

*Trio Motion Technology Ltd*  
*MOTION COORDINATOR*

*SERIES 2*  
*MC2 / MC204 / MC216*

*TECHNICAL REFERENCE MANUAL*

All goods supplied by TRIO are subject to TRIO's standard terms and conditions of sale.

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. The information in this manual applies to systems based on the *MOTION COORDINATOR* MC2, MC204, or MC216 with system software version 1.40, or higher, only.

Fifth Edition, Revision 1 September 1998

Copyright 1994..1998

Trio Motion Technology Ltd  
Unit 2  
Empire Way  
Gloucester  
GL2 5HY

Telephone (44) 1452 308332  
Fax (44) 1452 311884

---



## Contents

### 1.0 Introduction

1.1 Setup and Programming	1-1
1.2 Products	1-1
1.3 System Building	1-3
1.4 Features and Applications	1-3
1.5 Services	1-4

### 2.0 Installation of Modules

2.1 Packaging	2-1
2.1.1 Bulkhead Mounting	2-1
2.1.2 Module Interconnection	2-1
2.2 Environmental Considerations	2-1
2.3 Connection to Other TRIO Products	2-1
2.4 EMC Considerations	2-2
2.4.1 Background to EMC Directive	2-2
2.4.2 Testing Standards	2-3
2.4.3 Installation Requirements to Ensure Conformance	2-3

### 3.0 Motion Coordinator MC204/MC216/MC2 Master Modules

3.1 Motion Coordinator MC204	3-1
3.2 Motion Coordinator MC216	3-1
3.3 Motion Coordinator MC2	3-2
3.4 Axis Positioning Functions	3-3
3.5 Summary of Features MC204	3-3
3.6 Summary of Features MC216	3-3
3.7 Summary of Features MC2	3-4
3.8 Connections to Motion Coordinator SERIES 2 Master Modules	3-5
3.8.1 System 24v Power Input	3-5
3.8.2 Serial Port Connections	3-5
3.8.3 Amplifier Enable (Watchdog) Relay Outputs	3-7
3.8.4 Enabling Stepper Motor Amplifiers	3-7
3.8.5 MC2 Lower Front Panel Connections and Output Overload LED	3-8
3.8.6 MC204/MC216 Lower Front Panel Connections	3-8
3.8.7 24v Input Channels	3-9
3.8.8 24v Input/Output Channels	3-9
3.8.9 Network Interconnection	3-10
3.8.10 Analogue Inputs	3-10
3.8.11 Using End of Travel Limit Sensors	3-10

### 4.0 Axis Daughter Boards

4.1 Description	4-1
4.1.1 Fitting and Replacing Axis Daughter Boards	4-2
4.1.2 Axis Sequence in Motion Coordinator Modules	4-2
4.2 Stepper Daughter Board	4-3
4.2.1 Description	4-3
4.2.2 Connections to Stepper Daughter Board	4-3
4.3 Stepper Daughter Board with Position Verification	4-4
4.4 Servo Daughter Board	4-5
4.4.1 Description	4-5
4.4.2 Connections	4-5
4.5 Encoder Daughter Board	4-7
4.5.1 Description	4-7

4.6 Servo Resolver Daughter Board	4-8
4.6.1 Description	4-8
4.7 Voltage Output Daughter Board	4-10
4.7.1 Description	4-10
4.8 AbsoluteServo (SSI) Daughter Board	4-11
4.8.1 Description	4-11
4.8.2 SSI Connections	4-11
4.9 Differential Stepper Daughter Board	4-13
4.10 CAN Daughter Board	4-14
4.10 Hardware PSWITCH Daughter Board	4-14

## **5.0 Input/Output Modules and Operator Interfaces**

5.1 General Description	5-1
5.2 16 IO Digital Input/Output Module	5-1
5.2.1 Connection of 16IO Modules to System	5-3
5.2.2 Front Panel Indicators	5-3
5.2.3 Reading an Input / Setting an Output	5-2
5.2.4 Selection of IO Bank	5-4
5.2.5 Summary of Features	5-5
5.3 CAN 16-I/O Module	5-6
5.3.1 CAN I/O Connections	5-6
5.3.2 CAN Bus Wiring	5-7
5.3.3 DIP Switch Settings	5-8
5.3.4 LED Indicators	5-9
5.3.5 Software Interfacing	5-10
5.3.6 Troubleshooting	5-11
5.3.7 Specification	5-11
5.4 CAN Analog Input Module	5-12
5.4.1 CAN I/O Connections	5-12
5.4.2 CAN Bus Wiring	5-12
5.4.3 DIP Switch Settings	5-12
5.4.4 LED Indicators	5-12
5.4.5 Software Interfacing	5-13
5.4.6 Troubleshooting	5-13
5.4.7 Specification	5-13
5.5 Operator Interfaces on the Fibre Optic Network	5-14
5.5.1 Membrane Keypad	5-14
5.5.2 Mounting the Membrane Keypad	5-14
5.5.3 Connection of Membrane Keypad	5-14
5.6 Mini-Membrane Keypad	5-14
5.6.1 Mounting the Mini-Membrane Keypad	5-15
5.6.2 Connection of Mini-Membrane Keypad	5-15
5.6.3 Programming the Membrane Keypad and Mini-membrane Keypad	5-15
5.6.3.1 Writing to the Membrane and Mini-Membrane Display	5-15
5.6.3.2 Reading from the Membrane/Mini-Membrane Keypad	5-16
5.6.3.3 Keypad KEY ON - KEY OFF Mode	5-17
5.6.4 Summary of Features P503 Membrane Keypad	5-17
5.7 FO-VFKB Fibre-Optic Keypad/Display Interface	5-18
5.7.1 FO-VFKB Display Interface	5-18
5.7.2 FO-VFKB Keypad Interface	5-18
5.7.3 Power Requirements	5-18
5.7.4 Data Connection	5-19
5.7.5 Summary of Features P504 FO-VFKB	5-19
5.8 Membrane Cut-Outs and Overlays	5-20

## 6.0 System Setup and Diagnostics

6.1 Preliminary Concepts	6-1
6.2 System Setup	6-1
6.2.1 Preliminary checks	6-1
6.2.2 Checking Serial Communications and System Configuration	6-2
6.2.3 Input/Output Connections	6-3
6.2.4 Connecting a Servo Motor to a Servo Daughter Board	6-3
6.2.5 Setting Servo Gains	6-4
6.2.6 Gain Parameters	6-5
6.2.6.1 Proportional Gain	6-5
6.2.6.2 Integral Gain	6-5
6.4.6.3 Derivative Gain	6-5
6.4.6.4 Output Velocity	6-5
6.4.6.5 Velocity Feed Forward	6-5
6.3 Diagnostics	6-6
6.3.1 No Status LEDs	6-6
6.3.2 Status LEDs indicate an error condition	6-6
6.3.3 Motor runs away without issuing a move command	6-6
6.3.4 Motor runs away upon issuing a move command	6-6
6.3.5 Motor does not move upon issuing a move command	6-6
6.3.6 Axis goes out on following error after a time	6-6
6.3.7 Losing Position	6-6
6.3.8 Motion Perfect cannot "connect" with the controller	6-7

## 7.0 Multi-tasking BASIC Programming Concepts

7.1 Programming Concepts	7-1
7.1.1 Overview	7-1
7.1.2 Exercise: Editing and running 2 simple programs	7-2
7.1.3 Storage of programs on the SERIES 2 controllers	7-5
7.1.4 The Motion Coordinator screen editor and the Motion Perfect Editor	7-6
7.1.5 Compilation of SERIES 2 programs	7-7
7.1.6 Saving and loading programs to and from a PC	7-7
7.1.7 Commands exclusive to the MC2, MC216 and MC204	7-7
7.1.8 Local and global Variables and Labels	7-8
7.1.9 Table values on the SERIES 2 controllers	7-8
7.1.10 Setting SERIES 2 programs to run on power up	7-9
7.1.11 Forcing priority of program execution	7-10
7.1.12 Command line on SERIES 2 controllers	7-10
7.1.13 Typing commands for immediate execution	7-10
7.1.14 Simple Example BASIC Programs	7-11
7.2 AXIS, SYSTEM and PROCESS Parameters	7-12
7.2.1 Axis Parameters	7-12
7.2.2 System Parameters	7-12
7.2.3 Process Parameters	7-12

## **8.0 TRIO BASIC Programming - Command Reference**

8.1 Command Reference	8-1
8.1.1 Motion Commands	8-1
8.1.2 Input/Output Commands	8-1
8.1.3 Program Loop and Structures	8-2
8.1.4 Program and System Parameters and Functions	8-2
8.1.5 Mathematical Functions and Commands	8-4
8.1.6 Constants	8-4
8.1.7 Axis Parameters	8-5
8.2 Command and Function Description in Alphabetical Order	8-7

## **9.0 TRIO BASIC Example Programs**

9.1 Fetching an integer value from the membrane keypad	9-1
9.2 Fetching an real value from the membrane keypad	9-1
9.3 Rotating printhead with registration	9-2
9.4 Motion Coordinator SERIES 2 programs sharing data	9-4

## **10.0 *Motion Perfect***

10.1 What is <i>Motion Perfect</i> ?	10-1
10.2 What can <i>Motion Perfect</i> do for me ?	10-1
10.3 <i>Motion Perfect</i> PC Requirements	10-1
10.4 Obtaining <i>Motion Perfect</i>	10-4
10.5 Installing <i>Motion Perfect</i>	10-4
10.6 Connecting <i>Motion Perfect</i> to your Motion Coordinator	10-5
10.7 <i>Motion Perfect</i> Projects	10-6
10.8 <i>Motion Perfect</i> Project Manager	10-6
10.9 Check Project Options Dialog	10-8
10.10 A Simple <i>Motion Perfect</i> Session	10-11
10.11 <i>Motion Perfect</i> Tools	10-12
10.11.1 Axis Parameters	10-12
10.11.2 Controller Configuration	10-14
10.11.3 Editor	10-15
10.11.4 Debugger	10-20
10.11.5 Full Program Directory	10-23
10.11.6 IO Window	10-25
10.11.7 Jog Axes Window	10-26
10.11.8 Keypad Emulator	10-29
10.11.9 Oscilloscope	10-32
10.11.10 Terminal Windows	10-39
10.12 Hints and Tips for using <i>Motion Perfect</i>	10-40
10.13 Problems and Frequently Asked Questions	10-42
10.14 Loading a new system software version onto the Motion Coordinator	10-43

## **11.0 The TRIO Fibre Optic Network**

11.1	General Description	11-1
11.2	Connection of Network	11-1
11.2.1	Network Node Addressing	11-2
11.3	Network Programming	11-2
11.3.1	BASIC Commands	11-2
11.3.1.1	GET#n,VR(x)	11-2
11.3.1.2	KEY#n	11-3
11.3.1.3	PRINT#n,	11-3
11.3.1.4	SEND(n,type,data1 [,data2])	11-3
11.3.2	Examples of Network Programming	11-4
11.4	Network Specification	11-7
11.4.1	Message Format	11-7
11.4.2	Network Protocol	11-7
11.4.3	Message Types	11-7
11.4.4	Network Buffers	11-7
11.4.5	Network Status	11-8





## 1.0 Introduction

Trio Motion Technology's range of *Motion Coordinator* products were designed to enable the control of industrial machines with a minimum of external components. The products may be combined to build a control system capable of driving a multi-axis machine and its auxiliary equipment. The *Motion Coordinator* system allows you to control up to 16 servo or stepper motors, Digital I/O and additional equipment such as keypads and displays from a single master. Up to fifteen masters can be networked together using the TRIO fibre optic network allowing up to 180 axes of control. The controller is programmed using the TRIO BASIC programming language. This may be used to build "standalone programs" or commands can be sent from an external computer.

The TRIO control system is modular, allowing the user to tailor the controller to their specific needs, but also allowing the flexibility to incorporate new modules if needs should change.

### 1.1 Setup and Programming

To program the *Motion Coordinator* a dumb terminal or computer running a terminal emulation program must be connected via an RS-232-C serial link. The Trio Motion Technology *Motion Perfect* program is normally used to provide this and a wide range of other facilities, on an IBM PC or 100% compatible microcomputer running Windows or Windows 95.

Independent of the number of modules and axes, only one serial link is required between the host and the master module, because the individual modules are interconnected by a dedicated ribbon cable link.

Once connected to the master module, the user has direct access to the TRIO BASIC, 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, synchronized multi axis moves and unsynchronized multi axis moves as well as the control of the digital I/O.

The *Motion Coordinator* SERIES 2 range currently consists of the MC2, MC204 and MC216. These controllers feature multi-tasking BASIC. Multiple TRIO BASIC programs can be constructed and run simultaneously to make programming complex applications much easier.

### 1.2 Products

The range of Trio *Motion Coordinator* SERIES 2 products covered by this manual:

MC2	High speed, high performance master controller for 1-12 axes. 8 Opto-isolated Inputs and 8 Opto-isolated Input/Output channels are built in. Multi-tasking TRIO BASIC. I/O expansion to 96 channels.
MC204	Low cost, medium performance master controller for 1-4 axes. 8 Opto-isolated Inputs and 8 Opto-isolated Input/Output channels are built in. Multi-tasking TRIO BASIC. I/O expansion is via CAN bus.
MC216	Trio's highest performance master controller for 1-16 axes. 8 Opto-isolated Inputs and 8 Opto-isolated Input/Output channels are built in. Multi-tasking TRIO BASIC. I/O expansion is via CAN bus.

## ► Trio Motion Technology ◀

---

CAN 16 IO	DIN rail mounted 24v I/O expander module provides 16 opto-isolated channels each of which may be used as an Input or an Output. <u>This unit connects to the MC204/MC216 ONLY.</u>
CAN Analog Inputs	DIN rail mounted +/-10volt Analog Input module provides 8 opto-isolated channels. <u>This unit connects to the MC204/MC216 ONLY.</u>
16 IO	24v I/O expander module provides 16 opto-isolated channels each of which may be used as an Input or an Output. <u>This unit connects to the MC2 ONLY</u>
Axis Expander	Expansion module provides housing for up to 4 additional axis daughter boards. <u>Up to 2 connect to MC2 and up to 3 connect to MC216</u>

### Daughter Boards:

Servo Encoder	Mini board which provides interface for a servo motor with encoder feedback
Stepper	Mini board which provides interface for a stepper motor. Outputs are open-collector.
Reference Encoder	Mini board which provides an encoder reference input
Servo Resolver	Mini board which provides interface for a servomotor with resolver feedback
Stepper Encoder	Mini board which provides interface to a stepper motor with additional facility for encoder feedback and high speed registration using step-pulses.
Voltage Output	Mini board which provides a 12 bit +/-10v voltage output.
SSI Absolute	Interface to servo motor with absolute feedback using SSI absolute encoder system.
Differential Stepper	Stepper motor interface with differential output drivers.
CAN Daughter	Provides additional CAN interface. (MC204/MC216 Only)
Hardware PSWITCH	Switches 4 outputs in electronic hardware over position sector

### Operator Interfaces:

Mini-Membrane Keypad:	Compact operator keypad/display
Membrane Keypad:	High performance general purpose operator keypad/display.
Touchpro PC:	Colour graphics touchscreen operator interface (Contact Trio if you may have an application for this product - it is not covered by this manual)

### Options:

Analog Inputs:	Multiple channel analog input board. (MC2 Only)
----------------	---

### 1.3 System Building

The modules and boards may be mixed within the system rules:

- 1 Every system configuration must start with one MC2,MC204 or MC216 master unit as this contains the processor and logic power supply for the system.
- 2 Each master unit can house up to 4 daughter boards. These can be of any type, with the exception that the CAN daughter board is only compatible with the MC204/MC216.
- 3 MC2 can have up to 2 axis expander modules added to house up to 12 daughter boards. 4 being housed in the Master and 4 in each of the axis expanders. The MC216 can have up to 3 axis expander modules added to house up to 16 daughter boards.
- 4 The MC204 and MC216 can have up to 16 CAN-16I/O modules connected.
- 4 MC2 master module can have up to five 16 IO modules connected.
- 5 On the MC2 the total number of modules linked together by the dedicated ribbon cable, including the digital I/O, must not exceed six.

### 1.4 Features and Applications

The TRIO BASIC contains accurate motion control functions that permit the generation of complex movements of various types, including:

- Linear interpolation
- Circular and helical interpolation
- Variable speed and acceleration profiles
- Electronic gearboxes
- Electronic cam profiles

The operator interface may be achieved by any combination of the following:

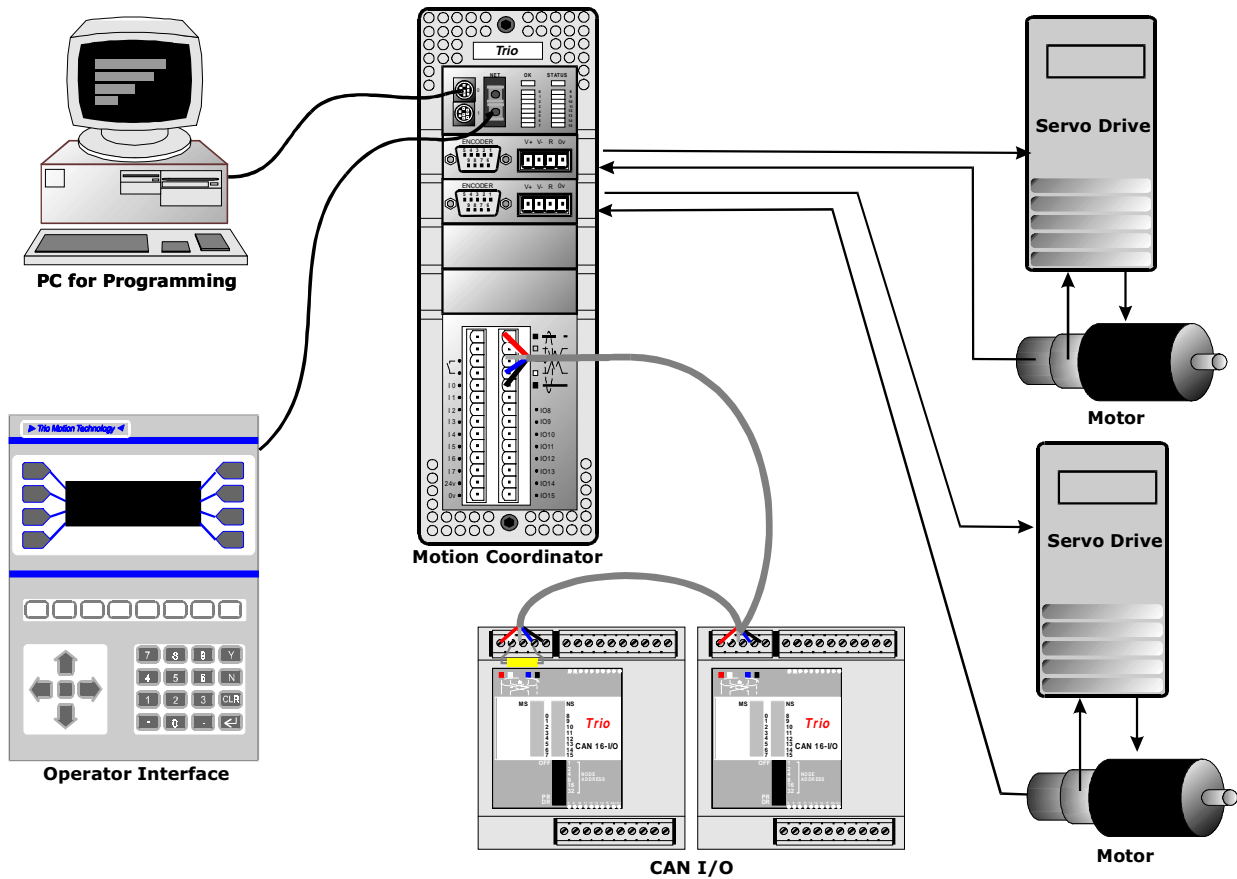
- Dedicated host computer
- Membrane Keypad with Vacuum Fluorescent Display
- Switches
- Thumbwheels
- Status lamps

The system is able to control a wide range of mechanisms and equipment including:

- Brushless servo motors
- Stepper motors
- Brushed DC servo motors
- Hydraulic servo valves
- Hydraulic proportional valves
- Pneumatic/hydraulic solenoids
- Relays/contactors

Typical applications include:

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• Automotive welding</li> <li>• Web control</li> <li>• Drilling</li> <li>• Glue laying</li> <li>• Laser guidance</li> <li>• Packaging</li> <li>• Spark erosion</li> </ul> | <ul style="list-style-type: none"> <li>• Coil winding</li> <li>• Cut to length</li> <li>• Electronic component assembly</li> <li>• Flying shears</li> <li>• Milling</li> <li>• Palletisation</li> <li>• Tension control</li> </ul> |
|--|--|



## 1.5 Services

By the time you have received this manual you will have been advised about product selection for your particular application. Trio's after sales services include:

- Installation
- Commissioning
- Application programming including the generation of software for stand alone applications.
- Custom built boards for specific requirements using the in-house CAD facilities.
- On-site servicing.

All Trio products are fully guaranteed for 12 months replacement or repair on a return to base system.

## 2.0 Installation of Modules

### 2.1 Packaging

All of the TRIO *MOTION COORDINATOR* modules, except the CAN DIN rail mounting modules, are supplied with the same exterior packaging which has been specifically designed for ease of use and flexibility of mounting. The dimensions are given in diagram 2.1 overleaf.

#### 2.1.1 Bulkhead Mounting

The module may be bulkhead mounted by securing in two locations through the rear panel of the unit with M4 screws. The unit should be mounted vertically and should not be subjected to mechanical loading. Care should be taken to ensure that there is a free flow of air vertically through the unit.

#### 2.1.2 Module Interconnection

Modules to be connected should be mounted on 70mm centres at the same height. The MASTER MC204 or MC2 of the group should be at the right hand end. After mounting on the bulkhead the interconnection bus cover may be removed by unscrewing the 2 securing screws. The appropriate length of ribbon cable will be supplied ready assembled and tested by TRIO. With the power off the ribbon cable can be inserted and the cable locked with the ears on each connector. The power may then be re-applied and all the interconnection bus covers replaced

**Note:** Modules must never be connected or disconnected whilst the power supply is on.

### 2.2 Connection To Other TRIO Products

Trio *MOTION COORDINATOR* products are NOT compatible with Trio 'CSC' product range and the two should never be mixed on the ribbon cable interconnection.

### 2.3 Environmental Considerations

Ensure that the area around the ventilation holes and top of the module are kept clear. Avoid violent shocks to, or vibration of, the system modules whilst in use or storage.

## **2.4 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.

- (i) Mount noise generators such as contactors, solenoid coils and relays as far away as possible from the modules.
- (ii) Where possible use solid-state contactors and relays.
- (iii) Fit suppressors across coils and contacts.
- (iv) Route heavy current power and motor cables away from signal and data cables.
- (v) Ensure all the modules have a secure earth connection.
- (vi) Where screened cables are used terminate the screen with a 360 degree termination, if possible, rather than a "pig-tail" and connect both ends of the screen to ground.

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 help the equipment manual should offer further assistance. For Trio specific installation instructions see section 2.4.3. The consideration of EMC implications is now 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.

### **2.4.1 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:

- (i) Equipment must be constructed to ensure that any electromagnetic disturbance it generates allows radio and telecommunications equipment and other apparatus to function as intended.
- (ii) 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 independant 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 relevent 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.

#### **2.4.2 Testing Standards**

For the purposes of testing a typical system configuration had to be 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:

- (i) Emissions - BS EN55022:1995 or BS EN50081-1:1992 depending on the particular product. Note that both standards specify the same limits for radiated emissions which is the only applicable part of the standards to Trio products. Most products conform to the Class A limits but some products, such as the range of membrane keypads, are within Class B limits.
- (ii) Immunity - BS EN50082-2:1995. This standard sets limits for immunity in an industrial environment and is a far more rigorous test than part 1 of the standard.

#### **2.4.3 Installation Requirements to Ensure 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:

- (i) The case of the modules should be earthed via their back panel. If the mounting panel is painted an area of paint should be removed to ensure a good electrical connection between the module and the cabinet.
- (ii) Ferrite cores must be fitted to 24V IO lines on MC2 and MC16IO. One ferrite core per bank of 8 I/O will provide sufficient attenuation. It should be placed as close to the terminal connector on the module as possible. If any IO 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 IO port can act as an antenna for noise.
- (iii) Screened cables should be used for encoder, resolver 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 should 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 one of the screw locks of the D-type shell of the encoder connector.
- (iv) Connection to the serial ports should be made with a Trio supplied cable. When a Motion Coordinator is not connected to a PC the serial cable must be removed as it will act as an antenna for electrical noise if it is left unterminated.

