts with programs running on the controller which might also require this section of the table, then the lower table value can be reset.

The lower table value is adjusted by setting focus to this text box and typing in the new value. The upper oscilloscope table value is subsequently automatically updated (this value cannot be changed by the user), based on the number of channels in use and the number of samples per grid division. If an attempt is made to enter a lower table value which causes the upper table value to exceed the maximum permitted value on the controller, then the original value is used by the oscilloscope.

Table Data Graph:

It is possible to plot controller table values directly, in which case the table limit text boxes enable the user to enter up to four sets of first/last table indices.

Parameter Checks:

If analogue inputs are being recorded, then the fastest oscilloscope resolution (sample rate) is the number of analogue channels in msec (ie; 2 analogue inputs infers the fastest sample rate is 2msec). The resolution is calculated by dividing the time base scale value by the number of samples per grid division.

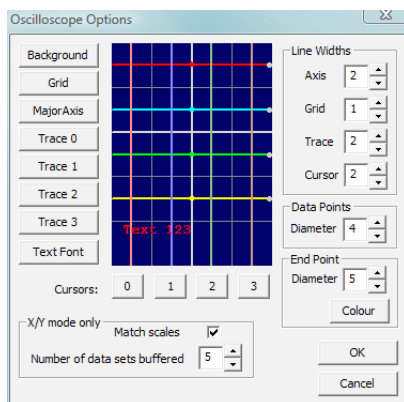
It is not possible to enter table channel values in excess of the controllers maximum **TABLE** size, nor to enter a lower oscilloscope table value. Increasing the samples per grid division to a value which causes the upper oscilloscope table value to exceed the controller maximum table value is also not permitted.

If the number of samples per grid division is increased, and subsequently the time base scale is set to a faster value which causes an unobtainable resolution, the oscilloscope automatically resets the number of samples per grid division.

Before the oscilloscope is triggered a sample quantization check is done to make sure that it is possible to gather the data at the sample interval requested. This may cause the number of samples per division to be adjusted so that the controller is able to gather the data at a sample period which is a whole number of servo cycles.

Options

The oscilloscope options are used to control the visual look of the oscilloscope display. Most colours and line thicknesses can be set, allowing the user to set up the oscilloscope to their own preference.



The X/Y mode only settings control the matching of the two channels used to capture X/Y data and the number of data sets buffered (and displayed) when in X/Y mode.

General Oscilloscope Information

Displaying Controller Table Points:

If the oscilloscope is configured for both table and motion parameters, then the number of points plotted across the display is determined by the time base (and samples per division). If the number of points to be plotted for the table parameter is greater than the number of points for the motion parameter, the additional table points are not displayed, but can be viewed by scrolling the table trace using the horizontal scrollbar.

Data Upload from the controller to the oscilloscope:

If the overall time base is greater than a predefined value, then the data is retrieved from the controller in blocks, hence the display can be seen to be updated in sections. The last point plotted in the current section is seen as a white spot.

If the oscilloscope is configured to record both motion parameters, and also to plot table data, then the table data is read back in one complete block, and then the motion parameters are read either continuously or in blocks (depending upon the time base).

Even if the oscilloscope is in continuous mode, the table data is not re-read, only the motion parameters are continuously read back from the controller.

Enabling/Disabling of oscilloscope controls:

Whilst the oscilloscope is running all the oscilloscope controls except the trigger button are disabled. Hence, if it is necessary to change the time base or vertical scale, the oscilloscope must be halted and re-started.

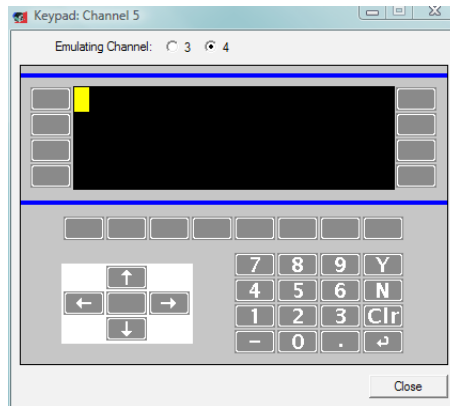
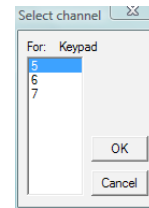
Display accuracy:

The controller records the parameter values at the required sample rate in the table, and then passes the information to the oscilloscope. Hence the trace displayed is accurate with respect to the selected time base. However, there is a delay between when the data is recorded by the controller and when it is displayed on the oscilloscope due to the time taken to upload the data via the communications link.

Keypad Emulation

The keypad requires one of the user communications channels, and so you will be prompted for the channel to use.

If the specified channel is already in use, either by another keypad or a terminal window, the window will not open. Once a channel has been reserved then the keypad will be shown.



In the TrioBASIC program the channel definition for the commands that are associated with the Keypad must be changed from 3 (or 4) to the channel that corresponds with the channel selected for the emulation. We recommend that the channel assignment be made through a variable, so when time comes to run the program on the real machine, only one program change will be required.

Example:

```
kpd=5
```

```
PRINT #kpd, "Press any key.."
```

Emulating Channel:

The normal operation of the keypad emulation returns the characters as if they were read from channel #3 with the **DEFKEY** translation. Alternatively, the *Motion Coordinator* can read the characters returned directly from the Keypad using channel 4. If the emulate #4 codes is selected then the keypad emulation will return the raw characters.



It is only possible to emulate the default DEFKEY table.

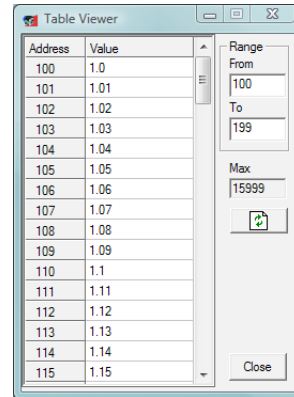
Key Functions

menu keys	This is a keypad menu key. Normally it is associated with a message on the display. This button can only be pressed by clicking the mouse over it.
function keys 1-8	This is the keypad function key 1. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '1' - '8' keys in the QWERTY area of the PC keyboard.
number keys	This is a keypad number key. It can be pressed by clicking the mouse over it or using the corresponding number in the numerical keypad of your PC keyboard.
Y/N keys	This is the keypad 'Y' and 'N' keys. This is usually used to respond YES or NO to some question on the display. It can be pressed by clicking the mouse over it or using the 'Y'/'N' keys in the QWERTY area of the PC keyboard.
CLR key	This is the keypad 'CLR' key. This is usually used to perform some form of CANCEL operation. It can be pressed by clicking the mouse over it or using the 'ESC' in the QWERTY area of the PC keyboard.
Return key	This is the keypad Return key. This is usually used to perform some form of ACCEPT operation. It can be pressed by clicking the mouse over it or using the 'Enter' in the QWERTY area or numerical keypad of the PC keyboard.
- key	This is the keypad '-' key. This is usually used for entering negative numbers. It can be pressed by clicking the mouse over it or using the '-' in the QWERTY area or numerical keypad of the PC keyboard.
. key	This is the keypad '.' key. This is usually used for entering fractional numbers. It can be pressed by clicking the mouse over it or using the '.' in the QWERTY area or numerical keypad of the PC keyboard.
arrow keys	This is the keypad up arrow key. This is usually used to select between options on the display. It can be pressed by clicking the mouse over it or using the appropriate arrow key of the PC keyboard.
centre button	This is the keypad centre key. It can only be pressed by clicking the mouse over it.

Table / VR Editor

The Table and VR Editor tools are very similar. You are presented with a spreadsheet style interface to view and modify a range of values in memory.

To modify a value, click on the existing value with the mouse and type in the new value and press return. The change will be immediate and can be made whilst programs are running.



Options

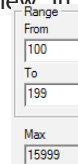
Range:

In both tools you have the option to set the start and end of the range to view. In the Table view tool the max value displays the highest value you can read (this is the system parameter `TSIZE`).

If the range of values is larger than the dialogue box can display, then the list will have a scrollbar to enable all the values to be seen.

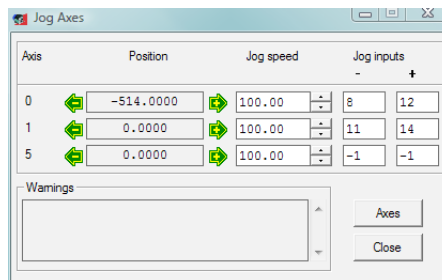
Refresh Button:

This screen does not update automatically, so if a Table or VR is changed by the program you will not see the new value until you refresh the display.



Jog Axes

This window allows the user to move the axes on the *Motion Coordinator*.



This window takes advantage of the bi-directional I/O channels on the *Motion Coordinator* to set the jog inputs. The forward, reverse and fast jog inputs are identified by writing to the corresponding axis parameters and are expected to be connected to NC switches. This means that when the input is on (+24V applied) then the corresponding jog function is DISABLED and when the input is off (0V) then the jog function is ENABLED.

The jog functions implemented here disable the fast jog function, which means that the speed at which the jog will be performed is set by the **JOGSPEED** axis parameter. What is more this window limits the jog speed to the range 0..demand speed, where the demand speed is given by the **SPEED** axis parameter.

Before allowing a jog to be initiated, the jog window checks that all the data set in the jog window and on the *Motion Coordinator* is valid for a jog to be performed.

Jog Reverse

This button will initiate a reverse jog. In order to do this, the following check sequence is performed:

If this is a **SERVO** or **RESOLVER** axis and the servo is off then set the warning message

If this axis has a daughter board and the WatchDog is off then set the warning message

If the jog speed is 0 the set the warning message

If the acceleration rate on this axis is 0 then set the warning message

If the deceleration rate on this axis is 0 then set the warning message

If the reverse jog input is out of range then set the warning message

If there is already a move being performed on this axis that is not a jog move then set the warning message

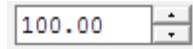
If there were no warnings set, then the message “Reverse jog set on axis?” is set in the warnings window, the **FAST _ JOG** input is invalidated for this axis, the **CREEP** is set to the value given in the jog speed control and finally the **JOG _ REV** output is turned off, thus enabling the reverse jog function.

Jog Forward

This button will initiate a forward jog. In order to do this, a check sequence identical to that used for Jog Reverse is performed.

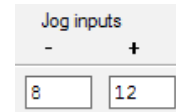
Jog Speed

This is the speed at which the jog will be performed. This window limits this value to the range from zero to the demand speed for this axis, where the demand speed is given by the **SPEED** axis parameter. This value can be changed by writing directly to this control or using the jog speed control. The scroll bar changes the jog speed up or down in increments of 1 unit per second



Jog Inputs

These are the inputs which will be associated with the forward / reverse jog functions.

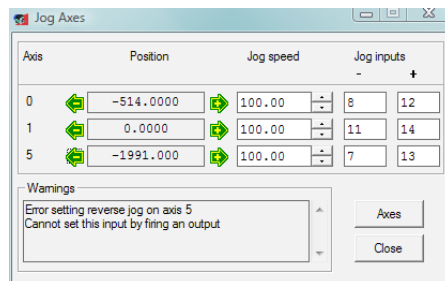


They must be in the range 8 to the total number of inputs in the system as the input channels 0 to 7 are not bi-directional and so the state of the input cannot be set by the corresponding output.

The input is expected to be **ON** for the jog function to be disabled and **OFF** for the reverse jog to be enabled. In order to respect this, when this is set to a valid input number, the corresponding output is set **ON** and then the corresponding **REV _ JOG** axis parameter is set.

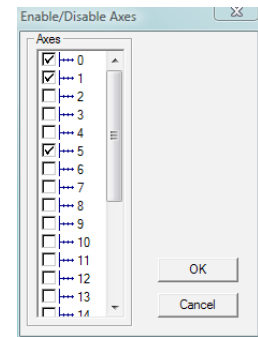
Warnings

This shows the status of the last jog request. For example, the screen below shows axis 0 with IO channel 7 selected. This is an Input-only channel and therefore cannot be used in the jog screen.



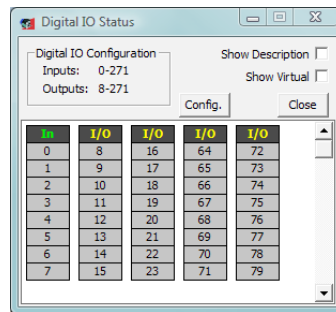
Axes

This displays an axis selector box which enables the user to select the axis to include in the jog axes display. By default, the physical axes fitted to the controller will be displayed.



Digital IO Status

This window allows the user to view the status of all the IO channels and toggle the status of the output channels. It also optionally allows the user to enter a description for each I/O line.



Digital Inputs

This shows the total number of input channels on the *Motion Coordinator*.

Digital Outputs

This shows the total number of output channels on the *Motion Coordinator*.

I/O Mimic

Input Banks (Green)

The LEDs show the status of the dedicated input channels. If an LED is **ON** then the corresponding input is **ON**. If an LED is **OFF** (grey) then the corresponding input is **OFF**.

I/O Banks (Yellow)

The LEDs show the status the bi-directional I/O channels. If an LED is **ON** then the corresponding input is **ON**. If an LED is **OFF** (grey) then the corresponding input is **OFF**. Under normal conditions the input status mimics the output status, except:

- If this input is connected to an external 24V then it may be **ON** without the corresponding output being **ON**.
- If the output chip detects an overcurrent situation, then the output chip will shut down and so the outputs will not be driven, even though they may be turned on.

If the LED is clicked with the mouse the status corresponding output channel is toggled, i.e. if the LED is **OFF** then the output will be turned **ON**, if the LED is **ON** then the output channel will be turned **OFF**.

Output Banks (Orange)

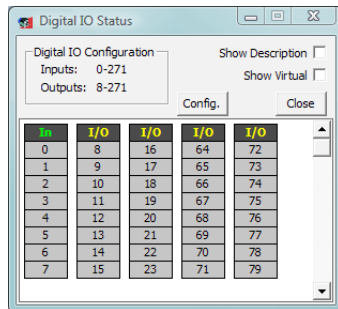
These banks of LEDs show the status the dedicated outut channels. If an LED is **ON** then the corresponding output is **ON**. If an LED is **OFF** (grey) then the corresponding output is **OFF**.

Virtual I/O Banks (Cyan)

These perform in a similar way to the I/O banks except for the fact that the I/O lines are not connected to any hardware. The state of the input always reflects the state of the output.

Descriptions

Checking the Show Description check box will show I/O line descriptions which are editable by the user. The descriptions are stored in the project file.

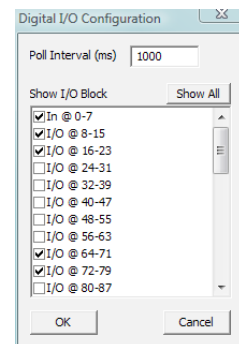


Configuration

The Config. button allows the user to show or hide I/O banks using the Digital I/O Configuration dialogue.

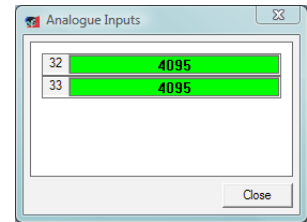
If the box next to an I/O bank is checked it will be shown, otherwise it will be hidden.

The I/O poll time can also be set from this dialogue.



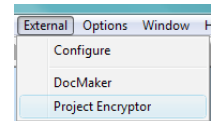
Analogue Input Viewer


The analogue input viewer is only available if the system has analogue inputs. It displays the input values of all analogue inputs in the system using a bar-graph with numeric display. All inputs have the range -2048 to 2047.



Linking to External Tools

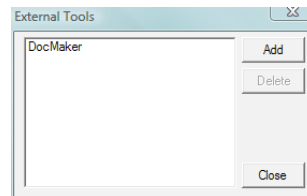
The **External** menu in *Motion* Perfect allows you to run other programs directly from the main *Motion* Perfect menu. In the example shown here, the menu has been configured to launch two other Trio applications, CAD2Motion and DocMaker. Further information on these applications is given at the end of this chapter.



 *Cad2Motion and DocMaker are available to download from the Trio Website at www.triomotion.com.*

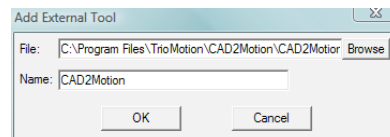
Configuring Items on the External menu

Clicking on the Configure item will bring up a list of all installed applications and from here we can add or delete items from this list.

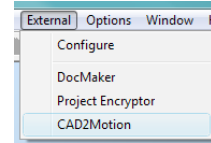


Adding a new programs to the menu

Clicking on the Add button will open the following dialogue:



You can either directly enter the path and program file name in the “File” box, or use the “Browse” option to open up a standard windows file selector box which you can use to locate the file on your computer.



Once you have selected the file, it will automatically appear in the External menu every time you run *Motion Perfect 2*.

Removing program items from the menu

To delete a program from the External menu, you simply need to click on the program name in the list and press the Delete button.

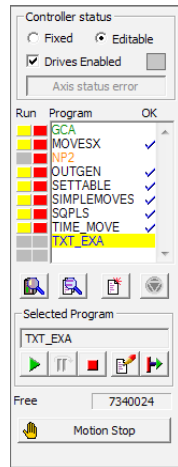


This simply removes the program from the menu. It does NOT affect the original program on disk!

Control Panel

The control panel appears on the left hand side of the main *Motion Perfect* window.

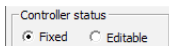
It provides direct links to many of the frequently used operations within *Motion Perfect*, in particular the file and directory functions.



Certain Control Panel Features behave differently on controller without a battery backup. The differences are described later in this section.

Control Panel Features

Fixed/Editable radio buttons



When the project is “fixed”, the programs are copied to the Flash EPROM on the *Motion Coordinator*, the *Motion Coordinator* is set to run from EPROM and the programs cannot be modified by *Motion Perfect*. Usually this is done when the machine programs are completed. The Flash EPROM provides a reliable permanent storage for the programs.

Drives radio button

Drives Enabled Drives Enabled **Drives Disabled** Drives Enabled

This radio button toggles the state of the enable (watchdog) relay on the controller, going between drives disabled (watchdog off) and drives enabled (watchdog on).

The LED mimic next to this control shows the status of the error LED on the *Motion Coordinator*. If it is yellow then the drives are disabled, if it is grey the drives are enabled and if it is flashing then there has been a motion error on at least one axis of the controller.

Axis Status Error Axis status error

This will normally be greyed out unless a motion error occurs on the controller.

When an error does occur you can use this button to clear the error condition.

Program directory

This is a scrollable list of the programs on the controller.

The list shows the program name followed by two optional indicators.

The colour of the program name text is black for TrioBASIC programs, red for encrypted programs, green for G-Code programs, blue for text files (not runnable) and orange for IEC programs. The program name is shown in italics if the program is running. The currently selected program is highlighted in the system highlight colour (yellow in the example above). If a program is listed more than one this represents more than one running instance of the same program.

Run	Program	OK
<input checked="" type="checkbox"/>	GCA	<input type="checkbox"/>
<input checked="" type="checkbox"/>	MOVESX	<input type="checkbox"/>
<input checked="" type="checkbox"/>	NP2	<input type="checkbox"/>
<input checked="" type="checkbox"/>	OUTGEN	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	SETTABLE	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	SIMPLEMOVES	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	SQPLS	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	TIME_MOVE	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	TXT_EXA	<input checked="" type="checkbox"/>

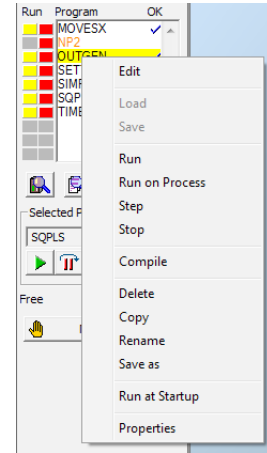
The first optional indicator is the number of the process on which the program is currently running. If it is not running then this space is blank.

The second indicator shows the status of the program. If it has a tick then the program has been compiled successfully and is runnable. If it has a cross then there was an error during the compilation of the program and it cannot be run. The indicator is blank if the program has not been compiled. For programs / files which are not runnable it is always blank.

If a program name is clicked then it will become the selected program if there are no programs running. If there are programs running the select will be ignored.

If a program name is double clicked then it will be opened for editing assuming that there are no programs running. If there are programs running then the editor will open in read-only mode.

Right clicking on an entry in the program list causes a pop-up menu to appear allowing easy access to operations commonly performed on programs. The right click operation highlights the entry in the program list under the cursor whilst the pop-up menu is visible. It reverts to the program which is currently selected on the controller when the menu is closed.



Run buttons

The run buttons provide short cut keys for running, stopping and single stepping programs. They can be in one of three states, red, green or yellow.







- | | |
|--------|---|
| RED | Click on the red button to start the corresponding program running. The button will turn Green. * |
| GREEN | Click on the green button to stop the corresponding program. The button will turn red. * |
| YELLOW | Click on the yellow button to single step through the program. |

* If the program goes into trace mode, through the use of a trace button, the selected program step button, the debug option of the program menu, the debug button on the tool bar or a **TRON/STEPLINE** command in a program or the terminal window, then the red/green run button will turn yellow. If the button is clicked when it is yellow then the program will be stepped one line.

General Options



-  Show Controller Configuration
-  Show a full directory of all programs in memory
-  Create a New Program. Same as the Program menu item.
-  **HALT** - Stop all programs which are running

Selected Program

The text box displays the currently selected program and the buttons below, the operations which can be performed on that program.



From left to right they are:

Run, Step, Stop, Edit and Power Up Mode.

Free Memory Free 7601215

Shows the total free memory available on the controller.

Motion Stop  Motion Stop

Stops all running programs, cancels moves on all axes and disables the watchdog relay.

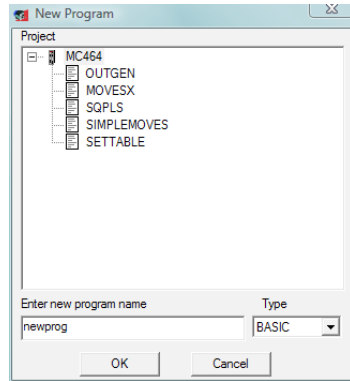


Motion Stop is a software function. It is not a substitute for a hardware E_Stop circuit and should not be used as an emergency stop.

Creating and Running a program

In order to create a new program on the controller, you must first have an active project. If you have already connected to the controller then you can use the default project which was created at this time.

You will be presented with a program selector dialogue and prompted to enter a name for your program file and a program type (BASIC, G-Code etc). It is a good idea to make this name representative of the task performed by the program, for example “mmi”, “motion”, “logic” or something similar. In the following example, we will add a program called “test” to the current project.



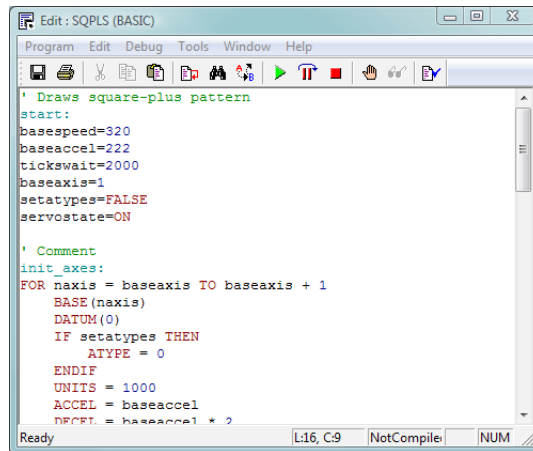
Once you have created a new program it will be added to both the controller and the *Motion Perfect* project file. You can now edit the file in *Motion Perfect Editor*, if appropriate for the program type selected, which will have been started up automatically when the new program was created.

The *Motion Perfect Editor*

You can start the Editor from the main Program Menu, the Edit button in the program section of the control panel or by right clicking in an entry in the control panel program list and selecting Edit from the pop-up menu.

If you launch the editor from the control panel it will start immediately. From the program menu you will first be prompted with a program selector dialogue to confirm the file you wish to edit.

The *Motion Perfect Editor* is designed to operate in a similar manner to any simple text editor found on a PC. Standard operations such as block editing functions, text search and replace and printing are all supported and conform to the standard Windows shortcut keys. In addition it provides TrioBASIC syntax highlighting, program formatting and program debugging facilities.



```

Edit: SQPLS (BASIC)
Program Edit Debug Tools Window Help
' Draws square-plus pattern
start:
basespeed=320
baseaccel=222
tickswait=2000
baseaxis=1
setatypes=FALSE
servostate=ON
' Comment
init_axes:
FOR naxis = baseaxis TO baseaxis + 1
  BASE(naxis)
  DATUM(0)
  IF setatypes THEN
    ATYPE = 0
  ENDIF
  UNITS = 1000
  ACCEL = baseaccel
  DECFI = baseaccel * 2
Ready L:16, C:9 NotCompile NUM

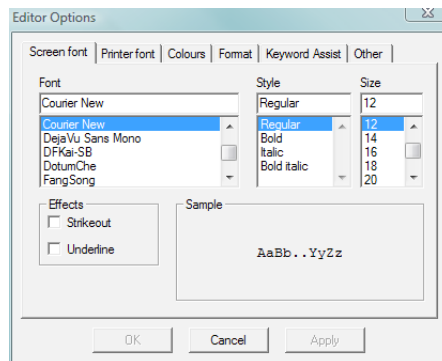
```

Editor Options

Options for the editor are controlled by the Editor Options dialogue. This can be opened by selecting Options/Editor from *Motion Perfect's* main menu. The dialogue allows to user to change the fonts used for screen display and printing, the colours used for syntax and line highlighting and the spacings used when automatically formatting a program.

Screen Font

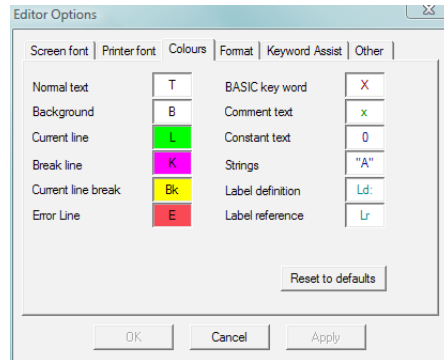
This is the font used by *Motion Perfect* to display text in the editor window on the screen. The font is restricted to fixed pitch fonts only.



Printer Font

This is the font used by *Motion Perfect* to print program listings. The font is restricted to fixed pitch fonts only.

Colours



Colours can be specified for the following:

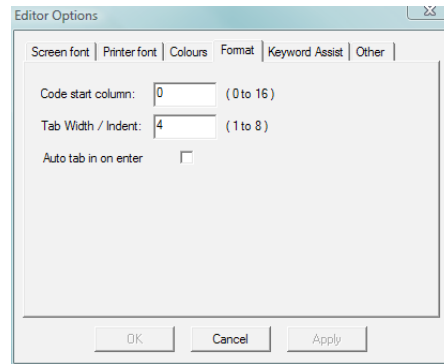
- Normal Text - Text which is not highlighted using syntax highlighting.
- Screen Background.
- Current Line (background) - the current line during debugging.
- Break Line (background) - a line containing a break (**TRON**) command.
- Current Line Break (background) - a line containing a break command which is also the current line.
- Error Line (background) - The first line containing a compilation error.
- BASIC key word - A key word in the TrioBASIC language, usually a command or some type of system variable.
- Comment Text
- Constant Text - text making up a constant value (number).
- Strings
- Label Definition - where a program label is defined.
- Label Reference - where a jump or branch (**GOTO**, **GOSUB** etc.) in program execution is required. The jump or branch changes the execution point to the place where the label is defined.

Format

The format options affect text entry and the automatic reformatting performed by *Motion Perfect*.

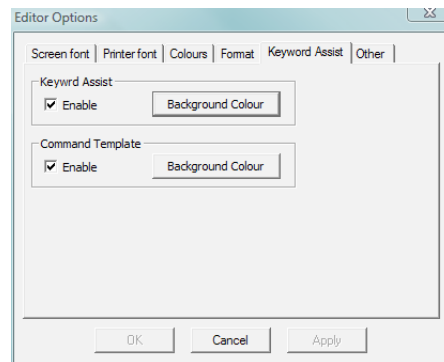
For automatic reformatting the code start column and the tab width are specifiable. As label definitions always start in column 0, the code start column can be used to indent all lines containing code thus making label definitions clearer.

The when Auto tab on enter check box is checked pressing the “enter” key will automatically indent the next (new) line to the same position as the current one.



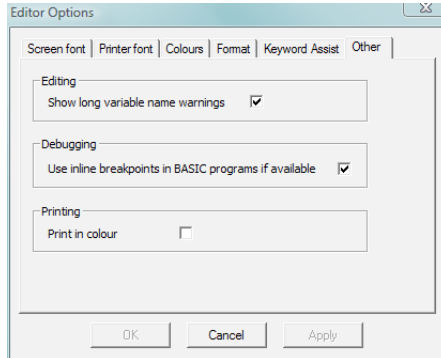
Keyword Assist

This controls the Keyword Assist and Command Template functions of the editor. The functions can be enabled or disabled using the check boxes and the background colour of the popup windows can be selected by clicking on the appropriate Background Colour button.



Other

The other options cover things which do not easily fit into any of the above categories.



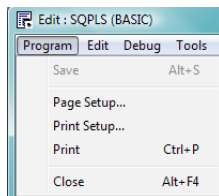
When the Show long variable name warnings box is checked, the editor will give a warning if a variable name exceeds the maximum number of characters which the controller checks for uniqueness of variable names (currently 16).

When the Use inline breakpoints in BASIC programs if available box is checked, the debugger uses inline breakpoints which the controller inserts without modifying the program. If the box is not checked then the debugger inserts a `TRON` statement into the program. The advantage of inline breakpoints is that, because they do not actually modify the program, they can be inserted whilst the program is running (or paused).

When the Print in colour box is checked any printing from the editor will be done in colour if the printer supports it. Otherwise printing is done in monochrome.

Editor Menus

Program



Save

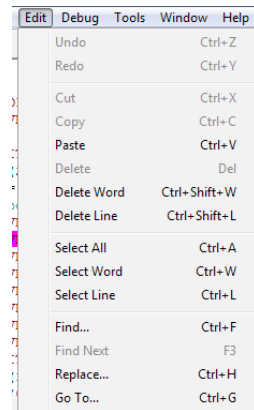
Normally the program is only saved to disk when the editor is closed or a program is run, however if you have modified the program the Save Button will be available and will force *Motion Perfect* to save the file immediately.

Printing

Use Page Setup to set the page margins, Print Setup to configure your printer settings and the Print option to send the program to the printer.

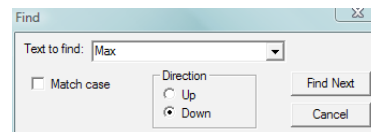
Edit

The edit menu functions are similar to many other text editors and provide the standard block cut/copy/paste operations as well as a simple text find/replace, and various select and delete functions.



Find/Replace

The options for the find & replace dialogues are very similar and feature many of the same options

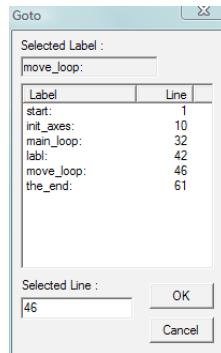


You should enter the text to search for in the “Find What” box, and if using find and replace, the text to replace it with in the “Replace With” box.

Normally the “Case Sensitive” search option is not selected, You should only use this option if you have an exact pattern to match, generally the default option is best.

Goto

The Goto option will bring up the following dialogue:

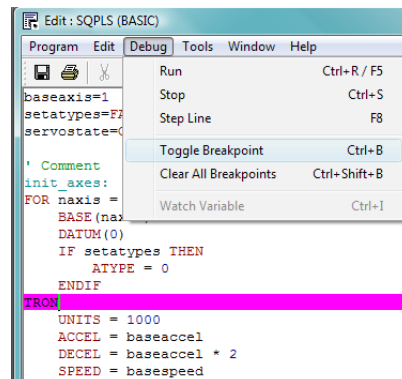


A list of labels defined in the program is displayed. You can either select a label from here, or enter a line number directly in the “Selected Line” field. Pressing the the button at the head of one of the columns in the list will cause the list to be sorted by the values in that column.

Pressing **OK** after a selection has been made will cause the cursor in the editor to jump directly to the beginning of the selected line.

Program Debugger

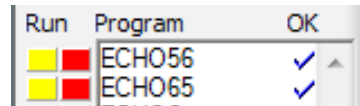
The *Motion* Perfect debugger allows you to run a program directly from the editor window in a special ‘trace mode, executing one line at a time (known as stepping) whilst viewing the line in the window. It is also possible to set breakpoints in the program, and run it at normal speed until it reaches the breakpoint where it will stop, and this line of code will be highlighted in the debug window.



When programs are running on the controller, any open editor windows will automatically switch to Debug Mode and will become read-only. Hence, breakpoints are set in the edit window, and the code viewed in the same window in debug mode when the program is running.

Stepping Through a program

To commence stepping a program:



Use the mouse to press the yellow button alongside the required program name in the list box on the control panel

if the required program is currently selected, press the 'Step' button on the control panel() or use the menu item 'Debug-Step line'

The currently executing line of code is indicated in the debug window by highlighting it with a green background, and a breakpoint is highlighted with a red background.

To continue stepping the program, repeatedly press the yellow button alongside the program name in the list box on the control panel, or press the 'Step' button or the 'F8' function key if the program required is currently selected on the *Motion Coordinator*.

Breakpoints

Breakpoints are special place markers in the code which allow a particular section (or sections) of the program to be identified when debugging the code. If a breakpoint is inserted, the program will pause at that point and return control to *Motion Perfect* where the controller may be interrogated or the program run in step mode as described above.

To insert a breakpoint, first position the text cursor on the line at which you want the break to occur, then use either Ctrl-B or the menu item to insert the breakpoint.

If the editor option Use inline breakpoints in BASIC programs if available has been selected and the controller supports inline breakpoints (this is true of most current controllers) then an inline breakpoint will be inserted. Otherwise the TrioBASIC instruction **TRON** is used to mark a breakpoint and **TROFF** to terminate a 'traced' block.

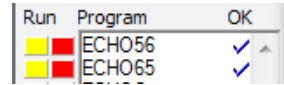


*Inline breakpoints can be added or removed whilst programs are running (or paused). It is not possible to add or remove **TRON** breakpoints whilst any programs are running or are paused.*

Running to a breakpoint

A program can be run to the next break point by:

- using the mouse to press the red button alongside the program name in the list box on the control panel.
- if it is the currently selected program on the *Motion Coordinator*, you can pressing the 'Run' button () on the control panel/editor tool bar, or by using the keyboard <F5> function key.
- by selecting the 'Debug->Run' menu option



Stopping a Program

If it is necessary to stop the program running before it reaches the breakpoint then:

- press the green button alongside the program name (running on the required process) in the list box on the control panel.
- press the stop button () on the control panel if the program is currently selected (this will stop all running copies of the program)
- use the 'Debug-Stop' menu option.




Alternatively all programs can be stopped by pressing the 'Halt' button on the control panel, or selecting the 'Program' 'Halt all programs' menu option, or using the <Ctrl><F> key combination.

Switching a running program into trace mode

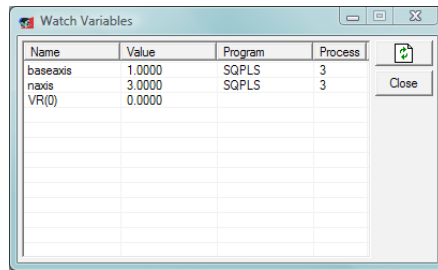
A running program can enter trace (stepping) mode by pressing the yellow button alongside the required program name in the list box on the control panel, or the 'Step' button if the required program is currently selected on the *Motion Coordinator*.

Viewing variables

The Variable Watch Tool is used to view the values of internal program variables and global VR variables. To add a variable to the watch tool highlight the variable name in the program then select Debug / Watch Variable from the editor menu or click on the  tool button in the editor.

Variable Watch Tool

The Variable Watch Tool is used to view the values of internal program variables and global VR variables.




Local variables in a program on a *Motion Coordinator* are held as part of the process information. This allows more than one instance of a program to run concurrently with each instance having its own local variables. Global VR variables have a single instance on the *Motion Coordinator* and allow values to be passed between programs / processes.


Adding Variables

Variables are normally added to the watch tool from the editor but can be added manually by entering the variable name, program name and process number (for local) or VR with number e.g. VR(99) (for global).

Removing Variables

A variable can be removed by deleting its name from the watch tool grid, then pressing the refresh button .

Values

The values are updated every time a program is paused or stepped using the Step button in the editor. They can also be refreshed manually using the refresh button .

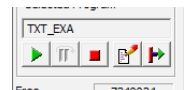
The value of a variable can be changed by entering a new value into the grid. The new value is written to the *Motion Coordinator* when the edit control which appears on the grid is closed (by pressing 'return' or clicking somewhere else in the grid).

Running Programs

You can start/stop programs running in one of four ways:

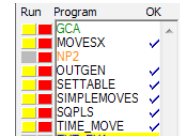
From the control panel

If the program is currently selected (highlighted in the control panel), you can press the green start arrow in the “selected program” box.



From the program list

Pressing the red button to the left of the program name in the list will start it running, the button will change to green and it will then function as a stop button for the same task.



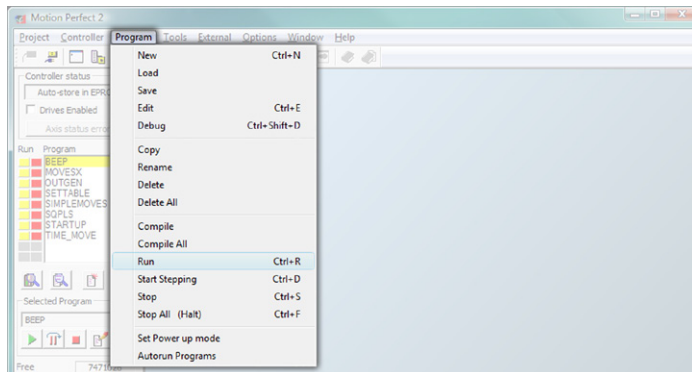
From the editor toolbar

If you have an editor or debug window open for the program you can use either the Debug menu or toolbar buttons to start the program running



From the Program Menu

The program menu provides us with a slightly different option when running the program as we are presented with a program selector box which includes an option to choose which task we want to run the program on.



Making programs run automatically

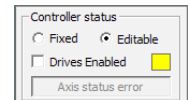
Set Powerup Mode

It is possible to make the programs on the controller run automatically when the system first starts up. From the Program Menu, select “Set Powerup Mode” to open the following dialogue.

Click on the program you want to auto run and a small drop-down list will appear to the right of the window. If you are happy to let the controller allocate which task to run on then you should choose “default” as the process number, otherwise you can specify the task explicitly in the box.

Storing Programs in the Flash EPROM

This is accomplished by selecting the “Fixed” option in the controller status section of the control panel, or the “Fix Program Into EPROM” option from the controller menu.



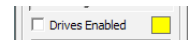
When the controller is fixed into EPROM, the programs actually still run from RAM. The information stored is copied into RAM when the controller is first started, therefore if the controller has been switched off for an extended period, or there is any corruption of the RAM, it will be refreshed with a correct copy of the programs.

When the controller is set to fixed you will not be able to edit any programs. In order to make changes you must select “Editable” from the control panel or “Enable Editing” from the controller menu.

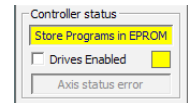
Variations for controllers without battery backup

On those controllers without battery backup, it is essential that you store your programs into the EPROM to avoid loss of data.

The control panel the fixed / editable radio buttons are replaced by a single button labelled “Store Programs into EPROM”.

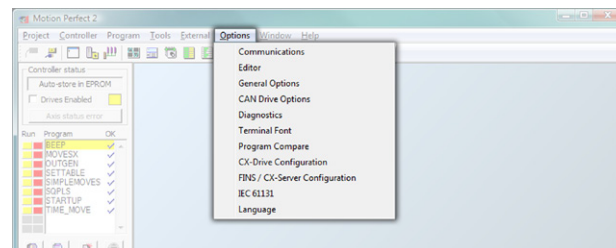


If the programs in memory have been edited, the button will be highlighted to remind you to fix into EPROM before exiting the program.



Configuring The *Motion Perfect 2* Desktop

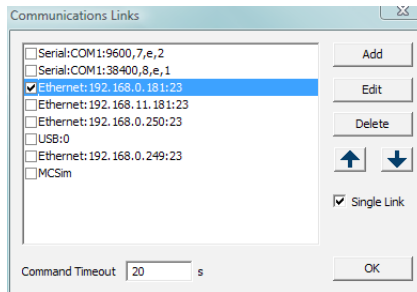
There are a number of ways in which you can configure *Motion Perfect 2* to suit your requirements. The Options menu provides a number of choices:-



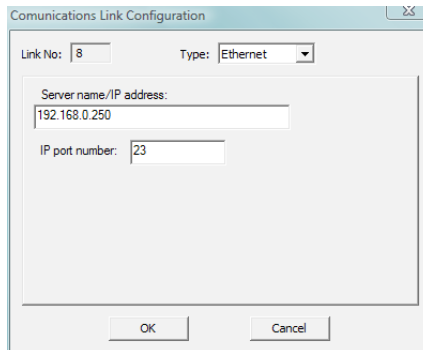
Communications

Set up the default communications device for *Motion Perfect 2* to use.

Motion Perfect 2 needs a connection to the controller in order to operate. This can be an RS-232 serial connection, a USB connection if your controller and PC have USB ports, an ethernet connection if your controller and PC have ethernet ports, or PCI if your controller is PCI based. The Communications Links tool (“Options / Communications” from *Motion Perfect*’s main menu) shows the communication links which have been configured.



If the port you wish to use is not shown, you need to select the Add Port option which will select the following dialogue.



The configuration parameters will change according to the interface type selected (serial, Ethernet etc.).

Select the interface type required, then change the parameters as required.

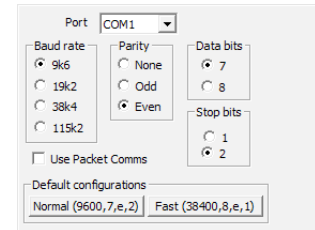
Communications message timeout

Certain operations on the controller may cause the controller to stop communicating for a few seconds. If you find that your PC seems to disconnect often, you can change *Motion Perfect 2's* default timeout value to allow the program to wait for a longer time before disconnecting.

Changing Communications Parameters

Serial

The default settings for serial communications on MC2 series controllers is: 9600 baud, 7 data bits, 2 stop bits, even parity. For MC3 series controllers it is 38400 baud, 8 data bits, 1 stop bit, even parity. If you wish to change these values you can do so with the configure button in the Configure Communications dialogue.



If you change the port setting to anything other than the default, you may encounter problems when the controller is reset because it will revert to the default values.

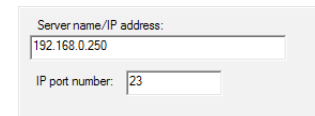
In order to avoid this your controller will need to set the comms parameters within an auto-running program. See the `SETCOM` instruction in the TrioBASIC reference for further information.

USB

There are no configuration options for USB communications.

Ethernet

For Ethernet communications the controller acts as a server and *Motion Perfect* a client. The “Server name / IP Address” should normally be set to the Ethernet IP address of the controller. The IP port is normally set to 23 (telnet port).



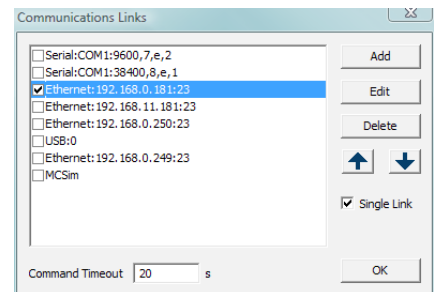
PCI

There are no configuration options for PCI communications.

Active Link Selection

If the “Single Link” box is ticked then it is only possible to select one communications link in the link list. Clicking on the check box of a link in the list will select it (or deselect it) leaving all other links deselected.

If the “Single Link” box is not ticked then clicking on the check box of a link in the list will select it (or deselect it) if it is



already selected). *Motion* Perfect will try all the selected links in the list in order until it finds a controller to connect to. The up and down arraws can be used to move the position of the currently highlighted link entry up and down in the list.

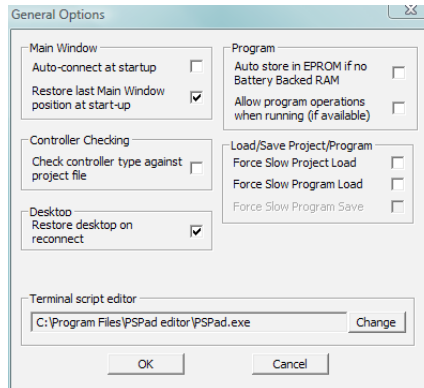
Editor Options

The Editor Options dialogue allows you to modify the appearance of the *Motion* Perfect 2 editor to suit your own personal taste. You can change both the default font used and the colours used by the syntax highlighting feature. See the Editor section for more details.

General Options

This dialogue allows the user to change a number of options relating to how *Motion* Perfect 2 starts up and handles projects.

When you select General Options you will be presented with the following screen.



The check-boxes enable the following features:

Auto-connect at startup

If this option is selected then *Motion* Perfect will try and connect automatically when it is started up. This is the default behaviour and is the same as older versions of *Motion* Perfect which did not have this option.

Restore last Main Window position at startup

If this option is selected *Motion* Perfect will save the Main Window position when it is closed and restore it when it is started again. If it is not selected, the Main Window will open maximized.

Check controller type against project file

Checks the type of the connected controller used against the one in the project file when a Check Project operation is performed.

Restore desktop on reconnect

If this option is selected, the program will attempt to automatically save the desktop layout when disconnecting from the controller.

When you reconnect, *Motion Perfect 2* will automatically restore the last desktop layout saved.

Auto EPROM if no Battery Backed RAM

If the controller does not feature battery-backed RAM memory and this option is selected, the program will attempt to automatically save the controller memory to EPROM before disconnecting from the controller.

Allow program program operations when running

If this option is selected, some program operations such as loading or editing can be used when programs are running (normally none are allowed when programs are running). It is advisable **NOT** to use this option as it is only available on some controllers and can cause *Motion Perfect* to become very slow and unreliable.

Force Slow Project Load

This option causes *Motion Perfect* to use the old, slow method of loading a project even on controllers which support the fast loadin method. This should only be used if fast loading is unreliable.

Force Slow Program Load

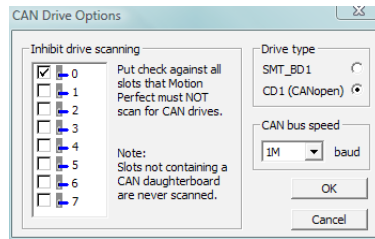
This option causes *Motion Perfect* to use the old, slow method of loading a program even on controllers which support the fast loadin method. This should only be used if fast loading is unreliable.

Terminal script editor

This allows the user to select a program with which to edit terminal scripts. The default is Windows Notepad.

CAN Drive Options

The CAN Drive Options dialogue controls how *Motion Perfect* interacts with CAN based drives.



Scanning for Drives

Normally *Motion Perfect* will scan all available **CAN** interfaces for drives. It is possible to inhibit scanning for drives on one or more of the **CAN** interfaces on the controller by ticking the appropriate check box. This is used when other **CAN** devices connected to a **CAN** interface (drives and other devices should not be connected to the same **CAN** interface).

CAN Drive Type

Motion Perfect currently only Infranor SMT_DB1 and Infranor CD1 drives. When the CD1 drive type is selected this gives limited functionality with other CANopen based drives.

CAN Bus Speed

This allows the **CAN** Bus speed to be set. Maximum speed is 1MBaud.

Diagnostics

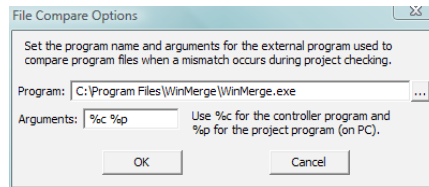
This is used to log communications and some of the internal workings of *Motion Perfect* as an aid to fault diagnosis. This should not be used except under direction from **Trio Motion Technology** as the data logging function has a significant effect on the speed of operation of *Motion Perfect*.

Terminal Font

This is used to set the font used in all terminal tool windows.

Program Compare

This is used to specify the external application used to display the differences between controller and PC versions of a program during the check project / resolve operation. The standard *Motion Perfect* installer installs a version of the **GPL** licensed program WinMerge to do this but you can choose another program if you prefer.



CX-Drive Configuration

Not required for operation with Trio controllers.

FINS Configuration

Not required for operation with Trio controllers.

Saving the Desktop Layout

When you have a number of windows open, you can save the layout so that it can be quickly restored later. Alternatively the desktop can be set to restore automatically on each re-connection by ticking the checkbox under the menu: Options/General options.

From the Window menu

Restore Last desktop	Ctrl+Shift+L
Restore Saved Desktop	Ctrl+Shift+R
Save Desktop	Ctrl+Shift+S
Clear Desktop	Ctrl+Shift+K

Restore Last Desktop	Restores the last desktop which was automatically saved by <i>Motion Perfect 2</i> when it disconnected from the controller.
Restore Saved Desktop	Restore the last desktop saved using the Save Desktop option.
Save Desktop	Saves the current desktop layout to a file on disk.
Clear Desktop	Closes all open tool windows.



Intelligent drive configuration windows are not restored.

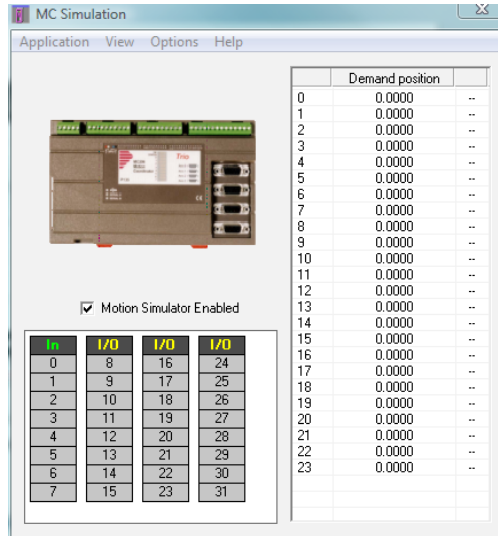
Running *Motion Perfect 2* Without a Controller

Normally you will run *Motion Perfect 2* on-line, that is connected to a controller. In fact *Motion Perfect 2* is designed to operate in this manner and has little functionality without the connection.

In order that you can view or edit your project programs without a controller connected there is a special application to simulate the controller operation and to allow *Motion Perfect 2* to operate in many ways as if a real controller were connected.