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

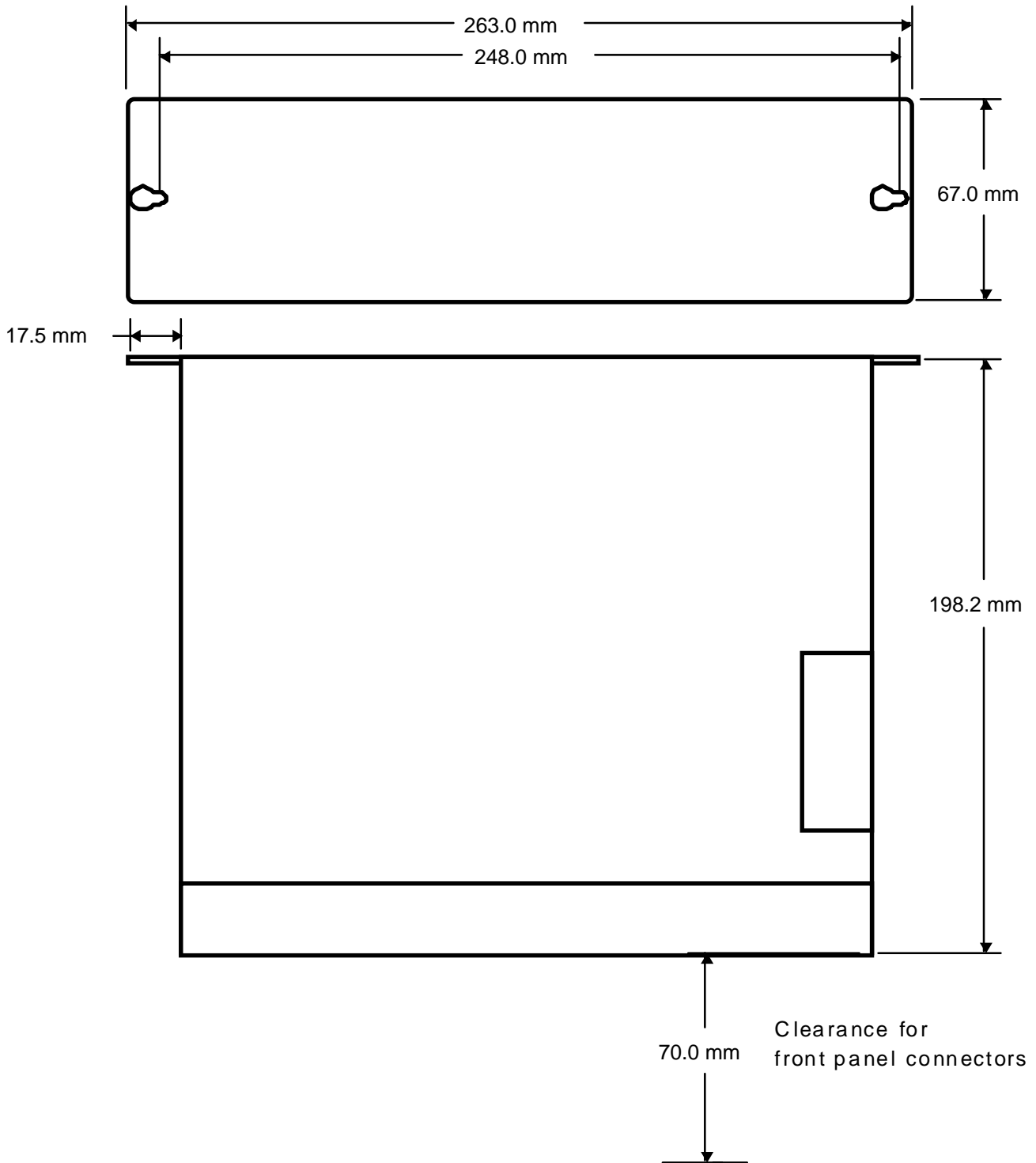


Fig 2.1 Mounting Details



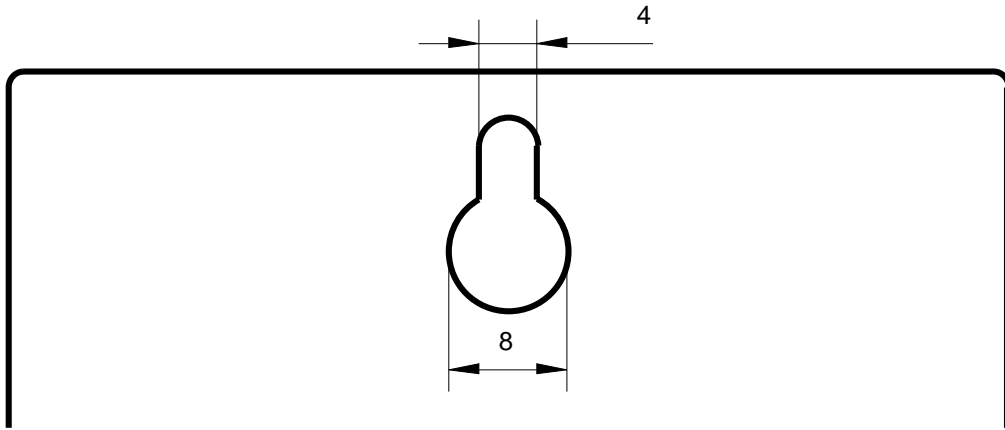


Fig 2.2 Details of Module Mounting



### **3.0 Motion Coordinator Description**

#### **3.1 Motion Coordinator MC204**

The *Motion Coordinator* MC204 is a modular servo control positioner with the ability to control up to 4 servo or stepper motors in any combination by the insertion of "Axis Daughter Boards" to suit the application. It is housed in a rugged metal 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 with a Multi-tasking TRIO BASIC using a PC, and then may be set to run "standalone" if an external computer is not required for the final system. The Multi-tasking version of Trio BASIC for the MC204 allows up to 5 BASIC programs to be run simultaneously on the controller using pre-emptive multi-tasking.

The Multi-tasking ability of the MC204 allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware.

The MC204 has 8 built in 24v inputs and 8 bi-directional IO channels. These may be used for system interaction or may be defined to be used by the controller for end of travel limits, datuming and feedhold functions if required. Each of the Input/Output channels has a status LED to make it easy to check them at a glance. The MC204 can have up to 512 external Input/Output channels connected using DIN rail mounted CAN 16-I/O modules. These units connect to the built-in CAN channel.

The MC204 has two built in RS-232 ports and one duplex RS-485 channel for simple factory communication systems. The TRIO fibre optic network system can be fitted as an option.

#### **3.2 Motion Coordinator MC216**

The *Motion Coordinator* MC216 is Trio's most powerful modular servo control positioner with the ability to control up to 16 servo or stepper motors in any combination by the insertion of "Axis Daughter Boards" to suit the application. It is housed in a rugged metal 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 with a Multi-tasking TRIO BASIC using a PC, and then may be set to run "standalone" if an external computer is not required for the final system. The Multi-tasking version of Trio BASIC for the MC204 allows up to 14 BASIC programs to be run simultaneously on the controller using pre-emptive multi-tasking.

The Multi-tasking ability of the MC216 allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware.

The MC216 has 8 built in 24v inputs and 8 bi-directional IO channels. These may be used for system interaction or may be defined to be used by the controller for end of travel limits, datuming and feedhold functions if required. Each of the Input/Output channels has a status LED to make it easy to check them at a glance. The MC216 can have up to 512 external Input/Output channels connected using DIN rail mounted CAN 16-I/O modules. These units connect to the built-in CAN channel.

The MC216 has two built in RS-232 ports and has provision for an external duplex RS-485 channel for simple factory communication systems. The TRIO fibre optic network system can be fitted as an option.

The MC216 features a number of enhancements over the MC204 in addition to the ability to control up to 16 axes. Its faster processor and faster memory allow it to run BASIC programs at more than twice the speed of the MC204. It also has a much larger user program memory and a real time clock function.

### 3.3 Motion Coordinator MC2

The *Motion Coordinator* MC2 is a modular servo control positioner with the ability to control up to 12 servo or stepper motors in any combination by the insertion of "Axis Daughter Boards" to suit the application. It is housed in a rugged metal 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 with Multi-tasking TRIO BASIC using a PC, and then may be set to run "standalone" if an external computer is not required for the final system. The version of Trio BASIC for the MC2 allows up to 14 BASIC programs to be run simultaneously on the controller using pre-emptive multi-tasking.

The Multi-tasking ability of the MC2 allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware.

The MC2 has 8 built in 24v inputs and 8 bi-directional IO channels. These may be used for system interaction or may be defined to be used by the controller for end of travel limits, datuming and feedhold functions if required. Each of the Input/Output channels has a status LED to make it easy to check them at a glance. The number of Input/Output channels can be extended up to 96 by connecting 16IO modules to the MC-2.

The MC2 has two built in RS-232 ports and one duplex RS-485 channel for simple factory communication systems. Other options are the TRIO fibre optic network system and 4 12bit analogue inputs.

### 3.4 Axis Positioning Functions

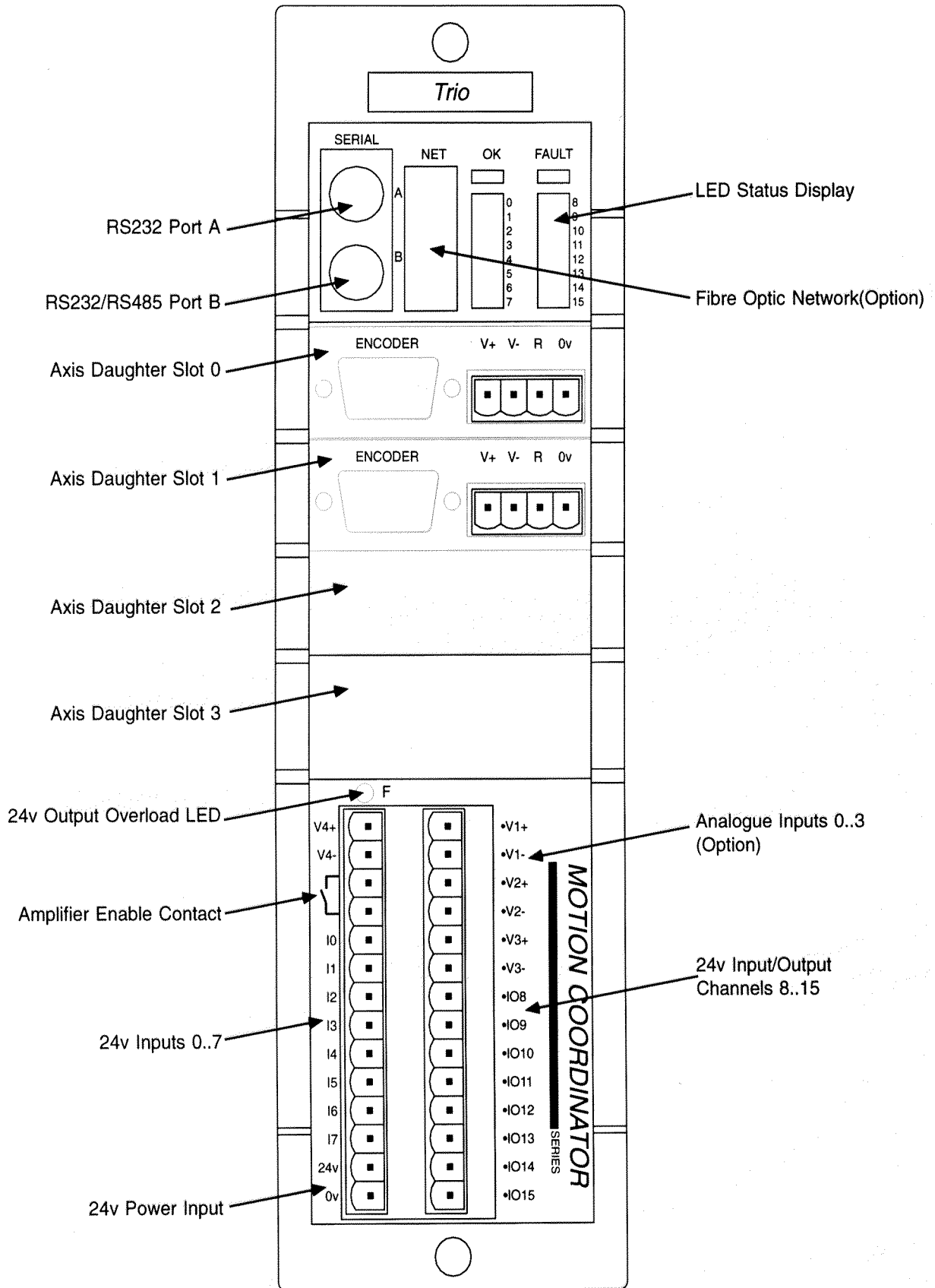
9 types of Axis Daughter Boards are currently available for the *Motion Coordinator* SERIES, these daughter boards may be fitted either into MC204,MC216 or MC2 (With the exception of the CAN daughter board):

Servo Encoder Daughter	-	Interface to servo-amplifier for motor with encoder feedback
Stepper Daughter	-	Interface to stepper motor amplifier
Reference Encoder Daughter	-	Encoder input only for position sensing
Servo Resolver Daughter	-	Interface to servo-amplifier for motor with resolver feedback
Stepper Encoder Daughter	-	Interface to stepper motor with encoder feedback
Analogue Output Daughter	-	+/-10v Output only
SSI Servo Encoder Daughter	-	Interface to servo axis with SSI absolute encoder
Differential Stepper Daughter	-	Interface to stepper motor amplifier with differential outputs
CAN daughter board	-	Additional CAN interface (for MC204/MC216 ONLY)
Hardware PSWITCH	-	Encoder input and 4 digital outputs which switch over sector

Details of the Axis Daughter Boards are given in chapter 4.

The motion control generation software receives instructions to move an axis or axes from the BASIC language which is running concurrently on the same processor. The motion generation software provides control during operation to ensure smooth, co-ordinated movements, velocity profiled as specified by the controlling program. Linear interpolation may be performed in as many axes as the controller provides, and circular or helical interpolation in any two 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.



The MOTION COORDINATOR MC2 in a 2 axis servo system configuration

### 3.5 Summary of Features *Motion Coordinator - MC204*

Size	262 mm x 68 mm x 198 mm (HxWxD)
Weight	0.75 kg
Operating Temp.	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	2 RS232 Channels: 1200, 9600,19200,38400 baud. 1 RS485 Channel (optional) 1 CAN channel built on to motherboard
Position Resolution	32 bit position count
Interpolation modes	Linear 1-4 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking TRIO BASIC system, maximum 6 tasks.
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Servo Cycle	1ms
Memory	122 Kbytes battery backed + flash program memory.
Power Input	500mA at 18..29 V d.c.
Amplifier Enable Output	NO relay contact rated 24Vdc @ 0.5A..
Digital Inputs	8 Opto-isolated 24v inputs.
Digital Input/Outputs	8 Opto-isolated 24v current sourcing 250 mA outputs/inputs

### 3.6 Summary of Features *Motion Coordinator - MC216*

Size	262 mm x 68 mm x 198 mm (HxWxD)
Weight	0.75 kg
Operating Temp.	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	2 RS232 Channels: 1200, 9600,19200,38400 baud. RS485 via externally buffered adapter 1 CAN channel built on to motherboard
Position Resolution	32 bit position count
Interpolation modes	Linear 1-16 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking TRIO BASIC system, maximum 15 tasks.
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Servo Cycle	Programmable - Default is 1ms
Memory	500 Kbytes battery backed + flash program memory.
Power Input	500mA at 18..29 V d.c.
Amplifier Enable Output	NO relay contact rated 24Vdc @ 0.5A..
Digital Inputs	8 Opto-isolated 24v inputs.
Digital Input/Outputs	8 Opto-isolated 24v current sourcing 250 mA outputs/inputs

### 3.7 Summary of Features *Motion Coordinator - MC2*

Size	262 mm x 68 mm x 198 mm (HxWxD)
Weight	0.75 kg
Operating Temp.	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	2 RS232 Channels: 1200, 9600,19200,38400 baud. 1 RS485 Channel (optional)
Position Resolution	32 bit position count
Interpolation modes	Linear 1-12 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking TRIO BASIC system, maximum 15 tasks.
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Servo Cycle	1ms default
Memory	122 Kbytes battery backed + flash program memory.(500k option)
Power Input	500mA at 18..29 V d.c.
Amplifier Enable Output	NO relay contact rated 24Vdc @ 0.5A..
Digital Inputs	8 Opto-isolated 24v inputs.
Digital Input/Outputs	8 Opto-isolated 24v current sourcing 350 mA outputs/inputs
Analog Inputs	An optional feature, incorporates 4 , 12 bit analog inputs.

### 3.8 Connections to *MOTION COORDINATOR SERIES 2* Master Modules

All connections other than the module interconnection bus are located on the front panel. All wires and cables should be of suitable size and type for the signals carried and the operating environment.

Suitable wires would include multicore screened cable for encoder feedback and the serial links. The analog outputs from the Axis Daughter Boards should be connected to the servo amplifier via a screened twisted pair. Amplifier enable outputs and 24v inputs do not have a large current requirement so the choice of wire is not critical. See 2.4.3

#### 3.8.1. System 24v Power Input

The MC204/MC216 /MC2 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 5v supply and other internal power supplies.

#### 3.8.2 Serial Port Connections on MC204/MC2

Trio supply ready made cables for connecting between a PC and the *Motion Coordinator* (product code P350).

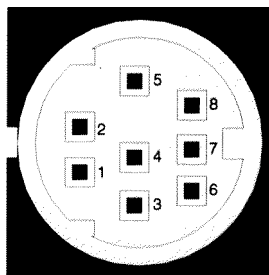
##### Serial Port 0:

1	-	Internal 5v MC216 Only	-	Do not connect
2	-	Internal 0v MC216 Only	-	Do not connect
3	-	RS232 Transmit		
4	-	RS232 Ground		
5	-	RS232 Receive		
6	-	Externally buffered output MC216 Only	-	Do not connect
7	-	Externally buffered input MC216 Only	-	Do not connect
8	-	Externally buffered control MC216 Only	-	Do not connect

##### Serial Port 1:

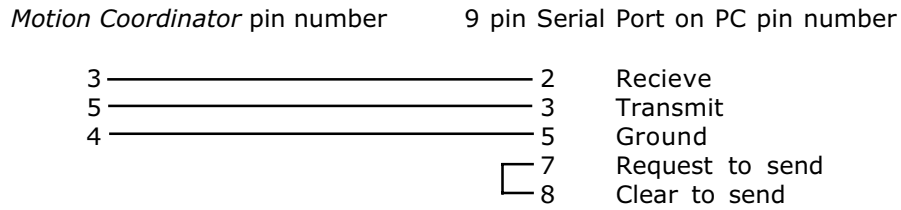
1	-	RS485 Data Input/Internal 5v on MC216 Only
2	-	RS485 Data Input (Inverted)/Internal 0v on MC216 Only
3	-	RS232 Transmit
4	-	RS232 Ground
5	-	RS232 Receive
6	-	RS232 +5v Output
7	-	RS485 Data Output (Inverted)/Externally buffered output 2 MC216 Only
8	-	RS485 Data Output/Externally buffered input 2 MC216 Only

There is no hardware handshake on the serial ports. An XON\XOFF protocol is used.

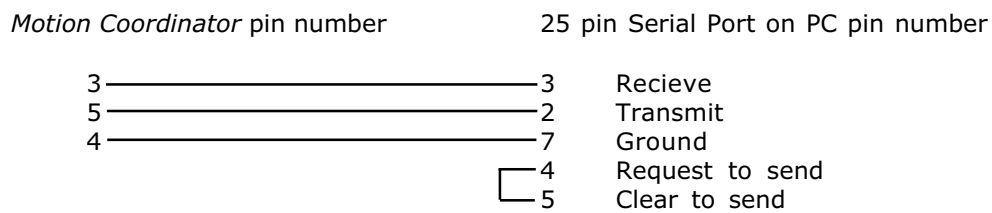


Trio recommend the use of their pre-made serial cables. If cables need to be made to connect to a PC serial port the following connections are required:

**Motion Coordinator to "AT" style PC with 9 pin serial port:**



**Motion Coordinator to "XT" style PC with 25 pin serial port:**



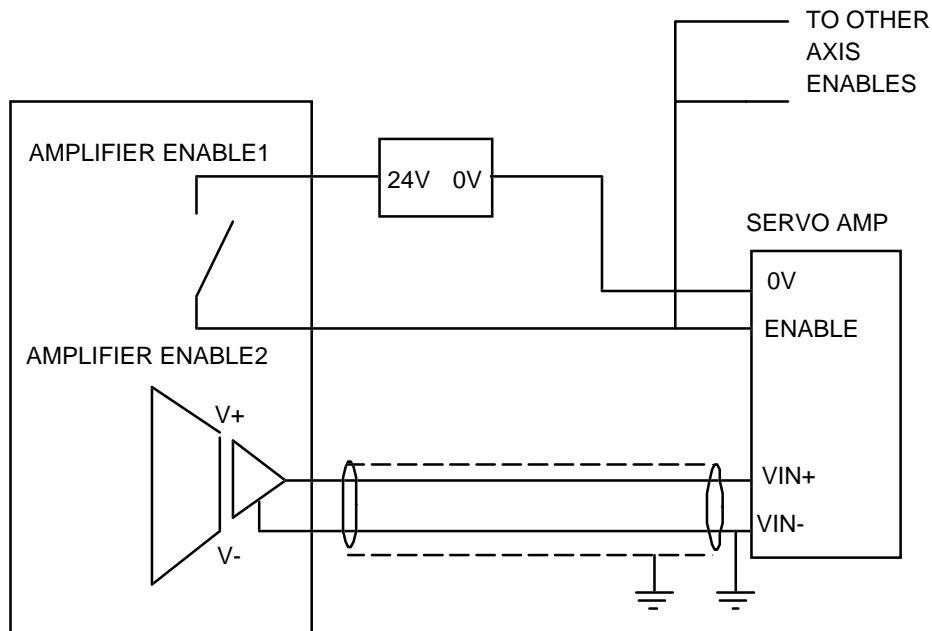


### 3.8.3 Amplifier Enable (Watchdog) Relay Output

An internal relay contact is used to enable external amplifiers when the controller has powered up correctly and the system and application software is ready. The amplifier enable is a single pole relay with a set of normally open contacts. The enable relay contact will be open circuit if there is no power on the controller OR a following 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 holdup 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**



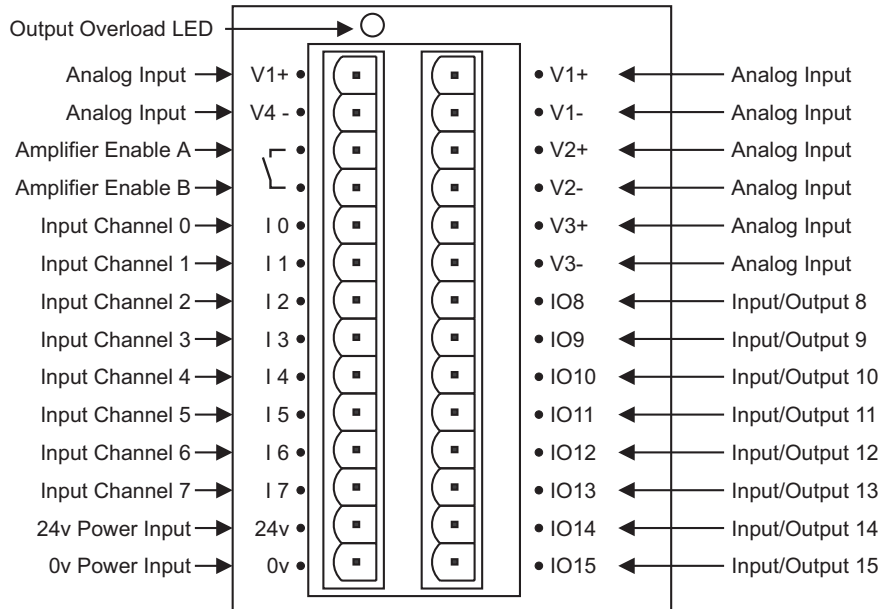
The ENABLE circuit is similar on a STEPPER Amplifier

Fig 3.4 Connection of Watchdog and Velocity Output to Servo Amp

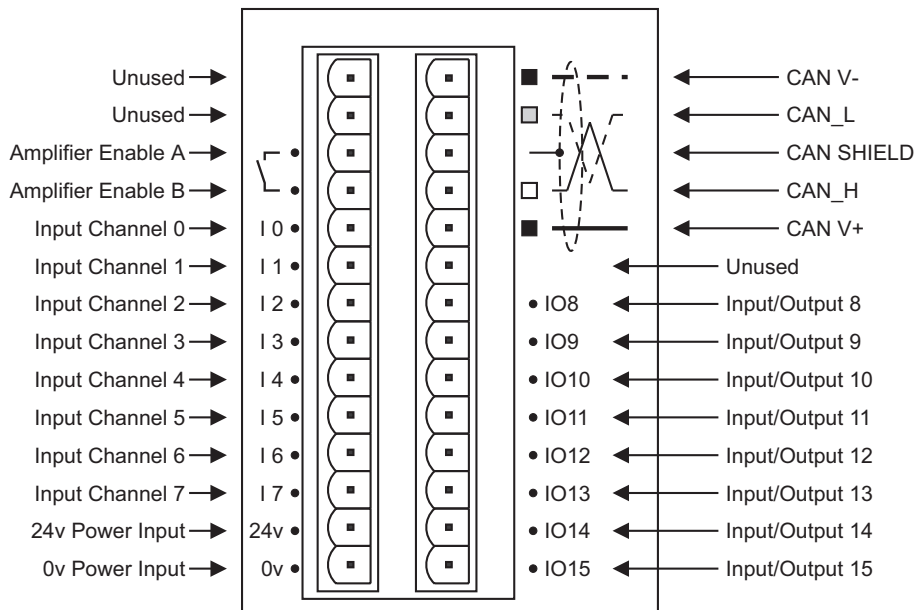
### 3.8.4 Enabling Stepper Motor Amplifiers

The watchdog relay contact can be used to enable both stepper and servo motor amplifiers. The stepper daughter boards provide an additional open-collector enable output which can be used to enable stepper motor amplifiers.