MC Simulation

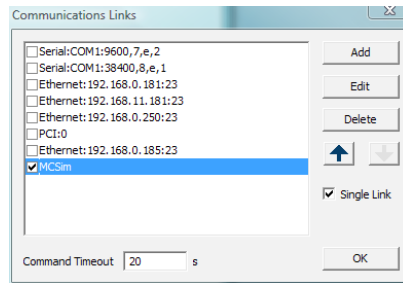
MC Simulation (MCSim) is a very simple program designed to run alongside *Motion Perfect 2* in the background. There are no options or configurations to worry about, you just have to run the program and connect as usual.



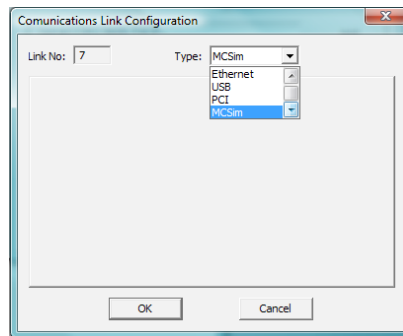
Starting MC Simulation from *Motion Perfect 2*

MCSimulation is automatically started (if it is not already running) when *Motion Perfect 2* tries to connect to it. To connect to MCSimulation either use the Connect to Simulator tool button or set up a Simulation link in the connection list.

Use the Add Port option to select a new port and choose "Simulation" as the Port Type.



The new device will normally appear at the end of the list. Use the “move up” button to make it the default option.



Limitations of MC Simulation

The MCSimulation program does not yet cover all the functionality present in a real controller. It does allow connection to *Motion Perfect* for program editing and the running of programs in the simulated environment.

There are some unsupported TrioBASIC commands (mainly those related to communications busses such as **CAN**).

The motion engine built into the simulation is still under development although it will handle all move types except linked moves. There is an axis demand position display which can be used to monitor the axes when moves are taking place. This can be toggled on and off by selecting View/Axes from the MCSimulation main menu. The motion engine can be enabled/disabled by checking/unchecking the the *Motion Simulator Enabled* check box.

PC Requirements

Operating System: Windows 2000, XP and Vista

Processor: 1.5GHz pentium class

RAM: 256MBytes for Windows 2000, more for others

The reliability of the connection between *Motion Perfect* and the Simulator is effected greatly by the performance of the PC and any other software running on it. The PC should have plenty of spare capacity in both RAM and processing power. Better performance can usually be obtained from a faster PC and also by running fewer applications at the same time.

Project Encryptor

Introduction

Motion Perfect Project Encryptor is a stand alone program running under Microsoft™ Windows which encrypts one or more programs in a *Motion Perfect* project so that the TrioBASIC source for that program cannot be read. This gives solution providers a way of protecting their work form possible reverse engineering attempts by third parties

Encryption Process

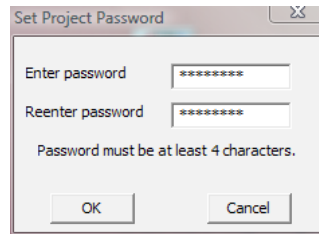
The encryption process uses a “Project Password” to encrypt one or more TrioBASIC programs in a project. The encryption of other types of program or file is not supported.

The encryption process creates a new project leaving the original unencrypted version intact. When the encrypted version of a project is loaded onto a motion *coordinator* a decryption key is required. This key, which is also generated by the encryption program, is used by the controller to decrypt the program for compilation purposes. The decryption key is generated from the “Project Password” and the security code of the target *Motion Coordinator* and is unique to that *Motion Coordinator*. The security code is derived from the unit’s serial number and a hardware identification code which is built into the motion *coordinator*.

Encrypting a Project

Entering the Project Password

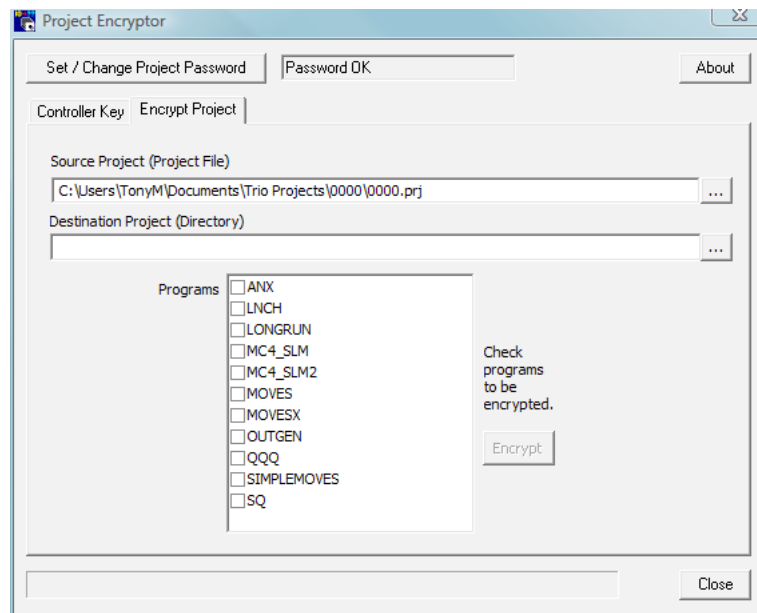
When the project encryptor is first started, or when the “Set / Change Project Password” button is pressed, the Password “Set Project Password” dialogue is displayed.



The password needs to be entered twice to reduce the chance of an entry error occurring. For security reasons the password is not displayed in the application's main window.

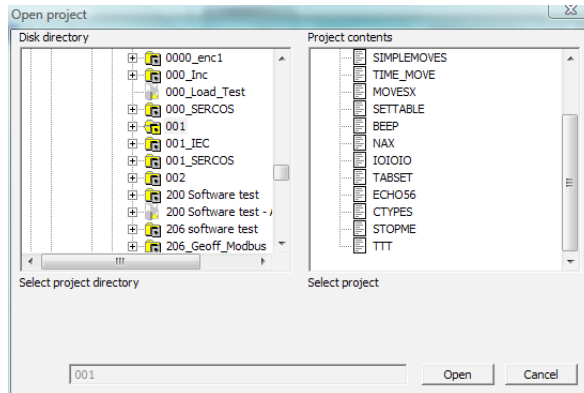
Selecting Source and Destination Projects

In order to be able to encrypt a project the user must select the project to be encrypted (source project) and the name and location of the encrypted project (destination project).



Source Project

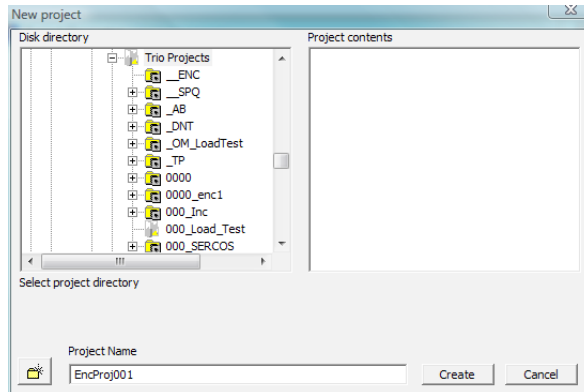
The source project is selected, either by entering the full path of the project file (which has a .prj extension) into the source project text entry box or by using the browse button (...) to the right of the source project text entry box to open up a file selection dialogue.



Select the source project by clicking on an entry in the disk directory tree.

Destination Project

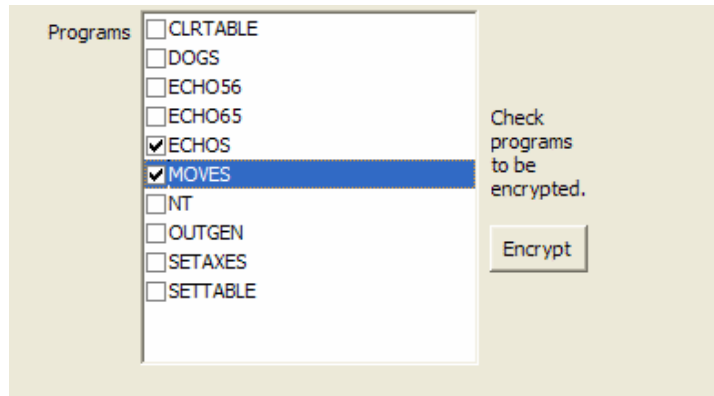
The destination project (which must be different from the source project) is selected, either by entering the full path of the new project directory into the destination project text entry box or by using the browse button (...) to the right of the destination project text entry box to open up a directory selection dialogue.



Select the parent directory for the project using the disk directory tree and enter the project name into the project name text entry box.

Selecting the programs to be encrypted

To select which programs to encrypt tick the check boxes next to the program names.



Encrypting

Click on the Encrypt button to encrypt the project. If it is not enabled then some of the data required to perform the encryption has not been entered.

CAD2Motion

Introduction

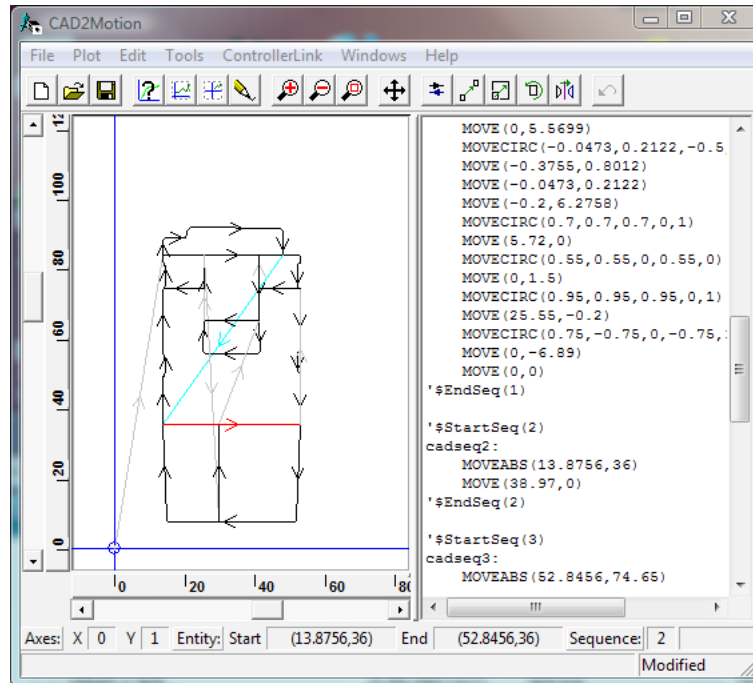
CAD to *Motion* is an application which displays a two dimensional motion path as specified in a TrioBASIC program. It also allows the importation of path data exported from a CAD system in **DXF** format.

It has a build-in editor and some tools for manipulating the sequence of movements which go to make up a movement path.

CAD2Motion is designed to be used in conjunction with **Trio Motion Technology's** *Motion* Perfect motion controller configuration program.

Main Screen

The main application form consists of a graph area to the left, a program list area to the right and a toolbar at the top.



Main Menu

Menu		Function
File	New	Create a new, blank program
	Open	Open an existing program
	Save	Save the current program
	As	Save the current program using a different name.
	Close	Close the current program file
	Import	Import CAD data
	Import Options	Change the options used when importing CAD data
	Append	Append another program or some CAD data to the current program
	Exit	Exit from <i>CAD2Motion</i>

Plot	Redraw	Redraw the graph display
	Zoom Extents	Zoom to display the extents of the current program
	Zoom Extents, Centre Zero	Zoom to display the extents of the current program with the point (0,0) in the centre of the display
	Zoom In	Zoom in a fixed amount
	Zoom Out	Zoom out a fixed amount
	Zoom Box	Zoom in to a box
	Pan	Pan the graph display
	Options	Manually set the display options










Edit	Cut	Cut selected text from the program list
	Copy	Copy selected text from the program list
	Paste	Paste text to the program list
	Delete	Delete selected text in program list
	Select All	Select all the text in the program list
	Undo	Undo the last change









Tools	Reverse Sequence	Reverse the current sequence
	Shift Sequence	Shift the current sequence
	Scale Sequence	Scale the current sequence
	Rotate Sequence	Rotate the current sequence
	Mirror Sequence	Mirror the current sequence
	Reorder Sequences	Reorder the sequences
	Reverse All	Reverse the whole motion path
	Shift All	Shift the whole motion path
	Scale All	Scale the whole motion path
	Rotate All	Rotate the whole motion path
	Mirror All	Mirror the whole motion path
	Auto Code Inserter	Automatically insert code at defined places in the program
	Check Program	Check the program for sequence coding errors

Windows	Toolbar	Show/hide the toolbar
	File Status Bar	Show/hide the file status bar
	Graph Status	Show/hide the graph status bar

Help	Help Topics	Show main help file for <i>CAD2Motion</i>
	About	Show <i>CAD2Motion</i> version information






Toolbar

-  Create a new program
-  Open an existing program file
-  Save the current program
-  Change the graph options
-  Zoom to the extents of the program data
-  Zoom to the extents of the program data with the point (0,0) in the centre of the display
-  Redraw the graph
-  Zoom in by a fixed amount
-  zoom out by a fixed amount

	Zoom in to a box
	Pan the graph display
	Reverse the current sequence
	Shift the current sequence
	Scale the current sequence
	Rotate the current sequence
	Mirror the current sequence
	Undo the last edit operation

Sequence Manipulation Tools

The Sequence Manipulation Tools are used to manipulate the sequences of moves which make up a single motion path. These tools allow for reversal, shifting, scaling, rotating and mirroring of single sequences of moves.

	Reverse	Reverse the current sequence. Any non move commands may not appear in the correct place in the reversed sequence.
	Shift	Shift the current sequence.
	Scale	Scale the current sequence.
	Rotate	Rotate the current sequence.
	Mirror	Mirror the current sequence about the X or Y axis.



*The functions of all of these tools are available from the Tools menu.
The whole of the file can be manipulated using the File Manipulation Tools.
Transformations can be undone using the undo button.*

Files

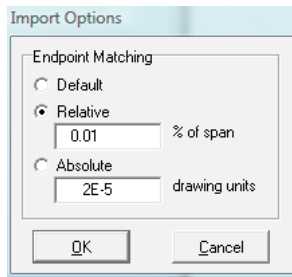
The program handles two types of file:

1. the TrioBASIC program file
2. the DXF drawing file.

It is possible to read and write TrioBASIC files and to import DXF files. It is also possible to append data from a TrioBASIC or DXF file.

Import Options

The import options are available through the File / Import Options menu. The options are set using the following dialogue:



Endpoint Matching

This is used to define the distance between the end point of one line and the start point of another line below which they will be considered as the same point. Three types of matching can be used:

3. Default specifies a tight relative tolerance which is good for most purposes.
4. Relative specifies the tolerance as a percentage of the span of the imported
5. drawing. Absolute specifies the tolerance in same units as used in the drawing.



WHEN ONE POINT IS MATCHED WITH ANOTHER THE TWO POINTS ARE CONSIDERED TO BE THE SAME AND ONE POINT IS EFFECTIVELY REMOVED. THE EFFECT OF THIS IS THAT IF THE MATCHING TOLERANCE IS LOOSE IT WILL APPEAR THAT THE END POINTS OF SOME OF THE LINES IN THE DRAWING HAVE BEEN MOVED. THIS EFFECT SHOULD BE CONSIDERED WHEN SETTING THE END POINT MATCHING TOLERANCE VALUE.

Graph

The graph display shows a two dimensional display of the motion path specified by the program shown in the program list.

If a line is selected on the graph the cursor in the program list is moved to the start of the line which produces the movement represented by the line. The line is highlighted in red in the graph display. If a line is double clicked the whole text of the appropriate program line is selected.

The current sequence will be highlighted in magenta with the leading **MOVEABS** highlighted in cyan. All other sequences will be displayed in black with the leading **MOVEABS** in light grey.

The graph has X and Y axis rulers to show values and has scroll bars to allow movement of the viewed area.

The graph view can also be changed using the graph display tools.

Preparing A Drawing For CAD2Motion

Because of the way CAD2Motion imports data from a **DXF** file it is important that the information in the CAD drawing used to produce the **DXF** file is constructed in the correct way.

A motion path must be drawn on a single layer. Nothing else can be on this layer except one or more other motion paths. The motion path must be continuous (have no gaps in it), although the lines which make up the motion path can be drawn in any order. CAD2Motion will only import straight lines and arcs so special curves must **NOT** be used. All objects used to make up the motion path must be simple objects (i.e. not groups or blocks).

CAD2Motion interprets the following **DXF** entities:

ARC

CIRCLE

LINE

LWPOLYLINE

POINT

POLYLINE

SPLINE Gives a series of straight lines joining the control points NOT full interpretation

VERTEX As part of POLYLINE

Program List

The program list contains the TrioBASIC statements which are interpreted to make up the motion path displayed in the graph.

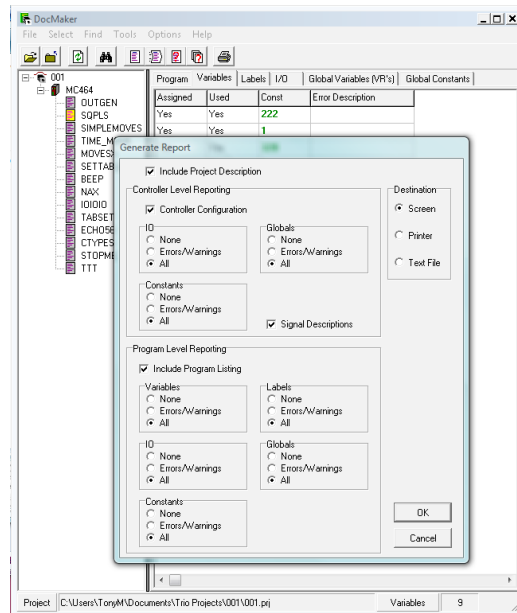
It is possible to use the program list as an editor to make manual changes to the program.

When the cursor is moved onto a line with a move command on it the appropriate line will be highlighted in the graph display. If the line is part of a sequence of moves the whole sequence will also be highlighted.

The program list editor follows normal Windows conventions and includes text cut and paste facilities. Multilevel undo is available using the Undo Button or the Undo option from the Edit Menu.

DocMaker

DocMaker is an application to assist in documenting a project created in *Motion Perfect* for a **Trio Motion Technology *Motion Coordinator***. It can be used to print program listings and to report on the programs (variables, labels, I/O and $\forall R$'s) and on overall I/O and $\forall R$ usage.



DocMaker analyses the content of the program files in the project. It can be used to print program listings and to report on the programs (variables, labels, I/O and $\forall R$'s) and on overall I/O and $\forall R$ usage. There is also a checking routine which does a quick check on the whole project and flags up possible errors

DocMaker Benefits

- Automatic Analysis of *Motion Perfect* Project Files
- Highlight potential errors due to labels or variables
- Generates fully cross-referenced reports
- Reformat programs with auto-indenting

Docmaker Hardware Requirements

- IBM PC or Compatible running Microsoft Windows 98 or higher
- Works with all current *Motion Coordinators*

CHAPTER
AUTO LOADER AND
MCLOADER ACTIVEX

10

AutoLoader and MCLoader ActiveX

Project Autoloader

Trio Project Autoloader is a stand alone *Motion Coordinator* program to load projects created using *Motion Perfect 2* onto a Trio *Motion Coordinator*

The program is intended for easy loading of projects onto controllers without the need to run *Motion Perfect* and so allows OEM manufacturers to update customers equipment easily.

Operation of the program is controlled using a script file which gives a series of commands to be processed, in order, by the program.

Using the Autoloader

General

The autoloader is primarily intended to be used to update controllers already installed in equipment to allow OEM manufacturers to update customers equipment easily. It can be used from a hard disk or CD-ROM.

Script File

The autoloader program uses a script file `AutoLoader.tas` as a source of commands. These commands are executed in order until all commands have been processed or an error has occurred.

If any command fails the execution terminates without completing the scripted command sequence.

Project

The project to be loaded using `LOADPROJECT` is in the form of a normal *Motion Perfect 2* project. This consists of a directory containing a project definition file and TrioBASIC program files. The directory must have the same name as the project definition file less the extension.

i.e. project definition file `TestProj.prj`, directory `TestProj`

The project directory must be in the `LoaderFiles` directory.

Timeout

If there are large programs in the project the command timeout may need to be increased from its default value of 10 seconds otherwise the project load may fail due to the long time it takes to select a new program on the controller. The `TIMEOUT` command should appear in the script file before any `LOADPROJECT` command.

Tables

Any tables to be loaded must be in the form of *.lst files produced by *Motion Perfect 2*. Normally these table files will be in the LoaderFiles directory.

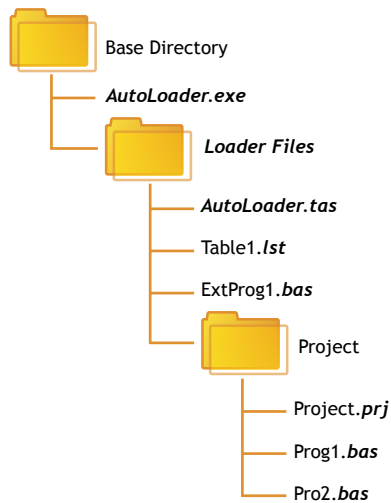
Extra Programs

Programs which need to be loaded using `LOADPROGRAM` because they are not in the project being loaded (or if no project is being loaded)

Normally these program files will be in the LoaderFiles directory.

Files

The autoloader is designed to work with the following file structure (fixed names are shown in *bold italic* type).



Where:

Base Directory is normally the root directory, but can be any directory.

Project is the *Motion Perfect 2* project directory for the project to be loaded using the `LOADPROJECT` command, Project.prj being the project file and Proj?.bas are the program files in the project.

Table?.lst are the table files to be loaded using the `LOADTABLE` command.

ExtProg?.bas are the extra programs to be loaded using the `LOADPROGRAM` command.

Any or all of the objects in the LoaderFiles directory can be located elsewhere as long as the file (or directory) name is specified using a full path. The script file can be specified as a single argument to the AutoLoader program.

Running the program

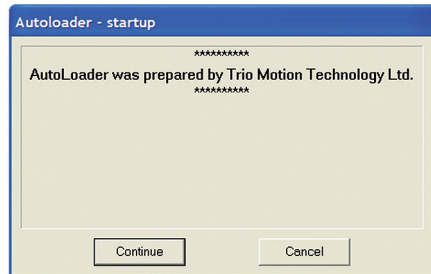
The program can be started in the same way as any other Windows program, in which case the LoaderFiles directory must be in the same directory as the AutoLoader executable file.

It can also be started from the command line with an optional argument which specifies the script file to process. e.g.

AutoLoader E:\MXUpdate\20051203\UpDate1.tas

Start Dialog

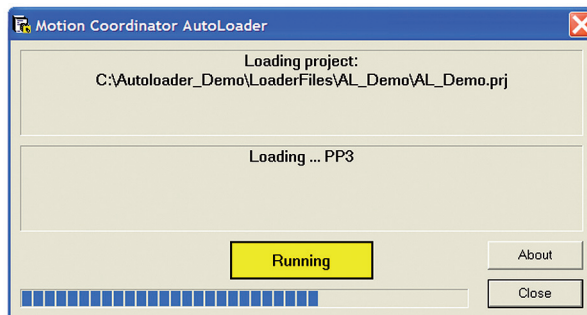
The start dialog displays a message specified in the script and has continue and cancel buttons so that the user can exit from the program without running the script.



Main Window

The program main window consists of two message windows; one to display the current command and the other to display the name of the program or file currently being loaded. There is a button to show the current status (Starting, running, pass or fail) and a progress bar to show the progress during file and table loading.

The close button closes the dialog. If it is pressed while a script is being processed then script processing will be terminated at the end of the current operation.



Script Commands

The following commands are available for use in script files

```
AUTORUN
CHECKPROJECT
CHECKTYPE
CHECKUNLOCKED
CHECKVERSION
COMMLINK (alternative COMMPORT)
COMPILEALL
COMPILEPROGRAM
DELETEALL (alternative NEWALL)
EPROM
FASTLOADPROJECT
HALTPROGRAMS
LOADPROGRAM
LOADPROJECT
LOADTABLE
SETPROJECT
SETRUNFROMEPROM
STARTUPMESSAGE
TIMEOUT
DELTABLE
```

All commands return a result of OK or Fail. An OK result allows script execution to continue, a Fail result will make script execution terminate at that point.

AUTORUN

Purpose: To run the programs on the controller which are set to run automatically at power-on.

Syntax: `AUTORUN`

CHECKPROJECT

Purpose: To check the programs on a controller against a project on disk.

Syntax: `CHECKPROJECT [<ProjectName>]`

Where <ProjectName> is the optional path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the project directory. If no <ProjectName> is specified then the current project, set by a previous `SETPROJECT` or `LOADPROJECT` command, is used. This operation is automatically performed by a `LOADPROJECT` operation.

Examples: `CHECKPROJECT`
`CHECKPROJECT TestProj`

CHECKTYPE

Purpose: To check the controller type.

Syntax: `CHECKTYPE <Controller List>`

Where <Controller List> is a comma separated list of one or more valid controller ID numbers.

i.e. 206,216

Examples: `CHECKTYPE 206`
`CHECKTYPE 202,216,206`

Controller ID Numbers

Each type of controller returns a different ID number in response to the TrioBASIC command `?CONTROL[0]` . The table below gives the ID number for current controllers.

Controller	CONTROL
MC302X	293
Euro205x	255
Euro209	259
MC206X	207
PCI208	208
MC224	224

CHECKUNLOCKED

Purpose: To check that the controller is not locked.

Syntax: CHECKUNLOCKED

CHECKVERSION

Purpose: To check the version of the controller system code.

Syntax: CHECKVERSION <Operator><Version>
CHECKVERSION <LowVersion>-<HighVersion>

Examples: CHECVERSION >1.49
CHECKVERSION >= 1.51
CHECKVERSION 1.42-1.50

Comment

Purpose: To allow the user to put descriptive comments into a script.

Syntax: ` <Text>
Where <Text> is any text.

Examples: ` This is a comment line

COMMLINK (alternative COMMPORT)

Purpose: To set the communications port and parameters.

Syntax: **Serial**

For a serial port this string is similar to COM1:9600,7,e,2 to specify the port, speed, number of data bits, parity and number of stop bits. 9600,7,e,2 are the default parameters for a controller.

USB

For a USB connection the string is USB:0 as only a single USB connection (0) is supported.

Ethernet

For an ethernet connection the string is similar to Ethernet:192.168.0.123:23 which specifies an ethernet connection to IP address 192.168.0.123 on port 23. The final ':' and the port number can be omitted, in which the port number defaults to 23.

PCI

For a PCI connection the string is similar to PCI:0 which specifies a connection to PCI card 0.

Examples: COMMLINK COM2:9600,7,e,2

COMMLINK USB:0

COMMLINK Ethernet:192.168.0.111

COMMLINK PCI:1

COMPILEALL

Purpose: To compile all the programs on the controller.

Syntax: COMPILEALL

COMPILEPROGRAM

Purpose: To compile a program on the controller.

Syntax: COMPILEPROGRAM <Program>

Where <Program> is the program name.

Examples: `COMPILEPROGRAM Prog`

The `LOADPROGRAM` command automatically compiles programs after they are loaded so under normal circumstances there is no need to use this command.

DELETEALL (alternative NEWALL)

Purpose: To delete all programs on the controller.

Syntax: `DELETEALL`

EPROM

Purpose: To store the project currently in controller RAM into EPROM

Syntax: `EPROM`

FASTLOADPROJECT

Purpose: To load a project from disk onto the controller.

Description: `FASTLOADPROJECT` is a faster alternative to `LOADPROJECT`. It is only compatible with system software version 1.63 or later for '2' series *Motion Coordinators*, and version 1.9013 or later for '3' series *Motion Coordinators*.

`FASTLOADPROJECT` must be used if a project contains encrypted programs.

Syntax: `FASTLOADPROJECT [<ProjectName>]`

Where `<ProjectName>` is the optional path of the project directory. If the project directory is in the same directory as the `ALoader.exe` executable then it is just the name of the of the project directory. If no `<ProjectName>` is specified then the current project, set by a previous `SETPROJECT` command, is used.

Examples: `FASTLOADPROJECT`

`FASTLOADPROJECT TestProj`

If `FASTLOADPROJECT` fails and the project only contains TrioBASIC source files then using `LOADPROJECT` may work.

HALTPROGRAMS

Purpose: To halt all programs on the controller.

Syntax: HALTPROGRAMS

This operation is automatically performed as part of **LOADPROJECT**, **LOADPROGRAM** and **DELTABLE** commands.

LOADPROJECT

Purpose: To load a project from disk onto the controller.

Syntax: LOADPROJECT <ProjectName>

Where <ProjectName> is the optional path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the of the project directory. If no <ProjectName> is specified then the current project, set by a previous **SETPROJECT** command, is used.

Examples: LOADPROJECT

LOADPROJECT TestProj

LOADPROJECT will only load TrioBASIC source files.

LOADPROGRAM

Purpose: To load a program not in a project onto the controller.

Syntax: LOADPROGRAM <ProgramFile>

Where <ProgramFile> is the path of the program file. If the program file is in the same directory as the ALoader.exe executable then this is just the file name of the program file.

Examples: LOADPROGRAM TestProg.bas

LOADPROGRAM will only load TrioBASIC source files.

LOADTABLE

Purpose: To load a table onto the controller.

Syntax: `LOADTABLE <TableFile>`

Where <TableFile> is the path of the table file. If the table file is in the LoaderFiles directory then this is just the file name of the table file.

This command should always be used after the `LOADPROJECT` command.

Examples: `LOADTABLE Tbl.lst`

SETPROJECT

Purpose: To set the current project for following commands.

Syntax: `SETPROJECT <ProjectName>`

Where <ProjectName> is the path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the of the project directory.

Examples: `SETPROJECT TestProj`

SETRUNFROMEPROM

Purpose: To set the controller to use the programs stored in its EPROM. (It actually copies the programs from EPROM into RAM at startup).

Syntax: `SETRUNFROMEPROM <State>`

Where <State> is 1 for copy from EPROM and 0 is use programs currently in RAM. A single @ character can be used to specify state in the project file.

Examples: `SETRUNFROMEPROM 1`

`SETRUNFROMEPROM @`

This command only applies to controllers which have battery backed RAM (controllers with no battery backed RAM will always copy programs from EPROM).

Startup Message

- Purpose:** To allow the user to display a custom message in the startup dialog.
- Multiple lines can be used to specify the message, they are displayed in the order that they appear in the script file. The message can be specified anywhere in the script file and the lines need not be together in the file.
- Syntax:** `# <Text>`
- Where <Text> is any text.
- Examples:** `# ***`
- `# This autoloader was set up by ABCD Inc. to change Valve Machine to left-hand thread`
- `# ***`

TIMEOUT

- Purpose:** To set the command timeout.
- Syntax:** `TIMEOUT time`
- Where time is the timeout value in seconds (default is 10).
- Examples:** `TIMEOUT 30`
- It will normally only be necessary to increase the timeout above 10 if there are large programs in the target controller or you are loading large programs onto it.

Script File

The autoloader program uses a script file AutoLoader.tas as a source of commands. These commands are executed in order until all commands have been processed or an error has occurred.

If any command fails the execution terminates without completing the scripted command sequence.

Sample Script

```
` Test Script
` *****
` Startup Message
# ***
# This autoloader was set up by TRIO to load a test project onto
a controller of fixed type.
# ***

COMMLINK COM1:9600,7,e,2

CHECKTYPE 206

CHECKVERSION > 1.45

CHECKUNLOCKED

LOADPROJECT LoaderTest

LOADTABLE tbl_1.lst

CHECKPROJECT LoaderTest

LOADPROGRAM flashop.bas

LOADPROGRAM clrtable.bas

LOADPROGRAM settable.bas

EPROM

SETRUNFROMEPROM @
```

For this script to work correctly the LoaderFiles directory must contain a project directory LoaderTest, a table file tbl_1.lst and three program files: flashop.bas, clrtable.bas and settable.bas.

MC Loader

Trio MC Loader is a Windows ActiveX control which can load projects (produced with *Motion Perfect*) and programs onto a Trio *Motion Coordinator*. Communication with the *Motion Coordinator* can be via Serial link, USB, Ethernet or PCI depending on the *Motion Coordinator*.

Requirements

- PC with one or more of USB interface, Ethernet network interface, or PCI based Motion Coordinator.
- Microsoft Windows XP, Vista or Windows 7 32bit versions (Windows 2000 or XP only for PCI connection)
- TrioUSB driver - for USB connection
- Trio PCI driver - for PCI connection (Windows 2000 and XP systems only)
- Knowledge of the Trio Motion Coordinator to which the TrioPC ActiveX controls will connect.
- Knowledge of the TrioBASIC programming language.

Installation of the MC Loader Component

Launch the program “Install_TrioMCLoader” and follow the on-screen instructions. The TrioUSB driver and TrioPC ocx will be installed and registered to your Windows environment. The Trio MC Loader 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 MC Loader 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 Project then Components... (or use shortcut ctrl+T).

When the Components dialogue box has opened, scroll down until you find “Trio MC Loader Control Module” then click in the block next to Trio MC Loader. (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 TrioMCLoader Control.

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

Properties

The control has the following properties:

CommLink

ControllerType

ControllerSystemVersion

DecryptionKey

Locked

ProjectFile

RunFromEPROM

Timeout

Events

The control does not generate any events.

CommLink

Type: BSTR (string)

Access: Read / write

Description: This property is used to get or set the configuration of the communications link. The format of the string depends on the type of communications link being used.

Serial

For a serial port this string is similar to COM1:9600,7,e,2 to specify the port, speed, number of data bits, parity and number of stop bits. 9600,7,e,2 are the default parameters for most controllers.

USB

For a USB connection the string is USB:0 as only a single USB connection (0) is supported.

Ethernet

For an ethernet connection the string is similar to Ethernet:192.168.0.123:23 which specifies an ethernet connection to IP address 192.168.0.123 on port 23. The final ':' and the port number can be omitted, in which the port number defaults to 23.

PCI

For a PCI connection the string is similar to PCI:0 which specifies a connection to PCI card 0.

Examples: Visual BASIC:

```
axLoader.CommLink = "Ethernet:192.168.22.11"
```

Visual C#:

```
axLoader.CommLink = "Ethernet:192.168.22.11";
```

ControllerType

Type: unsigned long

Access: Read

Description: This is a read-only property which returns the Controller Type code.

Examples: Visual BASIC:

```
Dim ConType As Long
ConType = axLoader.ControllerType
```

Visual C#:

```
ulong ulConType;
ulConType = axLoader.ControllerType;
```

ControllerSystemVersion

Type: double

Access: Read

Description: This is a read-only property which returns the controller system software version number.

Examples: Visual BASIC:

```
Dim Version As Double
Version = axLoader.ControllerSystemVersion
```

Visual C#:

```
double dVersion;
dVersion = axLoader.ControllerSystemVersion;
```

DecryptionKey

Type: BSTR (string)

Access: Read / write

Description: The DecryptionKey property sets/gets the decryption key for a subsequent fast mode LoadProject operations. The decryption key is only used when a project containing one or more encrypted programs is loaded onto a controller using fast LoadProject.

Examples: **Visual BASIC:**

```
axLoader.DecryptionKey = "hjiHU8700o"
```

Visual C#:

```
axLoader.DecryptionKey = "hjiHU8700o";
```



Decryption keys are derived from the key string used to encrypt the program(s) and the security code of the target controller. Decryption keys can be generated using the Project Encryptor tool distributed with Motion Perfect.

Locked

Type: Variant_bool

Access: Read

Description: This is a read-only property which returns the locked state of the controller (true for locked, false for unlocked).

Examples: **Visual BASIC:**

```
Dim IsLocked As Boolean  
IsLocked = axLoader.Locked
```

Visual C#:

```
bool bLocked;  
bLocked = axLoader.Locked;
```

ProjectFile

Type: BSTR (string)

Access: Read / write

Description: This property is used to get or set the current project file. The full path to the project file should be used when setting this property.

Examples: Visual BASIC:

```
If axLoader.ProjectFile.Length = 0 then
    axLoader.ProjectFile = "C:\Projects\PPX\PPX.prj"
End If
```

Visual C#:

```
if (axLoader.ProjectFile.Length == 0)
    axLoader.ProjectFile = "C:\\Projects\\PPX\\PPX.prj";
```

RunFromEPROM

Type: Variant_bool

Access: Read / write

Description: This property is used to control how the controller starts up. When set to false it uses programs stored in its RAM memory. When set to true the controller uses programs stored in its EPROM memory (overwriting the programs in RAM).

Examples: Visual BASIC:

```
If not axLoader.RunFromEPROM then
    axLoader.RunFromEPROM = True
End If
```

Visual C#:

```
if (!axLoader.RunFromEPROM)
    axLoader.RunFromEPROM = true;
```

Timeout

Type: unsigned long

Access: Read / write

Description: This property is used to set the command timeout for communications with the controller. The default value is 10 (seconds) but may need to be increased if you are using large programs or have a large project.

Examples: Visual BASIC:

```
If axLoader.Timeout < 20 Then  
    axLoader.Timeout = 25  
End If
```

Visual C#:

```
if (axLoader.Timeout < 20)  
    axLoader.Timeout = 25;
```

Methods

The control has the following methods:

```
AutoRun
CheckProject
CompileAll
CompileProgram
DeleteAll
DeleteTable
GetLastError
GetLastErrorString
HaltPrograms
LoadProgram
LoadProject
LoadTable
Lock
Unlock
```

AutoRun

Parameters: none

Return Type: Variant_bool

Description: This method is used to run any programs on the controller which are set to auto-run on startup.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.AutoRun Then
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.AutoRun())
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

CheckProject

Parameters: none

Return Type: Variant_bool

Description: This method is used to check the programs on the controller against the project previously set using the ProjectFile.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.CheckProject Then
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CheckProject())
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

CompileAll

Parameters: none

Return Type: Variant_bool

Description: This method is used to compile all programs on the controller.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.CompileAll Then
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CompileAll())  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

CompileProgram

Parameters: BSTR (string): ProgramName

Return Type: Variant_bool

Description: This method is used to compile a single program on the controller.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.CompileProgram("PROG") Then  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)  
End If
```

Visual C#:

```
if (!axLoader.CompileProgram("PROG"))  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

DeleteAll

Parameters: none

Return Type: Variant_bool

Description: This method is used to delete the all the programs on the controller.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.DeleteAll Then
```

```
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.DeleteAll())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

DeleteTable

Parameters: none

Return Type: Variant_bool

Description: This method is used to delete the table on the controller. It only works on controllers which do not have dedicated table memory.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples:**Visual BASIC:**

```
If Not axLoader.DeleteTable Then
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Variant_bool

```
if (!axLoader.DeleteTable())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

GetLastError

Parameters: none

Return Type: unsigned long

Description: This method is used to retrieve the error code after a method call has failed (returned false). The returned error code is only valid for the previous method call.

The following error codes can be returned:

Code	Error Description
0	No error
1	File does not exist
2	Error opening file
3	Invalid IP address
4	Invalid IP port
5	Invalid integer
6	Invalid communications port
7	Invalid communications parameters
8	Communications error
9	Invalid controller system version
10	Invalid controller type
11	Controller type not found
12	Invalid range
13	Failed version check
14	Controller locked
15	Failed to set project
16	Invalid command
17	Directory does not exist
18	No file specified
19	Program not in project
20	Program not on controller
21	CRC mismatch
22	Invalid directory

- 23 Failed to create directory
- 24 Invalid program file name
- 25 Error writing to file
- 26 Error reading CRC
- 27 Error calculating CRC
- 28 File not in project
- 29 Invalid program name
- 30 Failed to halt programs
- 31 Error reading directory
- 32 Program failed to compile
- 33 Failed to set communications parameters
- 34 Failed to get communications parameters
- 35 Transmit failure
- 36 Invalid connection type
- 37 Internal pointer error
- 38 Error sending string
- 39 Error sending command
- 40 Failed to select program

Further error information can be obtained by calling the `GetLastErrorString` method.

Examples:**Visual BASIC:**

```
If Not axLoader.CompileAll Then  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)  
End If
```

Visual C#:

```
if (!axLoader.CompileAll())  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

GetLastErrorString

Parameters: none

Return Type: BSTR (string)

Description: This method is used to retrieve additional information from the controller. The string contains extra information which can be used in conjunction with the error code returned by the GetLastError method.

Examples: Visual BASIC:

```
If Not axLoader.CompileAll Then
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CompileAll())
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

HaltPrograms

Parameters: none

Return Type: Variant_bool

Description: This method is used to halt all programs currently running on the controller. The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.HaltPrograms Then
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.HaltPrograms())
  DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

LoadProgram

Parameters: BSTR (string): ProgramFileName

Variant_bool: Compile

Return Type: Variant_bool

Description: This method is used to load a single program onto the controller. It is generally good practice to compile after loading the program.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.LoadProgram("C:\Programs\Prog.bas", True) Then
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.LoadProgram("C:\\Programs\\Prog.bas", true))
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

LoadProject

Parameters: Variant_bool: FastLoad

Return Type: Variant_bool

Description: This method is used to load the project previously set using the ProjectFile property onto the controller. If FastLoad is true, the loader will use the fast loading algorithm. Fast loading is not available some controllers and is only available in more recent versions of system software. All controllers will perform a normal (slow) load. Fast load must be used if the project contains one or more encrypted programs.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.LoadProject(False) Then
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.LoadProject(false))
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

LoadTable

Parameters: BSTR (string): TableFileName

Return Type: Variant_bool

Description: This method is used to load data into the table on the controller from a table list file (usually saved by *Motion Perfect*).

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the `GetLastError` and `GetLastErrorString` methods.

Examples: **Visual BASIC:**

```
If Not axLoader.LoadTable("C:\Tables\ThisTable.lst") Then
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.LoadTable("C:\\Tables\\ThisTable.lst"))
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

Lock

Parameters: unsigned long: Lock Code

Return Type: Variant_bool

Description: This method is used to lock the controller so that programs cannot be edited. The lock code used here must also be used if the controller is unlocked using the Unlock method.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.Lock(1234) Then  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)  
End If
```

Visual C#:

```
if (!axLoader.Lock(1234))  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

Unock

Parameters: unsigned long: LockCode

Return Type: Variant_bool

Description: This method is used to unlock a locked controller so that programs can be edited. The lock code used here must be the same as the code used to lock the controller.

The return value is true if the method call succeeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: Visual BASIC:

```
If Not axLoader.Unlock(1234) Then  
    DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)  
End If
```

Visual C#:

```
if (!axLoader.Unlock(1234))
    DisplayError(axLoader.GetLastError, axLoader.GetLastErrorString);
```


CHAPTER
USING THE PC MOTION
ACTIVEX CONTROL

11

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 versions) or Windows 7 (32 bit versions)
- Trio PCI driver - for PCI based *Motion Coordinators*
- 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

|                       |                                                                                                                                                                                               |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | <p>Sets the board number used to access a PCI card.</p> <p>The PCI cards in a PC are always enumerated sequentially starting at 0. It must be set before the <b>OPEN</b> command is used.</p> |
| <b>Type:</b>          | Long                                                                                                                                                                                          |
| <b>Access</b>         | Read / Write                                                                                                                                                                                  |
| <b>Default Value:</b> | 0                                                                                                                                                                                             |
| <b>Example:</b>       | <pre>Rem Open a PCI connection and refresh the TrioPC indicator If TrioPC.Board &lt;&gt; 0 Then     TrioPC.Board = 0 End If TrioPC_Status = TrioPC1.Open(3, 0) frmMain.Refresh</pre>          |

## HostAddress

|                       |                                                                                                                                                                                                     |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>   | <p>Used for reading or setting the IPV4 host address used to access a <i>Motion Coordinator</i> over an Ethernet connection. The SetHost command can also be used for setting the host address.</p> |
| <b>Type:</b>          | String                                                                                                                                                                                              |
| <b>Access:</b>        | Read / Write                                                                                                                                                                                        |
| <b>Default Value:</b> | "192.168.0.250"                                                                                                                                                                                     |
| <b>Example:</b>       | <pre>Rem Open a Ethernet connection and refresh the TrioPC indicator If TrioPC.HostAddress &lt;&gt; "192.168.0.111" Then     TrioPC.HostAddress = "192.168.0.111" End If</pre>                      |

```
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 property 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

---

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|-----------------------------------------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------|--------------------|-----------------------------------------------------------------------------------------------------|
| <b>Description</b>      | Performs the corresponding <b>MOVE(...)</b> command on the <i>Motion Coordinator</i> .                                                                                                                                                                                                                                                                                                                                                |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |
| <b>Syntax:</b>          | <code>MoveRel(Axes, Distance, [Axis])</code>                                                                                                                                                                                                                                                                                                                                                                                          |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |
| <b>Parameters:</b>      | <table><tr><td><b>short Axes:</b></td><td>Number of axes involved in the <b>MOVE</b> command.</td></tr><tr><td><b>Double Distance:</b></td><td>Distance to be moved, can be a single numeric value or an array of numeric values that contain at least Axes values.</td></tr><tr><td><b>Short Axis:</b></td><td>Optional parameters that must be a single numeric value that specifies the base axis for this move.</td></tr></table> | <b>short Axes:</b> | Number of axes involved in the <b>MOVE</b> command. | <b>Double Distance:</b> | Distance to be moved, can be a single numeric value or an array of numeric values that contain at least Axes values. | <b>Short Axis:</b> | Optional parameters that must be a single numeric value that specifies the base axis for this move. |
| <b>short Axes:</b>      | Number of axes involved in the <b>MOVE</b> command.                                                                                                                                                                                                                                                                                                                                                                                   |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |
| <b>Double Distance:</b> | Distance to be moved, can be a single numeric value or an array of numeric values that contain at least Axes values.                                                                                                                                                                                                                                                                                                                  |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |
| <b>Short Axis:</b>      | Optional parameters that must be a single numeric value that specifies the base axis for this move.                                                                                                                                                                                                                                                                                                                                   |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |
| <b>Return Value:</b>    | See TrioPC STATUS.                                                                                                                                                                                                                                                                                                                                                                                                                    |                    |                                                     |                         |                                                                                                                      |                    |                                                                                                     |

---

## Base

---

|                      |                                                                                                                                                                                                                                                                                                                 |                    |                                              |                     |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|----------------------------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>BASE(...)</b> command on the <i>Motion Coordinator</i> .                                                                                                                                                                                                                          |                    |                                              |                     |                                                                                                                                                           |
| <b>Syntax:</b>       | <code>Base(Axes, [Order])</code>                                                                                                                                                                                                                                                                                |                    |                                              |                     |                                                                                                                                                           |
| <b>Parameters:</b>   | <table><tr><td><b>short Axes:</b></td><td>Number of axes involved in the move command.</td></tr><tr><td><b>Short Order:</b></td><td>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.</td></tr></table> | <b>short Axes:</b> | Number of axes involved in the move command. | <b>Short Order:</b> | 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. |
| <b>short Axes:</b>   | Number of axes involved in the move command.                                                                                                                                                                                                                                                                    |                    |                                              |                     |                                                                                                                                                           |
| <b>Short Order:</b>  | 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.                                                                                                                                                       |                    |                                              |                     |                                                                                                                                                           |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                                                                                                                                                                                                                              |                    |                                              |                     |                                                                                                                                                           |

---

## MoveAbs

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>MOVEABS(...)</b> <b>AXIS(...)</b> command on the.                                                                                                                                                                                                                                                                                                                                           |
| <b>Syntax:</b>       | <b>MoveAbs(Axes, Distance, [Axis])</b>                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Parameters:</b>   | <p><b>short Axes:</b> Number of axes involved in the <b>MOVEABS</b> command.</p> <p><b>Double Distance:</b> 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 <b>Axes</b> values.</p> <p><b>Short Axis:</b> Optional parameters that must be a single numeric value that specifies the base axis for this move.</p> |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                                                                                                                                                                                                                                                                                                                                        |

---

## MoveCirc

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b> | Performs the corresponding <b>MOVECIRC(...)</b> <b>AXIS(...)</b> command on the <i>Motion Coordinator</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Syntax:</b>      | <b>MoveCirc(EndBase, EndNext, CentreBase, CentreNext, Direction, [Axis])</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Parameters:</b>  | <p><b>Double EndBase:</b> Distance to the end position on the base axis.</p> <p><b>Double EndNext:</b> Distance to the end position on the axis that follows the base axis.</p> <p><b>Double CentreBase:</b> Distance to the centre position on the base axis.</p> <p><b>Double CentreNext:</b> Distance to the centre position on the axis that follows the base axis.</p> <p><b>Short Dir:</b> 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.</p> <p><b>Short Axis:</b> Optional parameters that must be a single numeric value that specifies the base axis for this move.</p> |

**Return Value:** See TrioPC STATUS.

---

## AddAxis

---

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

**Syntax:** `AddAxis(LinkAxis, [Axis])`

**Parameters:**

|                              |                                                                                                     |
|------------------------------|-----------------------------------------------------------------------------------------------------|
| <code>short LinkAxis:</code> | A numeric value that specifies the axis to be “added” to the base axis.                             |
| <code>short Axis:</code>     | 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:**

|                                 |                                                                                               |
|---------------------------------|-----------------------------------------------------------------------------------------------|
| <code>Short TableStart:</code>  | The position in the table data on the <i>Motion Coordinator</i> where the cam pattern starts. |
| <code>Short TableStop:</code>   | The position in the table data on the <i>Motion Coordinator</i> where the cam pattern stops.  |
| <code>Double Multiplier:</code> | The scaling factor to be applied to the cam pattern.                                          |
| <code>Double LinkDist:</code>   | The distance the input axis must move for the cam to complete.                                |
| <code>Short LinkAxis:</code>    | Definition of the Input Axis.                                                                 |
| <code>Short LinkOption:</code>  | 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  | <b>CAMBOX</b> repeats automatically and bi-directionally when this bit is set.                      |
|                        | 8  | Pattern Mode.                                                                                       |
|                        | 32 | Link is only active during positive moves.                                                          |
| <b>Double LinkPos:</b> |    | The absolute position on the link axis where the cam will start.                                    |
| <b>Short Axis:</b>     |    | Optional parameters that must be a single numeric value that specifies the base axis for this move. |
| <b>Return Value:</b>   |    | See TrioPC STATUS.                                                                                  |

---

## Cam

---

|                      |                                                                                                        |                                                                                                                                                                                                                                   |
|----------------------|--------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | Performs the corresponding <b>CAM(...)</b> <b>AXIS(...)</b> command on the <i>Motion Coordinator</i> . |                                                                                                                                                                                                                                   |
| <b>Syntax:</b>       | <b>Cam(TableStart, TableStop, Multiplier, LinkDistance, [Axis])</b>                                    |                                                                                                                                                                                                                                   |
| <b>Parameters:</b>   | <b>Short TableStart:</b>                                                                               | The position in the table data on the <i>Motion Coordinator</i> where the cam pattern starts.                                                                                                                                     |
|                      | <b>Short TableStop:</b>                                                                                | The position in the table data on the <i>Motion Coordinator</i> where the cam pattern stops.                                                                                                                                      |
|                      | <b>Double Multiplier:</b>                                                                              | The scaling factor to be applied to the cam pattern.                                                                                                                                                                              |
|                      | <b>Double LinkDistance:</b>                                                                            | 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. |
|                      | <b>Short Axis:</b>                                                                                     | Optional parameters that must be a single numeric value that specifies the base axis for this move.                                                                                                                               |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                     |                                                                                                                                                                                                                                   |