### 3.8.5 MC2 Lower Front Panel Connections and Output Overload LED

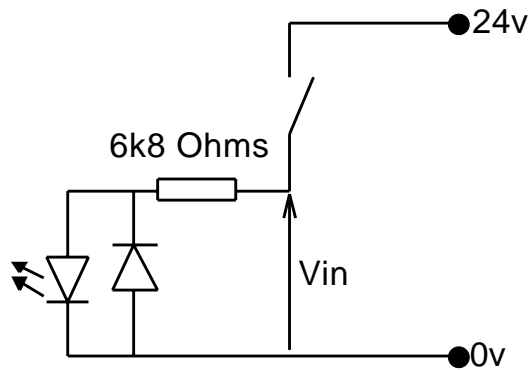


### 3.8.6 MC204/MC216 Lower Front Panel Connections



### 3.8.7 24v Input Channels

The *MOTION COORDINATOR* has 16 24v Input channels built into the master unit. These may be expanded to 528 Inputs by the addition of CAN-16I/O16 modules. (96 on MC2 using 16IO 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 Input/Output channel is to be used as an Input the output should not be switched on in the program.

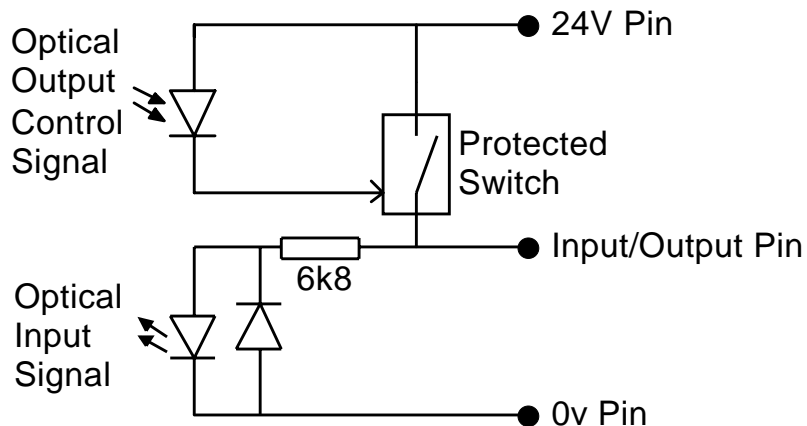


### 3.8.8 24v Input/Output Channels

Input/output channels 8..15 are bi-directional. The inputs have a protected 24v sourcing output connected to the same pin. If the output is unused it may be used as an input 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.

The MC2 output channel current limit is very sensitive and overloads as short as 10uSec can trip the over-current limit. This can create problems when driving incandescent blubs which have a very low resistance when cold. Typically this only allows 50mA blubs can be driven directly.

The MC204/MC216 feature a less sensitive current limit. Care should still 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.



### 3.8.9 Network Interconnection

The Fibre Optic Network Connection on the MC216,MC2 and MC204 is designed for communication between the master modules and membrane keypads on large machines, or small workgroups of machines. It uses the Hewlett-Packard "Versatile Link" format which offers moderate performance at low cost.

- 30m Range
- 38400 Baud transmission speed

The software for the network supports interconnection of up to 15 nodes in a token-ring network format. The nodes may consist of MC204,MC2 or MC216 masters, MC1 masters, Membrane Keypads, or CSC-DAC-MX controllers.

Any membrane keypads connected must have software version 2.01 or higher.

### 3.8.10 Analogue Inputs

The *Motion Coordinator* MC2 unit **only** may optionally be fitted with analogue inputs. The analog input channels can be programmed to operate as:

- 4 differential 12 bit channels or 7 single ended channels + common.
- 0..4.096 volt input or +/- 2.048 volt input.

The mode of operation is set by the AIN() BASIC command.

<u>Front Panel Label:</u>	<u>AIN channel Differential Mode:</u>	<u>Single Ended Mode:</u>
V1+	Channel 0 +	Channel 0
V1-	Channel 0 -	Channel 1
V2+	Channel 1+	Channel 2
V2-	Channel 1-	Channel 3
V3+	Channel 2+	Channel 4
V3-	Channel 2-	Channel 5
V4+	Channel 3+	Channel 6
V4-	Channel 3-	0v reference for channels 0..6

See the description of the AIN command for details of how to read these channels.

### 3.8.11 Using End of Travel Limit Sensors

Each axis of the Motion Coordinator system may have a 24v Input channel allocated to it for the functions:

- FORWARD Limit - Forward end of travel limit
- REVERSE Limit - Reverse end of travel limit
- DATUM Input - Used in datuming sequence
- FEEDHOLD Input - Used to suspend velocity profiled movements until the input is released

Switches used for the FORWARD/REVERSE/DATUM/FEEDHOLD inputs must be NORMALLY CLOSED type.

Each of the functions is optional and may be left unused if not required. Each of the 4 functions are available for each axis and can be assigned to any input channel in the range 0..31. An input can be assigned to more than one function if desired.

The axis parameters: FWD\_IN,REV\_IN, DATUM\_IN and FH\_IN are used to assign input channels to the functions. The axis parameters are set to -1 if the function is not required.

## 4.0 Axis Daughter Boards

### 4.1 Description

The axis daughter boards give the *Motion Coordinator* system enormous flexibility in its configuration. 10 types of daughter board are currently available. Each of the 4 potential slots in a MC204 master, 12 potential slots in a MC 2/AXIS EXPANDER combination, or 16 potential slots in an MC216/AXIS EXPANDER combination may be fitted with any type of daughter board in any combination, with the exception of the CAN daughter board. This daughter board is only compatible with the MC204/MC216.

The 10 types of axis daughter boards are:

	<b>Product Code:</b>
• Stepper	P230
• Stepper Encoder	P240
• Hardware Pswitch	P242
• Servo Encoder	P200
• Reference Encoder	P220
• Servo Resolver	P210
• Analogue Output	P260
• SSI Absolute Servo	P270
• Differential Stepper	P280
• CAN	P290

Trio can produce custom daughter boards for specific customer applications where required.

	Servo Encoder P200	Servo Resolver P210	Reference Encoder P220	Stepper P230	Stepper Encoder P240	Analogue Output P260	SSI Absolute P270	Differential Stepper P280	CAN P290
<b>Output Features</b>									
+/- 10v Differential Output	✓	✓	▪	▪	▪	✓	✓	▪	▪
Open Collector Stepper	▪	▪	▪	✓	✓	▪	▪	▪	▪
Differential Stepper	▪	▪	▪	▪	▪	▪	▪	✓	▪
CAN Bus Comms	▪	▪	▪	▪	▪	▪	▪	▪	✓
<b>Input Features</b>									
Differential Encoder	✓	▪	✓	▪	✓	▪	▪	▪	▪
SSI Absolute Encoder	▪	▪	▪	▪	▪	▪	✓	▪	▪
Resolver	▪	✓	▪	▪	▪	▪	▪	▪	▪
Stepper Pulse Feedback	▪	▪	▪	▪	✓	▪	▪	✓	▪
24v Registration Input	✓	✓	✓	▪	▪	▪	✓	▪	▪
5v Registration Input	✓	✓	✓	▪	✓	▪	✓	✓	▪
CAN Bus Comms	▪	▪	▪	▪	▪	▪	▪	▪	✓

#### 4.1.1 Fitting and Replacing Axis Daughter Boards

The axis daughter boards are normally supplied fitted into a Motion Coordinator System. If daughter boards need to be moved or replaced the following sequence must be followed. The axis daughter boards are supplied in an anti-static bag or box. This should be used to hold the axis daughter board at all times when not installed in the Motion Coordinator.

- 1 • Check there is no power on the module
- 2 • Disconnect any modules attached on the ribbon cable bus (See Chapter 2)
- 3 • Unscrew the 2 allen screws which secure the front moulding cover and remove the cover
- 4 • Unscrew the top ribbon cable bus cover
- 5 • Unscrew the 8 screws which secure the right hand side cover of the module
- 6 • Unscrew the single screw which secures the axis daughter board to the left hand side of the module. This is located by the left hand end of the axis daughter front panel

The sequence is reversed after inserting any new modules.

#### 4.1.2 Axis Sequence in the Motion Coordinator MC216/MC204/MC2/Axis Expander Module

The axis number of any axis daughter board is fixed by its position in the MC216/MC204/MC2 module.

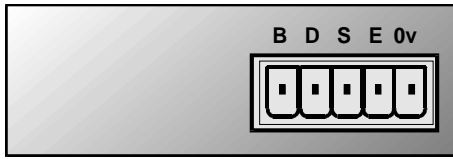
- Top Location • Axis 0
- Next Location • Axis 1
- Next Location • Axis 2
- Bottom Location • Axis 3

The axis number of any axis daughter board in the Axis Expander module is fixed by its position and the setting of the front panel selector switch on the Axis Expander.

	<b>Selector Switch-4..7</b>	<b>Selector Switch-8..11</b>	<b>Selector Switch 12-15(MC216 Only)</b>
Top Location	• Axis 4	Axis 8	Axis 12
Next Location	• Axis 5	Axis 9	Axis 13
Next Location	• Axis 6	Axis 10	Axis 14
Bottom Location	• Axis 7	Axis 11	Axis 15

## 4.2 Stepper Daughter Board

ATYPE parameter for Stepper Daughter Board = 1



### 4.2.1 Description

The stepper daughter board generates pulses to drive an external stepper motor amplifier. Single step, half step and micro-stepping drives can be used with the board. All four output signals are opto-isolated on the stepper daughter board and each features overcurrent protection.

In Single/Half step mode the stepper daughter board can generate pulses at up to 62 KHz.  
In microstep mode the stepper daughter can generate pulses at up to 500 KHz.

The Motion Coordinator recognises the type of axis daughter board fitted to each axis. An axis with a stepper daughter board fitted will return its ATYPE axis parameter as 1:

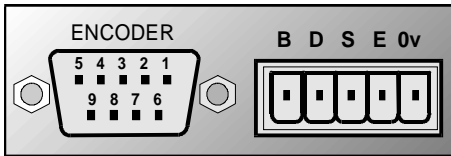
```
>>Print atype
1.0000
>>
```

### 4.2.2 Connections to Stepper Daughter Board:

- B Boost
- D Direction
- S Step
- E Enable
- 0v 0v reference for all connections

Each of the connections B,D,S,and E is an open-collector output. The outputs can be pulled up to 24v externally and can drive up to 100 mA. Overcurrent protection is provided on the outputs.

### 4.3 Stepper Daughter Board with Position Verification



#### ATYPE parameter for Position Verification Stepper Daughter Board = 4

The stepper daughter board with position verification has all the features of the simpler stepper daughter board. Position verification is added to a stepper axis by providing encoder feedback to check the position of the motor. An encoder port on the daughter board allows the position of the motor to be checked on every servo cycle. If the difference between the demanded number of steps and the measured position exceeds the programmed limit, the Motion Coordinator can take undertake a programmed error sequence. See section 4.2.2

#### Encoder Connections:

The encoder connections for a position verification stepper daughter board are identical to those on the encoder port of a servo daughter board shown in section 4.4

#### Choosing an encoder for position verification:

Stepper daughter boards generate fewer pulses than the number of steps the controller considers the axis to be moving. The Stepper daughter board divides the number of steps the controller moves by:

- a factor of 16 in default mode (MICROSTEP=OFF)
- a factor of 2 if MICROSTEP=ON

The encoder for position verification should have a number of edges equal to the number of steps the controller moves by, or an integer factor higher. If a factor higher is used the PP\_STEP axis parameter should be set.

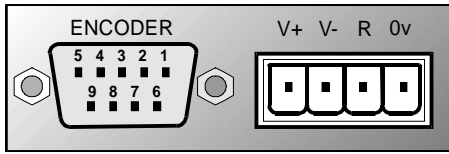
Example:

A stepper motor has 200 pulses/step and is driven with MICROSTEP=OFF. The controller therefore moves a distance of  $200 \times 16 = 3200$  to rotate the motor 1 turn. If an encoder is to be used for position verification the encoder would need 3200, 6400, 9600 or 12800 etc edges/turn. The encoder port will count 4 edges per encoder pulse. The ideal encoder would therefore have 800 pulses/rev.



## 4.4 Servo Daughter Board

ATYPE parameter for Servo Daughter Board = 2



### 4.4.1 Description

The servo daughter board provides the interface to a DC or Brushless servo motor fitted with an encoder or encoder emulation. The encoder port provides high speed differential receiver inputs. These inputs are opto-isolated to maximise the noise immunity of the system. The encoder port also provides an isolated 5v output capable of powering most encoders, simplifying wiring and eliminating external supplies.

The servo daughter board provides a hardware position capture function for both the Z input and a dedicated 24volt registration input. Transitions of either polarity on both these inputs can be used to record the position of the axis at the time of the event within less than 1 micro seconds. This practically eliminates time delays and avoids interrupting the processor frequently in multi-axis systems.

The servo daughter board provides a 12 bit +/-10v voltage output to drive most servo-amplifiers. The voltage output is opto-isolated as standard to maximise the noise immunity of the system.

### 4.4.2 Connections

#### Encoder Connections:

The encoder is connected via a 9 pin 'D' type socket mounted on the front panel. The plug supplied should be cable mounted and wired as shown below. The encoder port is designed for use with differential output 5 volt encoders.

9 pin 'D' type plug	function
1	channel A true
2	channel A complement
3	channel B true
4	channel B complement
5	0V
6	marker (Z) true
7	marker (Z) complement
8	+5V (150 mA max) - This output is short circuit protected
9	Register Input 5v Input pin
shell	protective ground

The encoder may be powered from the +5V supply output on the daughter board, provided it requires less than 150mA supply current. If the encoder is situated so far from the module that the supply is inadequate an external supply should be used and regulated locally to the encoder. In this case the +5V connection from pin 8 should not be used and the external supply 0v should be connected to pin 5 (0V).

If the encoder does not have complementary outputs, pins 2, 4 and 7 should be connected to a +2.5V bias voltage. This may be simply derived from a pair of 220 Ohm resistors in series with one end of the pair connected to 0v and the other end to +5v. The centre point of the pair will form approximately 2.5v.

If the encoder does not have a marker pulse, pins 6 and 7 may be left unconnected.

## Voltage Output

The +/-10v output voltage to drive external servo amplifiers is generated between the V+ and V- pins. The voltage output is isolated and floating but if multiple servo daughter boards are fitted it should be noted that the boards share a common power supply for generation of the +/-10v. This means that the V- pins should not be referenced to different external voltages.

## Registration Input

The registration input is a 24v dc input connected through high-speed opto-isolation into the encoder counter circuit. An alternative 5v input pin is available on the encoder port. The internal circuitry can be used to capture the position at which the registration input makes a transition from low to high or vice-versa. This function is accessed in software from the REGIST command. The input is measured relative to the 0v input on the servo daughter board which must be connected if the registration input is used. Note that this is the same 0v as the encoder port.

## Software Considerations

The servo daughter board returns axis parameter ATYPE of 2. The servo function can be switched on or off with the SERVO axis parameter:

```
>>SERVO =ON
```

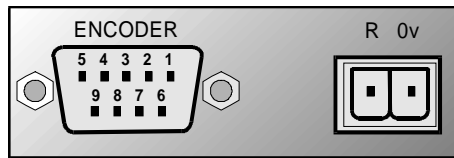
When the servo function is OFF the value in the axis parameter DAC is written to the 12 bit digital to analogue converter:

```
>>DAC=2047    -    This will set -10volts on voltage output when servo is OFF  
>>DAC=0       -    This will set 0volts on voltage output when servo is OFF  
>>DAC=-2048   -    This will set 10volts on voltage output when servo is OFF
```

**Note:**        **The daughter board hardware inverts the voltage relative to the DAC parameter.**

## 4.5 Encoder Daughter Board

**ATYPE parameter for Encoder Daughter Board = 3**



### 4.5.1 Description

The encoder daughter board provides an encoder input without a servo feedback facility for measurement, registration and synchronization functions on conveyors, drums, flying shears, etc. The encoder port provides high speed differential receiver inputs. These inputs are opto-isolated to maximise the noise immunity of the system. The encoder port also provides an isolated 5v output capable of powering most encoders. This simplifies wiring and eliminates external power supplies.

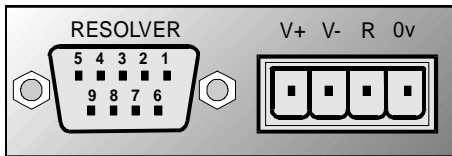
The encoder daughter board provides a hardware position capture function for both the encoder Z input and the dedicated 24volt registration input. Transitions of either polarity on both these inputs can be used to record the position of the axis at the time of the event within less than 1 micro second. This practically eliminates time delays and avoids interrupting the processor frequently in multi-axis systems.

### 4.5.2 Connections

The encoder and registration input connections are identical to those of the servo daughter board. See section 4.4

## 4.6 Servo Resolver Daughter Board

**ATYPE parameter for Servo Resolver Daughter Board = 5**



### 4.6.1 Description

The servo daughter board provides the interface to a DC or Brushless servo motor fitted with a resolver. The resolver port provides absolute position feedback within one motor turn. The resolver inputs are opto-isolated to maximise the noise immunity of the system. The resolver port also provides an oscillator output capable of driving many resolvers.

The resolver daughter board provides a hardware position capture function for both the zero marker input and a dedicated 24volt registration input. Transitions of either polarity on both these inputs can be used to record the position of the axis at the time of the event within less than 1 micro seconds. This practically eliminates time delays and avoids interrupting the processor frequently in multi-axis systems.

The resolver daughter board provides a 12 bit +/-10v voltage output to drive most servo-amplifiers. The voltage output is opto-isolated as standard to maximise the noise immunity of the system.

### 4.6.2 Resolver Connections

#### Voltage Output

The +/-10v output voltage to drive external servo amplifiers is generated between the V+ and V- pins. The voltage output is isolated and floating but if multiple servo daughter boards are fitted it should be noted that the boards share a common power supply for generation of the +/-10v. This means that the V- pins should not be referenced to different external voltages.

#### Registration Input

The registration input is a 24v dc input connected through high-speed opto-isolation into the resolver circuit. An alternative 5v input pin is available on the encoder port. The internal circuitry can be used to capture the position at which the registration input makes a transition from low to high or vice-versa. This function is accessed in software from the REGIST command. The input is measured relative to the 0v input on the servo daughter board which must be connected if the registration input is used. Note that this is the same 0v as the resolver port.

#### Resolver Port Pinout

1	-	SIN +	5	-	0v
2	-	SIN -	6	-	REF +
3	-	COS +	7	-	REF -
4	-	COS -	8	-	Analog Velocity Output
			9	-	Register 5v Input

The reference oscillator runs at 2kHz and produces a +/- 2volt output.

## Software Considerations

The resolver daughter board returns axis parameter ATYPE of 5. The servo function can be switched on or off with the SERVO axis parameter:

```
>>SERVO =ON
```

When the servo function is OFF the value in the axis parameter DAC is written to the 12 bit digital to analogue converter. Therefore:

```
>>DAC=2047    -    This will set -10volts on voltage output when servo is OFF
>>DAC=0       -    This will set 0volts on voltage output when servo is OFF
>>DAC=-2048   -    This will set +10volts on voltage output when servo is OFF
```

**Note:**      **The daughter board hardware inverts the voltage relative to the DAC parameter.**

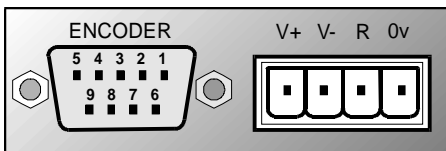
The measured position of the axis MPOS is not reset to zero on power up but reset to a value in the range 0..4095. All other software functions are similar to the servo daughter board. The resolver converter circuit uses a fixed 12 bit conversion. The motor resolution will therefore be 4096 "edges"/turn.

## 4.7 Voltage Output Daughter Board

**ATYPE parameter for Voltage Output Daughter Board = 6**

### 4.7.1 Description

The voltage output daughter board provides a 12 bit +/-10v voltage output for driving inverters and other devices. The board is a simplified servo daughter board and the connections are similar. The encoder port does not function even if the connector is fitted.



### Voltage Output

The +/-10v output voltage to drive external servo amplifiers is generated between the V+ and V- pins. The voltage output is isolated and floating but if multiple daughter boards are fitted it should be noted that the boards share a common power supply for generation of the +/-10v. This means that the V- pins should not be referenced to different external voltages.

### Software Considerations

The voltage output daughter board returns axis parameter ATYPE of 6.

The board functions like a servo daughter board with the SERVO= OFF the value in the axis parameter DAC is written to the 12 bit digital to analogue converter. Therefore:

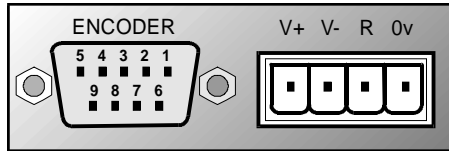
```
>>DAC=2047    -    This will set -10volts on voltage output
>>DAC=0       -    This will set 0volts on voltage output
>>DAC=-2048   -    This will set +10volts on voltage output
```

**Note:** The daughter board hardware inverts the voltage relative to the DAC parameter.

## 4.8 Absolute Servo (SSI) Daughter Board

**ATYPE parameter for Absolute Servo (SSI) Daughter Board = 7**

### 4.8.1 Description



The SSI daughter board provides the interface to a DC or Brushless servo motor fitted with an absolute encoder or encoder emulation. The encoder port provides high speed differential synchronous serial interface (SSI). This port is opto-isolated to maximise the noise immunity of the system.

The SSI port is programable so that any number of bits from 1 to 24 can be clocked into the position registers.

If the registration function is to be used then a minimum of 12 bits is required for correct operation.

The SSI daughter board provides a hardware position capture function for both a 5V differential input and a dedicated 24volt registration input. Transitions of either polarity on both these inputs can be used to record the position of the axis at the time of the event within less than 1 micro seconds. This practically eliminates time delays and avoids interrupting the processor frequently in multi-axis systems. It should be noted however that the SSI system overall does not support this level of accuracy due to delays in clocking data from the encoders.

The servo daughter board provides a 12 bit +/-10v voltage output to drive most servo-amplifiers. The voltage output is opto-isolated as standard to maximise the noise immunity of the system.

### 4.8.2 SSI Connections:

The encoder is connected via a 9 pin 'D' type socket mounted on the front panel. The socket supplied should be cable mounted and wired as shown below. The SSI encoder port is designed for use with differential output 5 volt encoders.

9 pin 'D' type plug	function
1	data true
2	data complement
3	clock true
4	clock complement
5	0V- must be connected even if an external PSU is used to power the encoder.
6	marker (Z) true - can be used as a second general purpose regist input.
7	marker (Z) complement - can be used as a second general purpose regist input.
8	+5V (150 mA max) - This output is short circuit protected
9	Register Input 5v Input pin
shell	protective ground

The encoder may be powered from the +5V supply output on the daughter board, provided it requires less than 150mA supply current. If the encoder is situated so far from the module that the supply is inadequate an external supply should be used and regulated locally to the encoder. In this case the +5V connection from pin 8 should not be used and the external supply 0v should be connected to pin 5 (0V).

## Voltage Output

The +/-10v output voltage to drive external servo amplifiers is generated between the V+ and V- pins. The voltage output is isolated and floating but if multiple servo daughter boards are fitted it should be noted that the boards share a common power supply for generation of the +/-10v. This means that the V- pins should not be referenced to different external voltages.

## Registration Input

The registration input is a 24v dc input connected through high-speed opto-isolation into the encoder counter circuit. An alternative 5v input pin is available on the encoder port. The internal circuitry can be used to capture the position at which the registration input makes a transition from low to high or vice-versa. This function is accessed in software from the REGIST command. The input is measured relative to the 0v input on the servo daughter board which must be connected if the registration input is used. Note that this is the same 0v as the encoder port. Note: the Z pulse input normally used with incremental encoders is still available for use as a general purpose differential 5V regist input.

## Software Considerations

The number of bits to be shifted in from the encoder attached to the SSI port is set using the SSI\_BITS axis parameter. This parameter defaults to 0 on power up and this disables the reading of the encoder position into the MPOS parameter. As soon as SSI\_BITS is set to a value in the range 1..24 the MPOS parameter will be set with the current position of the encoder and will then follow the position of the encoder until the controller is powered down or SSI\_BITS is set to 0. For example if a 12 bit multiturn encoder with 12 bits per turn is being used the command:

```
SSI_BITS=24
```

should be put at the start of the program or typed on the command line.

The SSI daughter board returns axis parameter ATYPE of 7. The servo function can be switched on or off with the SERVO axis parameter:

```
>>SERVO =ON
```

When the servo function is OFF the value in the axis parameter DAC is written to the 12 bit digital to analogue converter. Therefore:

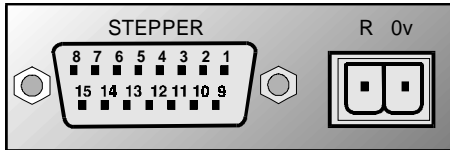
```
>>DAC=2047    -    This will set -10volts on voltage output when servo is OFF  
>>DAC=0       -    This will set 0volts on voltage output when servo is OFF  
>>DAC=-2048   -    This will set +10volts on voltage output when servo is OFF
```

**Note:**        **The daughter board hardware inverts the voltage relative to the DAC parameter.**



## 4.9 Differential Stepper Daughter Board

**ATYPE parameter for Differential Stepper Daughter Board = 4**



The differential stepper daughter board is a stepper daughter board with the output signals provided as differential 5 volt signals on a 15 way 'D' connector. The daughter board does not feature an encoder port for position verification, but does have a registration input to allow for capture of the number of step pulses when a registration signals arrives.

### Connections:

The differential stepper daughter board is fitted with a 15 way female 'D' connector:

Pin	Use
1	Step+
2	Direction+
3	Boost+
4	no connection
5	Fault input
6	no connection
7	no connection
8	Screen
9	Step-
10	Direction-
11	Enable+
12	Enable-
13	0v
14	0v
15	Boost-

All outputs are RS422 differential lines without terminating resistors. A 26LS31 line driver is employed. The fault input and registration inputs are single ended opto-isolated inputs that sink approximately 8mA.

### WARNING:

**The Fault input and Registration input are designed for 5volt operation. Applying 24 volts will damage the input circuit.**

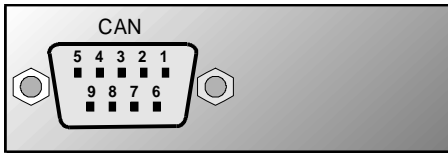
For 24 volt operation a 4k7 0.5Watt resistor must be added in series with the input.

A 2 pin terminal terminal connector is provided for the registration input.

### Software Considerations

The differential daughter board uses the same axis gate array as a stepper encoder daughter board as is therefore recognised by the MC216/MC204/MC2 as this type (ATYPE=4). It does not however have an encoder inputs fitted so may only be used in the VERIFY=OFF mode where the stepper pulses are fed back into the encoder counter circuit.

#### 4.10 CAN Daughter Board



#### ATYPE parameter for CAN Daughter Board = 8

The CAN daughter board is the first of a new *extended* type of daughter board. These extended daughter boards feature a higher data capability through the provision of extra pins on the daughter board to motherboard connector. At present only the MC204/216 controller supports the extended daughter board format. The MC204 and 216 feature CAN built onto the motherboard. The combination of the motherboard CAN channel and the ability to add up to 4 CAN daughter boards allows the MC204 to potentially control up to 5 CAN channels, and 17 on the MC216.

#### Multiple CAN Channels:

The CAN daughter board is particularly aimed at providing dedicated CAN channels for controlling servo amplifiers. It features opto-isolation and the ability to support communications at 1M Baud. The built-in CAN channel is designed for Input/Output expansion and to allow communication between controllers and between Motion Coordinators and PLC's

#### CAN Connections:

The CAN connection of the CAN daughter board is via a 9 way Male 'D' type connector. Pinout is compatible with that suggested by the CAN in AUTOMATION organisation:

Pin	Use
1	no connection
2	CAN_L
3	CAN_GND
4	no connection
5	Shield
6	no connection
7	CAN_H
8	no connection
9	no connection

No terminating resistor is fitted internally in the CAN daughter board.

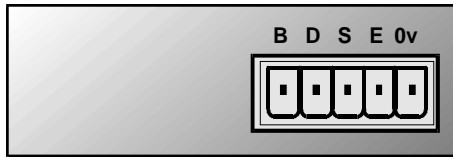
#### Software Considerations

The CAN daughter board is recognised with ATYPE of 8:

```
>>? ATYPE AXIS(0)
8.0000
>>
```

The daughter board CAN channel can be controlled using the Trio BASIC CAN command. Trio are also building into the system software dedicated communication software to support particular communication protocols. At present the only one available is for "INFRANOR" amplifiers.

#### 4.11 Hardware PSWITCH Daughter Board



#### ATYPE parameter for Hardware PSWITCH Daughter Board = 10

The hardware PSWITCH daughter board allows 4 open-collector outputs to be switched ON and OFF at programmed positions. This function is similar to the software PSWITCH command which is implemented in the system software and allows outputs to be switched ON and OFF over position sectors. The Hardware PSWITCH daughter board performs the position comparison in electronics hardware on the daughter board. This allows the pulses generated to be very accurately timed.

#### Encoder Connections:

The encoder connections for a hardware PSWITCH daughter board are identical to those on the encoder port of a servo daughter board shown in section 4.4. The daughter board encoder input does NOT have registration facilities. The encoder edge rate for hardware PSWITCH functioning is limited to 500kHz.

#### Output Connections:

The hardware PSWITCH daughter board is built as a Stepper Encoder daughter board with an alternative gate array fitted. The 5 way miniature disconnect connector is used to bring out the open-collector outputs:

B Output Channel 0  
 D Output Channel 1  
 S Output Channel 2  
 E Output Channel 3  
 0v Common 0v for outputs

#### Software Considerations:

The Hardware PSWITCH daughter board requires controller system software of 1.33 or higher. This software version supports the Hardware PSWITCH daughter board, it also extends the maximum number of PSWITCHes allowed from 8 to 16.

Each of the 4 channels is programmed individually with a PSWITCH command. This is similar to the software PSWITCH command:

**PSWITCH(*sw, en, axis, opno, opst, setpos, rpos*)**

sw: The switch number in the range 0..15  
 en: Switch enable - 3 to enable as Hardware PSWITCH, 1 or ON to enable as software PSWITCH, 0 or OFF to disable. Each switch may individually be chosen as hardware or software in any combination.  
 axis: Axis number of Hardware PSWITCH daughter board  
 opno: Select output 0..3 on Hardware PSWITCH daughter board  
 opst: Selects the state to set the output to, if 1 then output set ON else set it OFF  
 setpos: The position at which output is set, in user units  
 rpos: The position at which output is reset, in user units



## 5.0 Input/Output Modules and Operator Interfaces

### 5.1 General Description

Trio can supply a range of Input/Output Modules and Operator Interface Units. The MC2 controller features Input/Output modules connected to the unit via its ribbon cable expansion bus at the top of the module. The MC204 and MC216 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. The operator interface units all communicate with MC2/MC204 controllers using the Trio fibre-optic network system. Alternatively third party operator interface units may be connected via a serial port. A third option is for machine manufacturers to build a dedicated operator interface for their application. Dedicated operator interfaces can be easily connected to the Trio fibre-optic network by building in a flexible interface board: FO-VFKB.

Product Range:	Product Code:
* 16IO 24volt Input/Output Module	P310
* CAN 16-I/O Module	P315
* CAN Analog Input Module	P325
* Membrane Keypad	P503
* Mini-Membrane Keypad	P502
* Trio Touchpro PC	P590
* Application Specific Keypad using FO-VFKB	P504

The Trio Touchpro PC is a graphical touchscreen operator interface built around a standard PC with 6.5" LCD active matrix LCD screen. It is not covered by this manual.

### 5.2 16IO 24v Digital Input/Output Module

Up to five 16IO modules (P310) may be connected to an MC2 system to provide for up to 96 IO channels (Including the input/output channels on the master module). Convenient front panel screw plug terminals permit connection to the 16 opto-isolated input/output channels. Each of the channels is bi-directional and can be used EITHER as an input OR as an output.

To use any of the channels of the 16 IO as an input simply do not switch the output channel of the same number ON in the program. The input channels permissible voltage range is 12Vdc to 24Vdc. The opto-isolator provides isolation to 3500 Volts. A diode across each input protects against reverse polarity. Typically a 24 Vdc supply will be used and in this case the current consumption of a single input will be about 4 mA. The Input/Output channels are divided into two sections of eight inputs each. The inputs are debounced and individual front panel LED status indicators mimic the state of each channel.

Each of the 16 IO channels may also be used in any combination as an output. If an input channel is set ON as an output it will not be damaged but will always return ON. The output channels are arranged in banks of 8. Each of the output channels can source 350mA and has electronic overcurrent protection which operates at 400mA. The bank of 8 is also limited in the total current that it can provide is approximately 800mA. An overload fault status LED is illuminated if any of the 8 outputs in the bank has shut down. **The output channel current limit is very sensitive and overloads as short as 10uSec can trip the over-current limit. This can create problems when driving incandescent blubs which have a very low resistance when cold. Typically this only allows 50mA bulbs to be directly driven.**

A rotary switch is used to select which of 5 ranges are provided by the module. Up to five modules may reside in the system. The first 16 input/output channels are provided by the Motion Coordinator SERIES 2 master module. **The 16IO modules (P310) cannot be connected to the MC204 or MC216 Motion Coordinators.**

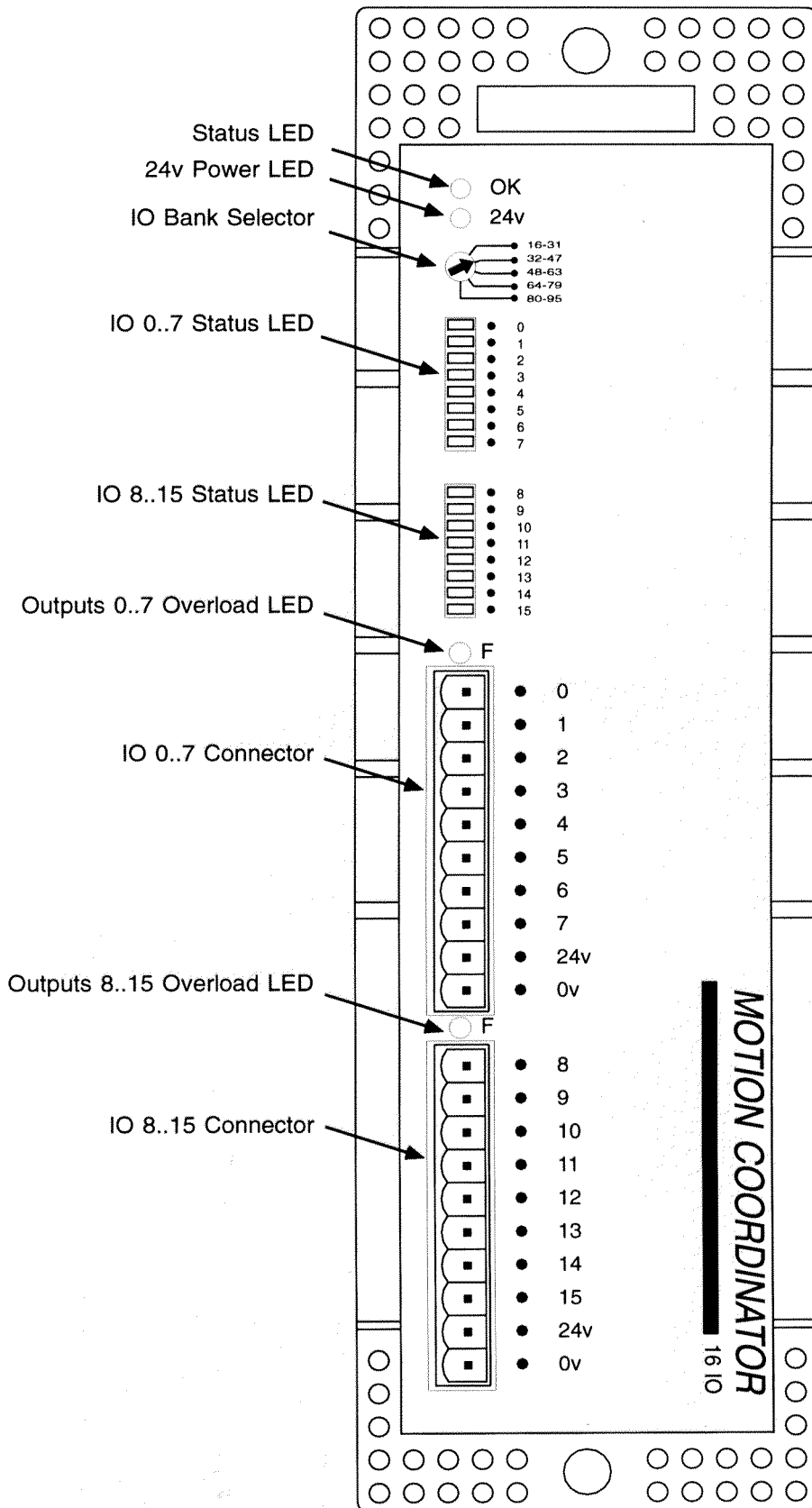


Fig 5.1 16IO Front Panel

### 5.2.1 Connection of 16 IO Modules to System

All connections other than the ribbon cable bus are made via screw terminal plugs mounted on the module front panel. The 16IO features two ten way connectors each of which connects eight input/outputs and the common zero volts and 24v supply connection. The connectors are polarised so that they may only be inserted in one orientation.

The inputs should always be at a positive potential with respect to the common 0v. The 0v and 24v terminals are connected internally. The power supply to drive the 24v circuits must be provided externally.

To connect the ribbon cable bus the metal cover must first be removed from the top of the MC2 master module and the 16 IO module using a screwdriver. Using the ready made cable supplied by TRIO with your order, push the connector into the receptacles on top of the unit. The two locking levers should be pushed outwards before insertion of the cable mounted socket. The socket may only be inserted one way around. To lock the socket in place push the levers together. The module bus cover may then be screwed back on leaving the cable emerging from the slot at the side of the module.

### 5.2.2 Front Panel Indicators

The 16 IO features 20 status indicators on the front panel. The topmost indicator is illuminated by the MC2 Master module after a successful initialisation. On start-up the Master determines how many modules are connected to the system. The module may be interrogated by a host computer or TRIO BASIC program to check how many 16 IO units were found. If this does not equal the number of units actually on the system the host may choose to report this fact. If the 16 IO module did not respond when the MC2 attempted to initialise it, the topmost indicator on the module will not be lit.

The 24v Power LED indicates that 24v has been wired to the module and is present.

The overload fault indicators illuminate if any of the 8 output channels in a bank go overcurrent or the driver circuit goes overtemperature. The 24v supply must be removed to clear the indicator. The remaining 16 indicators each mimic the current state of an input. The indicators report the state that will be read by the Master module ie: after the input signal has been debounced and conditioned. If the channel is used as an output the input channel will mimic the output. The indicators therefore indicate the status of the channel in both cases.

### 5.2.3 Reading an Input / Setting an Output

A host computer or TRIO BASIC program may read the inputs by commanding the Master to return the input state using the IN command. Output channels can be set ON or OFF using the commands OP or PSWITCH. None of the input channels issue interrupts to the Master module. The number of input/output channels can be read with the NIO system variable.

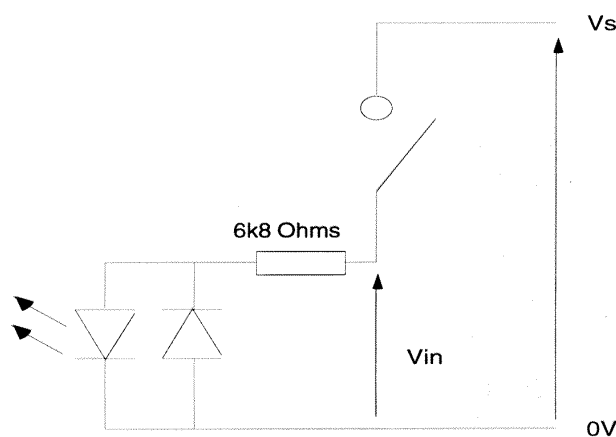


Fig 5.2 16IO Input Circuit

### 5.2.4 Selection of IO Bank

Up to five 16 IO modules may be installed in a system. As the modules are identical the Master must have some means of identifying individual modules. This is done by selecting an IO bank using the selector switch located on the front panel.

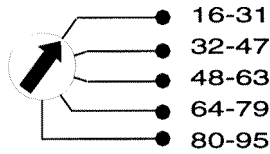


Fig 5.3 Bank Selector Switch

Input/Output channels 0..15 are always provided by the SERIES2 master module. The selector switch is turned with a small screwdriver to select the input/output range required for that module.

#### Example:

A system has two 16 IO modules and a SERIES 2 master module. The first 16 IO modules switch would be set to point to 16-31 and the second 16 IO modules switch would be set to point to 32..47. An output connected to output number 6 in bank 32..47 would be accessed in software as output 38.

#### Input/Output Channels:

Module Label:	Bank 16-31	Bank 32-47	Bank 48-63	Bank 64-79	Bank 80-96
0	16	32	48	64	80
1	17	33	49	65	81
2	18	34	50	66	82
3	19	35	51	67	83
4	20	36	52	68	84
5	21	37	53	69	85
6	22	38	54	70	86
7	23	39	55	71	87
8	24	40	56	72	88
9	25	41	57	73	89
10	26	42	58	74	90
11	27	43	59	75	91
12	28	44	60	76	92
13	29	45	61	77	93
14	30	46	62	78	94
15	31	47	63	79	95



**5.2.5 Summary of Features**

Size	263mm x 68mm x 197mm (HxWxD)
Weight	1.450 kg
Operating temperature	0 - 45 degrees C
Input/Output Channels	16
Input voltage	Vin = 12 Vdc to 32 Vdc (min. Vin for guaranteed ON = 12v)
Maximum current	7mA @ 32 Vdc
Opto- isolation	3500 V
Maximum output current	350mA (50mA driving incandescent lamps)
Output Protection	Thermal and Overcurrent protection with indicator LEDs
Status indicators	1 per channel
Debounce	Hardware
Connection	Two 10- way screw terminal plugs
IO Bank setting	Selected by rotary switch

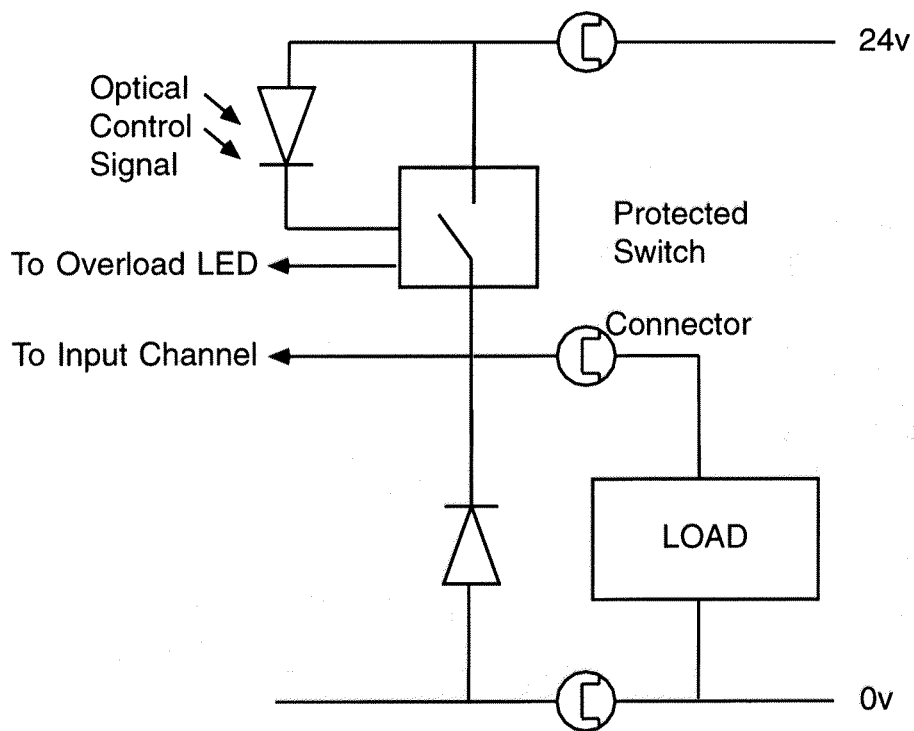


FIG 5.4 Output Channel

### 5.3 CAN 16-I/O Module (P315)

The CAN 16-I/O Module allows the 24volt digital inputs and outputs of the MC204 and MC216 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 MC204/MC216. Convenient disconnect terminal s are used for the I/O connections. Each of the 16 channels in each module is bi-directional and can be used EITHER as an input OR as an output. The CAN 16-I/O Module may also be used as an I/O expander for Lenze drives with an appropriate CAN interface.

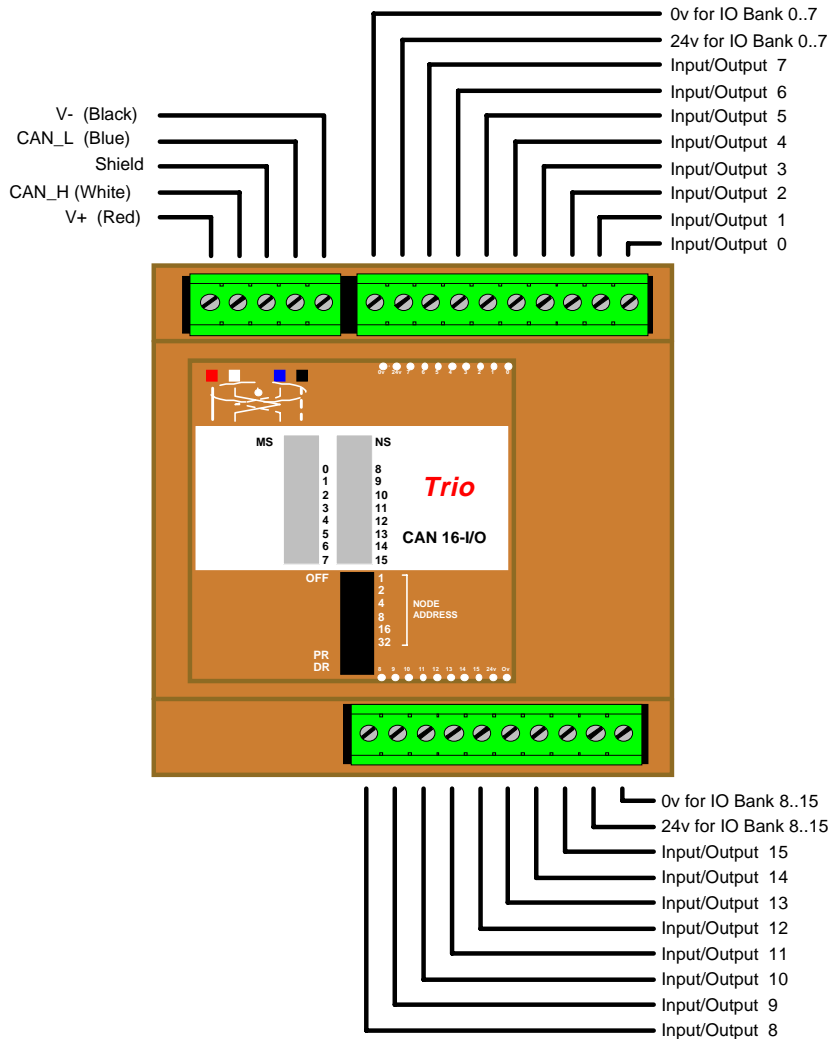
#### 5.3.1 I/O Connections

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



### 5.3.2 Bus Wiring

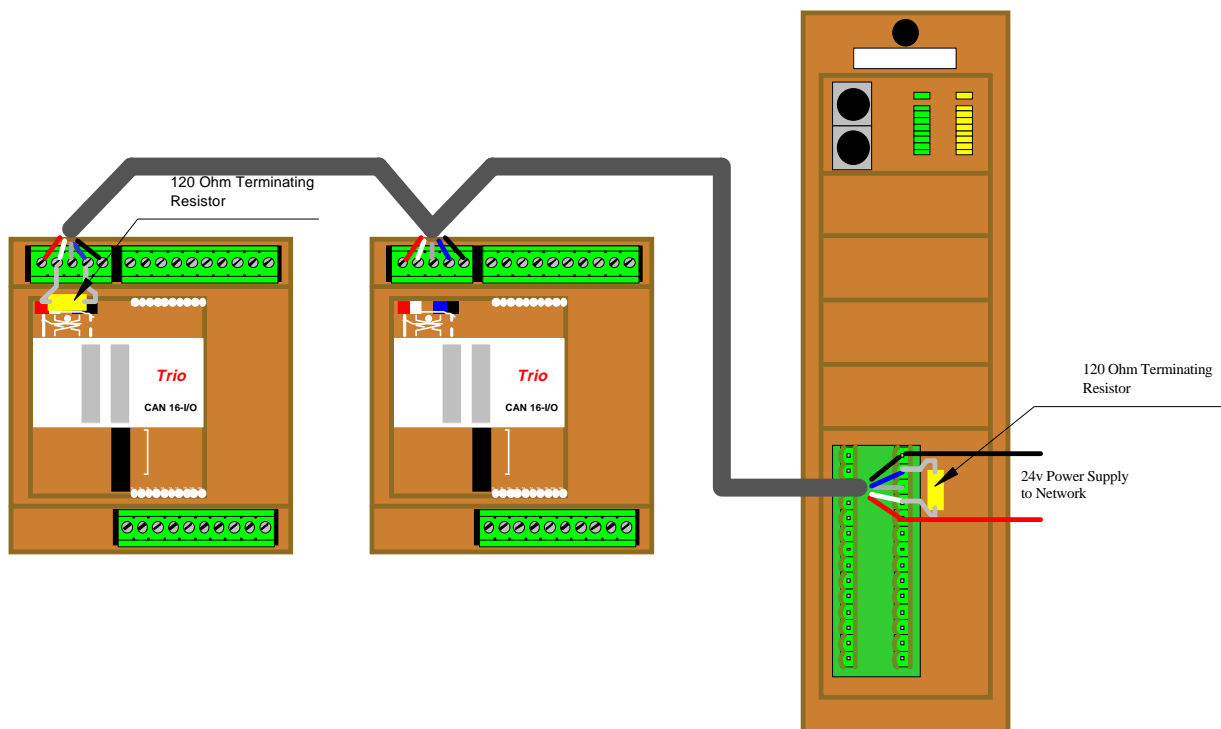
The CAN 16-I/O Modules and the Motion Coordinator are connected together on a network which matches the physical specification of DeviceNet running at 500kHz. The network is of a linear bus topology. That is the devices are daisy-chained together with spurs from the chain. The total length is allowed 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. Contact: Anixter Distribution, 15 Chesford Grange, Woolston, Warrington WA1 4RQ, UK. Phone: (44) 1925 810121

The CAN 16-I/O modules are powered from the network. The 24 volts supply for the network must be externally connected. **The MC204/MC216 does NOT provide the network power.** In many installations the power supply for the MC204/MC216 will also provide the network power. It is recommended NOT to use the I/O power supply to power the network as switching noise from the I/O devices will be carried into the network.