## Cancel

---

|                      |                                                                                                                                                                                                                                                                                                                                                                         |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|--------------|--|----------------------------------------------|--|-----------------------------------------------|--------------------|-----------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>CANCEL(...)</b> <b>AXIS(...)</b> command on the <i>Motion Coordinator</i> .                                                                                                                                                                                                                                                               |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
| <b>Syntax:</b>       | <code>Cancel(Mode, [Axis])</code>                                                                                                                                                                                                                                                                                                                                       |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
| <b>Parameters:</b>   | <table><tr><td><b>Short Mode:</b></td><td>Cancel mode.</td></tr><tr><td></td><td>0 cancels the current move on the base axis.</td></tr><tr><td></td><td>1 cancels the buffered move on the base axis.</td></tr><tr><td><b>Short Axis:</b></td><td>Optional parameters that must be a single numeric value that specifies the base axis for this move.</td></tr></table> | <b>Short Mode:</b> | Cancel mode. |  | 0 cancels the current move on the base axis. |  | 1 cancels the buffered move on the base axis. | <b>Short Axis:</b> | Optional parameters that must be a single numeric value that specifies the base axis for this move. |
| <b>Short Mode:</b>   | Cancel mode.                                                                                                                                                                                                                                                                                                                                                            |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
|                      | 0 cancels the current move on the base axis.                                                                                                                                                                                                                                                                                                                            |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
|                      | 1 cancels the buffered move on the base axis.                                                                                                                                                                                                                                                                                                                           |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
| <b>Short Axis:</b>   | Optional parameters that must be a single numeric value that specifies the base axis for this move.                                                                                                                                                                                                                                                                     |                    |              |  |                                              |  |                                               |                    |                                                                                                     |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                                                                                                                                                                                                                                                                                      |                    |              |  |                                              |  |                                               |                    |                                                                                                     |

---

## Connect

---

|                        |                                                                                                                                                                                                                                                                                                               |                      |                               |                        |                   |                    |                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-------------------------------|------------------------|-------------------|--------------------|-----------------------------------------------------------------------------------------------------|
| <b>Description:</b>    | Performs the corresponding <b>CONNECT(...)</b> <b>AXIS(...)</b> command on the <i>Motion Coordinator</i> .                                                                                                                                                                                                    |                      |                               |                        |                   |                    |                                                                                                     |
| <b>Syntax:</b>         | <code>Connect(Ratio, LinkAxis, [Axis])</code>                                                                                                                                                                                                                                                                 |                      |                               |                        |                   |                    |                                                                                                     |
| <b>Parameters:</b>     | <table><tr><td><b>Double Ratio:</b></td><td>The gear ratio to be applied.</td></tr><tr><td><b>Short LinkAxis:</b></td><td>The driving axis.</td></tr><tr><td><b>Short Axis:</b></td><td>Optional parameters that must be a single numeric value that specifies the base axis for this move.</td></tr></table> | <b>Double Ratio:</b> | The gear ratio to be applied. | <b>Short LinkAxis:</b> | The driving axis. | <b>Short Axis:</b> | Optional parameters that must be a single numeric value that specifies the base axis for this move. |
| <b>Double Ratio:</b>   | The gear ratio to be applied.                                                                                                                                                                                                                                                                                 |                      |                               |                        |                   |                    |                                                                                                     |
| <b>Short LinkAxis:</b> | The driving axis.                                                                                                                                                                                                                                                                                             |                      |                               |                        |                   |                    |                                                                                                     |
| <b>Short Axis:</b>     | Optional parameters that must be a single numeric value that specifies the base axis for this move.                                                                                                                                                                                                           |                      |                               |                        |                   |                    |                                                                                                     |
| <b>Return Value:</b>   | 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

---

|                      |                                                                                                                |                                                                                                                                                                                     |
|----------------------|----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | Performs the corresponding <b>MOVEHELICAL(...)</b> <b>AXIS(...)</b> command on the <i>Motion Coordinator</i> . |                                                                                                                                                                                     |
| <b>Syntax:</b>       | <b>MoveHelical(FinishBase, FinishNext, CentreBase, CentreNext, Direction, LinearDistance, [Axis])</b>          |                                                                                                                                                                                     |
| <b>Parameters:</b>   | <b>Double FinishBase:</b>                                                                                      | Distance to the finish position on the base axis.                                                                                                                                   |
|                      | <b>Double FinishNext:</b>                                                                                      | Distance to the finish position on the axis that follows the base axis.                                                                                                             |
|                      | <b>Double CentreBase:</b>                                                                                      | Distance to the centre position on the base axis.                                                                                                                                   |
|                      | <b>Double CentreNext:</b>                                                                                      | Distance to the centre position on the axis that follows the base axis.                                                                                                             |
|                      | <b>Short Direction:</b>                                                                                        | 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. |
|                      | <b>Double LinearDistance:</b>                                                                                  | The linear distance to be moved on the base axis + 2 whilst the other two axes are performing the circular move.                                                                    |
|                      | <b>Short Axis:</b>                                                                                             | Optional parameters that must be a single numeric value that specifies the base axis for this move.                                                                                 |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                             |                                                                                                                                                                                     |

---

## MoveLink

---

|                     |                                                                                                                          |                                           |
|---------------------|--------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| <b>Description:</b> | Performs the corresponding <b>MOVELINK(...)</b> <b>AXIS(...)</b> command on the <i>Motion Coordinator</i> .              |                                           |
| <b>Syntax:</b>      | <b>MoveLink(Distance, LinkDistance, LinkAcceleration, LinkDeceleration, LinkAxis, LinkOptions, LinkPosition, [Axis])</b> |                                           |
| <b>Parameters:</b>  | <b>Double Distance:</b>                                                                                                  | Total distance to move on the base axis.  |
|                     | <b>Double LinkDistance:</b>                                                                                              | Distance to be moved on the driving axis. |

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Double LinkAcceleration</b> | Distance to be moved on the driving axis during the acceleration phase of the move.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Double LinkDeceleration</b> | Distance to be moved on the driving axis during the deceleration phase of the move.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Short LinkAxis:</b>         | The driving axis for this move.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Short LinkOptions:</b>      | Specifies special processing for this move: <ul style="list-style-type: none"> <li>0 no special processing.</li> <li>1 link commences exactly when registration event occurs on link axis.</li> <li>2 link commences at an absolute position on link axis (see param 7).</li> <li>4 <b>MOVELINK</b> repeats automatically and bi-directionally when this bit is set. (This mode can be cleared by setting bit 1 of the <b>REP _ OPTION</b> axis parameter).</li> <li>32 Link is only active during positive moves on the link axis.</li> </ul> |
| <b>Double LinkPosition:</b>    | The absolute position on the link axis where the move will start.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Short Axis:</b>             | 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:**

|                         |                                                                                                     |
|-------------------------|-----------------------------------------------------------------------------------------------------|
| <b>Double Position:</b> | Absolute position of the end of move for the base axis.                                             |
| <b>Short Axis:</b>      | 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

---

|                      |                                                                                       |                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>RUN(...)</b> command on the <i>Motion Coordinator</i> . |                                                                                                                 |
| <b>Syntax:</b>       | <b>Run(Program, Process)</b>                                                          |                                                                                                                 |
| <b>Parameters:</b>   | <b>String Program:</b>                                                                | String that specifies the name of the program to be run.                                                        |
|                      | <b>Short Process:</b>                                                                 | Optional parameter that must be a single numeric value that specifies the process on which to run this program. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                    |                                                                                                                 |

## Stop

---

|                      |                                                                                        |                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>STOP(...)</b> command on the <i>Motion Coordinator</i> . |                                                                                                                    |
| <b>Syntax:</b>       | <b>Stop(Program, Process)</b>                                                          |                                                                                                                    |
| <b>Parameters:</b>   | <b>String Program:</b>                                                                 | String that specifies the name of the program to be stopped.                                                       |
|                      | <b>Short Process:</b>                                                                  | Optional parameter that must be a single numeric value that specifies the process on which the program is running. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                     |                                                                                                                    |

# Variable Commands

## GetTable

|                      |                                                                       |                                                                                                                                                              |
|----------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Retrieves and writes the specified table values into the given array. |                                                                                                                                                              |
| <b>Syntax:</b>       | <code>GetTable(StartPosition, NumberOfValues, Values)</code>          |                                                                                                                                                              |
| <b>Parameters</b>    | <b>Long StartPosition:</b>                                            | Table location for first value in array.                                                                                                                     |
|                      | <b>Long NumberOfValues:</b>                                           | Size of array to be transferred from Table Memory.                                                                                                           |
|                      | <b>Double Values:</b>                                                 | 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. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                    |                                                                                                                                                              |

## GetVariable

|                      |                                                                                                                                   |                                            |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| <b>Description:</b>  | Returns the current value of the specified system variable. To specify different base axes, the <b>BASE</b> command must be used. |                                            |
| <b>Syntax:</b>       | <code>GetVariable(Variable, Value)</code>                                                                                         |                                            |
| <b>Parameters:</b>   | <b>String Variable:</b>                                                                                                           | Name of the system variable to read.       |
|                      | <b>Double Value:</b>                                                                                                              | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                                                |                                            |

## GetVr

---

|                      |                                                         |                                            |
|----------------------|---------------------------------------------------------|--------------------------------------------|
| <b>Description:</b>  | Returns the current value of the specified VR variable. |                                            |
| <b>Syntax:</b>       | <code>GetVr(Variable, Value)</code>                     |                                            |
| <b>Parameters:</b>   | <b>Short Variable:</b>                                  | Number of the VR variable to read.         |
|                      | <b>Double Value:</b>                                    | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                      |                                            |

## SetTable

---

|                      |                                                                     |                                                                                                                                    |
|----------------------|---------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Sets the specified table variables to the values given in an array. |                                                                                                                                    |
| <b>Syntax:</b>       | <code>SetTable(StartPosition, NumberOfValues, Values)</code>        |                                                                                                                                    |
| <b>Parameters</b>    | <b>Long StartPosition:</b>                                          | Table location for first value in array.                                                                                           |
|                      | <b>Long NumberOfValues:</b>                                         | Size of array to be transferred to Table Memory.                                                                                   |
|                      | <b>Double Values:</b>                                               | A single numeric value or an array of numeric values that contain at least NumberOfValues values to be placed in the Table Memory. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                  |                                                                                                                                    |

## SetVariable

---

|                     |                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b> | Sets the current value of the specified system variable. To specify different base axes, the <b>BASE</b> command must be used. |
| <b>Syntax:</b>      | <code>SetVariable(Variable, Value)</code>                                                                                      |

---

|                      |                         |                                                 |
|----------------------|-------------------------|-------------------------------------------------|
| <b>Parameters:</b>   | <b>String Variable:</b> | Name of the system variable to write.           |
|                      | <b>Double Value:</b>    | Variable in which the value to write is stored. |
| <b>Return Value:</b> | See TrioPC STATUS.      |                                                 |

---

## SetVr

---

|                      |                                                  |                                                 |
|----------------------|--------------------------------------------------|-------------------------------------------------|
| <b>Description:</b>  | Sets the value of the specified Global variable. |                                                 |
| <b>Syntax:</b>       | SetVr(Variable, Value)                           |                                                 |
| <b>Parameters:</b>   | <b>Short Variable:</b>                           | Number of the VR variable to write.             |
|                      | <b>Double Value:</b>                             | Variable in which the value to write is stored. |
| <b>Return Value:</b> | See TrioPC STATUS.                               |                                                 |

---

## GetProcessVariable

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                           |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| <b>Description:</b> | 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 <i>Motion Coordinators</i> . 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. |                                                           |
| <b>Syntax:</b>      | GetProcessVariable(VariableIndex, Process, Value)                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                           |
| <b>Parameters:</b>  | <b>Short VariableIndex:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                          | The index of the variable in the process variables table. |
|                     | <b>Short Process:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                | The process number of the running process.                |
|                     | <b>Double Value:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Variable in which to store the value read.                |
| <b>Example:</b>     | Let us assume that there is the program “T1” on the <i>Motion Coordinator</i> 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:**

|                         |                                            |
|-------------------------|--------------------------------------------|
| <b>String Variable:</b> | The name of the variable.                  |
| <b>Short Axis:</b>      | The axis number.                           |
| <b>Double Value:</b>    | 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:**

|                         |                           |
|-------------------------|---------------------------|
| <b>String Variable:</b> | The name of the variable. |
| <b>Short Axis:</b>      | The axis number.          |



---

|                      |                      |               |
|----------------------|----------------------|---------------|
|                      | <b>Double Value:</b> | Value to set. |
| <b>Return Value:</b> | 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:**

|                         |                                            |
|-------------------------|--------------------------------------------|
| <b>String Variable:</b> | The name of the variable.                  |
| <b>Short Process:</b>   | The process number of the running process. |
| <b>Double Value:</b>    | 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:**

|                         |                                            |
|-------------------------|--------------------------------------------|
| <b>String Variable:</b> | The name of the variable.                  |
| <b>Short Process:</b>   | The process number of the running process. |
| <b>Double Value:</b>    | 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:**

|                               |                                            |
|-------------------------------|--------------------------------------------|
| <code>String Variable:</code> | The name of the variable.                  |
| <code>Short Slot:</code>      | The slot number.                           |
| <code>Double Value:</code>    | 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:**

|                               |                           |
|-------------------------------|---------------------------|
| <code>String Variable:</code> | The name of the variable. |
| <code>Short Slot:</code>      | The slot number.          |
| <code>Double Value:</code>    | 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.

---

---

|                      |                                                     |                                            |
|----------------------|-----------------------------------------------------|--------------------------------------------|
| <b>Syntax:</b>       | <code>GetPortVariable(Variable, Port, Value)</code> |                                            |
| <b>Parameters:</b>   | <b>String Variable:</b>                             | The name of the variable.                  |
|                      | <b>Short Port:</b>                                  | The port number.                           |
|                      | <b>Double Value:</b>                                | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                  |                                            |

---

## SetPortVariable

---

|                      |                                                                                                                                                                                                                                                                       |                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| <b>Description:</b>  | For a system variable that accepts the <code>PORT</code> 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 <code>PORT</code> modifier, then this method will fail. |                           |
| <b>Syntax:</b>       | <code>SetPortVariable(Variable, Port, Value)</code>                                                                                                                                                                                                                   |                           |
| <b>Parameters:</b>   | <b>String Variable:</b>                                                                                                                                                                                                                                               | The name of the variable. |
|                      | <b>Short Port:</b>                                                                                                                                                                                                                                                    | The port number.          |
|                      | <b>Double Value:</b>                                                                                                                                                                                                                                                  | Value to set.             |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                                                                                                                                                                                    |                           |

# Input / Output Commands

---

## Ain

---

|                      |                                                                                       |                                            |
|----------------------|---------------------------------------------------------------------------------------|--------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>AIN(...)</b> command on the <i>Motion Coordinator</i> . |                                            |
| <b>Syntax:</b>       | <b>Ain(Channel, Value)</b>                                                            |                                            |
| <b>Parameters:</b>   | <b>Short Channel:</b>                                                                 | <b>AIN</b> channel to be read.             |
|                      | <b>Double Value:</b>                                                                  | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                    |                                            |

---

## Get

---

|                      |                                                                                       |                                            |
|----------------------|---------------------------------------------------------------------------------------|--------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>GET #...</b> command on the <i>Motion Coordinator</i> . |                                            |
| <b>Syntax:</b>       | <b>Get(Channel, Value)</b>                                                            |                                            |
| <b>Parameters:</b>   | <b>Short Channel:</b>                                                                 | Comms channel to be read.                  |
|                      | <b>Short Value:</b>                                                                   | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                    |                                            |

---

## In

---

|                     |                                                                                      |                                          |
|---------------------|--------------------------------------------------------------------------------------|------------------------------------------|
| <b>Description:</b> | Performs the corresponding <b>IN(...)</b> command on the <i>Motion Coordinator</i> . |                                          |
| <b>Syntax:</b>      | <b>In(StartChannel, StopChannel, Value)</b>                                          |                                          |
| <b>Parameters:</b>  | <b>Short StartChannel:</b>                                                           | First digital I/O channel to be checked. |
|                     | <b>Short StopChannel:</b>                                                            | Last digital I/O channel to be checked.  |

---

---

|                      |                    |                                   |
|----------------------|--------------------|-----------------------------------|
|                      | <b>Long Value:</b> | Variable to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS. |                                   |

---

## Input

---

|                      |                                                                                         |                                            |
|----------------------|-----------------------------------------------------------------------------------------|--------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>INPUT #...</b> command on the <i>Motion Coordinator</i> . |                                            |
| <b>Syntax:</b>       | <b>Input(Channel, Value)</b>                                                            |                                            |
| <b>Parameters:</b>   | <b>Short Channel:</b>                                                                   | Comms channel to be read.                  |
|                      | <b>Double Value:</b>                                                                    | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                      |                                            |

---

## Key

---

|                      |                                                                                       |                                            |
|----------------------|---------------------------------------------------------------------------------------|--------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>KEY #...</b> command on the <i>Motion Coordinator</i> . |                                            |
| <b>Syntax:</b>       | <b>Key(Channel, Value)</b>                                                            |                                            |
| <b>Parameters:</b>   | <b>Short Channel:</b>                                                                 | Comms channel to be read.                  |
|                      | <b>Double Value:</b>                                                                  | Variable in which to store the value read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                    |                                            |

---

## Linput

---

|                      |                                                                                       |                                                                                |
|----------------------|---------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>LINPUT #</b> command on the <i>Motion Coordinator</i> . |                                                                                |
| <b>Syntax:</b>       | <b>Linput(Channel, StartVr)</b>                                                       |                                                                                |
| <b>Parameters:</b>   | <b>Short Channel:</b>                                                                 | Comms channel to be read.                                                      |
|                      | <b>Short StartVr:</b>                                                                 | Number of the <b>VR</b> variable into which to store the first key press read. |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                    |                                                                                |

---

## Mark

---

|                      |                                                                                                           |                                                          |
|----------------------|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>MARK(...)</b> command on the <i>Motion Coordinator</i> .                    |                                                          |
| <b>Syntax:</b>       | <b>Mark(Axis, Value)</b>                                                                                  |                                                          |
| <b>Parameters:</b>   | <b>Short Axis number:</b>                                                                                 | Axis number.                                             |
|                      | <b>Short Value:</b>                                                                                       | The stored capture value for a registration first event. |
| <b>Return Value:</b> | See TrioPC STATUS. <b>FALSE</b> if no value has been captured (no registration first event has occurred). |                                                          |

---

## MarkB

---

|                     |                                                                                         |                                                           |
|---------------------|-----------------------------------------------------------------------------------------|-----------------------------------------------------------|
| <b>Description:</b> | Performs the corresponding <b>MARKB(...)</b> command on the <i>Motion Coordinator</i> . |                                                           |
|                     | Syntax: <b>MarkB(Axis, Value)</b>                                                       |                                                           |
|                     | <b>Parameters:</b>                                                                      | <b>Short Axis number:</b> Axis number.                    |
|                     | <b>Short Value:</b>                                                                     | 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:**

|                     |                                                                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Long Output:</b> | 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. |
| <b>Short State:</b> | 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:**

|                            |                                                                                                            |
|----------------------------|------------------------------------------------------------------------------------------------------------|
| <b>Short Switch:</b>       | Switch to be set.                                                                                          |
| <b>Short Enable:</b>       | 1 to enable, 0 to disable.                                                                                 |
| <b>Short Axis:</b>         | Optional numeric value that specifies the base axis for this command.                                      |
| <b>Short OutputNumber:</b> | Optional numeric value that specifies the number of the output to set.                                     |
| <b>Short OutputStatus:</b> | Optional numeric value that specifies the signalled status of the output, 0 implies off, not-0 implies on. |
| <b>Double SetPosition:</b> | 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:**

|                          |                                                                          |
|--------------------------|--------------------------------------------------------------------------|
| <b>Short PortNumber:</b> | Number of the comms port to read (0 or 1).                               |
| <b>Short StartVr:</b>    | Number of the first variable to receive values read from the comms port. |
| <b>Short NumberVr:</b>   | Number of variables to receive.                                          |
| <b>Short Format:</b>     | 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:**

|                    |                                               |
|--------------------|-----------------------------------------------|
| <b>Short Mode:</b> | 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:**

|                           |                                                                                                                                                     |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short Destination:</b> | Address to which the data will be sent.                                                                                                             |
| <b>Short Type:</b>        | type of message to be sent: <ul style="list-style-type: none"> <li>1 Direct variable transfer.</li> <li>2 Keypad offset.</li> </ul>                 |
| <b>Short Data1:</b>       | 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. |
| <b>Short Data2:</b>       | 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:**

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

**Return Value:** See TrioPC STATUS.

# General commands

## Execute

|                      |                                                                                                                                                                                                                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Performs the corresponding <b>EXECUTE...</b> command on the <i>Motion Coordinator</i> .                                                                                                                                                                                                                                  |
| <b>Syntax:</b>       | <b>Execute(Command)</b>                                                                                                                                                                                                                                                                                                  |
| <b>Parameters:</b>   | <b>String Command:</b> String that contains a valid TrioBASIC command.                                                                                                                                                                                                                                                   |
| <b>Return Value:</b> | Boolean; <b>TRUE</b> if the command was sent successfully to the <i>Motion Coordinator</i> and the <b>EXECUTE</b> command on the <i>Motion Coordinator</i> was completed successfully and the command specified by the <b>EXECUTE</b> command was tokenised, parsed and completed successfully. Otherwise <b>FALSE</b> . |

## GetData

|                      |                                                                                                                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | This method is used when an asynchronous connection has been opened, to read data received from the <i>Motion Coordinator</i> over a particular channel. The call will empty the appropriate channel receive data buffer held by the ActiveX control. |
| <b>Syntax:</b>       | <b>GetData(channel, data)</b>                                                                                                                                                                                                                         |
| <b>Parameters:</b>   | <b>Short channel:</b> Channel over which the required data was received (0,5,6,7, or 9).<br><b>String data:</b> data received by the control from the <i>Motion Coordinator</i> .                                                                     |
| <b>Return Value:</b> | Boolean; <b>TRUE</b> - if the given channel is valid, the connection open and the data read correctly from the buffer. Otherwise <b>FALSE</b> .                                                                                                       |

## SendData

---

|                      |                                                                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | This method is used when the connection has been opened in the asynchronous mode, to write data to the <i>Motion Coordinator</i> over a particular channel. |
| <b>Syntax:</b>       | <code>SendData(channel, data)</code>                                                                                                                        |
| <b>Parameters:</b>   | <b>Short channel:</b> channel over which to send the data (0,5,6,7, or 9).<br><b>String data:</b> data to be written to the <i>Motion Coordinator</i> .     |
| <b>Return Value:</b> | Boolean; <b>TRUE</b> - if the given channel is valid, the connection open, and the data written out correctly. Otherwise <b>FALSE</b> .                     |

---

## Scope

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Initialises the data capture system in the <i>Motion Coordinator</i> for future data capture on a trigger event by executing a <i>SCOPE</i> command on the <i>Motion Coordinator</i> . A trigger event occurs when the <i>Motion Coordinator</i> executes a <b>TRIGGER</b> command.                                                                                                                                                                                                                                                         |
| <b>Syntax:</b>       | <code>Scope(OnOff, [SamplePeriod, TableStart, TableEnd, CaptureParams])</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Parameters:</b>   | <b>Boolean OnOff:</b> <b>TRUE</b> to set up and enable data capture, <b>FALSE</b> to disable it.<br><b>Long SamplePeriod:</b> Data sample period (in servo periods).<br><b>Long TableStart:</b> The table index for the start of the block of <b>TABLE</b> memory which will be used to hold captured data.<br><b>Long TableEnd:</b> The table index for the start of the block of <b>TABLE</b> memory which will be used to hold captured data.<br><b>String CaptureParams:</b> A string of up to 4 comma separated parameters to capture. |
| <b>Example:</b>      | Rem Set up to capture <b>MPOS</b> and <b>DOPS</b> on axis 5<br><code>TrioPC_Status = TrioPC1.Scope(True, 10, 0, 1000, "MPOS AXIS(5), DPOS AXIS(5)")</code>                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Return Value:</b> | See TrioPC STATUS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

# Trigger

---

|                      |                                                                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description:</b>  | Sends a <b>TRIGGER</b> command to the <i>Motion Coordinator</i> to start data capture previously configured using a <b>SCOPE</b> command. |
| <b>Syntax:</b>       | <b>Trigger()</b>                                                                                                                          |
| <b>Parameters:</b>   | None.                                                                                                                                     |
| <b>Return Value:</b> | 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

---

|                     |                                                                                                                                                                                                           |                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <b>Description:</b> | 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. |                                               |
| <b>Syntax:</b>      | <code>OnOnProgress</code>                                                                                                                                                                                 |                                               |
| <b>Parameters:</b>  | <b>Description:</b>                                                                                                                                                                                       | Textual description of the associated process |
|                     | <b>Percentage:</b>                                                                                                                                                                                        | Progress of the process in percent.           |

# Intelligent Drive Commands

---

## 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 Coordinators* that support the MehchatroLink interface.
- Syntax:** **MechatroLink(Module, Function, NumberOfParameters, MLParameters, Result)**
- Parameters:**
- |                                  |                                                                 |
|----------------------------------|-----------------------------------------------------------------|
| <b>Short Module:</b>             | Number of the MechatroLink interface module.                    |
| <b>Short Function:</b>           | MechatroLink function number.                                   |
| <b>Short NumberOfParameters:</b> | Number of parameters to use in the <b>MECHATROLINK</b> command. |
| <b>Double MLParameters:</b>      | Array of parameters to use for the <b>MECHATROLINK</b> command. |
| <b>Double Result:</b>            | 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     | Synchronous on specified port number.               |
|           | 3240  |                                                     |
|           | 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.