### 5.3.3 DIP Switch Settings

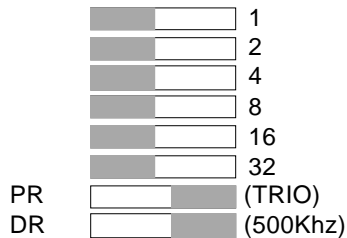
The DIP switches can be set up at present to allow for 2 different protocols. The switch marked "PR" selects the protocol. Switched right it selects the TRIO protocol, switched left the switches select the module to act as a LENZE drive expansion I/O.

#### TRIO Protocol:

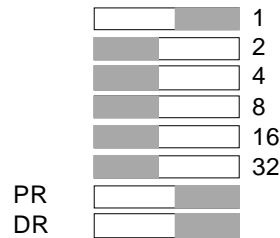
The top 6 DIP switches on the CAN 16-I/O set the module address. Only addresses 0..15 are valid CAN 16-I/O addresses. The switch marked PR is set ON to select TRIO protocol. The switch marked DR sets 125kHz or 500kHz. Only 500kHz is valid with the TRIO protocol.

The addresses for I/O modules MUST be set 0,1,2... in sequence. Therefore the first CAN 16-I/O Module should have the switch setting:

ADDRESS=0  
I/O Channels 16..31



ADDRESS=1  
I/O Channels 32..47

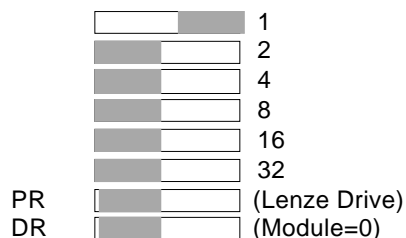


Note that the I/O Channels referred to above are the I/O channels including the 0..15 channels on the MC204/MC216 itself.

#### LENZE Drive Protocol:

The top 6 DIP switches on the CAN 16-I/O set the drive number. This should be set to 1..63. If 0 is set as the drive number the module will transmit to drive 1. The switch marked PR is set OFF to select LENZE drive protocol. The switch marked DR selects which of 2 potential I/O modules can transmit to each drive. The drive should be set to use 500kHz baudrate.

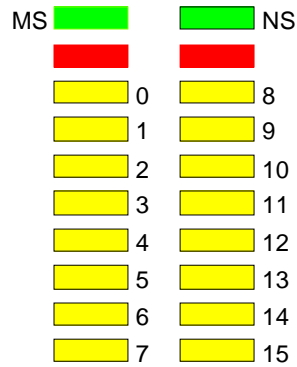
Drive Number=1  
Module Number=0



Drive Address=10  
Module Number=1

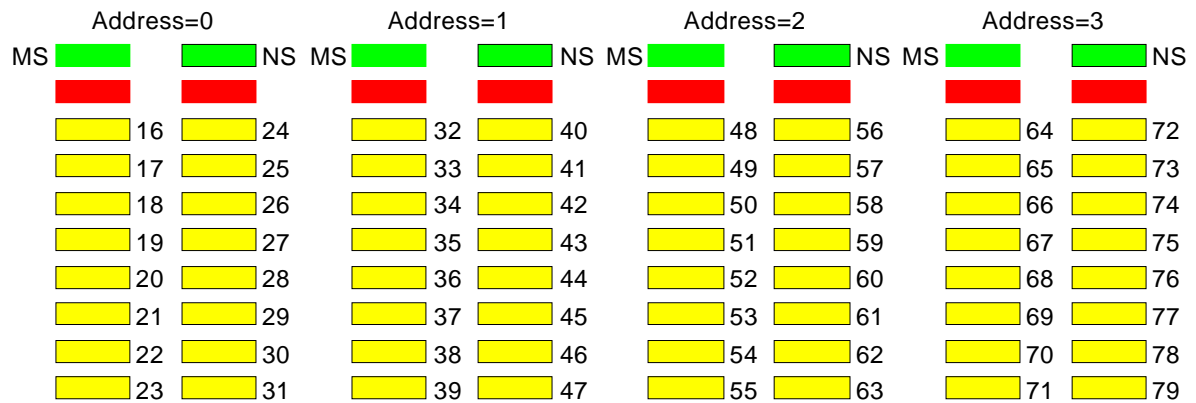


### 5.3.4 LED Indicators

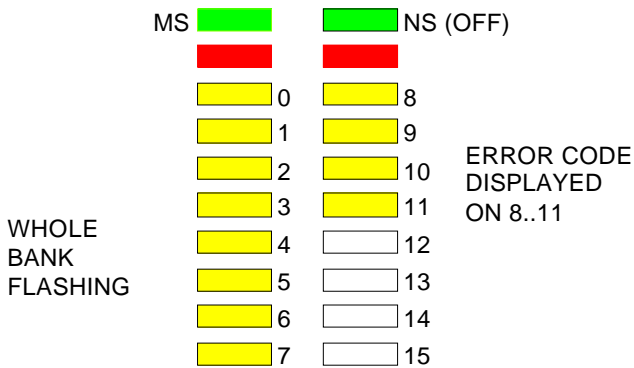


MS    Represents "Module Status"    - ON when module powered on OK  
 NS    Represents "Network Status"    - ON when module powered on OK and initialised.

When NS is ON LEDs marked 0..15 represent the input channels 0..15 of the module. The input channel number in the total network depends on the address:



Address:	Start:	End:
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



Error Codes:

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

**5.3.5 Software Interfacing**

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 new system parameters are available to facilitate the use of the CAN 16-I/O:

```
CANIO_STATUS
CANIO_ADDRESS
CANIO_ENABLE
```

Chapter 8 has a definition of these new system parameters.

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

Inputs 0..31 ONLY are available for use with system parameters which specify an inputs such as FWD\_IN.

Outputs 8..31 ONLY are available for use with the PSWITCH command.

The built-in I/O channels have the fastest operation <1mS

CAN I/O channels 16..64 have the next fastest operation <2mS

CAN I/O channels 64..191 have the next fastest operation <8mS

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

### 5.3.6 Troubleshooting

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 MC204/MC216 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... ?

System Software Version 1.40 (or higher) ?

Motion Coordinator CANIO\_ADDRESS=32 ?

### 5.3.7 Specification

Inputs:	16	24volt inputs channels with 2500v isolation
Outputs:	16	24volt 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 overtemperature protected	
Indicators:	Individual status LED's	
Address Setting:	Via DIP switches	
Power Supply:	24v / 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 (1995) Class A Industrial Noise Emissions	

## 5.4 CAN ANALOG INPUTS Module (P325)

The CAN Analog Input Module allows the MC204 and MC216 to be expanded with banks of 8 analog input channels. Up to 4 x P325 Modules may be connected allowing up to 32 x 12 bit analog channels. Convenient disconnect terminals are used for the I/O connections. The input channels are designed for +/-10volt operation. Each bank of 8 channels is opto-isolated from the CAN bus

### 5.4.1 I/O Connections

The CAN Analog Input Module has 3 disconnect terminal connectors:

DeviceNet physical format 5 way CAN connector

Input Bank 0..7 with 0v reference and earth on 10 way connector

The lower 10 way connector is unused

### 5.4.2 Bus Wiring

See 5.3.2

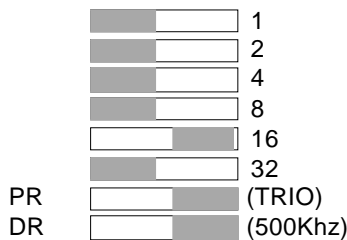
### 5.4.3 DIP Switch Settings

The switch marked "PR" selects the protocol, but is currently unused as only the TRIO protocol is available.

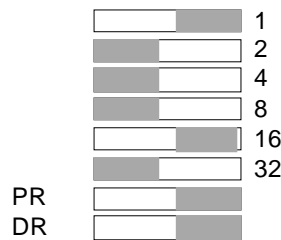
The switch marked DR sets 125kHz or 500kHz. Only 500Khz is valid with the TRIO protocol.

The addresses for P325 modules MUST be set 16,17,18... in sequence. Therefore the first P325 Module should have the switch setting:

ADDRESS=16  
Analog Inputs 0..7



ADDRESS=17  
Analog Inputs 8..15



**Note: P325 modules and P315 (16-I/O) modules may be mixed on the network. The P315 addresses will be 0 to 15 in sequence and the P325 modules will be 16 to 19 in sequence.**

### 5.4.4 LED Indicators



MS Represents "Module Status" - GREEN ON when module powered on OK  
NS Represents "Network Status" - GREEN ON when module powered on OK and initialised.

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



### 5.4.5 Software Interfacing

The Motion Coordinator will automatically detect and allow the use of correctly connected P325 modules. The number of connected analog input channels is reported in the startup message and is also available to the programmer via an additional system parameter "NAIO".

The analog input resolution is fixed at +10volts to -10volts will return values 2047..-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.

The P325 works "single ended" and does not return differential values.

It is not possible to mix the P325 module which is running the TRIO I/O protocol with DeviceNet equipment on the same network.

### 5.4.6 Troubleshooting

If the network configuration is incorrect 2 indications will be seen: The P325 module will indicate that it is uninitialised and the MC204/MC216 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... ?

System Software Version 1.42 (or higher) ?

Motion Coordinator CANIO\_ADDRESS=32 ?

### 5.4.7 Specification

Analog Inputs:	8	+/-10 volt inputs channels with 500v isolation from CAN bus
Resolution:	12 bit	
Protection:	Inputs are protected against 24v overvoltage.	
Address Setting:	Via DIP switches	
Power Supply:	24v / 1.5W	
Mounting:	DIN rail mount	
Size:	95mm wide x 45mm deep x 105mm high	
Weight:	200g	
CAN:	500kHz, Up to 32 analog input channels	
EMC:	BSEN50082-2 (1995) Industrial Noise Immunity / BS EN55022 (1995) Class A Industrial Noise Emissions	

## **5.5 Operator Interfaces on the Fibre Optic Network**

Trio can supply a range of operator interfaces:

- \* Membrane Keypad
- \* Mini-membrane keypad
- \* Trio Touchpro PC (Not covered by this manual)
- \* Fibre-optic interface module (Allows users to design their own keypad on Trio fibre-optic network)

### **5.5.1 Membrane Keypad**

The Membrane Keypad brings together all the elements required for an effective man-machine interface in one package thus minimising the time taken to mount and connect to the rest of the system. The keypad has 37 tactile keys, eight of which can be defined by the user by inserting key legends from the rear of the keypad. Incorporated into the keypad is a four line by twenty character vacuum fluorescent display. Connection to the control system is via a fibre optic cable to the Motion Coordinator MC2, MC216 or MC204 master module or network of master modules. The interfacing to the master is provided by a built in fibre-optic interface module. The only other connection necessary is a 24 Volts DC power supply input.

### **5.5.2 Mounting the Membrane Keypad**

To mount the Membrane Keypad a rectangular cutout and four holes are required, as shown in fig 5.5. The Keypad is offered up to the front of the panel and fixed with the four studs in the corners of the Keypad. A depth of 50mm behind the front panel is needed to mount the Keypad, with an extra 50mm clearance for the fibre optic connector on the back.

### **5.5.3 Connection of Membrane Keypad**

The fibre optic link to a mater module or network of masters is the Hewlett-Packard "Versatile Link" format. The fibre optic modules are colour coded:

Blue - Receiver  
Grey or Black- Transmitter

The receiver on the membrane keypad must be connected a transmitter and vice-versa. The fibre optic link is running at 38400 baud and will operate over a distance of up to 30m. Care must be taken when installing the fibre cable, making sure it is not bent in a tighter radius than 100mm. Failure to observe this restriction could lead to a break in the cable or at very least attenuation of a signal giving a reduced distance over which the link will operate. An excessive number of bends in the cable will attenuate the signal and hence reduce the operating distance.

Power is applied to the two pin disconnect terminals on the side of the Keypad.

## **5.6 Mini-Membrane Keypad**

The Mini-Membrane Keypad is a lower cost alternative to the full membrane keypad. The keypad has 25 tactile keys. Incorporated into the keypad is a two line by twenty character vacuum fluorescent display. Connection to the control system is via a fibre optic cable to the Motion Coordinator MC2 or MC204 master module or network of master modules. The interfacing to the master is provided by a built in fibre-optic interface module. The only other connection necessary is a 24 Volts DC power supply input.

### 5.6.1 Mounting the Mini-Membrane Keypad

The Mini-Membrane Keypad can be either mounted into a rectangular cutout or may be mounted into a 3U rack. If rack mounted the 4 blind holes in the panel need to be extended to allow 4 2.5mm diameter screws (Not supplied) to be fitted to secure the unit into a rack. To mount the Mini-Membrane Keypad into a rectangular cutout the figure 5.7 should be followed. The Keypad is offered up to the front of the panel and fixed with the four studs in the corners of the Keypad. A depth of 50mm behind the front panel is needed to mount the Keypad, with an extra 50mm clearance for the fibre optic connector on the back.

### 5.6.2 Connection of Mini-Membrane Keypad

See 5.5.2. The connections are similar

### 5.6.3 Programming the Membrane Keypad and Mini-Membrane Keypad

The Keypads make use of standard TRIO BASIC commands to write to the display and read from the keypad. The output /input device should be specified as 4 or 3 in any PRINT , GET, or KEY statement E.G.

```
>>PRINT#4, "Hello"
```

Alternatively the membrane keypad can be used as part of a network, in which case see chapter 11 for further details.

#### 5.6.3.1 Writing to the Membrane and Mini-Membrane Display

The BASIC command PRINT (see chapter 8) is used to write to the display. By using the CHR command with the PRINT it is possible to send control codes to the display to perform certain functions as described below:

<i>CHR(..)</i>	<i>Function</i>	<i>Description</i>
8	Back Space	The cursor moves one character to the left.
9	Horizontal Tab	The cursor moves one character to the right.
10	Line Feed	The cursor moves to the same column on the next line down.
12	Form Feed	The cursor moves to the top left hand corner.
13	Carriage Return	The cursor moves to the end on the same line.
14	Clear	All displayed characters are cleared. The cursor doesn't move.
17	Overwrite Mode	When the cursor reaches the bottom right hand corner it moves to the top left hand corner.
18	Scroll Up Mode	When the cursor reaches the bottom right hand corner the display scrolls up one line and the cursor moves to the left hand end of the next line.
20	Cursor _	Cursor is displayed as an _ character (Mini-Membrane only)
21	Cursor Visible	Cursor is displayed as a blinking all dot character.
22	Cursor Invisible	Cursor is turned off.
23	Cursor Flashing _	Cursor is displayed as a blinking _ character (Mini Only)
27+72+0..79	Position Cursor	The cursor is moved to the position specified by the last number (in the range 0..79 on Membrane keypad, 0..39 on Mini-Membrane) where each position on the screen is numbered, starting with zero in the top left hand corner to 79 or 39 in the bottom right hand corner.

**Note:** The CURSOR command provides a easy method of controlling the cursor. For Example:

```
PRINT CURSOR(10);
```

This will send the cursor to the 10th position on the first row. Note the use of the semicolon to suppress the carriage return which the PRINT command would normally send as well.

**5.6.3.2 Reading from the Membrane Keypad/Mini-Membrane Keypad**

Use the KEY command to test if a key has been pressed and the GET command to read which key has been pressed. **For simplicity and consistency it is recommended to use KEY and GET with the #4 channel number.** It is also possible to use the #3 channel number, in which case the numbers returned can be modified using DEFKEY.

Key	Key Number	GET#4 value	GET#3 value
Up Arrow	1	33	20
Left Arrow	2	34	22
Center Button	3	35	24
Right Arrow	4	36	23
Down Arrow	5	37	21
*Undefined 1 (Left most)	12	44	30
*Undefined 2	13	45	31
*Undefined 3	14	46	32
*Undefined 4	15	47	33
*Undefined 5	16	48	34
*Undefined 6	17	49	35
*Undefined 7	18	50	36
*Undefined 8 (Right most)	19	51	37
7	23	55	55
8	24	56	56
9	25	57	57
Y	26	58	89
4	27	59	52
5	28	60	53
6	29	61	54
N	30	62	78
1	34	66	49
2	35	67	50
3	36	68	51
CLR	37	69	27
-	38	70	45
0	39	71	48
.	40	72	46
<enter>	41	73	13
*Menu Select 4 (Bottom Left)	45	77	50
*Menu Select 3	46	78	51
Menu Select 2	47	79	52
Menu Select 1 (Top Left)	48	80	53
Menu Select 5 (Top Right)	49	81	54
Menu Select 6	50	82	55
*Menu Select 7	51	83	56
*Menu Select 8 (Bottom Right)	52	84	57

Keys marked \* are not present on Mini-Membrane

### 5.6.3.3 Keypad KEY ON - KEY OFF Mode

Keypads with software version 3.00 (see back panel for version number) and higher support a mode of operation where the keypad returns the key pressed and then a further character (31) is returned when the key is released. The keypad has to be set into this mode. On power up the keypad is in the normal mode where it returns just the key number.

KEY ON-KEY OFF mode:

To set KEY ON-KEY OFF mode: PRINT#4,CHR(140);CHR(127);CHR(136);

To return to default mode: PRINT#4,CHR(140);CHR(0);CHR(136);

Note that in this mode the key presses MUST be fetched with a GET#4 rather than GET#3. This is because the KEY RELEASED character (31) is not included in the DEFKEY table used with GET#3.

### 5.6.4 Summary of Features P503 Membrane Keypad

Size	230mm x 180mm x 50mm
Weight	1.450Kg
Operating Temperature	0-45 degrees C
Power supply	24 VDC 500mA
Number of Keys	37
Type of Keys	Metal dome tactile, debounced
Display	4x20 Vacuum Fluorescent, with anti-glare filter
Environmental	Sealed to IP65, provided mating face to panel is sealed
Material	Polyester top layer resistant to most solvents
Data Output	38400 baud, fibre optic link

## 5.7 FO-VFKB Fibre Optic Keypad/Display Interface

This is not packaged as a module like the rest of the TRIO range. Instead it is a single PCB designed to fix directly on to the back of a Vacuum Fluorescent display to allow customers to easily build their own design of membrane keypad on the Trio fibre-optic network. The connectors and mounting holes on the board are specifically intended for mounting on the following displays:

ITRON	2 x 20
	4 x 20

Other displays can be supported but connection may have to be made via a short cable from the display to the FO-VFKB. To give reliable, noise free transmission of data to and from the FO-VFKB, the link to the master module is made with a fibre optic cable.

### 5.7.1 FO-VFKB Display Interface

Display Interface:

DIL connector:	<u>FUNCTION</u>	<u>PIN NO.</u>	<u>FUNCTION</u>
	D7	1 2	D6
	D5	3 4	D4
	D3	5 6	D2
	D1	7 8	D0
	WR	9 10	GND
	N/C	11 12	BUSY
	GND	13 14	GND
	+5V	15 16	+5V

### 5.7.2 FO-VFKB Keypad Interface

The keypad interface permits connection of proprietary or custom keypads with matrix outputs of up to 5 rows by 11 columns. Up to five common output keypads may be connected provided that each one has no more than 11 keys. The keypad is easily accessed via TRIO BASIC commands which enables a program to accept run-time data from an operator or scroll through a menu for example. Connection to the keypad is made via a short ribbon cable between the 16 way IDC plug on the back of the FO-VFKB module and the similar connector on the back of the keypad.

<u>Function</u>	<u>IDC plug pin no.</u>		<u>Function</u>
Row 4	1	2	Column 0
Row 1	3	4	Column 7
Column 4	5	6	Row 2
Column 1	7	8	Column 8
Column 5	9	10	Row 3
Column 2	11	12	Column 9
Column 6	13	14	Column 3
Row 0	15	16	Column 10

### 5.7.3 Power Requirements

A 24V DC power supply is required to drive the FO-VFKB. This is also the supply for the display so must be capable of delivering at least 750mA. Power is applied on the two way disconnect terminal. The board is diode protected against reverse voltages being applied to the power connector. With power applied to the board the green LED on the top of the board should be lit.

#### **5.7.4 Data Connection**

See 5.2.2 . The data connection is identical to that of a membrane keypad.

#### **5.7.5 Summary of Features P504 FO-VFKB**

Size	150mm x 64mm
Weight	0.065 Kg
Operating Temperature	0-45 degrees C
Keypad	5x11 debounced keypad decoder with key on/key off option
V/F Display	Direct connection on DIL header
Data Connection	Fibre optic data link
Power supply	24V DC external power supply required