CHAPTER  
COMMUNICATIONS  
PROTOCOLS

12





# Introduction to Modbus

A growing number of programmable keypads and HMI's provide the user with a choice of interface protocols to enable communication with various PLCs and Industrial Computers. One such protocol is Modbus. The *Motion Coordinator* system software provides built-in support for the Modbus protocol.

## Modbus RTU

The Modbus RTU protocol provides single point to point communication between a programmable keypad/display and the *Motion Coordinator*. Implementation of the protocol is provided on serial port 1 for RS232 and port 2 for RS485. Baud rate and slave address can be set in the TrioBASIC program during serial port initialisation.

### Initialisation and Set-up

The Modbus protocol is initialised by setting the mode parameter of the `SETCOM` instruction to 4, 7 or 9. The `ADDRESS` parameter must also be set before the Modbus protocol is activated.

#### Example:

```
ADDRESS=1
```

```
SETCOM(9600,8,1,2,1,mode) `Port 1 as MODBUS port at 9600 baud
```

```
ADDRESS=1
```

```
SETCOM(19200,8,1,2,2,mode) `set up the RS485 port at 19200 baud
```

The protocol can be de-selected by setting the option to 0 in the `SETCOM` command.

```
SETCOM(19200,8,1,2,2,0) `set the RS485 port to normal mode
```

Mode can be set as follows:

mode = 4 16bit signed integer

mode = 7 IEEE floating point

mode = 9 32bit signed long

**See Also:** `SETCOM` for information about other parameters.

## Modbus TCP

The Modbus TCP protocol provides single point to point communication between a programmable keypad/display and one or more *Motion Coordinators*. Implementation of the protocol is provided on the Ethernet Port using the standard Ethernet “port 502” connection for Modbus. When the remote device opens the Modbus connection over Ethernet, the data transfers will be 16-bit signed integers routed to the `VR` memory area unless this is changed by a setting of the Ethernet command in a BASIC program run at power up.

### Initialisation and Set-up

The Modbus TCP session is started when the remote device opens Ethernet “port 502”. No initialisation is required in the *Motion Coordinator* unless the required data type and memory area differ from the default.

Change the Modbus TCP mode.

```
write=1
slot=-1 `mc464 default port
ETHERNET(write,slot,7,1) ` Set the Modbus TCP link to transfer
Floating Point Data
```

Change the Modbus TCP data area.

```
ETHERNET(write,slot,9,1) ` Set the Modbus TCP link to access
TABLE memory
```

**See also:** `ETHERNET` command for full details of all options.

**Example:** The following shows a typical set-up for a HMI panel running a Modbus Link. All references below are to the programming software supplied by the HMI manufacturer and are not specific to any individual programming environment. See your HMI programming instructions for the actual set-up sequence.

In the Controller Driver section choose “Modicon Modbus”, choose any Modicon PLC type from the PLC setup section.

Program the panel to display a variable and open up a dialog box to Define

| Choose                                          | Example          |
|-------------------------------------------------|------------------|
| Input bits, Output bits, Holding Register.      | Holding Register |
| Data size/type                                  | WORD (Binary)    |
| Address Offset.                                 | 13               |
| Display format and field width to be displayed. | Numeric 4 digits |

The *Motion Coordinator* is the slave so it will always wait for the HMI to request the data required. With the set-up shown above, the display should poll the *Motion Coordinator* for the value of  $\text{VR}(12)$  and display the data as a 4 digit number.

# Modbus Technical Reference

This section lists the *Motion Coordinator's* response to each supported Modbus Function.

## Modbus Code Table

The following Modbus Function Codes are implemented:

| Code | Function Name                 | Action                                                                                             |
|------|-------------------------------|----------------------------------------------------------------------------------------------------|
| 1    | Read Coil Status              | Returns output bit pattern                                                                         |
| 2    | Read Input Status             | Returns input bit pattern                                                                          |
| 3    | Read Holding Registers        | Returns data from VR() variables                                                                   |
| 5    | Write Single Coil             | Sets single output ON/OFF                                                                          |
| 6    | Write Single Register         | Sets the value of a single VR() variable                                                           |
| 15   | Write multiple coils          | Sets multiple output ON/OFF                                                                        |
| 16   | Write Multiple Registers      | Sets the values of a group of VR() variables                                                       |
| 23   | Read/write multiple registers | Sets the value of a group of VR() variables AND returns the values from a group of VR() variables. |



TABLE() memory can be defined as the target data area instead of VR() mamory.

### (1) Read Coil Status

|                        |                                                              |
|------------------------|--------------------------------------------------------------|
| Modbus Function Code   | 1                                                            |
| Mapped Trio Function   | Read output state: READ_OP(nn, mm)                           |
| Starting Address Range | 0 to NIO-1 (NIO = Number of Input/Output Bits on Controller) |
| Number of Points Range | 1 to (NIO-1) - Starting Address                              |
| Returned Data          | Bytes containing "Number of Points" bits of data             |

### (2) Read Input Status

|                        |                                                              |
|------------------------|--------------------------------------------------------------|
| Modbus Function Code   | 2                                                            |
| Mapped Trio Function   | Read input word: IN(nn,mm)                                   |
| Starting Address Range | 0 to NIO-1 (NIO = Number of Input/Output Bits on Controller) |
| Number of Points Range | 1 to (NIO-1) - Starting Address                              |
| Returned Data          | Bytes containing "Number of Points" bits of data             |

**(3) Read Holding Registers**

|                        |                                                                                                             |
|------------------------|-------------------------------------------------------------------------------------------------------------|
| Modbus Function Code   | 3                                                                                                           |
| Mapped Trio Function   | Read Variable (VR or TABLE)                                                                                 |
| Starting Address Range | 0 to 65535                                                                                                  |
| Number of Points Range | 1 to 127 (Number of variables to be read)                                                                   |
| Returned Data          | 2 to 254 bytes containing up to 127 16-bit Signed Integers or up to 63 32bit long words or up to 63 floats. |

**(5) Write Single Coil**

|                        |                                  |
|------------------------|----------------------------------|
| Modbus Function Code   | 5                                |
| Mapped Trio Function   | Set Single Output: OP(n,ON/OFF)  |
| Starting Address Range | 8 to 271                         |
| Data                   | 00 = Output OFF, ffH = Output ON |
| Returned Data          | None                             |

**(6) Write Single Register**

|                        |                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------|
| Modbus Function Code   | 6                                                                                              |
| Mapped Trio Function   | Set Variable: VR(addr)=data or TABLE(addr, data)                                               |
| Register Address Range | 0 to 65535                                                                                     |
| Data                   | -32768 to 32767 (16 bit signed) or $-2^{31}$ to $2^{31}-1$ (32bit signed) or 32bit IEEE float. |
| Returned Data          | None                                                                                           |

**(15) Write Multiple Coils**

|                        |                                                              |
|------------------------|--------------------------------------------------------------|
| Modbus Function Code   | 15                                                           |
| Mapped Trio Function   | Set Multiple Outputs: OP(addr, ON/OFF)... OP(addr+n, ON/OFF) |
| Starting Address Range | 8 to 271                                                     |
| Number of Points Range | 1 to 264                                                     |
| Data                   | Bit pattern                                                  |
| Returned Data          | None                                                         |

**(16) Write Multiple Registers**

|                        |                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------|
| Modbus Function Code   | 16                                                                                             |
| Mapped Trio Function   | Set Variables: VR(addr)=data1 ..... VR(addr+n)=datan or TABLE(addr, data, ..., datan)          |
| Starting Address Range | 0 to 65535                                                                                     |
| Number of Points Range | 1 to 127                                                                                       |
| Data1 to Datan         | -32768 to 32767 (16 bit signed) or $-2^{31}$ to $2^{31}-1$ (32bit signed) or 32bit IEEE float. |
| Returned Data          | None                                                                                           |

**(23) Read/Write Multiple Registers**

|                        |                                                                                                                                                                                                                   |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modbus Function Code   | 23                                                                                                                                                                                                                |
| Mapped Trio Function   | Set Variables: VR(addr)=data1 ..... VR(addr+n)=datan or TABLE(addr, data, ..., datan)<br><br>Read Variable (VR or TABLE)                                                                                          |
| Starting Address Range | 0 to 65535                                                                                                                                                                                                        |
| Number of Points Range | 1 to 127 (Number of variables to be written)<br><br>1 to 127 (Number of variables to be read)                                                                                                                     |
| Data1 to Datan         | -32768 to 32767 (16 bit signed) or $-2^{31}$ to $2^{31}-1$ (32bit signed) or 32bit IEEE float.<br><br>2 to 254 bytes containing up to 127 16-bit Signed Integers or up to 63 32bit long words or up to 63 floats. |
| Returned Data          | As Data1 to Datan                                                                                                                                                                                                 |

**Glossary**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <b>HMI</b>              | Human - Machine Interface.                                                  |
| <b>MODBUS</b>           | A communications protocol developed by Modicon, part of Groupe Schneider.   |
| <b>RTU</b>              | One of two serial transmission modes used by Modbus, the other being ASCII. |
| <b>TCP</b>              | Protocol used when Modbus is transmitted over Ethernet.                     |
| <b>Holding Register</b> | A read/write variable as defined for Modicon PLC.                           |
| <b>Coil</b>             | A programmable output as defined for Modicon PLC.                           |

# DeviceNet

The DeviceNet option allows the *Motion Coordinator* to be attached as a slave node to a DeviceNet factory network. If the built-in CANbus port is used for DeviceNet, it will not be available for CAN I/O expansion, so the digital I/O will be limited to the 8 in and 8 bi-directional on the *Motion Coordinator* itself.

## Installation and Set-up

The `DEVICENET` TrioBASIC command must be in a program that runs at power-up. See the command reference in chapter 8 for information about the use of the `DEVICENET` command. In order to prevent the *Motion Coordinator* from acting as a CANIO master and generating non-DeviceNet CANbus messages on power-up, set the `CANIO _ ADDRESS` to 33. This parameter is written directly into Flash EPROM and so it is only necessary to set `CANIO _ ADDRESS` once.

e.g. in an intialisation program:

```
IF CANIO _ ADDRESS<>33 THEN CANIO _ ADDRESS = 33
DEVICENET(slot, 0, baudrate, macid, pollbase, pollin,pollout)
```

## DeviceNet Information

The *Motion Coordinator* operates as a slave device on the DeviceNet network and supports Explicit Messages of the predefined master/slave connection set and Polled I/O. It does not support the Explicit Unconnected Message Manager (UCMM).

Polled I/O allows the master to send up to 4 integer variables to the *Motion Coordinator* and to read up to 4 integer variables from the *Motion Coordinator*. These values are mapped to the `TABLE` memory in the *Motion Coordinator*. The values are transferred periodically at a rate determined by the DeviceNet Master. The Global variables (VRs) and `TABLE` memory are also accessible over DeviceNet individually by way of the Explicit Messaging service.

## Connection Types Implemented

There are 3 independent connection channels in this DeviceNet implementation:

1. Group 2 predefined master/slave connection  
This connection will only handle Master/Slave Allocate/Release messages. The maximum message length for this connection is 8 bytes.
2. Explicit message connection  
This connection will handle explicit messaging for the DeviceNet objects defined below. The maximum message length for this connection is 242 bytes.
3. I/O message connection  
This connection will handle the I/O poll messaging. The maximum message length for this connection is 32 bytes.

## DeviceNet Objects Implemented

The *Motion Coordinator* supports the following DeviceNet object classes.

| Class | Object     | Description                                                                                                                                        |
|-------|------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x01  | Identity   | Identification and general information about the device                                                                                            |
| 0x02  | Router     | Provides a messaging connection point through which a Client may address a service to any object class or instance residing in the physical device |
| 0x03  | DeviceNet  | Provides the configuration and status of a DeviceNet port                                                                                          |
| 0x04  | Assembly   | Permits access to the I/O poll connection from the explicit message channel                                                                        |
| 0x05  | Connection | Manages the characteristics of the communications connections                                                                                      |
| 0x8a  | MC         | Permits access to the VR variables and TABLE data on the <i>Motion Coordinator</i>                                                                 |

### Identity Object

Class Code: 0x01

#### Instance Services

| Id   | Service              | Description                          |
|------|----------------------|--------------------------------------|
| 0x05 | Reset                | Reinitialises the DeviceNet protocol |
| 0x0E | Get Attribute Single | Used to read the instance attributes |

#### Instance Attributes

| Attribute ID | Access Rule | Name                                         | DeviceNet Data Type             | Data Value                                              |
|--------------|-------------|----------------------------------------------|---------------------------------|---------------------------------------------------------|
| 1            | Get         | Vendor                                       | UINT                            | 0x0115 (277)                                            |
| 2            | Get         | Product Type                                 | UINT                            | Generic Device (0x0000)                                 |
| 3            | Get         | Product Code                                 | UINT                            | The MC type as returned by the CONTROL system variable. |
| 4            | Get         | Revision<br>Major Revision<br>Minor Revision | Structure of:<br>USINT<br>USINT | 3<br>2                                                  |
| 5            | Get         | Status                                       | WORD                            | Only bit 0 (owned) is implemented                       |



| Attribute ID | Access Rule | Name                                           | DeviceNet Data Type                  | Data Value                                                                                   |
|--------------|-------------|------------------------------------------------|--------------------------------------|----------------------------------------------------------------------------------------------|
| 6            | Get         | Serial Number                                  | UDINT                                | The MC Serial Number                                                                         |
| 7            | Get         | Product Name<br>String Length<br>ASCII String1 | Structure of:<br>USINT<br>STRING(30) | 11<br>"Trio MC_<product code>", where <product code> is the same as defined for attribute 3. |

## DeviceNet Object

Class Code: 0x03

### Class Services

| Id   | Service              | Description                       |
|------|----------------------|-----------------------------------|
| 0x0E | Get Attribute Single | Used to read the class attributes |

### Class Attributes

| Attribute ID | Access Rule | Name     | DeviceNet Data Type | Data Value |
|--------------|-------------|----------|---------------------|------------|
| 1            | Get         | Revision | UINT                | 2          |

Number of Instances: 1

### Instance Services

| Id   | Service                              | Description                                                                                                                                                       |
|------|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x0E | Get Attribute Single                 | Used to read the instance attributes                                                                                                                              |
| 0x10 | Set Attribute Single                 | Used to write the instance attributes                                                                                                                             |
| 0x4B | Allocate Master/Slave Connection Set | Requests the use of the Predefined Master/Slave Connection set                                                                                                    |
| 0x4C | Release Group 2 Identifier Set       | Indicates that the specified Connections within the Predefined Master/Slave Connection Set are no longer desired. These Connections are to be released (Deleted). |

## Instance Attributes

| Attribute ID | Access Rule | Name                   | DeviceNet Data Type            | Data Value                                             |
|--------------|-------------|------------------------|--------------------------------|--------------------------------------------------------|
| 1            | Get         | MAC ID                 | USINT                          | DeviceNet node address. Software defines               |
| 5            | Get         | Allocation Information | Structure of:<br>BYTE<br>USINT | 0-63 = master address<br>The current allocation choice |

## Allocation\_byte

|       |                  |                          |
|-------|------------------|--------------------------|
| bit 0 | explicit message | Supported, 1 to allocate |
| bit 1 | Polled           | Supported, 1 to allocate |
| bit 2 | Bit_strobed      | Not supported, always 0  |
| bit 3 | reserved         | always 0                 |

## Assembly Object

Class Code: 0x04

Number of Instances: 2

There are 2 instances implemented. Instance 100 is a static input object, associated with the I/O poll producer. Instance 101 is a static output object, associated with the I/O poll consumer.

## Instance Services

| Id   | Service              | Description                           |
|------|----------------------|---------------------------------------|
| 0x0E | Get Attribute Single | Used to read the instance attributes  |
| 0x10 | Set Attribute Single | Used to write the instance attributes |

## Instance Attributes

| Attribute ID | Access Rule | Attribute | Description                                                                                                                                                                                                                                                                          |
|--------------|-------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3            | Get / Set   | Data      | <p>Get Instance 100 : The I/O poll producer is executed and the output buffer returned.</p> <p>Set Instance 100: Error.</p> <p>Get Instance 101: The last received I/O poll buffer is returned.</p> <p>Set Instance 101: The buffer received is passed to the I/O poll consumer.</p> |

## Connection Object

Class Code: 0x05

### Instance Services

| Id   | Service              | Description                           |
|------|----------------------|---------------------------------------|
| 0x0E | Get Attribute Single | Used to read the instance attributes  |
| 0x10 | Set Attribute Single | Used to write the instance attributes |

Number of Instances: 2

The values for these attributes are defined in the “Predefined master/slave connection set” of the “ODVA DeviceNet specification”.

## Instance Attributes (Instance 1)

Instance Type : Explicit Message

| Attribute ID | Access Rule | Name                            | DeviceNet Data Type | Data Value                                                             |
|--------------|-------------|---------------------------------|---------------------|------------------------------------------------------------------------|
| 1            | Get         | State                           | USINT               | 0 = nonexistent<br>1 = configuring<br>3 = established<br>4 = timed out |
| 2            | Get         | Instance Type                   | USINT               | 0 = explicit message                                                   |
| 3            | Get         | Transport Class Trigger         | USINT               | 83 hex                                                                 |
| 4            | Get         | Produced Connection ID          | UINT                | 10xxxxxx011 binary<br>xxxxxx = node address                            |
| 5            | Get         | Consumed Connection ID          | UINT                | 10xxxxxx100 binary<br>xxxxxx = node address                            |
| 6            | Get         | Initial Comm Characteristics    | USINT               | 21 hex                                                                 |
| 7            | Get         | Produced Connection Size        | UINT                | 7                                                                      |
| 8            | Get         | Consumed Connection Size        | UINT                | 7                                                                      |
| 9            | Get / Set   | Expected Packet Rate            | UINT                | 2500 default (msec)<br>with timer resolution<br>of 1mS                 |
| 12           | Get         | Watchdog Timeout Action         | USINT               | 1 = autodelete                                                         |
| 13           | Get         | Produced Connection Path Length | USINT               | 0                                                                      |
| 14           | Get         | Produced Connection Path        |                     | Null (no data)                                                         |
| 15           | Get         | Consumed Connection Path Length | USINT               | 0                                                                      |
| 16           | Get         | Consumed Connection Path        |                     | Null (no data)                                                         |

## Instance Attributes (Instance 2)

Instance Type : Polled I/O

| Attribute ID | Access Rule | Name                            | DeviceNet Data Type | Data Value                                                             |
|--------------|-------------|---------------------------------|---------------------|------------------------------------------------------------------------|
| 1            | Get         | State                           | USINT               | 0 = nonexistent<br>1 = configuring<br>3 = established<br>4 = timed out |
| 2            | Get         | Instance Type                   | USINT               | 1 = Polled I/O                                                         |
| 3            | Get         | Transport Class Trigger         | USINT               | 0x83                                                                   |
| 4            | Get         | Produced Connection ID          | UINT                | 01111xxxxxx binary<br>xxxxxx = node address                            |
| 5            | Get         | Consumed Connection ID          | UINT                | 10xxxxxx101 binary<br>xxxxxx = node address                            |
| 6            | Get         | Initial Comm Characteristics    | USINT               | 0x01                                                                   |
| 7            | Get         | Produced Connection Size        | UINT                | 0x08                                                                   |
| 8            | Get         | Consumed Connection Size        | UINT                | 0x08                                                                   |
| 9            | Get / Set   | Expected Packet Rate            | UINT                | 2500 default (msec)<br>with timer resolution of<br>1 msec              |
| 12           | Get         | Watchdog Timeout Action         | USINT               | 0                                                                      |
| 13           | Get         | Produced Connection Path Length | USINT               | 0                                                                      |
| 14           | Get         | Produced Connection Path        |                     | Null (no data)                                                         |
| 15           | Get         | Consumed Connection Path Length | USINT               | 0                                                                      |
| 16           | Get         | Consumed Connection Path        |                     | Null (no data)                                                         |
| 17           | Get         | Production Inhibit Time         | USINT               | 0                                                                      |

## MC Object

Class Code: 0x8A

### Instance Services

| Id   | Service            | Description                                                                                                                   |
|------|--------------------|-------------------------------------------------------------------------------------------------------------------------------|
| 0x05 | Reset              | Performs EX on the <i>Motion Coordinator</i> . This will reset the DeviceNet as well.                                         |
| 0x33 | Read Word - TABLE  | Reads the specified number of TABLE entries and sends their values in 16 bit 2s complement format                             |
| 0x34 | Read Word - VR     | Reads the specified number of TABLE entries and sends their values in 16 bit 2s complement format                             |
| 0x35 | Read IEEE - TABLE  | Reads the specified number of TABLE entries and sends their values in 32 bit IEEE floating point format                       |
| 0x36 | Read IEEE - VR     | Reads the specified number of VR entries and sends their values in 32 bit IEEE floating point format                          |
| 0x37 | Write Word - TABLE | Receives the specified number of values in 16 bit 2s complement format and writes them into the specified TABLE entries       |
| 0x38 | Write Word - VR    | Receives the specified number of values in 16 bit 2s complement format and writes them into the specified VR entries          |
| 0x39 | Write IEEE - TABLE | Receives the specified number of values in 32 bit IEEE floating point format and writes them into the specified TABLE entries |
| 0x3A | Write IEEE - VR    | Receives the specified number of values in 32 bit IEEE floating point format and writes them into the specified VR entries    |

The following sections describe the message body area of the Explicit Message used to specify the different services. This ignores all of the fragmentation protocol.

### Read word format

Request

|        | bit 7                                                    | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|----------------------------------------------------------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0 | 0                                                        | Service code = 0x33, or 0x34 |       |       |       |       |       |       |
| byte 1 | Class ID = 0x8A                                          |                              |       |       |       |       |       |       |
| byte 2 | Instance ID = 0x01 (this is the only instance supported) |                              |       |       |       |       |       |       |
| byte 3 | bits 15-8 of Source Address                              |                              |       |       |       |       |       |       |

|        | bit 7                            | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|----------------------------------|-------|-------|-------|-------|-------|-------|-------|
| byte 4 | bits 7-0 of Source Address       |       |       |       |       |       |       |       |
| byte 5 | ignored                          |       |       |       |       |       |       |       |
| byte 6 | Number of word values to be read |       |       |       |       |       |       |       |

### Response

|            | bit 7                | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|------------|----------------------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0     | 1                    | Service code = 0x33, or 0x34 |       |       |       |       |       |       |
| byte 1     | bits 15-8 of Value 0 |                              |       |       |       |       |       |       |
| byte 2     | bits 7-0 of Value 0  |                              |       |       |       |       |       |       |
| ...        |                      |                              |       |       |       |       |       |       |
| byte n     | bits 15-8 of Value m |                              |       |       |       |       |       |       |
| byte n + 1 | bits 7-0 of Value m  |                              |       |       |       |       |       |       |

### Write word format

Request

|            | bit 7                                                    | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|------------|----------------------------------------------------------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0     | 0                                                        | Service code = 0x37, or 0x38 |       |       |       |       |       |       |
| byte 1     | Class ID = 0x8A                                          |                              |       |       |       |       |       |       |
| byte 2     | Instance ID = 0x01 (this is the only instance supported) |                              |       |       |       |       |       |       |
| byte 3     | bits 15-8 of Source Address                              |                              |       |       |       |       |       |       |
| byte 4     | bits 7-0 of Source Address                               |                              |       |       |       |       |       |       |
| byte 5     | ignored                                                  |                              |       |       |       |       |       |       |
| byte 6     | Number of word values to be written                      |                              |       |       |       |       |       |       |
| byte 7     | bits 15-8 of Value 0                                     |                              |       |       |       |       |       |       |
| byte 8     | bits 7-0 of Value 0                                      |                              |       |       |       |       |       |       |
| ...        |                                                          |                              |       |       |       |       |       |       |
| byte n     | bits 15-8 of Value m                                     |                              |       |       |       |       |       |       |
| byte n + 1 | bits 7-0 of Value m                                      |                              |       |       |       |       |       |       |

Response

|        | bit 7 | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0 | 1     | Service code = 0x37, or 0x38 |       |       |       |       |       |       |

## Read IEEE format

Request

|        | bit 7                                                    | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|----------------------------------------------------------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0 | 0                                                        | Service code = 0x35, or 0x36 |       |       |       |       |       |       |
| byte 1 | Class ID = 0x8A                                          |                              |       |       |       |       |       |       |
| byte 2 | Instance ID = 0x01 (this is the only instance supported) |                              |       |       |       |       |       |       |
| byte 3 | bits 15-8 of Source Address                              |                              |       |       |       |       |       |       |
| byte 4 | bits 7-0 of Source Address                               |                              |       |       |       |       |       |       |
| byte 5 | ignored                                                  |                              |       |       |       |       |       |       |
| byte 6 | Number of IEEE values to be read                         |                              |       |       |       |       |       |       |

Response

|            | bit 7                 | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|------------|-----------------------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0     | 1                     | Service code = 0x35, or 0x36 |       |       |       |       |       |       |
| byte 1     | bits 7-0 of Value 0   |                              |       |       |       |       |       |       |
| byte 2     | bits 15-8 of Value 0  |                              |       |       |       |       |       |       |
| byte 3     | bits 23-16 of Value 0 |                              |       |       |       |       |       |       |
| byte 4     | bits 31-24 of Value 0 |                              |       |       |       |       |       |       |
|            | ...                   |                              |       |       |       |       |       |       |
| byte n     | bits 7-0 of Value m   |                              |       |       |       |       |       |       |
| byte n + 1 | bits 15-8 of Value m  |                              |       |       |       |       |       |       |
| byte n + 2 | bits 23-16 of Value m |                              |       |       |       |       |       |       |
| byte n + 3 | bits 31-24 of Value m |                              |       |       |       |       |       |       |

## Write IEEE format

Request

|        | bit 7                                                    | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|----------------------------------------------------------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0 | 0                                                        | Service code = 0x39, or 0x3A |       |       |       |       |       |       |
| byte 1 | Class ID = 0x8A                                          |                              |       |       |       |       |       |       |
| byte 2 | Instance ID = 0x01 (this is the only instance supported) |                              |       |       |       |       |       |       |
| byte 3 | bits 15-8 of Source Address                              |                              |       |       |       |       |       |       |
| byte 4 | bits 7-0 of Source Address                               |                              |       |       |       |       |       |       |
| byte 5 | ignored                                                  |                              |       |       |       |       |       |       |
| byte 6 | Number of IEEE values to be written                      |                              |       |       |       |       |       |       |



|            | bit 7                 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|------------|-----------------------|-------|-------|-------|-------|-------|-------|-------|
| byte 7     | bits 7-0 of Value 0   |       |       |       |       |       |       |       |
| byte 8     | bits 15-8 of Value 0  |       |       |       |       |       |       |       |
| byte 9     | bits 23-16 of Value 0 |       |       |       |       |       |       |       |
| byte 10    | bits 31-24 of Value 0 |       |       |       |       |       |       |       |
| ...        |                       |       |       |       |       |       |       |       |
| byte n     | bits 7-0 of Value m   |       |       |       |       |       |       |       |
| byte n + 1 | bits 15-8 of Value m  |       |       |       |       |       |       |       |
| byte n + 2 | bits 23-16 of Value m |       |       |       |       |       |       |       |
| byte n + 3 | bits 31-24 of Value m |       |       |       |       |       |       |       |

Response

|        | bit 7 | bit 6                        | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|------------------------------|-------|-------|-------|-------|-------|-------|
| byte 0 | 1     | Service code = 0x39, or 0x3A |       |       |       |       |       |       |

## Ethernet

Ethernet is the primary connection port to the *Motion Coordinator*. This section describes how to set up a simple Ethernet connection.

### Default IP Address

The IP address (Internet Protocol address) is a 32-bit address that has two parts: one part identifies the network, with the network number, and the other part identifies the specific machine or host within the network, with the host number. An organization can use some of the bits in the machine or host part of the address to identify a specific subnet. Effectively, the IP address then contains three parts: the network number, the subnet number, and the machine number.

The 32-bit IP address is often depicted as a dot address (also called dotted quad notation) - that is, four groups of decimal digits separated by points.

For example, the Trio Ethernet daughter board has a default IP address of:

192.168.000.250

Each of the decimal numbers represents a string of eight binary digits. Thus, the above IP address really is this string of 0s and 1s:

11000000.10101000.00000000.11111010

As you can see, points are inserted between each eight-digit sequence just as they are in the decimal version of the IP address. Obviously, the decimal version

of the IP address is easier to read and that's the form most commonly used (192.168.000.250).

Part of the IP address represents the network number or address and another part represents the local machine address (also known as the host number or address). IP addresses can be one of several classes, each determining how many bits represent the network number and how many represent the host number. IP addresses are grouped by classes A,B,C, D and E. The Trio Ethernet is set up for a Class C address.

Using the above example, here's how the IP address is divided:

```
<-Network address->.<-Host address->
192.168 . 000.250
```

The beginning Network Address portion of 192 begins with the first three bits as 110... and classifies it as a Class C address. This means you can have up to 256 host addresses on this particular network.

If you wanted to add sub-netting to this address, then some portion (in this example, eight bits) of the host address could be used for a subnet address. Thus:

```
<-Network address->.<-Subnet address->.<-Host address->
192.168 . 000 . 250
```

To simplify this explanation, the subnet has been divided into a neat eight bits but an organization could choose some other scheme using only part of the third quad or even part of the fourth quad.

A subnet (short for "sub-network") is an identifiably separate part of an organization's network. Typically, a subnet may represent all the machines at one geographic location, in one building, or on the same local area network (**LAN**).

## The Subnet Mask

A router or switch knows which bits to look at (and which not to look at) by looking at a subnet mask. In a binary mask, a "1" over a number says "Look at the number underneath"; a "0" says "Don't look." Using a mask saves the router having to handle the entire 32-bit address; it can simply look at the bits selected by the mask.

Using the Trio default IP address, the combined network number and subnet number occupy 24 bits or three of the quads. The default subnet mask carried along with the packet is:

```
255.255.255.000
```

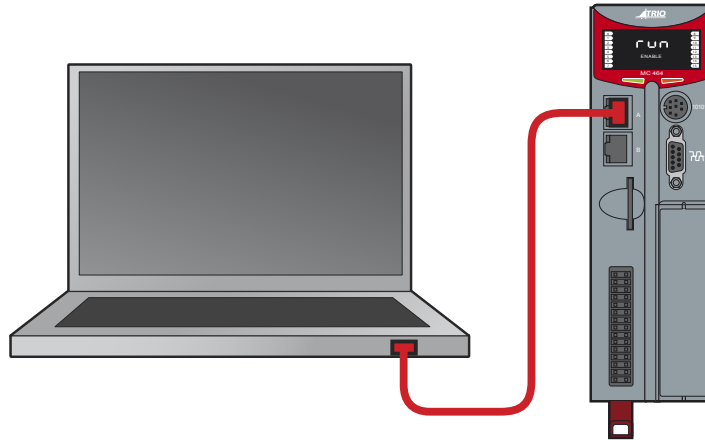
Or a string of all 1's for the first three quads (telling the router to look at these) and 0's for the host number (which the router doesn't need to look at).

## Connecting to the Trio *Motion Coordinator*

The following steps can be followed to establish an Ethernet connection from a PC to the *Motion Coordinator*.

### 1. One-to-One Connection

The Ethernet connection in the *Motion Coordinator* will adapt to the cable. Either straight or cross-over cable can be used.



The IP address of the Host PC can be set to match the default value of the Trio ethernet card.

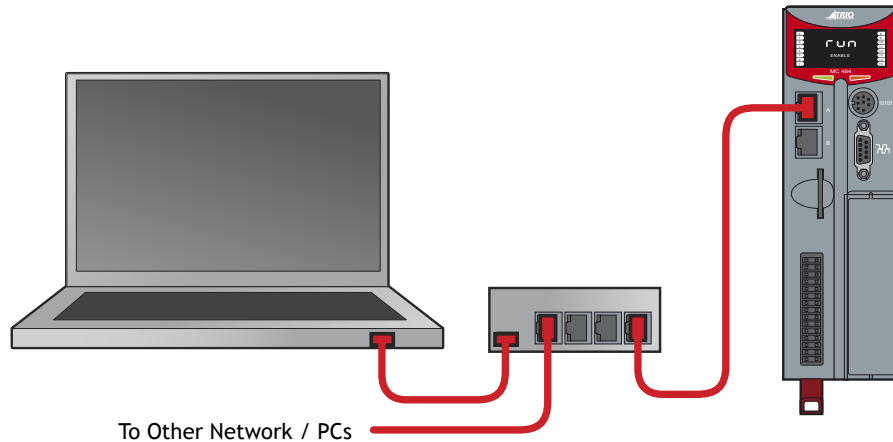
**Host PC IP:** 192.168.000.251  
**Subnet:** 255.255.255.000

**Trio IP:** 192.168.000.250  
**Subnet:** 255.255.255.000

If leaving the Trio's IP address as default, proceed to step 6 to test communications.

### 2. Connecting the Trio to Network through an Ethernet hub/switch

When connecting the Trio *Motion Coordinator* to an existing Ethernet network on a hub, simply add the connection using a high quality Ethernet cable.



The IP address of the Trio *Motion Coordinator* can be set to match the network address. The Trio's default subnet (255.255.255.000) is generic and allows any host PC to communicate with the controller regardless of a specific sub-network mask. Below is a typical example.

|             |                 |
|-------------|-----------------|
| Host PC IP: | 92.200.185.001  |
| Subnet:     | 255.255.255.224 |
| Trio IP:    | 192.200.185.a   |
| Subnet:     | 255.255.255.000 |

Where: a = Valid IP address for the Trio ethernet board on the given network

### 3. Select a valid IP address for the Trio

For this network example, the 224 in the subnet indicates the network can have up to (6) sub-networks (224 = 11100000). The (5) remaining bits within the 224 mask will allow up to 30 valid host addresses ranging from 1 to 30.

Valid IP Addresses (a) for above example:

|              |                 |    |       |          |
|--------------|-----------------|----|-------|----------|
| 002 =        | 11100010        | to | 030 = | 11111110 |
| New Trio IP: | 192.200.185.002 |    |       |          |
| Trio Subnet: | 255.255.255.000 |    |       |          |

### 4. Checking and Setting The Trio's IP Address

The IP address of the *Motion Coordinator* can be verified using the command line interface ">>" of the *Motion Coordinator*. The command line can be accessed via the terminal 0 in *Motion Perfect2*.

At the command line, use the **ETHERNET** command and type:

```
>>ETHERNET(0,0,0)
```

When connected correctly the controller will respond with the line:

```
>>192.168.000.250
```

The sequence (192.168.000.250) is the IP address of the *Motion Coordinator*.

## 5. To change the IP address to a different one

Set a new IP address to match the network:

At the command line, use the `ETHERNET` command and type:

```
>>ETHERNET(1,0,0,192,200,185,2)
```

Verify the new IP address:

```
>>ETHERNET(0,0,0)
```

The new IP address value prints out:

```
>>192.200.185.002
```

Cycle power to the *Motion Coordinator* for the new IP address to take effect.

## 6. Test the Communications

The easiest way to test the ethernet link is to “ping” the *Motion Coordinator*. This can be done using the ping command at the Windows command prompt.

From the `START` button in Windows, select Accessories and then Command Prompt utility.

At the prompt type ping followed by the *Motion Coordinators* IP address:

```
C:\>ping 192.168.0.250
```

Successful reply from controller:

```
Pinging 192.168.0.250 with 32 bytes of data:
```

```
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
```

```
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
```

```
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
```

```
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
```

```
Ping statistics for 192.168.0.250:
```

```
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
Approximate round trip times in milli-seconds:
```

```
Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

If the ping command is unsuccessful you will see:

```
C:\>ping 192.168.0.250

Pinging 192.168.0.250 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.0.250:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

## 7. Motion Perfect Terminal

If the controller was successfully ‘pinged’, then *Motion Perfect* can be used to open a remote command-line prompt connection to the controller. This tests the TCP socket connection.

- Start in disconnected mode and configure the communication link.
- From the *Motion Perfect* menu, select Options... Communications. In the Communications Links window, click the Add button, select type Ethernet and enter the IP address of the *Motion Coordinator*. Leave the IP port number as 23.
- Click ok and select this link in the list. Click ok again.
- Now open a terminal with Tools... Terminal. Make sure it shows your selected IP address at the top.
- Press the <return> key and the characteristic Trio command-line prompt ('>') should be seen.

## 8. Ethernet Ports

The controller uses various ports for different communications protocols. You should ensure that the following are all allowed on your network.

| Port  | Function              |
|-------|-----------------------|
| 23    | Telnet/ MP2/ MCLoader |
| 80    | HTTP                  |
| 502   | Modbus                |
| 3240  | Trio PC <i>Motion</i> |
| 41100 | Multiprog             |

## Anybus

The P875 Anybus adapter module allows a growing range of communication types to be added to the MC464 *Motion Coordinator*. It is designed to take an Anybus CompactCom module. This provides a standard interface into the *Motion Coordinator* while the module takes care of the detailed fieldbus operation. To allow the programmer to make configuration changes, the **ANYBUS** command in BASIC can be used to set up and modify the behaviour of the Anybus CompactCom module.

The following Anybus CompactCom modules are available.

| Module Name       | Function                                    | Supported    |
|-------------------|---------------------------------------------|--------------|
| Fieldbus Versions | ll the following are Slave modules (server) |              |
| CANopen           | CANopen slave DS301 specification (v4.02)   | Contact Trio |
| CC-Link           | Conformance to BTP-050227-B specification   | Yes          |
| ControlNet        | ControlNet slave. CIP functionality         | Contact Trio |
| DeviceNet         | DeviceNet slave. CIP functionality          | Yes          |
| Modbus-RTU        | RTU (8bit) and ASCII (7bit) support         | Contact Trio |
| Profibus          | Up to 244 bytes cyclic data transfer        | Yes          |

| Industrial Ethernet Versions | All the following are Slave Modules (server)   |              |
|------------------------------|------------------------------------------------|--------------|
| EtherCAT                     | EtherCAT I/O Slave. Max 256 Byte               | Contact Trio |
| EtherNet/IP                  | Ethernet/IP (CIP), webserver and email sending | Contact Trio |
| Modbus-TCP                   | Modbus TCP, webserver and email sending        | Contact Trio |
| Profinet-IO                  | Profinet, webserver and email sending          | Contact Trio |
| Profinet-IO 2-port           | Profinet, webserver and email sending          | Contact Trio |
| Sercos III                   | SERCOS III, webserver and email sending        | Contact Trio |
| Other Versions               | All the following function as serial ports     |              |
| Bluetooth                    | Bluetooth Class 2 SPP. Virtual Serial port     | Yes          |
| RS-232                       | Serial port                                    | Yes          |
| RS-485                       | Serial port                                    | Yes          |
| USB                          | Virtual serial port                            | Yes          |

For full information, go to the Anybus website at this [URL](http://www.anybus.com/products/abcctech.shtml):

<http://www.anybus.com/products/abcctech.shtml>

## Anybus Configuration

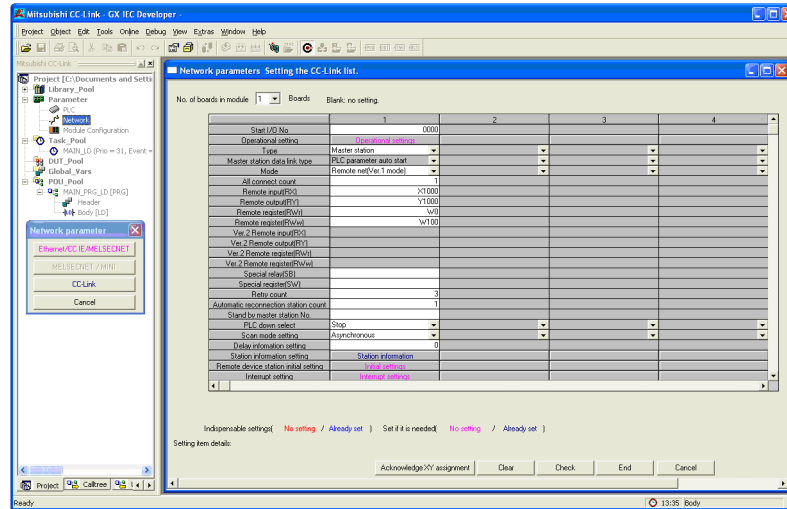
The Anybus module is automatically recognised by the *Motion Coordinator* and most modules are also configured by the system software, so there is not much setup required in the BASIC. Mostly the configuration is limited to setting the network parameters, e.g. speed, node address, and setting the target data area within the *Motion Coordinator*.

The **ANYBUS** BASIC command is used to configure the modules. Specific examples can be found in chapter 8 under the **ANYBUS** command description.

## CC-Link master configuration

The CC-Link module requires some handshaking to enter the process active state. This section details how to configure the handshaking in the master to enable communications. It is assumed that the user is experienced with the master and that they can configure the CC-Link as per the manufacturers' information.

The master must be configured as per the manufacturers' instructions. In this example the master is configured as shown below.



## Slave configuration

The current implementation in the MC464 only supports CC-link Version 1. With up to 4 stations addressable. The final 16 bits are used for handshaking and the final word write point is used for an error word.

When you map data to the *Motion Coordinator* it automatically determines how many stations to activate from the amount of data.

The maximum amount of data that can be mapped is detailed below.



| Occupied Stations | Bit points read | Bit points write | Word points read | Word points write |
|-------------------|-----------------|------------------|------------------|-------------------|
| 1                 | 16              | 16               | 4                | 3                 |
| 2                 | 48              | 48               | 8                | 7                 |
| 3                 | 80              | 80               | 12               | 11                |
| 4                 | 112             | 112              | 16               | 15                |

## Handshaking

The handshaking is performed using the final 16 bits read and write.

| Bits  | Slave -> Master                      | Master -> Slave                       |
|-------|--------------------------------------|---------------------------------------|
| 0 - 7 | Reserved                             | Reserved                              |
| 8     | Initial data processing request flag | Initial data processing complete flag |
| 9     | Initial data setting complete flag   | Initial data setting request flag     |
| A     | Error status flag                    | Error reset request flag              |
| B     | Remote READY                         | Reserved                              |
| C - F | Reserved                             | Reserved                              |

Using the configuration in previously explained (CC-Link master configuration), the formula for the memory offset is  $10(m+n)0$  Hex, where  $m$  is depending on the station number and  $n$  on the number of occupied stations.

The relations are:

$$m = (\text{station number} - 1) * 2 \text{ and}$$

$$n = \text{number of occupied stations} * 2 - 1.$$

In this example the station number is 1 therefore the  $m$ -factor is zero. The  $n$ -factor will have the values 1,3,5,7 for 1,2,3,4 occupied station(s). In this example the number of occupied stations is 2 and the  $m+n$ -factor consequently is 3. So the memory offset for the handshaking area is 1030 Hex and the complete address for the remote ready flag will be 103B.

The handshaking requires of waiting for bit 8, Initial data processing complete flag to be true. Then set bit 8 in the write area. You should then see bit B, remote ready become true and the Anybus module will enter process active.

This can be manually done using the Entry Data Monitor (on a Mitsubishi PLC) or automatically in a program. The following example could be used in structured text.

```

IF (X1038=1 AND X103B=0) THEN
 Y1038=1;
ELSE
 Y1038=0;
END _ IF;

```

## DeviceNet / Profibus Master configuration

Setting up the master is quite straight-forward because no handshake is required. Follow the guidelines provided by the PLC vendor. EDS and GSD files can be provided. Check the website for details.

Configure DeviceNet with 2 16-bit integer inputs and 2 16-bit integer outputs. This data is transmitted cyclically using the 'Polled Connection' method. Ensure to configure the master identically to the slave otherwise the data will not transmit.

## Anybus Status byte

The following example shows how the Anybus states can be read. This checks for a change in state and if the module is supervised on the network. It displays the information on one of the terminal channels.

```

Read_state:
 VR(0)=ANYBUS(3,slotnum)
 rdanybus_state=READ_BIT(2,0)*$4+READ_BIT(1,0)*$2+READ_BIT(0,0)
 IF rdanybus_state<>anybus_state THEN
 anybus_state=rdanybus_state
 PRINT#term, "ANYBUS CC CHANGED STATE"
 PRINT#term, " Anybus State = ";
 ON anybus_state+1 GOSUB s0,s1,s2,s3,s4,s5,s6,s7
 PRINT#term, ""
 anybus_state=rdanybus_state
 ENDF

 'check for change in supervisory bit
 IF supbit<>READ_BIT(3,readbit) THEN
 supbit=READ_BIT(3,readbit)
 IF READ_BIT(3,readbit)=0 THEN
 PRINT#term, "Module is not supervised"
 ELSE
 PRINT#term, "Module is supervised by another network
device"
 ENDF
 ENDF

RETURN

'Anybus State list
s0:
 PRINT#term, "SETUP"
 RETURN
s1:
 PRINT#term, "NW_INIT"
 RETURN
s2:
 PRINT#term, "WAIT_PROCESS"
 RETURN
s3:

```

```
 PRINT#term, "IDLE"
 RETURN
s4:
 PRINT#term, "PROCESS _ ACTIVE"
 RETURN
s5:
 PRINT#term, "ERROR"
 RETURN
s6:
 PRINT#term, "(reserved)"
 RETURN
s7:
 PRINT#term, "EXCEPTION"
 RETURN
```



CHAPTER  
APPENDIX

13



# Reference

## Communications Ports

| Chan | Device:-                                                     |
|------|--------------------------------------------------------------|
| 0    | Serial Port 0 - RS232 - <i>Motion Perfect</i> / Command Line |
| 1    | Serial Port 1                                                |
| 2    | Serial Port 2                                                |
| 3    | Fibre optic port (value returned defined by DEFKEY)          |
| 4    | Fibre optic port (returns raw keycode of key pressed)        |
| 5    | <i>Motion Perfect</i> user channel                           |
| 6    | <i>Motion Perfect</i> user channel                           |
| 7    | <i>Motion Perfect</i> user channel                           |
| 8    | Used for <i>Motion Perfect</i> internal operations           |
| 9    | Used for <i>Motion Perfect</i> internal operations           |
| 10   | Fibre optic network data                                     |

## Error Codes

| Number | Message                         |
|--------|---------------------------------|
| 1      | Command not recognized          |
| 2      | Invalid transfer type           |
| 3      | Error programming Flashl        |
| 4      | Operand expected                |
| 5      | Assignment expected             |
| 6      | QUOTES expected                 |
| 7      | Stack overflow                  |
| 8      | Too many variables              |
| 9      | Divide by zero                  |
| 10     | Extra characters at end of line |
| 11     | ] expected in PRINT             |

| Number | Message                               |
|--------|---------------------------------------|
| 12     | Cannot modify a special program       |
| 13     | THEN expected in IF/ELSEIF            |
| 14     | Error erasing Flash                   |
| 15     | Start of expression expected          |
| 16     | ) expected                            |
| 17     | , expected                            |
| 18     | Command line broken by ESC            |
| 19     | Parameter out of range                |
| 20     | No process available                  |
| 21     | Value is read only                    |
| 22     | Modifier not allowed                  |
| 23     | Remote axis is in use                 |
| 24     | Command is command line only          |
| 25     | Command is runtime only               |
| 26     | LABEL expected                        |
| 27     | Program not found                     |
| 28     | Duplicate label                       |
| 29     | Program is locked                     |
| 30     | Program(s) running                    |
| 31     | Program is stopped                    |
| 32     | Cannot select program                 |
| 33     | No program selected                   |
| 34     | No more programs available            |
| 35     | Out of memory                         |
| 36     | No code available to run              |
| 37     | Command out of context                |
| 38     | Too many nested structures            |
| 39     | Structure nesting error               |
| 40     | ELSE/ELSEIF/ENDIF without previous IF |
| 41     | WEND without previous WHILE           |
| 42     | UNTIL without previous REPEAT         |
| 43     | Variable expected                     |
| 44     | TO expected after FOR                 |



| Number | Message                                           |
|--------|---------------------------------------------------|
| 45     | Too may nested FOR/NEXT                           |
| 46     | NEXT without FOR                                  |
| 47     | UNTIL/IDLE expected after WAIT                    |
| 48     | GOTO/GOSUB expected                               |
| 49     | Too many nested GOSUB                             |
| 50     | RETURN without GOSUB                              |
| 51     | LABEL must be at start of line                    |
| 52     | Cannot nest one line IF                           |
| 53     | LABEL not found                                   |
| 54     | LINE NUMBER cannot have decimal point             |
| 55     | Cannot have multiple instances of REMOTE          |
| 56     | Invalid use of \$                                 |
| 57     | VR(x) expected                                    |
| 58     | Program already exists                            |
| 59     | Process already selected                          |
| 60     | Duplicate axes not permitted                      |
| 61     | PLC type is invalid                               |
| 62     | Evaluation error                                  |
| 63     | Reserved keyword not available on this controller |
| 64     | VARIABLE not found                                |
| 65     | Table index range error                           |
| 66     | Features enabled do not allow ATYPE change        |
| 67     | Invalid line number                               |
| 68     | String exceeds permitted length                   |
| 69     | Scope period should exceed number of Ain params   |
| 70     | Value is incorrect                                |
| 71     | Invalid I/O channel                               |
| 72     | Value cannot be set. Use CLEAR_PARAMS command     |
| 73     | Directory not locked                              |
| 74     | Directory already locked                          |
| 75     | Program not running on this process               |
| 76     | Program not running                               |
| 77     | Program not paused on this process                |

| Number | Message                                                          |
|--------|------------------------------------------------------------------|
| 78     | Program not paused                                               |
| 79     | Command not allowed when running <i>Motion Perfect</i>           |
| 80     | Directory structure invalid                                      |
| 81     | Directory is locked                                              |
| 82     | Cannot edit program                                              |
| 83     | Too many nested OPERANDS                                         |
| 84     | Cannot reset when drive servo on                                 |
| 85     | Flash Stick Blank                                                |
| 86     | Flash Stick not available on this controller                     |
| 87     | Slave error                                                      |
| 88     | Master error                                                     |
| 89     | Network timeout                                                  |
| 90     | Network protocol error                                           |
| 91     | Global definition is different                                   |
| 92     | Invalid program name                                             |
| 93     | Program corrupt                                                  |
| 94     | More than one program running when trying to set GLOBAL/CONSTANT |
| 95     | Program encrypted                                                |
| 96     | BASIC TOKEN definition incorrect                                 |
| 97     | ( expected                                                       |
| 98     | Number expected”;                                                |
| 99     | AS expected”;                                                    |
| 100    | STRING, VECTOR or ARRAY expected                                 |
| 101    | String expected                                                  |
| 102    | Download Abort or Timeout                                        |
| 103    | Cannot specify program type for an existing program              |
| 104    | File error: Invalid COFF image file                              |
| 105    | Variable defined outside include file                            |
| 106    | Command not allowed within INCLUDE file                          |
| 107    | Serial Number must be -1                                         |
| 108    | Append block inconsistent                                        |
| 109    | Invalid range specified                                          |
| 110    | Too many items defined for block                                 |

| Number | Message                                                     |
|--------|-------------------------------------------------------------|
| 111    | Invalid MSPHERICAL input                                    |
| 112    | Too many labels                                             |
| 113    | Symbol table locked                                         |
| 114    | Incorrect symbol type                                       |
| 115    | Variables not permitted on Command Line                     |
| 116    | Invalid program type                                        |
| 117    | Parameter expected                                          |
| 118    | Firmware error: Device in use                               |
| 119    | Device error: Timeout waiting for device                    |
| 120    | Device error: Command not supported by device               |
| 121    | Device error: CRC error                                     |
| 122    | Device error: Error writing to device                       |
| 123    | Device error: Invalid response from device                  |
| 124    | Firmware error: Cannot reference data outside current block |
| 125    | Disk error: Invalid MBR                                     |
| 126    | Disk error: Invalid boot sector                             |
| 127    | Disk error: Invalid sector/cluster reference                |
| 128    | File error: Disk full                                       |
| 129    | File error: File not found                                  |
| 130    | File error: Filename already exists                         |
| 131    | File error: Invalid filename                                |
| 132    | File error: Directory full                                  |
| 133    | Command only allowed when running <i>Motion Perfect</i>     |
| 134    | # expected                                                  |
| 135    | FOR expected                                                |
| 136    | INPUT/OUTPUT/APPEND/FIFO_READ/FIFO_WRITE expected           |
| 137    | File not open                                               |
| 138    | End of file                                                 |
| 139    | File already open                                           |
| 140    | Invalid storage area                                        |
| 141    | Invalid Floating-Point operation                            |
| 142    | Invalid System Code - wrong controller                      |
| 143    | IEC error - invalid variable access                         |

| Number | Message                                  |
|--------|------------------------------------------|
| 144    | Numerical error : Not-a-Number(NaN) used |
| 145    | Numerical error : Infinity used          |
| 146    | Numerical error : Subnormal value used   |
| 147    | MAC EEPROM is locked                     |
| 148    | Invalid mix of data types                |
| 149    | Invalid startup configuration command    |

## Data Formats and Floating-Point Operations

The TMS320C3x processor used by the *Motion Coordinator* features several different data types. In the *Motion Coordinator* we use two main formats. The following descriptions are taken directly from the TI documentation.