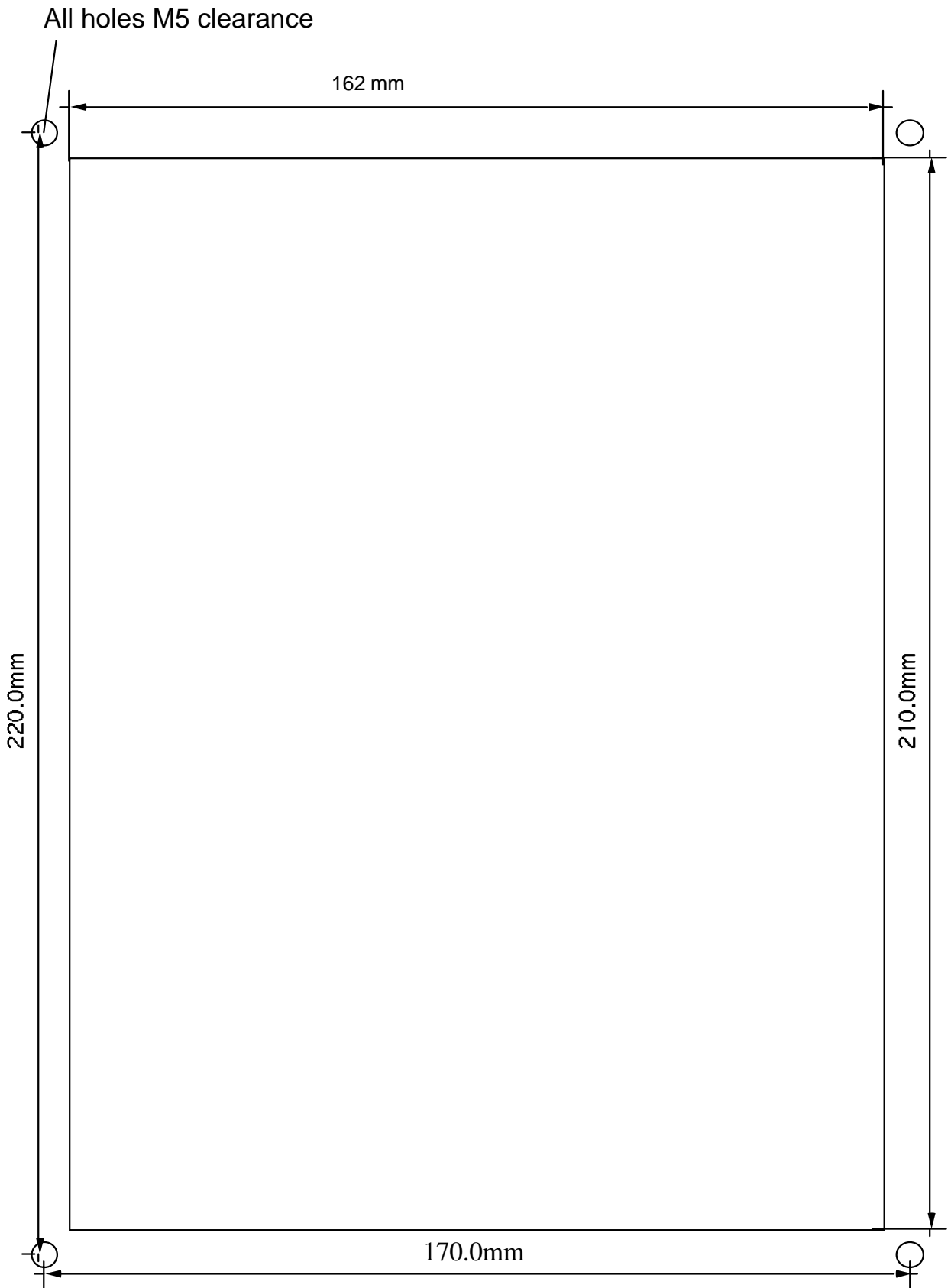


Fig 5.5 Membrane Keypad Mounting Holes



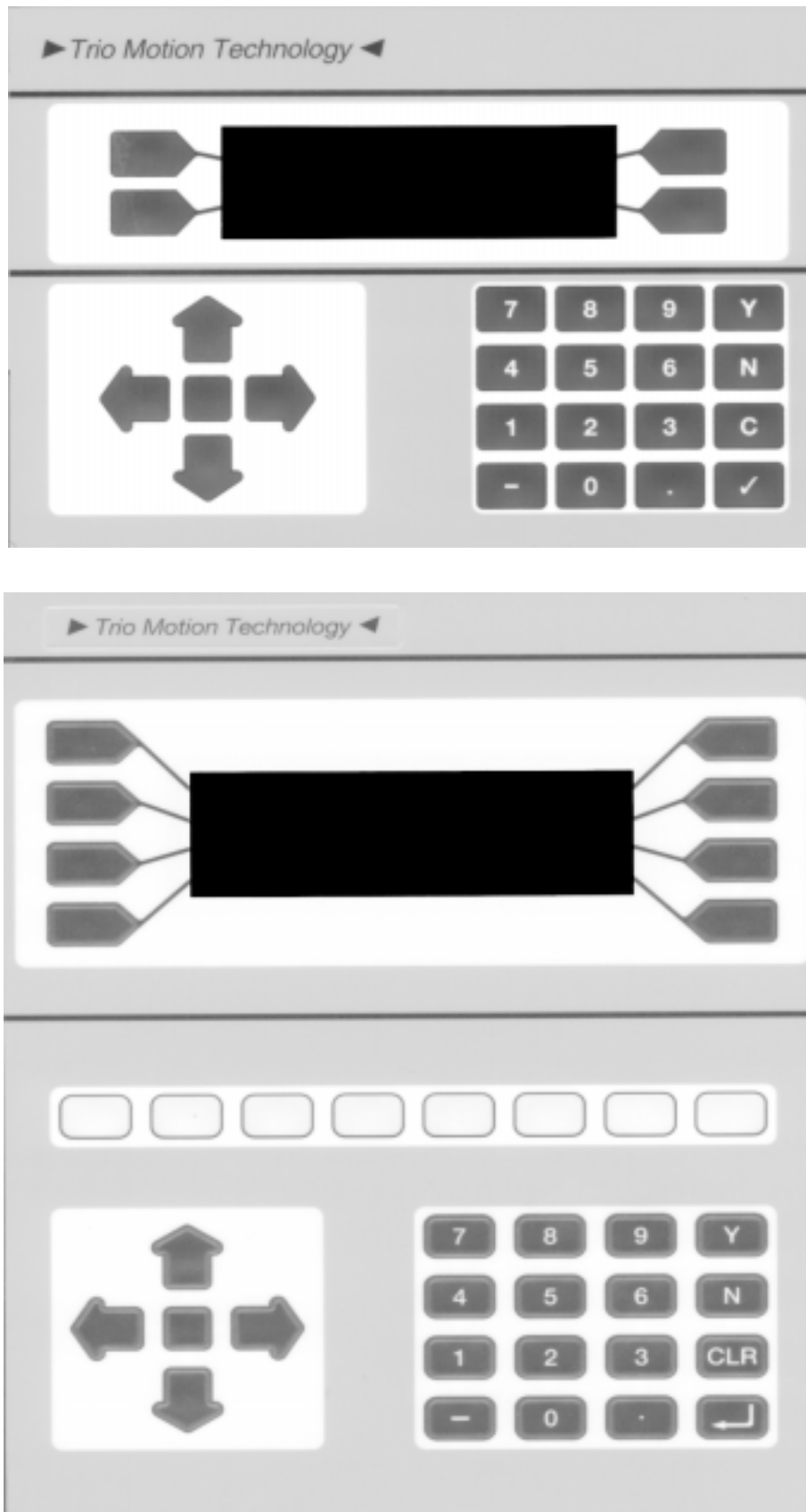


Fig 5.6 Mini-Membrane and Membrane Keypad. Not to scale

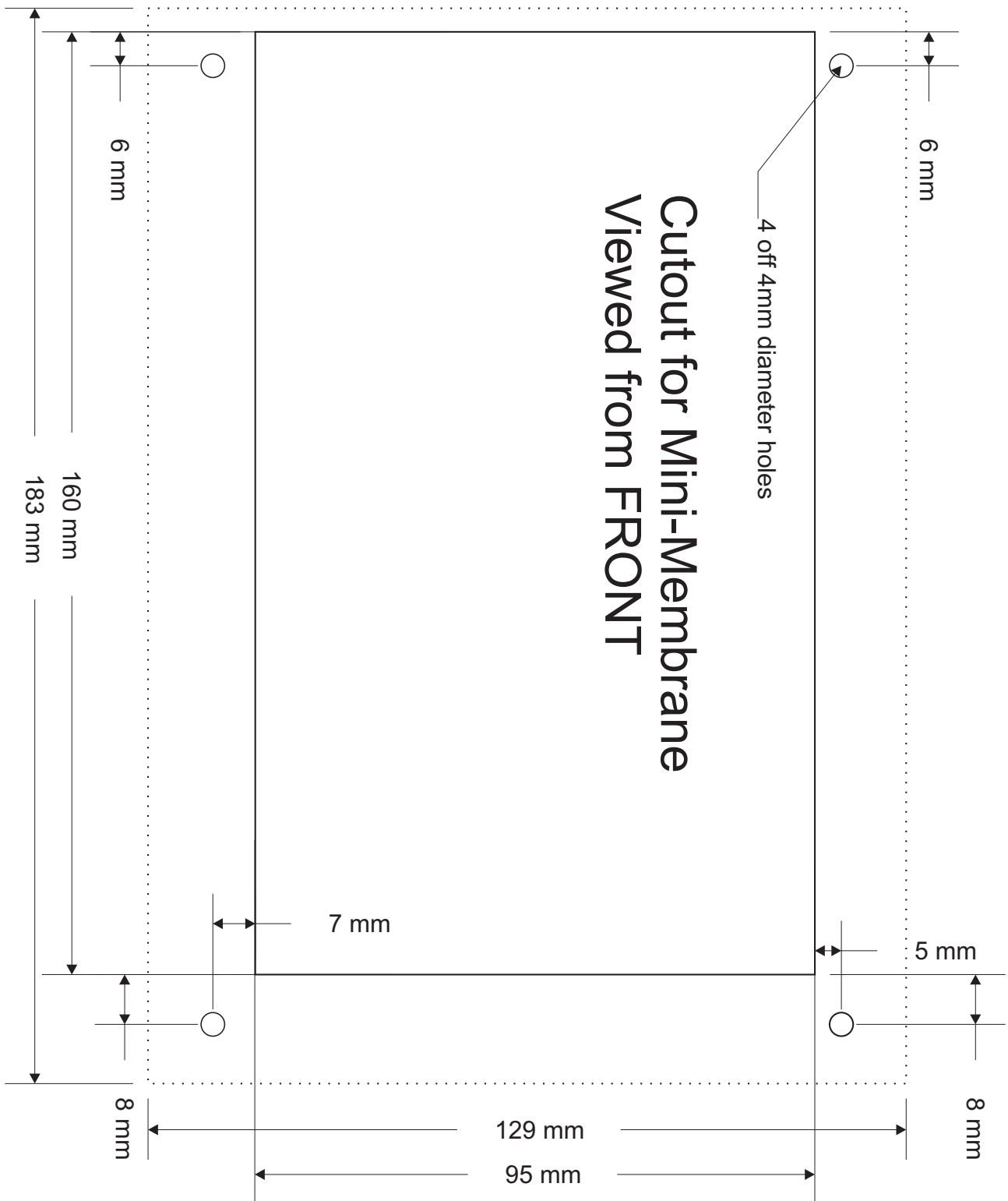


Fig 5.7 Mini-Membrane Mounting

## 6.0 System Setup and Diagnostics

### 6.1 Preliminary Concepts:

- Host Computer A Windows PC running *Motion Perfect*.
  
- Motor This shall be understood as a tuned amplifier motor configuration for a servo axis or a stepper motor and amplifier combination
  
- Prompt When the SERIES 2 controller is ready to receive a new command prompt >> will appear on the left hand side of the current line in the "terminal" under the "tools" menu .
  
- Axis Parameters The functioning of each axis is governed by the corresponding axis parameters. Each axis parameter has a name and the parameter 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: ? P\_GAIN.For further information see chapter 9.

### 6.2 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 both to all SERIES 2 modules. TRIO recommend that this section is read in full before attempting to operate the system for the first time.

#### 6.2.1 Preliminary checks

All wiring should be checked for possible misconnection and integrity before any power is applied.

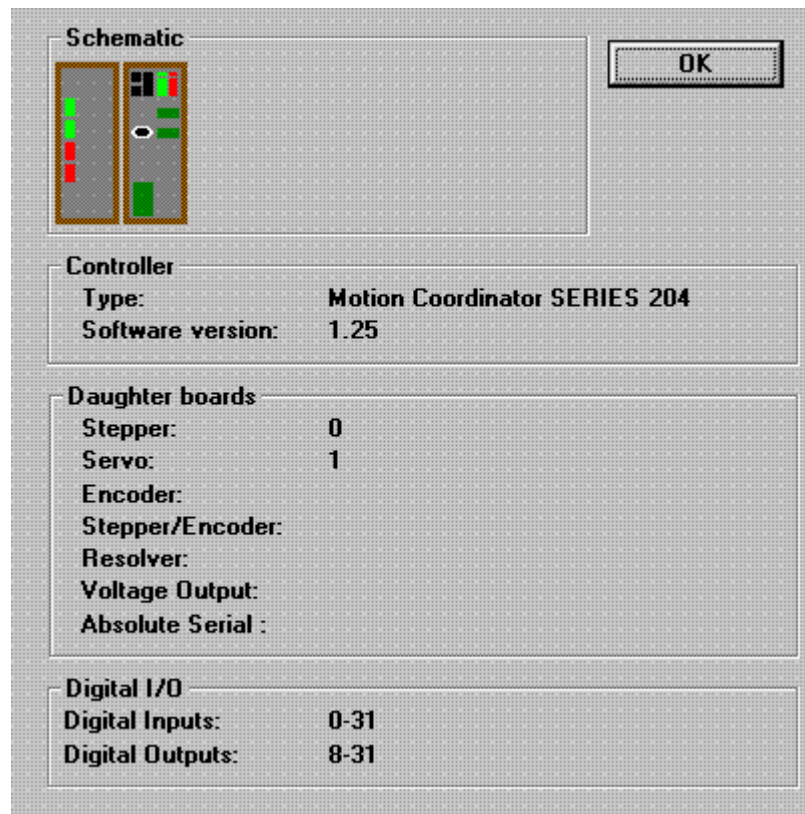
In the following procedure make sure that everything is disconnected until it is required to be connected.

- Disconnect all connectors from the system apart from ribbon cable bus
- Apply power to system with connectors disconnected at check 24v power input on master
- Check bank selector switches on IO16 or Axis Expander modules
- Apply 24v to SERIES 2 master only and connect serial lead between serial port A on master and an unused serial port on your PC.
- Run "*Motion Perfect*" and check it finds your Motion Coordinator.

### 6.2.2 Checking Serial Communications and System Configuration

- Make sure that the serial lead is connected between the SERIES 2 master serial port A and a serial port on the computer.
- With the 24v power only applied to the master run *Motion Perfect* on the computer. When *Motion Perfect* detects a controller press the OK 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:
  - (1) *Motion Perfect* has made a serial connection between your PC and the controller.
  - (2) *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 mimic of your system:

Example:



This message would be produced by a Motion Coordinator MC204 with a servo daughter board in axis slot 0 and a stepper daughter board in axis slot 1. One CAN16 I/O module is connected to expand the I/O to 32/24.

Check that the system description corresponds with the modules that are actually present. If this is not so, check the ribbon cable connection and the settings of the DIP switches on any 16IO or axis expander modules.

### 6.2.3 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. On the MC2 if the red fault LED lights in any case the output is probably overloaded or short - circuited. Remove the power and check the wiring and load before starting again. The red LED can only be reset by removing the power.

### 6.2.4 Connecting a Servo Motor to an Servo Daughter Board

Each servo axis should be connected in turn. This description assumes the motor/ amplifier combination are already tested and functioning.

- With the servo amplifier off or inhibited connect the motor encoder only. 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 parameter 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 +/-10volt output for the axis. Now connect the servo amplifier to the V+V- connections.
- Enable the servo amplifier by clicking the "Drives disabled" button on the control panel. If the axis runs away the motor/amplifier combination must be re-checked. (Note: clicking "Drives disabled/ 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 as the position servo has been switched OFF.
- Set a small positive output voltage by setting DAC=25. 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=-25.
- 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:
  - (1) - Swap A and /A connections on the encoder input  
or
  - (2) - Swap BOTH motor terminals and tacho terminals (DC motors only !) On many digital brushless motors the direction can be reversed by an amplifier setting  
or
  - (3) - If amplifier has differential inputs reverse voltage as it enters amplifier. (This can cause problems with some servo-amplifiers. The V- pins of the daughter board are internally connected inside the Motion Coordinator so the axis voltage outputs cannot float relative to each other)  
or
  - (4) - set a negative PP\_STEP axis parameter. (This is not possible using SSI encoders)
- We are now ready to apply the position servo as described in 6.2.5 :

### 6.2.5 Setting Servo Gains

The servo system controls the motor by constantly adjusting the voltage output which gives a speed demand to the servo amplifier. The speed demand is worked out by looking at the measured position of the axis from the encoder and 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 voltage output 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:	(Note that fractional values are allowed)
Proportional Gain	1.0	
Integral Gain	0.0	
Derivative Gain	0.0	
Output Velocity Gain	0.0	
Velocity Feedforward Gain	0.0	

A simple test program can be used to generate movement to and fro for examination of the motion profile generated on an oscilloscope. The oscilloscope should be connected to the tachometer or the velocity output provided by the servo amplifier.

```

start:    PRINT "Enter Axis Number ":INPUT VR(0)
          BASE(VR(0))
          SPEED=20000
          ACCEL=200000
          DECEL=200000
loop:     MOVE(1000)
          WAIT IDLE
          WA(100)
          MOVE(-1000)
          WAIT IDLE
          WA(100)
          GOTO loop
    
```

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, replacing the name UNTITLEDx. Now click on the EDIT button and an edit window will be opened where the program shown above may be typed in. See section 10.11.3 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.

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 adds smoothing damping at the expense of higher following errors.
- Low following errors during complete motion cycle  
This can be achieved by using velocity feedforward 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 decelerations.

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 below.

## 6.2.6 Gain Parameters

### 6.2.6.1 Proportional Gain

The proportional gain creates an output voltage,  $O_p$  that is proportional to the following error  $E$ .  $O_p = K_p \times E$

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

Axis parameter is called P\_GAIN

### 6.2.6.2 Integral gain

The Integral gain creates an output  $O_i$  that is proportional to the sum of the errors that have occurred during the system operation.  $O_i = K_i \times SE$

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

Axis parameter is called I\_GAIN

### 6.2.6.3 Derivative gain

This produces an output  $O_d$  that is proportional to the change in the following error and speeds up the response to changes in error whilst maintaining the same relative stability.  $O_d = K_d \times DE$

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

Axis parameter is called D\_GAIN

### 6.2.6.4 Output Velocity

This increases the system damping, creating an output that is proportional to the change in measured position.  $O_{ov} = K_{ov} \times DP_m$ . This parameter can be useful for smoothing motions but will generate high following errors.

Axis parameter is called OV\_GAIN

### 6.2.6.5 Velocity Feed Forward

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.  $O_{vff} = K_{vff} \times DP_d$

Axis parameter is called VFF\_GAIN

The VFF\_GAIN parameter can be set by minimising the following error at a constant machine speed AFTER the other gains have been set.

## 6.3 Diagnostics

### 6.3.1 No Status LEDs

- On any module- **Power Supply**
- LEDs lit on Master but not on other modules - **Ribbon cable / Bank select switch**

### 6.3.2 Status LEDs indicate an error condition

- Green OK LED ON and orange Status LED flashing - **Following error on at least one axis.**  
The axis demand position and measured position exceed the programmed limit.

### 6.3.3 Motor runs away without issuing a move command

- tacho/amplifier polarity
- encoder/controller polarity
- gains (amplifier and/or controller)

### 6.3.4 Motor runs away upon issuing a move command

- tacho feedback
- encoder feedback
- gains (amplifier and/or controller)

### 6.3.5 Motor does not move upon issuing a move command

- wiring (enables/inhibits/limits on amplifier and controller)-check status on all axes
- amplifier power
- feedhold applied
- speed, acceleration and/or deceleration set to zero
- servo set off/watchdog set off
- gains (amplifier and/or controller)
- axis is already running a move which has not completed - Check MTYPE and NTYPE

### 6.3.6 Axis goes out on following error after a time

- speed being requested requires more than 10v - check amplifier tacho gain and motor/ amplifier speed characteristics.
- amplifier shutting down on current limit after a time

### 6.3.7 Axis losing position

- encoder coupling
- encoder signal (wire length, differential/single ended encoder)
- mechanics



### 6.3.8 Motion Perfect cannot "connect" with the controller

- Controller running a program which transmits on serial port A. If this prevents *Motion Perfect* connecting to the controller, open Terminal screen in *Motion Perfect* unconnected mode and type a "halt" command at the command prompt.
- Faulty or unconnected serial cable
- *Motion Perfect* baudrate, databits, parity or stopbits have been changed. - Adjust to default of 9600 baud, 7 data bits, 2 stop bits, even parity under the "Options" menu.



---

## 7.0 Motion Coordinator MC2/MC204/MC216 - Multi-tasking BASIC programming:

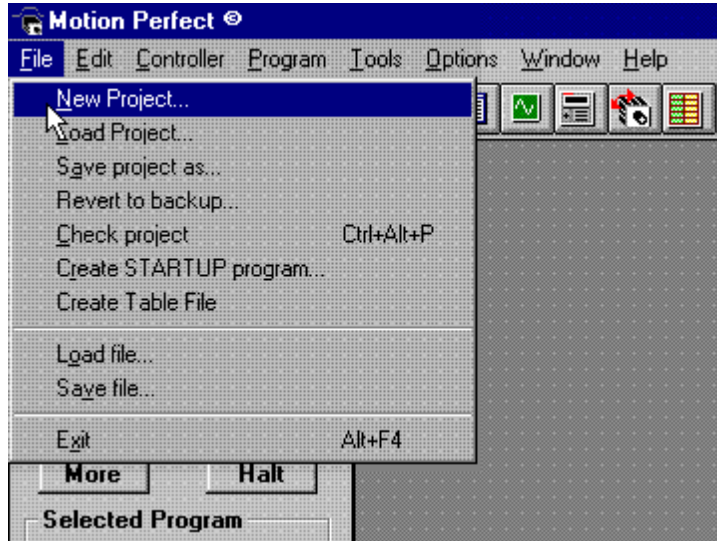
### 7.1.1 Overview:

- The Motion Coordinator MC216 and MC204 feature a multi-tasking version of the TRIO BASIC language. The multi-tasking is simple to set up and use and allows very complex machines to be programmed. The MC2 and MC216 has up to 15 "processes" available, the MC204 has up to 6 "processes" available. On both controllers one process (process 0) is reserved for accepting and responding to BASIC commands on a serial port. This is called the "Command Line" task. The other tasks may all run a separate BASIC program at the same time.
- The controllers have facilities for storing multiple programs in their memory. The controllers have simple file handling instructions for managing these program files rather like the DOS filing system on a PC. The programs may be stored to or loaded from a PC for archiving, printing and some editing jobs. Motion Perfect is an motion application development program designed to complement the facilities of the Motion Coordinator. *Motion Perfect* assists in maintaining the group of BASIC programs which run a machine by keeping a copy of them organised in a folder (sub-directory) on your PC's hard disk.
- Your BASIC programs are held in flash eprom or battery-backed RAM. The BASIC program is built by connecting a PC to the serial port A of the controller and using the facilities of *Motion Perfect* to write and test one or more programs. The programs may be set to run automatically on power up at different priorities. If required the PC may be left connected as an operator interface or may be removed and the programs run "standalone".
- The entire system software for the controller is held in flash EPROM as well as your BASIC programs. For most applications this is not necessary, but allows 2 benefits:
  - The system software can be updated by downloading a file from a computer disk. ( This is achieved using the "Load system software" facility under the "Controller" menu). Updates are downloadable from Trio's website at [www.triomotion.com](http://www.triomotion.com)
  - If the user has the necessary programming tools and TRIO files, users may make additions to the system software in the 'C' language to suit their own specialised requirements. A typical example of this are the transformation functions required by robots. Programming extensions in 'C' requires detailed knowledge of the SERIES 2 and specialised compilers and development tools. Please contact TRIO for details of the costs and benefits.

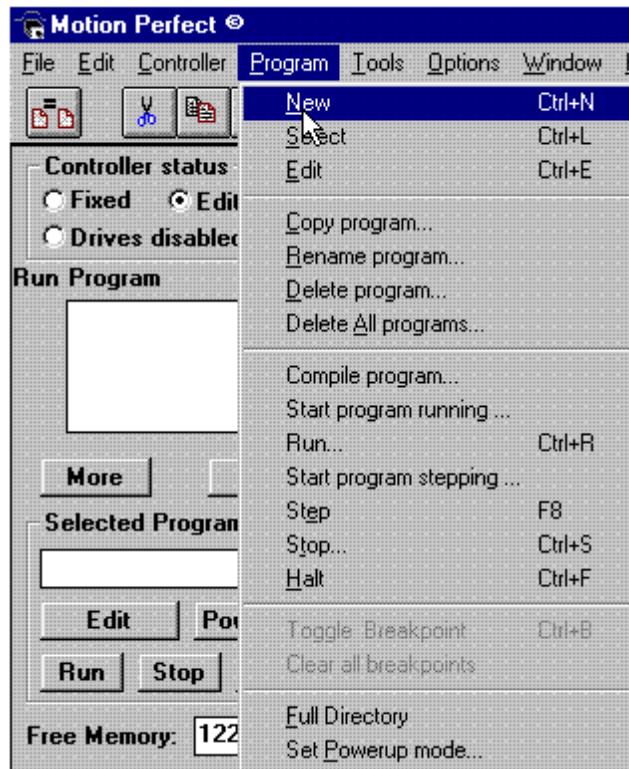
**7.1.2 Exercise: Editing and running 2 simple programs:**

As an introduction to the SERIES 2 programming system this section takes you through entering and running two simple programs using *Motion Perfect* then running them simultaneously. As a starting point it is assumed you have a controller with Motion Perfect connected.

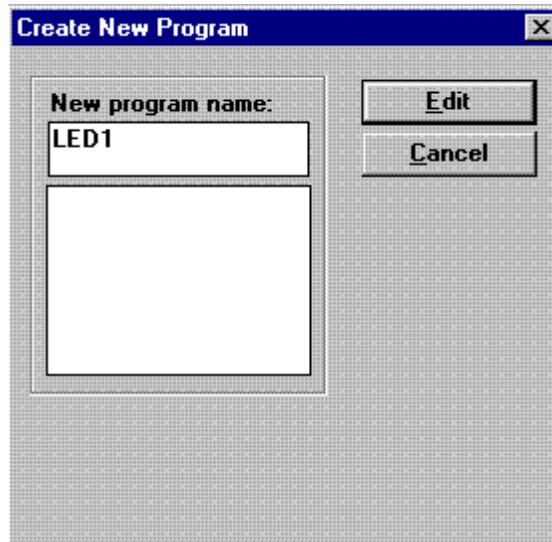
Firstly we create a new "project" using the "New project..." option under the "File" menu:



Confirm that you want to erase the contents of the controller. Give your project a title "project1" for example. Now create a new program (which is part of the project) by using "New..." under the "Program" menu



The program is given a name and the *Motion Perfect* editor is opened automatically. Name your program LED1:



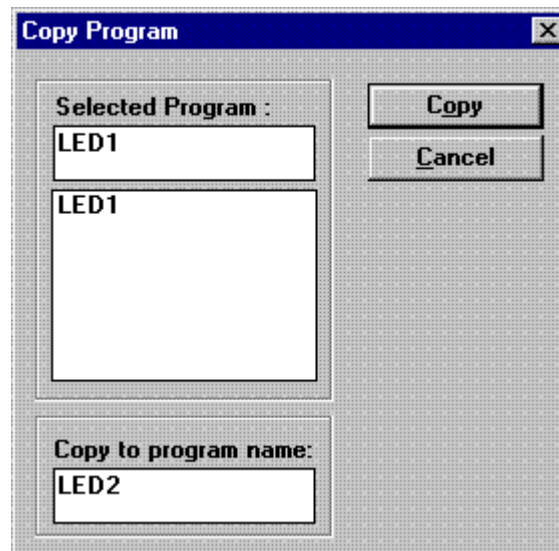
Using the editor enter a simple program to flash the LED connected to output 8:

## ► Trio Motion Technology ◀

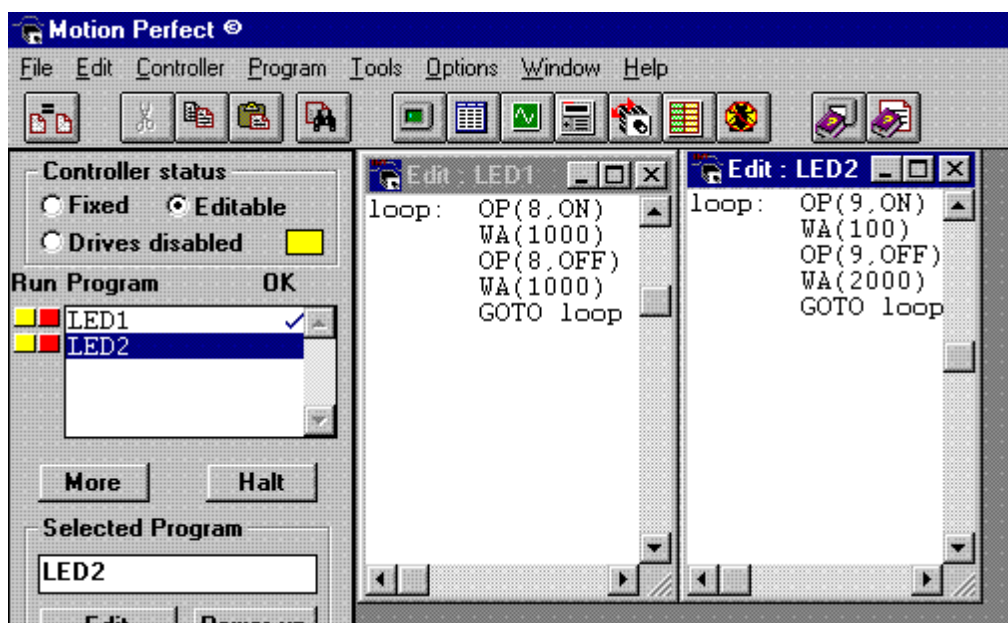
The system compiles and links the program before running it. If the compiler detects an error in the program it will not run, but will print the line number at which the error occurred. That line can be located by looking at the current line number in the editor which is displayed in the bottom right of the status bar or by using the "Goto.." facility under the "Edit" menu. When the program runs the LED indicator for output 8 will flash. Note how the command line remains available for immediate commands if a channel 0 terminal window is opened.