### Single-Precision Floating Point Format

In the single precision format, the floating-point number is represented by an 8-bit exponent field (e) and a twos complement 24-bit mantissa field (man) with an implied significant non-sign bit.

Operations are performed with an implied binary point between bits 23 and 22.

When the implied most significant non-sign bit is made explicit, it is located to the immediate left of the binary point.

The floating point number 'x' is given by:

$$\begin{aligned}
 x = & \quad 01.f \times 2^e & \quad \text{if } s=0 \\
 & \quad 10.f \times 2^e & \quad \text{if } s=1 \\
 & \quad 0 & \quad \text{if } e=-128
 \end{aligned}$$

The following examples illustrate the range and precision of the single-precision floating-point format:

$$\text{Most Positive:} \quad x = (2 - 2^{-23}) \times 2^{127} = 3.4028234 \times 10^{38}$$

$$\text{Least Positive:} \quad x = 1 \times 2^{-127} = 5.8774717 \times 10^{-39}$$

$$\text{Least Negative:} \quad x = (-1 - 2^{-23}) \times 2^{-127} = -5.8774724 \times 10^{-39}$$

$$\text{Most Negative:} \quad x = -2 \times 2^{127} = -3.4028236 \times 10^{38}$$

## Single-Precision Integer Format

In the single precision integer format, the integer is represented in twos complement notation.

|    |   |
|----|---|
| 31 | 0 |
| s  |   |

The range of an integer  $x$ , represented in the single-precision integer format, is:

$$-2^{31} \leq x \leq 2^{31} - 1$$

## Product Codes

| Processors |       |
|------------|-------|
| P860       | MC464 |

| Expansion Modules |                      |
|-------------------|----------------------|
| P871              | RTEX Interface       |
| P872              | SERCOS Interface     |
| P873              | SLM interface        |
| P874              | FlexAxis 8 Interface |
| P875              | Anybus-CC Module     |
| P876              | EtherCAT Interface   |
| P878              | Blanking Module      |
| P879              | FlexAxis 4 Interface |

| Options - I/O |                         |
|---------------|-------------------------|
| P316          | CAN 16-I/O              |
| P317          | CAN 16-Out Digital      |
| P318          | CAN 16-In Digital       |
| P319          | CAN 16-I/O Digital      |
| P326          | CAN 8-In/4-Out Analogue |
| P327          | CAN 8-Relay Out         |

| Keypads & Cables |                            |
|------------------|----------------------------|
| P381             | FlexAxis Splitter Splitter |

| Software |                                                |
|----------|------------------------------------------------|
| P877     | IEC 61131 Runtime FEC                          |
| P680     | KW Multiprog IEC 61131 Programming Environment |

A range of Fibre-Optic cables can be supplied for both the Trio FO Network and to the SERCOS specification. Contact your Trio Distributor for details.

CHAPTER  
INDEX

14





# Index

## SYMBOLS

\$ (Dollar) 8-158  
 + (Add) 8-278  
 : (Colon) 8-155  
 ' (Comment) 8-156  
 / (Divide) 8-279  
 = (Equals) 8-280  
 > (Greater Than) 8-282  
 >= (Greater Than or Equal) 8-282  
 # (Hash) 8-157  
 < (Less Than) 8-283  
 <= (Less Than or Equal) 8-283  
 \_ (Line Cont) 8-141  
 \* (Multiply) 8-279  
 <> (Not Equal) 8-281  
 ^ (Power) 8-280  
 .. (Range) 8-109  
 - (Subtract) 8-278

## A

ABS 8-284  
 ACC 8-13  
 ACCEL 8-310  
 ACOS 8-285  
 ADDAX 8-16  
 ADDAX\_AXIS 8-310  
 AddAxis 11-12  
 ADD\_DAC 8-14  
 ADDRESS 8-158  
 AFF\_GAIN 8-311  
 Ain 11-28  
 AIN 8-109  
 AND 8-285  
 Anybus 12-25  
 ANYBUS 8-159  
 AOUT 8-164  
 ASIN 8-287  
 ATAN 8-287  
 ATAN2 8-288  
 ATYPE 8-311  
 AutoRun 10-23  
 AUTORUN 8-164, 10-8  
 AXIS 8-20

AXIS\_ADDRESS 8-313  
 AXIS\_DEBUG\_A 8-313  
 AXIS\_DEBUG\_B 8-313  
 AXIS\_DISPLAY 8-314  
 AXIS\_ENABLE 8-314  
 AXIS\_ERROR\_COUNT 8-315  
 AXIS\_MODE 8-316  
 AXIS\_OFFSET 8-165  
 Axis Positioning Functions 2-4  
 AXISSTATUS 8-317  
 AXISVALUES 8-21

## B

BACKLASH 8-22  
 BACKLASH\_DIST 8-318  
 Backlit Display 2-9  
 Base 11-10  
 BASE 8-23  
 BASICERROR 8-141  
 Battery 2-9  
 BATTERY\_LOW 8-165  
 Board 11-7  
 BOOT\_LOADER 8-166  
 BREAK\_ADD 8-166  
 BREAK\_DELETE 8-167  
 BREAK\_LIST 8-167  
 BREAK\_RESET 8-168  
 B\_SPLINE 8-288

## C

Cam 11-13  
 CAM 8-24  
 CamBox 11-12  
 CAMBOX 8-29  
 CAN 8-168  
 Cancel 11-14  
 CANCEL 8-37  
 CANIO\_ADDRESS 8-174  
 CANIO\_ENABLE 8-175  
 CANIO\_STATUS 8-175  
 CANOPEN\_OP\_RATE 8-176  
 CHANGE\_DIR\_LAST 8-318  
 CHANNEL\_READ 8-110

CHANNEL\_WRITE 8-111  
 CheckProject 10-24  
 CHECKPROJECT 10-8  
 CHECKSUM 8-176  
 CHECKTYPE 10-8  
 CHECKUNLOCKED 10-9  
 CHECKVERSION 10-9  
 CLEAR 8-176  
 CLEAR\_BIT 8-291  
 CLEAR\_PARAMS 8-177  
 Close 11-5  
 CLOSE 8-112  
 CLOSE\_WIN 8-319  
 CLUTCH\_RATE 8-319  
 CmdProtocol 11-8  
 Comment 10-9  
 CommLink 10-18  
 COMMLINK 10-10  
 COMMPORT 10-10  
 COMMSERROR 8-177  
 COMMSPOSITION 8-178  
 COMMSTYPE 8-178  
 Communications 11-44  
 Communications Ports 13-3  
 COMPILE 8-180  
 CompileAll 10-24  
 COMPILE\_ALL 8-180  
 COMPILEALL 10-10  
 CompileProgram 10-25  
 COMPILEPROGRAM 10-10  
 Connect 11-14  
 CONNECT 8-40  
 Connection 11-43  
 Connections to the MC464 2-5  
 CONNPATH 8-42  
 CONSTANT 8-292  
 CONTROL 8-180  
 ControllerSystemVersion 10-19  
 ControllerType 10-19  
 COORDINATOR\_DAT 8-320  
 COPY 8-181  
 CORNER\_MODE 8-320  
 CORNER\_STATE 8-321  
 COS 8-293  
 CPU\_EXCEPTIONS 8-181  
 CRC16 8-293  
 CREEP 8-322

## D

DAC 8-325  
 DAC\_OUT 8-326  
 DAC\_SCALE 8-326  
 Data Formats and Floating-Point Operations 13-8  
 DATE 8-182  
 DATE\$ 8-183  
 Datum 11-15  
 DATUM 8-44  
 DATUM\_IN 8-327  
 DAY 8-184  
 DAY\$ 8-185  
 DECEL 8-328  
 DECEL\_ANGLE 8-328  
 DecryptionKey 10-20  
 DEFPOS 8-49  
 DEL 8-185  
 DeleteAll 10-25  
 DELETEALL 10-11  
 DeleteTable 10-26  
 DEMAND\_EDGES 8-329  
 DEMAND\_SPEED 8-330  
 DeviceNet 12-9  
 DEVICENET 8-186  
 D\_GAIN 8-323  
 Dir 11-42  
 DIR 8-187  
 DISABLE\_GROUP 8-51  
 DISPLAY 8-188  
 DLINK 8-189  
 DPOS 8-330  
 DUMP 8-193  
 D\_ZONE\_MAX 8-323  
 D\_ZONE\_MIN 8-324

## E

EDPROG 8-194  
 EMC considerations 3-7  
 ENCODER 8-331  
 ENCODER\_BITS 8-331  
 ENCODER\_CONTROL 8-332  
 ENCODER\_FILTER 8-333  
 ENCODER\_ID 8-333  
 ENCODER\_RATIO 8-54  
 ENCODER\_READ 8-334  
 ENCODER\_STATUS 8-334  
 ENCODER\_TURNS 8-335  
 ENCODER\_WRITE 8-56  
 END\_DIR\_LAST 8-335  
 ENDMOVE 8-336

ENDMOVE\_BUFFER 8-337  
 ENDMOVE\_SPEED 8-337  
 EPROM 8-197, 10-11  
 EPROM\_STATUS 8-197  
 ERROR\_AXIS 8-197  
 Error Codes 13-3  
 ERROR\_LINE 8-198  
 ERRORMASK 8-338  
 Ethernet 12-19  
 ETHERNET 8-198  
 EX 8-206  
 Execute 11-35  
 EXECUTE 8-207  
 EXP 8-295  
 Expansion Module Assembly 4-3
 

- Anybus-CC Module (P875) 4-14
- EtherCAT Interface (P876) 4-16
- Fitting Expansion Modules 4-4
- FlexAxis Interface (P874 / P879) 4-11
- Module SLOT Numbers 4-3
- RTEX Interface (P871) 4-5
- SERCOS II Interface (P872) 4-7
- SLM Interface (P873) 4-9

## F

FALSE 8-307  
 FASTDEC 8-340  
 FAST\_JOG 8-339  
 FASTLOADPROJECT 10-11  
 FastSerialMode 11-9  
 FE 8-340  
 FEATURE\_ENABLE 8-207  
 FE\_LATCH 8-341  
 FE\_LIMIT 8-342  
 FE\_LIMIT\_MODE 8-343  
 FE\_RANGE 8-343  
 FHOLD\_IN 8-344  
 FHSPEED 8-345  
 FILE 8-112  
 FLAG 8-118  
 FLAGS 8-119  
 FLASH\_DUMP 8-209  
 FLASHTABLE 8-209  
 FLASHVR 8-210  
 FLEXLINK 8-57  
 FlushBeforeWrite 11-8  
 FORCE\_SPEED 8-346  
 FOR..TO.. STEP.. NEXT 8-142  
 Forward 11-16

FORWARD 8-59  
 FPGA\_VERSION 8-211  
 FPU\_EXCEPTIONS 8-211  
 FRAC 8-295  
 FRAME 8-211  
 FRAME\_TRANS 8-212  
 FREE 8-212  
 FS\_LIMIT 8-347  
 FULL\_SP\_RADIUS 8-348  
 FWD\_IN 8-349  
 FWD\_JOG 8-349

## G

Get 11-28  
 GET 8-120  
 GetAxisVariable 11-24  
 GetConnectionType 11-6  
 GetData 11-35  
 GetLastError 10-27  
 GetLastErrorString 10-29  
 GetPortVariable 11-26  
 GetProcessVariable 11-23  
 GetProcVariable 11-25  
 GetSlotVariable 11-26  
 GetTable 11-21  
 GetVariable 11-21  
 GetVr 11-22  
 GLOBAL 8-296  
 GOSUB..RETURN 8-144  
 GOTO 8-145 ... *See also GOSUB..RETURN*

## H

HALT 8-213  
 HaltPrograms 10-29  
 HALTPROGRAMS 10-12  
 Hardware Overview 2-3
 

- Backlit Display 2-10
- Battery 2-9
- Connections to the MC464 2-5
- MC464 Feature Summary 2-11

 HLM\_COMMAND 8-214  
 HLM\_READ 8-216  
 HLM\_STATUS 8-217  
 HLM\_TIMEOUT 8-218  
 HLM\_WRITE 8-218  
 HLS\_MODEL 8-220  
 HLS\_NODE 8-220  
 HostAddress 11-7

HTTP 8-220  
 HW\_PSWITCH 8-121

## I

IDLE 8-146  
 IEEE\_IN 8-297  
 IEEE\_OUT 8-297  
 IF..THEN..ELSEIF..ELSE..ENDIF 8-146  
 I\_GAIN 8-350  
 In 11-28  
 IN 8-122  
 INCLUDE 8-220  
 INDEVICE 8-221  
 INITIALISE 8-222  
 Input 11-29  
 INPUT 8-123  
 INPUTS0 / INPUTS1 8-124  
 InsertLine 11-42  
 Installation of the MC464 3-3  
   Mounting 3-5  
   Packaging 3-3  
 INT 8-298  
 INTEGER\_READ 8-299  
 INTEGER\_WRITE 8-299  
 Introduction to the MC464 1-3, 13-3  
   Features 1-5  
   Website 1-6  
 INVERT\_IN 8-125  
 INVERT\_STEP 8-350  
 I/O Capability 2-4  
 IsOpen 11-5

## J

JOGSPEED 8-351

## K

Key 11-29  
 KEY 8-126

## L

LAST\_AXIS 8-222  
 LIMIT\_BUFFERED 8-352  
 LINK\_AXIS 8-352  
 Linput 11-30  
 LINPUT 8-127

LIST 8-223  
 LIST\_GLOBAL 8-223  
 LN 8-300  
 LOADED 8-353  
 LoadProgram 10-30, 11-41  
 LOADPROGRAM 10-12  
 LoadProject 10-30, 11-41  
 LOAD\_PROJECT 8-224  
 LOADPROJECT 10-12  
 LoadSystem 11-41  
 LOADSYSTEM 8-224  
 LoadTable 10-31  
 LOADTABLE 10-13  
 Lock 10-32  
 LOCK 8-225  
 Locked 10-20  
 LOOKUP 8-226

## M

Mark 11-30  
 MARK 8-353  
 MarkB 11-30  
 MARKB 8-354  
 MC464 Features 1-5  
 MC464 Packaging 3-3  
 MC464 Serial Connections 2-5  
 MechatroLink 11-40  
 MERGE 8-355  
 MHELICAL 8-61  
 MHELICALSP 8-64  
 MOD 8-300  
 Modbus RTU 12-3  
 Modbus TCP 12-4  
 MODULE\_IO\_MODE 8-128  
 MOTION\_ERROR 8-226  
 Motion Perfect 2 9-3  
   Axis Parameters 9-29  
   Configuring The Desktop 9-63  
   Controller Configuration 9-13  
   Creating and Running a program 9-51  
   Digital IO Status 9-45  
   Ethernet Configuration 9-14  
   Feature Enable 9-16  
   General Options 9-66  
     CAN Drive 9-67  
     CX-Drive Configuration 9-69  
     Diagnostics 9-68  
     FINS Configuration 9-69  
     menu 9-69  
     Program Compare 9-68

Terminal Font 9-68  
 Keypad Emulation 9-40  
 Linking to External Tools 9-47  
 Loading New System Software 9-19  
 Lock / Unlock Controller 9-22  
 Main Menu 9-10  
 Memory Card Support 9-17  
 Oscilloscope 9-31  
 Project Check Window 9-6  
 Projects 9-5  
 Running Programs 9-61  
 Running Without a Controller 9-70  
   CAD2Motion 9-75  
   DocMaker 9-81  
   Project Encryptor 9-72  
 Table / VR Editor 9-42  
 Terminal 9-25  
 The *Motion* Perfect Desktop 9-9  
 The *Motion* Perfect Editor 9-52  
 Tools 9-24  
 MOVE 8-67  
 MoveAbs 11-11  
 MOVEABS 8-69  
 MOVEABSSP 8-72  
 MoveCirc 11-11  
 MOVECIRC 8-73  
 MOVECIRCSP 8-76  
 MoveHelical 11-17  
 MoveLink 11-17  
 MOVELINK 8-76  
 MoveModify 11-18  
 MOVEMODIFY 8-81  
 MoveRel 11-10  
 MOVES\_BUFFERED 8-356  
 MOVESP 8-85  
 MOVETANG 8-86  
 MPE 8-227  
 MPOS 8-356  
 MSPEED 8-357  
 MSPHERICAL 8-88  
 MSPHERICALSP 8-92  
 MTYPE 8-357

## N

NAIO 8-229  
 N\_ANA\_IN 8-228  
 N\_ANA\_OUT 8-229  
 NEG\_OFFSET 8-359  
 New 11-41

NEW 8-230  
 NEWALL 10-11  
 NEXT 8-148 ... See also FOR..TO.. STEP.. NEXT  
 NIO 8-231  
 NOT 8-301  
 NTYPE 8-359

## O

OFF 8-307  
 OFFPOS 8-360  
 ON 8-308  
 OnBufferOverrunChannel0/5/6/7/9 11-38  
 ON.. GOSUB / GOTO 8-148  
 OnProgress 11-39  
 OnReceiveChannel0/5/6/7/9 11-38  
 Op 11-31  
 OP 8-129  
 Open 11-4  
 OPEN 8-131  
 OPEN\_WIN 8-361  
 OR 8-301  
 OUTDEVICE 8-231  
 OUTLIMIT 8-362  
 OV\_GAIN 8-362

## P

PEEK 8-232  
 P\_GAIN 8-363  
 PI 8-308  
 PLC\_ERROR 8-232  
 PLC\_READ 8-233  
 PLC\_STATUS 8-234  
 PLM\_OFFSET 8-363  
 PMOVE 8-235  
 POKE 8-238  
 PORT 8-239  
 POS\_OFFSET 8-364  
 POWER\_UP 8-239  
 PP\_STEP 8-364  
 PRINT 8-133  
 PRMBLK 8-240  
 PROC 8-236  
 PROCESS 8-240  
 PROC\_LINE 8-236  
 PROCNUMBER 8-237  
 PROC\_STATUS 8-236  
 Product Codes 13-9  
 ProjectFile 10-21

PROJECT\_KEY 8-241  
 PROTOCOL 8-241  
 PS\_ENCODER 8-365  
 Pswitch 11-31  
 PSWITCH 8-135

## R

RAISE\_ANGLE 8-368  
 RapidStop 11-19  
 RAPIDSTOP 8-92  
 READ\_BIT 8-302  
 READ\_OP 8-137  
 ReadPacket 11-32  
 READPACKET 8-242  
 Record 11-32  
 REG\_INPUTS 8-369  
 Regist 11-32  
 REGIST 8-96  
 REGIST\_CONTROL 8-371  
 REGIST\_DELAY 8-372  
 REGIST\_SPEED 8-373  
 REGIST\_SPEEDB 8-373  
 REG\_POS 8-370  
 REG\_POSB 8-371  
 REMAIN 8-374  
 REMOTE 8-244  
 REMOTE\_PROC 8-245  
 Removable Storage 2-4  
 RENAME 8-246  
 REP\_DIST 8-375  
 REPEAT.. UNTIL 8-150  
 REP\_OPTION 8-376  
 RESET 8-237  
 Reverse 11-16  
 REVERSE 8-104  
 REV\_IN 8-377  
 REV\_JOG 8-377  
 R\_MARK 8-365  
 R\_REGISTSPEED 8-366  
 R\_REGPOS 8-367  
 RS\_LIMIT 8-378  
 Run 11-20  
 RUN 8-246  
 RUN\_ERROR 8-238  
 RunFromEPROM 10-21  
 RUNTYPE 8-247

SCHEDULE\_TYPE 8-248  
 Scope 11-36  
 SCOPE 8-249  
 SCOPE\_POS 8-250  
 Select 11-42  
 SELECT 8-250  
 Send 11-33  
 SendData 11-36  
 SERCOS 8-251  
 SERCOS\_PHASE 8-256  
 SERIAL\_NUMBER 8-257  
 SERVO 8-379  
 SERVO\_PERIOD 8-257  
 SERVO\_READ 8-107  
 SetAxisVariable 11-24  
 SET\_BIT 8-303  
 Setcom 11-34  
 SETCOM 8-138  
 SetHost 11-6  
 SetPortVariable 11-27  
 SetProcVariable 11-25  
 SETPROJECT 10-13  
 SETRUNFROMEPROM 10-13  
 SetSlotVariable 11-26  
 SetTable 11-22  
 SetVariable 11-22  
 SetVr 11-23  
 SGN 8-303  
 SIN 8-304  
 SLOT 8-258  
 SLOT\_NUMBER 8-379  
 SPEED 8-380  
 SPEED\_SIGN 8-380  
 SPHERE\_CENTRE 8-380  
 SQR 8-305  
 SRAMP 8-381  
 START\_DIR\_LAST 8-382  
 STARTMOVE\_SPEED 8-382  
 Startup Message 10-14  
 STEP 8-258  
 STEPLINE 8-259  
 STEP\_RATIO 8-107  
 STICK\_READ 8-259  
 STICK\_READVR 8-260  
 STICK\_WRITE 8-261  
 STICK\_WRITEVR 8-262  
 Stop 11-20  
 STOP 8-263  
 STOP\_ANGLE 8-383  
 STORE 8-264

## S

SYSTEM\_ERROR 8-264  
SYSTEM\_VARIABLE 8-264

## T

TABLE 8-265  
TABLE\_POINTER 8-267  
TABLEVALUES 8-268  
TAN 8-305  
TANG\_DIRECTION 8-384  
THEN 8-150  
TICKS 8-269  
TIME 8-270  
Timeout 10-22  
TIMEOUT 10-14  
TIMER 8-139  
TO 8-151 ... *See also FOR..TO.. STEP.. NEXT*  
TOKENTABLE 8-270  
TRANS\_DPOS 8-384  
Trigger 11-37  
TRIGGER 8-270  
TrioPC Motion ActiveX Control 11-3  
TRIOPTTESTVARIAB 8-385  
TROFF 8-271  
TRON 8-272  
TRUE 8-308

## U

UNITS 8-385  
Unock 10-32  
UNTIL 8-151

## V

VECTOR\_BUFFERED 8-386  
VERIFY 8-386  
VERSION 8-274  
VFF\_GAIN 8-387  
VIEW 8-275  
VP\_SPEED 8-387  
VR 8-275  
VRSTRING 8-277

## W

WA 8-152  
WAIT 8-152  
WDOG 8-277

WEND 8-153 ... *See also WHILE*  
WHILE 8-154

## X

XOR 8-306