**Programs cannot be edited or selected with any programs running. Note that the "Halt" button stops all running programs.**

A similar second program can now be made. This can be done most quickly by copying the first LED1 program using "Copy program..." under the "Program" menu:



The LED2 program is automatically the "SELECTed" program. It can therefore be opened in an edit window using the "Edit" button.



Change the second program LED2 to control a different output channel with different time periods.

The programs can now be run independently. Note that only the SELECTed program can be run or stepped using the "Run" or "Step" buttons. For any others it is necessary to using the red/yellow buttons or to use the menus. It is not necessary (or even useful) to close the editor windows when the programs are running. *Motion Perfect* will automatically set them to "read only" when any programs are running.

To finish this exercise press the "More..." button when the programs are running. A "Full directory" is displayed with a list of all programs and processes.

### 7.1.3 Storage of programs on the MC216/MC204/MC2 controller

All the SERIES 2 *Motion Coordinators* can hold up to 14 programs if memory size permits. The controllers have a number of BASIC commands to allow you to create, manipulate and delete programs. **Motion Perfect provides buttons which makes typing the commands unnecessary and inadvisable** since *Motion Perfect* always updates your project when, for example, copying a program. If a command **is** typed directly users should do a "Check project" under the "File" menu to resolve any differences.

Note: parameters enclosed in marks: <programe> are optional.

SELECT programe	-	Selects a named program for editing/ creates a fresh program
NEW <programe>	-	Deletes selected program or a named program
NEW ALL	-	Deletes all programs
DIR	-	Directory of all programs, their sizes and selected program
COPY programe,programe2	-	Duplicates a program
RENAME oldname,newname	-	Renames a program
DEL <programe>	-	Deletes selected program or a named program
EPROM	-	Makes a copy of all programs into flash eprom
LIST <programe>	-	Lists selected program, or named program to the PC screen

Using the DIR command you can see how the SERIES 2 uses memory space for holding both each entered program called the "source" and a compiled version, the "object". The total user program memory space is 122k bytes. On the MC2 only the program space may be expanded to 500k bytes. On the MC216 the memory space is 500k as standard.

#### **7.1.4 The *Motion Coordinator* SERIES 2 Screen Editor and the *Motion Perfect* Editor**

The SERIES 2 controllers have a built-in simple screen editor allows programs to be edited on a VT100 terminal. **This editor should NOT be used with *Motion Perfect*.** *Motion Perfect* has its own "Windows" style editor which allows for multiple program editing, copying and pasting to/from the clipboard, find/replace and most importantly is linked into your project so that changes are made to both controller and PC copies simultaneously.

The commands of the built-in editor are recorded here for reference only.

Entering the editor:

```
>>EDIT <line sequence number>
```

The line sequence numbers are given in error messages.

Editor Commands:

Del	-	Delete right
Backspace	-	Delete left
Right Arrow	-	Cursor right
Left Arrow	-	Cursor left
Return	-	Creates new line
Control-Y	-	Delete Line

*Type Control-K then D to leave the editor.*

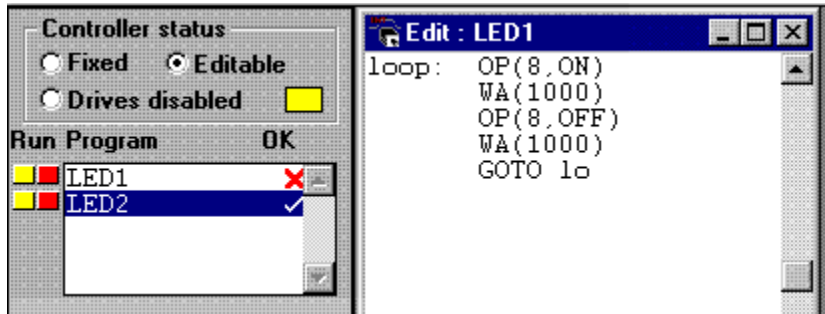


### 7.1.5 Compilation of *Motion Coordinator* SERIES 2 programs

The SERIES 2 system compiles programs automatically when required. It is not normally required to force the controller to compile programs but this can be done under the "Program" menu in *Motion Perfect*. The controller automatically compiles:

- Prior to running after editing the selected program will be compiled automatically
- When another program is selected after editing the original program will be compiled

The program's syntax and structure are checked during compilation. If compilation is unsuccessful a message is issued and no code is generated. A red cross appears in the *Motion Perfect* directory box: (In this example because the label "lo" is missing).



Programs cannot be run when this occurs. The error(s) should be corrected and the program rerun.

The compilation process includes:

- Removing comments
- Numbers are compiled into the processor format
- Expressions are converted into RPN format for execution
- Variable locations are pre-calculated
- Loop structure destinations are calculated and embedded

### 7.1.6 Saving and Loading Programs to and from a PC

Programs on the SERIES 2 controller are held in battery-backed memory or flash EPROM between power-ups. *Motion Perfect* automatically makes a "project" from the programs for an application. Whenever you create or edit a program, *Motion Perfect* edits both the controller copy and the copy on your PC. In this way it is not normally necessary to "save" or "load" programs. *Motion Perfect* checks that the two versions of your project are identical using a CRC (cyclic redundancy check). If the two differ *Motion Perfect* allows you to decide whether to copy the controller version to disk or vice-versa.

Programs are stored on the PC as an ASCII text file. They may therefore be printed, edited and copied using the PC's facilities. The source programs are held on the controller in tokenised format so the size of the source program in memory will be less than the PC file size.

#### 7.1.7 Program Commands exclusive to the MC2, MC216 and MC204 Motion Coordinators:

The MC2 and MC216 have commands for the real time clock:  
Real time clock commands:

DATE, DAY, TIME, DAY\$, DATE\$, TIME\$

The MC204 and MC216 have the CAN commands:

CAN, CANIO\_ADDRESS, CANIO\_STATUS, CANIO\_ENABLE, CAN\_ADDRESS, CAN\_ENABLE

### 7.1.8 Local and Global Variables and Labels

The SERIES 2 programs have their own local labels and local variables. For example the two programs:

```
Start:  FOR a=1 to 100
        MOVE(a)
        WAIT IDLE
        NEXT a
        GOTO Start
```

```
Start:  REPEAT
        a=a+1
        PRINT a
        UNTIL a=300
        GOTO Start
```

These two program when run at the same time each have their own version of the variable "a" and the label "Start". If you need to hold data in common between two or more programs the fixed variable array VR() should be used or alternatively if a large amount of data is to be held the TABLE() function can be used.

Note that the VR() and TABLE() values are battery-backed as well as being global. They are therefore NOT cleared to zero on power-up.

To make programs more readable when using the VR() array a named local variable can be used as a constant offset into the array of each program. Note however that the constant needs to be declared in each program using the variable. In the example below VR(3) is used to hold a length parameter:

```
Start:  GOSUB Initial
        VR(length)=x

        ...Body of program

Initial: length=3
        RETURN
```

```
Start:  GOSUB Initial
        MOVE(VR(length))
        PRINT VR(length)

        ...Body of program

Initial: length=3
        RETURN
```

### 7.1.9 Table Values on the Motion Coordinator SERIES 2

Table values can be written to and read from with the TABLE command. See chapter 8 for details. The TABLE is held in memory as a special class of program. The TABLE program cannot be edited, listed, copied or renamed but it can be deleted:

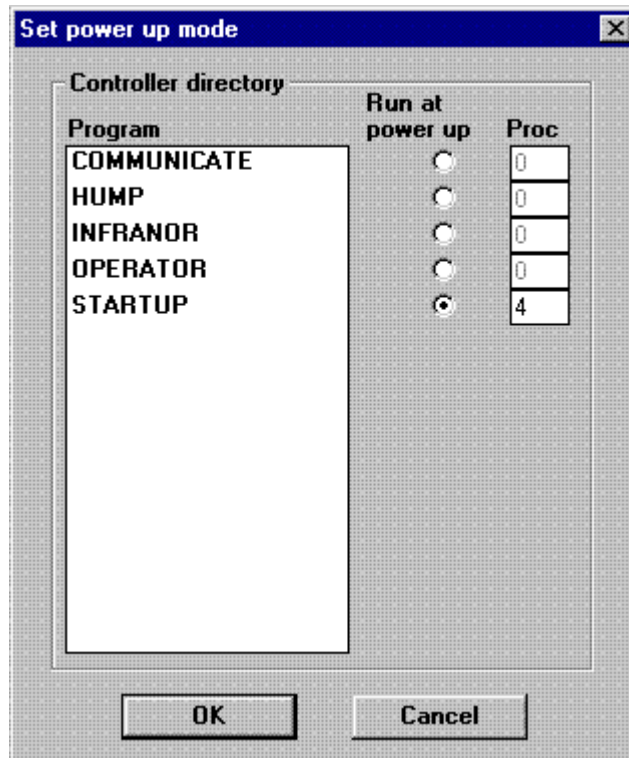
```
DEL "TABLE"
```

*Note that the name "TABLE" must be in quotes in this special case.*

The maximum table size on the SERIES 2 controllers is 16000 entries. The Trio BASIC will not allow values outside the highest value that has been written to to be read.

**7.1.10 Setting Motion Coordinator 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 (See Chapter 8): For example:

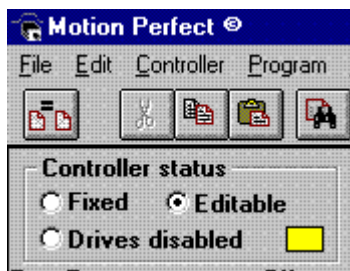


Typically only ONE program is set to run on power up. This program can then start the others under program control:

```

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

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



### 7.1.11 Forcing priority of program execution

The *Motion Coordinator* SERIES 2 controllers will allocate programs to processes automatically to make the system easier to use. This will be sufficient for many applications especially when there are less than 4 programs.

For more complex applications it can be useful to allocate priorities to programs. The SERIES 2 controllers have a default servo period of 1 mSec. This period is internally divided into 3 subdivision time slots which default to 1/3 mSec each. Two of the processes (number 14 and number 13) get a fixed time slot allocated to them (if they are present). These are called the "fast" tasks. They should be used for jobs which require:

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

The other processes (including the command line) share the third time slot. Execution speed will hence slow down as the number of programs running increases. However very useful execution speed is still obtained. Processes 1..12 are "slow" tasks. Programs can be forced to run on a process using the commands RUNTYPE or RUN:

```
>>RUNTYPE "progname",ON,7  
>>RUN "progname",7
```

*If equal time is required to be given to all programs, processes 14 and 13 should NOT be used. The time available is automatically evenly divided between the remaining programs.*

The MC204 has a maximum of 5 programs running. Processes 5 and 4 are allocated 1/3 of the servo period each and are equivalent to numbers 14 and 13 on the MC2/MC216. Processes 1, 2 & 3 and the command line share the remaining third. These programs and the command line use the available time slot with equal priority.

### 7.1.12 Command Line on *Motion Coordinator* SERIES 2 Controllers

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. Note that the command line of the SERIES 2 features a buffer of the last 10 commands. 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* !

### 7.1.13 Typing Commands for Immediate Execution

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

Example:

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

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

### 7.1.14 Simple Example BASIC Programs

```

'
' Example Program 1:
'
start:    TICKS=0
          PRINT "Press a key"
          WAIT UNTIL KEY
          GET k
          PRINT "It is ";-TICKS/1000;" seconds since you were asked to press a key"
          GOTO start

'
' Example Program 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 BASIC program, not the motion. The first line is a comment. It has no effect on the program execution.

```

'
' Example Program 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

```

## **7.2 AXIS, SYSTEM and PROCESS Parameters**

### **7.2.1 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 32 bit integer. The parameters are all set to default values on every power up.

Parameters are read from and written to like variables. The BASIC 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
>>? MERGE AXIS(3)
```

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

### **7.2.2 System Parameters**

Trio BASIC 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.

### **7.2.3 Process Parameters**

Trio BASIC also holds a small number of parameters which are held separately for each PROCESS. The process assumed is the current process the command is using.

```
TICKS
PROCNUMBER
PMOVE
ERROR_LINE
INDEVICE
OUTDEVICE
BASE
```

## Trio BASIC Programming - Command Reference

Commands marked in *italics* are provided for compatibility with older Trio controllers, but are not advised for new applications.

Commands marked:     **[MP]** are generally only used by Motion Perfect for its operations  
                               **[MC2]** are only available on MC2 controllers  
                               **[MC204/216]** are only available on MC204/MC216 controllers

### 1 Motion Commands

ACC	Set acceleration rate (use ACCEL)	-10
ACCEL	Read/set acceleration rate	-11
ADDAX	Superimpose 2 or more motion profiles	-11
BASE	Choose axes for commands	-18
CAM	CAM profile movement	-20
CAMBOX	CAM profile movement linked to position	-22
CANCEL	Cancel a move	-26
CONNECT	Gearbox connection	-29
DATUM	Datum axis	-33
DEC	Set deceleration rate (use DECEL)	-34
FORWARD	Continuous forward move	-44
MHELICAL	Helical move	-59
MOVE	Incremental move	-61
MOVEABS	Absolute position move	-62
MOVECIRC	Circular arc move	-63
MOVELINK	Move linked to position	-65
MOVEMODIFY	Move with modifiable end	-68
RAPIDSTOP	Cancel on all axes	-81
REVERSE	Continuous reverse move	-85
SP	Set speed (use SPEED)	-92

### 2 Input/Output Commands

AIN	Read analog input[MC2]	-13
CURSOR	Cursor Position	-30
FLAG	Set/read single PLC flag	-42
FLAGS	Set/read all of PLC flags	-42
GET	Get character from serial port	-47
IN	Read digital inputs	-51
INPUT	Get number from serial port	-52
KEY	Key pressed flag	-54
LINPUT	Get string from serial port	-55
OP	Control digital outputs	-73
PRINT	print to serial channel	-77
PSWITCH	Set output in position sector	-79
READPACKET	Read transmitted variables	-81
RECORD	Record registration transitions	-81
REGIST	Setup registration capture	-83
SEND	Send fibre network message	-89
SETCOM	Set serial port baudrate etc	-91

### 3 Program Loop and Structures

ELSE	Part of IF structure	-38
ENDIF	Terminates multi-line IF	-39
FOR	Starts FOR..NEXT loop	-43
GOSUB	Execute subroutine	-48
GOTO	Branch to line in program	-49
IF	Starts conditional test	-50
NEXT	Terminates FOR..NEXT loop	-71
ON...	Multiple GOTO/GOSUB via variable	-72
REPEAT	Start REPEAT UNTIL loop	-84
RETURN	Return from subroutine	-85
STEP	Set FOR..NEXT loop size	-93
THEN	Part of IF structure	-96
TO	Part of FOR..NEXT structure	-96
UNTIL	Ends REPEAT-UNTIL structure	-99
WA	Wait for time	-102
WAIT IDLE	Wait for move to finish	-102
WAIT LOADED	Wait for move to start	-102
WAIT UNTIL	Wait for condition	-103
WEND	Terminates WHILE..WEND	-103
WHILE	Starts WHILE..WEND	-104

### 4 Program and System Parameters and Functions

ADDRESS	RS485 Address	-12
AIN0..3/AINBIO..3	Analog inputs as parameter	-14
APPENDPROG	Used by Motion Perfect Editor [MP]	-15
AUTORUN	Runs programs set to run by RUNTYPE	-16
AXIS	Allocates a single command to an axis	-16
AXISVALUES	[MP] uses to read axis parameters	-17
BASICERROR	Set routine to run if BASIC error	-19
CAN	Control CAN network channel [MC204/MC216]	-25
CANIO_ADDRESS	CAN I/O net address [MC204/216]	-26
CANIO_ENABLE	CAN I/O status	-27
CANIO_STATUS	CAN I/O status	-27
CHECKSUM	Battery backed RAM checksum	-27
CLEAR	Clear global variables to zero	-27
COMMSERROR	Communications error parameter	-28
COMPILE	Forces compilation of a program	-28
CONTROL	Controller type	-29
COPY	Copy a program	-29
DATE/DATE\$	Returns date [MC2/MC216]	-32
DAY/DAY\$	Returns day of week	-34
DEL	Delete a program	-36
DIR	Directory of programs	-37
EDIT	Screen edit without Motion Perfect	-38
EDPROG	Used by Motion Perfect Editor[MP]	-38
EPROM	Transfer program to flash eprom	-39
ERROR_AXIS	Axis on which first motion error	-39
ERROR_LINE	Line where BASIC error occurred	-39
EX	Software reset	-40
FRAME	Coordinate transformation frame num	-44
FREE	Returns free memory	-45
HALT	Halts all programs	-49
INDEVICE	Set default serial input device	-51
INITIALISE	Resets all axis params to defaults	-52



INPUTS0/INPUTS1	First 32 inputs as parameter	-52
JOINT	Returns transform config	-54
LAST_AXIS	Highest axis in use	-54
LIST	List a program	-55
LOADSYSTEM	Load system software	-56
LOCK	Hide programs	-57
MATCH	Compare transition pattern	-57
MOTION_ERROR	Motion error indicator	-60
MPE	Motion Perfect mode control	-68
NAIO	Number analog input channels[MC204/216]	-70
NETSTAT	Network status	-70
NEW	Deletes program	-71
NIO	Number input/output channels	-71
OUTDEVICE	Set default serial output	-74
PMOVE	Process move buffer loaded flag	-75
POWER_UP	Sets program to run from EPROM	-76
PROC	Redirect command to a process	-78
PROCESS	List process status	-78
PROCMOVE	Returns axes with moves pending	-78
PROCNUMBER	Returns process to a program	-79
RENAME	Rename a program	-84
RESET	Clear local variables	-84
RUN	Run a program	-86
RUN_ERROR	Number of last BASIC error	-86
RUNTYPE	Set program to run on power up	-87
SCOPE	Store parameters into memory	-88
SCOPE_POS	Scope position in data	-88
SELECT	Select program to edit	-89
SERVO_PERIOD	Set controller servo period	-90
STEPLINE	Single step program	-93
STOP	Stop program execution	-93
STORE	Store system software	-94
TABLEVALUES	Used by [MP] to read table	-95
TICKS	Clock ticks for process	-96
TIME	Returns time [MC2/MC216]	-96
TRIGGER	Trigger SCOPE from program	-97
TROFF	Trace on	-97
TRON	Trace on	-98
TSIZE	Returns highest TABLE loaded	-98
VERSION	System software version number	-100
VIEW	Lists programs incl.tokens	-100
WDOG	Control amplifier enable relay	-103

## 5 Mathematical Functions and Commands

*	Multiply	-7
+	Add	-7
-	Subtract	-7
/	Divide	-8
<	Is less than	-9
<=	Is less than or equal to	-9
<>	Is not equal to	-10
=	Assignment/Equals comparison	-8
>	Is greater than	-8
>=	Is greater than or equal to	-9
ABS	Returns the magnitude of a number	-10
ACOS	Arccos	-11

AND	Logical and bitwise AND	-14
ASIN	Arcsin	-15
ATAN	Arctangent	-15
ATAN2	ATAN(x/y)	-15
COS	Cosine	-30
EXP	Exponential Function	-40
FRAC	Fractional part function	-44
INT	Returns the integer part of a number	-53
LN	Natural log function	-55
MOD	Modulus function	-60
NOT	Logical and bitwise NOT	-71
OR	Logical and bitwise OR	-74
SGN	Returns the sign of a number	-91
SIN	Sine of an angle	-91
SQR	Returns the square root	-92
TABLE	Access data table	-94
TAN	Tangent of an angle	-95
VR	Battery backed global variable	-101
XOR	Logical and bitwise exclusive OR	-104

## 6 Constants

FALSE	0	-40
OFF	Logical off/0	-71
ON	Logical on/1	-72
PI	3.14159	-75
TRUE	-1	-98

## 7 Axis Parameters

AFF_GAIN	Accel feedforward gain	-12
ATYPE	Axis type	-16
AXISSTATUS	Axis status bits	-17
BOOST	Stepper amplifier boost control	-19
CAN_ADDRESS	Remote drive CAN address	-26
CAN_ENABLE	Remote drive enable control	-26
CLOSE_WIN	Close register window	-28
CREEP	Creep speed	-30
D_GAIN	Derivative gain	-37
DAC	Setting for DAC output when servo off	-31
DAC_OUT	Current output to voltage output DAC	-31
DATUM_IN	Define datum input	-34
DECEL	Read/set deceleration	-35
DEFPOS	Set absolute value of current position	-36
DEMAND_EDGES	Demand position in edge units	-37
DPOS	Demand position	-37
ENCODER	Raw encoder register	-39
ENDMOVE	Position of end of move	-39
ERRORMASK	Status mask for errors	-40
FAST_JOG	Fast jog input	-40
FASTDEC	Fast Deceleration	-41
FE	Following error	-41
FE_LIMIT	Following error limit	-41
FEGRAD	Following error limit gradient	-41
FEMIN	Following error limit at zero speed	-41
FERANGE	Following error report range	-41

FH_SPEED	Feedhold speed	-42
FHOLD_IN	Feedhold input	-42
FS_LIMIT	Forward software limit	-46
FWD_IN	Forward limit input	-47
FWD_JOG	Forward jog input	-47
I_GAIN	Integral gain	-53
INVERT_STEP	Invert active step edge	-53
JOGSPEED	Slow jogging speed	-53
LINKAX	Link axis for gearbox etc	-54
MARK	Test if registration event	-57
MERGE	Merge movements	-58
MICROSTEP	Microstep movement control	-59
MPOS	Measured position	-69
MSPEED	Measured speed	-69
MTYPE	Move type	-70
NTYPE	Buffer move type	-71
OFFPOS	Offset axis position	-72
OPEN_WIN	Open register window	-73
OUTLIMIT	Voltage output limit	-75
OV_GAIN	Output velocity gain	-75
P_GAIN	Proportional gain	-81
PP_STEP	Encoder ratio scaling	-76
REG_MATCH	Match factor for RECORD sequence	-82
REG_POS	Registration position	-82
REMAIN	Remainder of move	-83
REP_OPTION	Set repeat mode	-84
REPDIST	Repeat distance	-84
REV_IN	Reverse limit input	-85
REV_JOG	Reverse jog input	-85
RS_LIMIT	Reverse software limit	-86
SERVO	Position loop on/off	-90
SPEED	Demand speed	-92
SRAMP	S ramp factor	-92
SSI_BITS	Number of bits in SSI absolute enc.	-92
TRANS_DPOS	Transformed demand position	-97
TRANSITIONS	Number of transitions	-97
UNITS	Unit conversion factor	-99
VERIFY	Step verify mode control	-99
VFF_GAIN	Velocity feedforward gain	-100
VP_SPEED	Velocity profile speed	-100



## Command and Function Description in Alphabetical Order

### Expression1 + expression2

Type: Arithmetic operation

Description: Adds two expressions

Parameters:

*Expression1*: Any valid TRIO BASIC expression

*Expression2*: Any valid TRIO BASIC expression

Example: `result=10+(2.1*9)`

The BASIC evaluates the parentheses first giving the value 18.9 and then adds the two expressions.

Therefore `result` holds the value 28.9

### Expression1 - expression2

Type: Arithmetic operation

Description: Subtracts expression2 from expression1

Parameters:

*Expression1*: Any valid TRIO BASIC expression

*Expression2*: Any valid TRIO BASIC expression

Example: `VR(0)=10-(2.1*9)`

The BASIC evaluates the parentheses first giving the value 18.9 and then subtracts this from 10.

Therefore `VR(0)` holds the value -8.9

### Expression1 \* expression2

Type: Arithmetic operation

Description: Multiplies expression1 by expression2

Parameters:

*Expression1*: Any valid TRIO BASIC expression

*Expression2*: Any valid TRIO BASIC expression

Example: `factor=10*(2.1+9)`

The BASIC evaluates the brackets first giving the value 11.1 and then multiplies this by 10.

Therefore `factor` holds the value 111

**Expression1 / expression2**

Type: Arithmetic operation

Description: Divides expression1 by expression2

Parameters:

*Expression1:* Any valid TRIO BASIC expression

*Expression2:* Any valid TRIO BASIC expression

Example: `a=10/(2.1+9)`

The BASIC evaluates the parentheses first giving the value 11.1 and then divides 10 by this number

Therefore a holds the value .9009

**Expression1 = expression2**

Type: Arithmetic and logical operation

Description: Returns TRUE if expression1 is equal to expression2, otherwise returns false.

Note: TRUE is defined as -1, and FALSE as 0

Parameters:

*Expression1:* Any valid TRIO BASIC expression

*Expression2:* Any valid TRIO BASIC expression

Example: `IF IN(7)=ON THEN GOTO label`

If input 7 is ON then BASIC execution will continue at line starting "label:"

**Expression1 > expression2**

Type: Logical operation

Description: Returns TRUE if expression1 is greater than expression2, otherwise returns false.

Note: TRUE is defined as -1, and FALSE as 0

Parameters:

*Expression1:* Any valid TRIO BASIC expression

*Expression2:* Any valid TRIO BASIC expression

Example 1: `VR(0)=1>0`

1 is greater than 0 and therefore VR(0) holds the value -1

Example 2: `WAIT UNTIL MPOS>200`

The BASIC program will wait until the measured position is greater than 200

### Expression1 >= expression2

Type:	Logical operation
Description:	Returns TRUE if expression1 is greater than or equal to expression2, otherwise returns false.
Note:	TRUE is defined as -1, and FALSE as 0
Parameters:	
<i>Expression1:</i>	Any valid TRIO BASIC expression
<i>Expression2:</i>	Any valid TRIO BASIC expression
Example:	<pre>IF target&gt;=120 THEN MOVEABS(0)</pre> <p>If variable target holds a value greater than or equal to 120 then move to the absolute position of 0.</p>

### Expression1 < expression2

Type:	Logical operation
Description:	Returns TRUE if expression1 is less than expression2, otherwise returns false.
Note:	TRUE is defined as -1, and FALSE as 0
Parameters:	
<i>Expression1:</i>	Any valid TRIO BASIC expression
<i>Expression2:</i>	Any valid TRIO BASIC expression
Example:	<pre>IF AIN(1)&lt;10 THEN GOSUB rollup</pre> <p>If the value returned from analog input 1 is less than 10 then execute sub-routine "rollup"</p>

### Expression1 <= expression2

Type:	Logical operation
Description:	Returns TRUE if expression1 is less than or equal to expression2, otherwise returns false.
Note:	TRUE is defined as -1, and FALSE as 0
Parameters:	
<i>Expression1:</i>	Any valid TRIO BASIC expression
<i>Expression2:</i>	Any valid TRIO BASIC expression
Example:	<pre>maybe=1&lt;=0</pre> <p>1 is not less than or equal to 0 and therefore variable <b>maybe</b> holds the value 0</p>

**Expression1 <> expression2**

Type:	Logical operator
Description:	Returns TRUE if expression1 is not equal to expression2, otherwise returns false.
Note:	TRUE is defined as -1, and FALSE as 0
Parameters:	
<i>Expression1:</i>	Any valid TRIO BASIC expression
<i>Expression2:</i>	Any valid TRIO BASIC expression
Example:	<pre>IF MTYPE&lt;&gt;0 THEN GOTO scoop</pre> <p>If axis is not idle (MTYPE=0 indicates axis idle) then goto label "scoop"</p>

**ABS(expression)**

Type:	Function
Description:	The ABS function converts a negative number into its positive equal. Positive numbers are unaltered.
Parameters:	
<i>Expression:</i>	Any valid TRIO BASIC expression
Example:	<pre>IF ABS(AIN(0))&gt;100 THEN PRINT "Analog Input Outside +/-100"</pre>

**ACC(rate of acc)**

Type:	Command
Note:	This command is provided to aid compatibility with older Trio controllers. Acceleration rate and deceleration rate are recommended to be set with the ACCEL and DECEL axis parameters.
Description:	Sets both the acceleration and deceleration rate simultaneously
Parameters:	
<i>rate of acc:</i>	The units of the parameter are dependant on the UNITS axis parameter. The acceleration factor is entered in UNITS/SEC/SEC. Therefore if, for example, the unit conversion factor is set to the number of encoder edges in 1 <b>rev</b> and the ACC() command is issued ACC(10) it will take 2 seconds to accelerate to 20 <b>revs/sec</b> .
Example:	<pre>ACC(100)</pre>



## 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: 

```
ACCEL=130:' Set acceleration rate
PRINT " Acceleration rate is ";ACCEL;" mm/sec/sec"
```

## ACOS(expression)

Type: Function

Description: The ACOS 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 0..PI

Parameters:

*Expression:* Any valid TRIO BASIC expression.

Example: 

```
>>PRINT ACOS(-1)
3.1416
>>
```

## ADDAX(axis)

Type: Command

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 axis to which the command is issued. The specified axis can be any axis and does not have to physically exist in the system. 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. To break the link an ADDAX(-1) command is issued.

The ADDAX command therefore allows an axis to perform the moves specified on TWO axes added together. When the axis parameter SERVO is set to OFF on an axis with an encoder interface the measured position MPOS is copied into the demand position DPOS. This allows ADDAX to be used to sum encoder inputs.

Parameter:

*axis:* Axis to sum changes from

Note: The ADDAX command sums the movements in encoder edge units.

Example 1: 

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

'   Axis 0 will move 1*1000+2*20 = 1040 edges
```

Example 2: Pieces are placed onto a continuously moving belt and further along the line are picked up. A detection system gives an indication as to whether a piece is in front of or behind its nominal position, and how far.

```
FORWARD AXIS(0)'           set continuous move
ADDAX(2)
REPEAT
    GOSUB getoffset'       Get offset to apply
    MOVE(offset) AXIS(2)
UNTIL IN(2)=ON
```

Axis 0 in this example executes a continuous FORWARD and a superimposed MOVE on axis 2 is used to to apply offsets

Example 3: A CAMBOX movement is linked to an encoder input. In order to achieve registration an gradual offset is required to be applied *as if the link encoder is being moved*. This can be achieved by linking the CAMBOX to an unused imaginary axis on the controller. The encoder position is added to the imaginary axis with ADDAX. Offset moves to achieve the registration are run on the imaginary axis:

```
' Axis 0 runs CAMBOX moves
' Axis 1 has encoder daughter board
' Axis 2 is "imaginary" axis

SERVO AXIS(1)=OFF
ADDAX(1) AXIS(2)
...
CAMBOX(1000,1100,4,600,2) AXIS(0)
```

## ADDRESS

Type: System Parameter

Description: Sets the RS485 multi-drop address for the board. This parameter should be in the range of 1..32

Example: ADDRESS=5

## AFF\_GAIN

Type: Axis Parameter

Description: This axis parameter is a reserved keyword, but is not currently implemented.

## AIN(analog chan)

Type: Function

Description: MC2: If the MC2 has the analogue input option fitted this function will return the value of the analog input specified. The value returned is a decimal representation of the voltage input and is in the range 0 to 4095 corresponding to voltage inputs in the range 0V to 4.096V. On the MC216/MC204 analog input modules (P325) are connected on the CAN bus. The P325 has 8 channels of +/-10v analog inputs which return 2047..-2048.

Parameters:

*analog chan*: Analogue input channel number 0..3 on MC2, 0..31 on MC204/216( But see note )

Example:

The speed of a production line is to be governed by the rate at which material is fed onto it. The material feed is via a lazy loop arrangement which is fitted with an ultra-sonic height sensing device. The output of the ultra-sonic sensor is in the range 0V to 4V where the output is at 4V when the loop is at its longest.

```
MOVE(-5000)
REPEAT
    a=AIN(1)
    IF a<0 THEN a=0
    SPEED=a*0.25
UNTIL MTYPE=0
```

Note that 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.

Note (MC2 only) The analog input channels are implemented with a MAX186 integrated circuit. This chip can be programmed for +/- 2.048 v input, or may be set for single ended analog inputs. Channel values above 127 are passed directly as command codes to the chip and the reply returned by the AIN() function. For a full list to the functions of the MAX186 the chip data sheet should be consulted. Some useful codes are:

```
AIN(131) reads channel 0 as +/- 2.048 volts
AIN(147) reads channel 1 as +/- 2.048 volts
AIN(163) reads channel 2 as +/- 2.048 volts
AIN(179) reads channel 3 as +/- 2.048 volts
```

**AIN0/AIN1/AIN2/AIN3/AINBI0/AINBI1/AINBI2/AINBI3**

- Type: Analog Inputs as System Parameter
- Description: On an MC2 with the analog input option fitted this function will return the value of the analogue input channel 0 to channel 3. The value returned is a decimal representation of the voltage input and is in the range 0 to 4095 corresponding to voltage inputs in the range 0V to 4.096V. With the alternative forms AINBI0..AINBI3 The value returned is a decimal representation of the voltage input and is in the range -2048 to 2047 corresponding to voltage inputs in the range -2.048V to 2.047V. On the MC204/MC216 the system parameters return analog input channels 0..7.
- Note: These system parameters duplicate the AIN() command. They are provided in system parameter format to allow the SCOPE function (Which can only store parameters) to read the analog inputs.

**Expression1 AND expression2**

- Type: Logical and bitwise operator
- Description: This performs an AND function between corresponding bits of the integer part of two valid TRIO BASIC expressions.

The AND function between two bits is defined as follows:

AND	0	1
0	0	0
1	0	1

Parameters:

*Expression1:* Any valid TRIO BASIC expression

*Expression2:* Any valid TRIO BASIC expression

Example 1: IF (IN(6)=ON) AND (DPOS>100) THEN tap=ON

Example 2: VR(0)=10 AND (2.1\*9)

The basic 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

The AND is a bitwise operator and so the binary action taking place is:

```

      01010
AND  10010
-----
      00010
    
```

Therefore VR(0) holds the value 2

Example 3: IF MPOS AXIS(0)>0 AND MPOS AXIS(1)>0 THEN GOTO cycle1

### APPENDPROG [string]

Type: System Command (This function is used by the Motion Perfect editor)

Description: This command appends a line to the currently selected program.

Parameters:

*string*: The text, enclosed in quotation marks, that is to be appended to the program

### ASIN(expression)

Type: Function

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 (Numbers outside the +/-1 input range will return zero)

Parameters:

*Expression*: Any valid TRIO BASIC expression.

Example: 

```
>>PRINT ASIN(-1)
-1.5708
>>
```

### ATAN(expression)

Type: Function

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:

*Expression*: Any valid TRIO BASIC expression.

Example: 

```
>>PRINT ATAN(1)
0.7854
>>
```

### ATAN2(expression1,expression 2)

Type: Function

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 TRIO BASIC expression.

Example: 

```
>>PRINT ATAN2(0,1)
0.0000
>>
```

## ATYPE

Type: Axis Parameter

Description: The ATYPE axis parameter may be used to test the type of each axis fitted. It will return the values:

0	-	No axis daughter board fitted
1	-	Stepper daughter board
2	-	Servo daughter board
3	-	Encoder daughter board
4	-	Stepper daughter with position verification/Differential Stepper
5	-	Resolver daughter board
6	-	Voltage output daughter board
7	-	Absolute SSI Servo daughter board
8	-	CAN daughter board
9	-	REMOTE CAN axis
10	-	PSWITCH daughter board

The ATYPE axis parameter is set by the system software at power up and is not normally written to.

Example: 

```
>>PRINT ATYPE AXIS(2)
1.0000
>>
```

This would show that an stepper daughter board is fitted in this axis slot.

## AUTORUN

Type: System Command

Description: Starts running all the programs that have been set to run at power up. See RUNTYPE.

## AXIS(expression)

Type: Function

Description: Assigns ONE command or axis parameter read or assignment to a particular axis.

Note: If it is required to change every subsequent command the BASE command should be used instead.

Parameters:

*Expression:* Any valid TRIO BASIC expression. The expression should be an axis number.

Example 1: 

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

Example 2: 

```
MOVE(300) AXIS(2)
```

Example 3: 

```
REPDIST AXIS(3)=100
```

The AXIS command may be used to modify any axis parameter expression and the axis dependent commands: ACC, ADDAX, CANCEL, CAM, CAMBOX, CONNECT, DATUM, DEC, DEFPOS, FORWARD, MOVEABS, MOVECIRC, MHELICAL, MOVELINK, MOVE, MOVEMODIFY, REGIST, REVERSE, SP, WAIT IDLE, WAIT LOADED

## AXISSTATUS

Type: Axis Parameter (READ ONLY)

Description: The AXISSTATUS axis parameter may be used to check various status bits held for each axis fitted:

Bit Number:	Description:	Value:
0	Unused	1
1	Following error warning range	2
2	Unused	4
3	Unused	8
4	In forward limit	16
5	In reverse limit	32
6	Datuming	64
7	Feedhold	128
8	Following error exceeds limit	256
9	In forward software limit	512
10	In reverse software limit	1024
11	Cancelling move	2048

The AXISSTATUS axis parameter is set by the system software and should not be written to.

Example: `IF (AXISSTATUS AND 16)>0 THEN PRINT "In forward limit"`

## AXISVALUES(axis,bank)

Type: Command (Used by Motion Perfect to read axis parameters)

Description : Reads banks of axis parameters. There are 2 banks of parameters for each axis, bank 0 displays the data that is only changed by the TRIO BASIC, bank 1 displays the data that is changed by the motion generator. The data is given in the format:

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

where:

`<Parameter>` is the name of the parameter

`<type>` is the type of the value. The types are:

i integer

f float

u float that when changed means that the bank 0 data must be updated

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

**BASE(axis no<,second axis><,third axis>...)**

Type: Command

Description: The BASE command is used to direct subsequent motion commands and axis parameter read/writes to a particular axis, or group of axes. The default setting is a sequence: zero,one,two...

**Each BASIC process has its own BASE group of axes and each program can set values independently.**

The BASIC program is separate from the MOTION GENERATOR program which controls motion in the axes. The motion generator has separate functions for each axis, so each axis is capable of being programmed with its own speed, acceleration, etc and moving independently and simultaneously OR they can be linked together by interpolation or linked moves.

The AXIS() command also redirects commands to different axes but applies to just the single command it precedes, and to a single axis. The BASE() command redirects all subsequent commands unless they are specified with AXIS.

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.

Example 1:

```

BASE(1)
UNITS=2000'      Set unit conversion factor axis 1
SPEED=100'       Set speed axis 1
ACCEL=5000'      Set acceleration rate axis 1
BASE(2)
UNITS=2000'      Set unit conversion factor axis 2
SPEED=125'       Set speed axis 2
ACCEL=10000'     Set acceleration rate axis 2
    
```

Example 2:

```

BASE(0,4,6)
MOVE(100,-23.1,1250)
    
```

Note: In example 2 axis 0 will move 100 units, axis 4 will move -23.1 and axis 6 will move 1250 units. The axes will move along the resultant path at the speed and acceleration set for axis 0.

Note 2: The BASE command sets an internal array of axes held for each process. The default array for each process is 0,1,2..up to the number of controller axes. If the BASE command does not specify all the axes, the BASE command will "fill in" the remaining values automatically. Firstly it will fill in any remaining axes above the last declared value, then it will fill in any remaining axes in sequence:

Example 3:

```

'      Set BASE array on a 16 axis MC216 controller
BASE(2,6,10)
    
```

This will set the internal array of 16 axes to:  
**2,6,10,11,12,13,14,15,0,1,3,4,5,7,8,9**



Note 3: On the command line process ONLY the BASE array may be seen by typing:

```
>>BASE
(0,2,3,1)
>>
```

This example is from an MC204 with 4 axes

### **BASICERROR**

Type: Command

Description: This command may only be used as part of an ON ... GOSUB or ON ... GOTO command. When used in this context it defines a routine to be run when an error occurs in a BASIC command.

Example:

```
' if an error occurs in a BASIC command then
' run the error routine

ON BASICERROR GOTO error_routine

....(rest of program)

error_routine:
    PRINT "The error ";RUN_ERROR[0];
    PRINT " occurred in line ";ERROR_LINE[0]
    STOP
```

### **BOOST**

Type: Axis Parameter

Description: Sets the boost output on a stepper daughter board. The boost output is a dedicated open collector output on the stepper and stepper encoder daughter boards. The open collector can be switched on or off for each axis using this command.

Example: BOOST AXIS(11)=ON

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

Type: Command

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 2 to 16000. The controller interpolates between the values in the table to allow small numbers of points to define a smooth profile.

Parameters:

*start point:* The cam table may be used to hold several profiles and/or other information. To allow freedom of use each command specifies where to start in the table.

*end point:* Specifies end of values in table. Note that 2 or more CAM() commands executing simultaneously can use the same values in the table.

*table multiplier:* The table values are absolute positions from the start of the motion and are normally specified in encoder edges. The table multiplier may be set to any value to scale the values in the table.

*distance:* The distance factor controls the speed of movement through the table. The time taken to execute the CAM() command is dependent on the current axis SPEED and this distance (which is in user units). Say for example the system is being programmed in mm and the speed is set to 10mm/sec. If a *distance* of 100mm is specified the CAM command will take 10 seconds to execute. The speed may be changed at any time to any value as with other motion commands. 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.**

Example: 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:

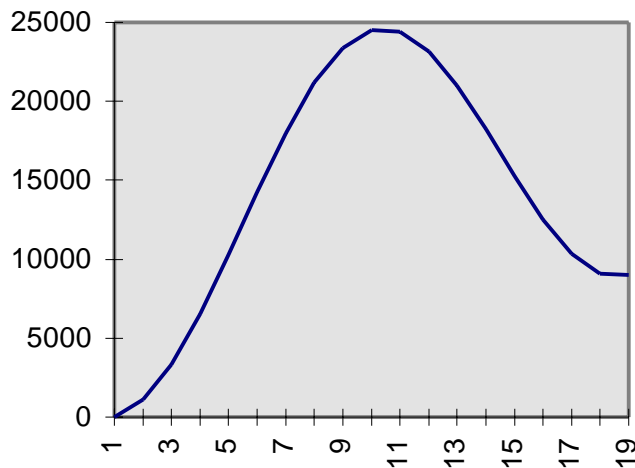
```
GOSUB camtable

loop:  CAM (1,19,1,200)
      GOTO loop
```

Note: The subroutine `camtable` loads the data into the CAM TABLE.

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

Note2: When the CAM command is executing the ENDMOVE parameter is set to the end of the PREVIOUS move



**CAMBOX(start point, end point, table multiplier, link distance , link axis<,link options><, link pos>)**

- Type: Command
- 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 2 to 16000. The controller interpolates between the values in the table to allow small numbers of points to define a smooth profile.
- Parameters:
- start point:* The cam table may be used to hold several profiles and/or other information. To allow freedom of use each command specifies where to start in the table.
- end point:* Specifies end of values in table. Note that 2 or more CAMBOX commands executing simultaneously can use the same values in the table.
- table multiplier:* The table values are absolute positions from the start of the motion and are **specified in encoder edges units**. The table multiplier may be set to any value to scale the values in the table.
- link distance:* The link distance specifies the distance the link axis must move to complete the specified output movement. **The link distance is in the user units of the link axis** and should always be specified as a positive distance.
- link axis:* This parameter specifies the axis to link to. It should be set to 0..3 (MC204) , 0..11 (MC2) 0..15 (MC216)
- link options:* Settings:
- 1 -link commences exactly when registration event occurs on link axis
  - 2 -link commences at an absolute position on link axis (see param 7)
  - 4 -CAMBOX repeats automatically and bi-directionally when this bit is set.
- (This mode can be cleared by setting bit 1 of the REP\_OPTION axis parameter)
- link pos:* This parameter is the absolute position where the CAMBOX link is to be started when parameter 6 is set to 2.
- Note: 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.

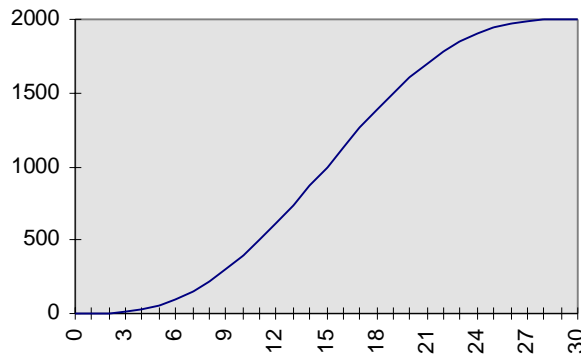
***Parameters 6 and 7; link options and link pos, are optional.***

Example 1:

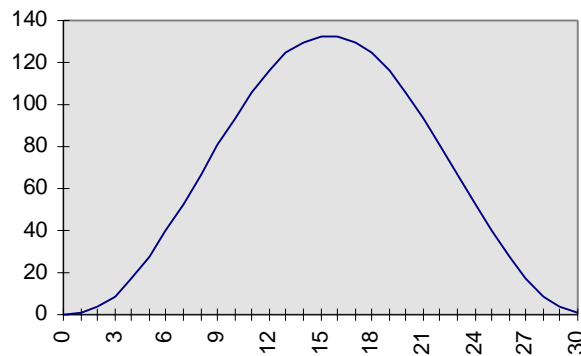
```

num_p=30
scale=2000
'
' Subroutine to generate a SIN shape speed profile
'
' Uses: p is loop counter
'       num_p is number of points stored in tables positions
0..num_p
'       scale is distance travelled scale factor
'
FOR p=0 TO num_p
    TABLE(p,((-SIN(PI*2*p/num_p))/(PI*2))+p/num_p)*scale)
NEXT p

```



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 rotating drum feeding labels is activated when a product conveyor reaches a position held in the variable "start". This example uses the table points 0..30 generated in Example 1:

```

CAMBOX(0,30,800,80,15,2,start)

```

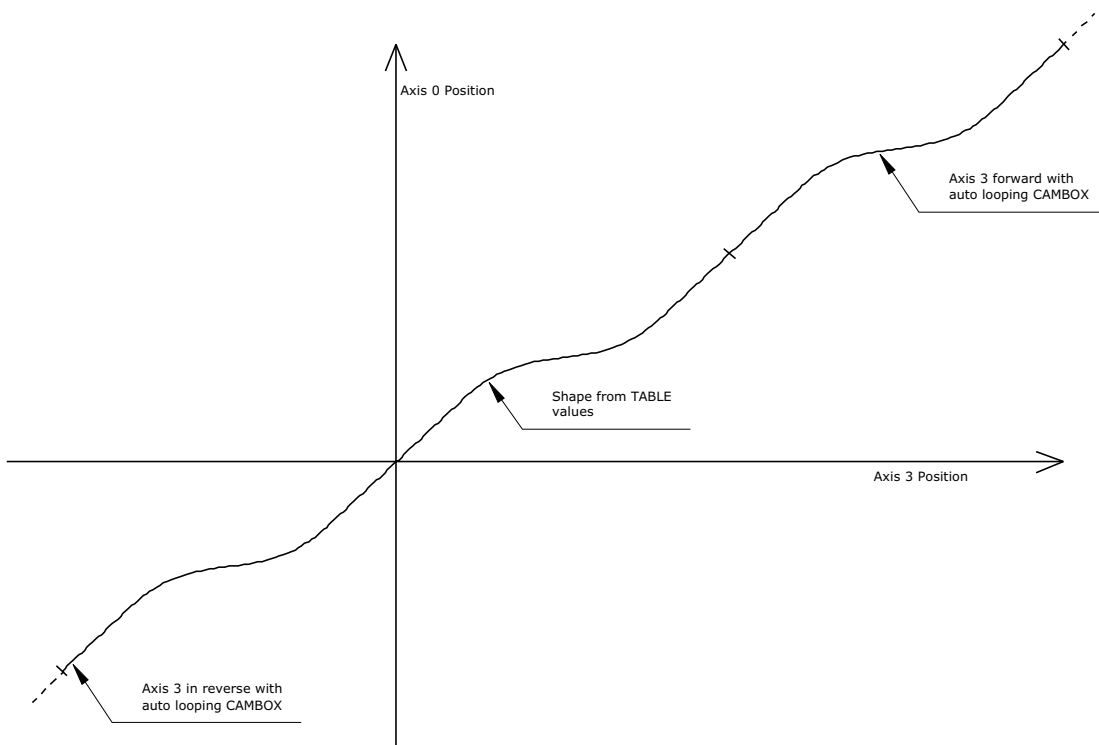
Note:

Parameter 1 is 0	The start of the profile shape in the TABLE
Parameter 2 is 30	The end of the profile shape in the TABLE
Parameter 3 is 800	This scales the TABLE values. Each CAMBOX motion would therefore total 800*2000 <i>encoder edges steps</i> .
Parameter 4 is 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.
Parameter 5 is 15	This specifies the axis to link to.
Parameter 6 is 2	This is the link option setting - Start at absolute position on the link axis.
Parameter 7 is	variable "start". The motion will execute when the position "start" is reaches on axis 15.

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..1100.

```
CAMBOX(1000,1100,45.68,360,3,4) AXIS(0)
```



REP\_OPTION=2' This will cancel repeating mode.

Note that the system software resets bit 1 of REP\_OPTION to 0 when the mode has been cancelled.

**CAN(channel,function#{parameters})**

Type: Function (MC204/MC216 Only)

Description: This function allows the CAN communication channels to be controlled from the Trio BASIC programming system. The MC204 and MC216 have the capability to support CAN daughter boards. The MC204 and MC216 also has a CAN interface built into the motherboard. With up to 4 CAN daughter boards plus the built-in CAN channel the units can control a maximum of 5 CAN channels:

<b>Channel: Baudrate:</b>	<b>Channel Number:</b>	<b>Maximum</b>
Built-in CAN	-1	500 KHz
Daughter Slot 0	0	1 Mhz
Daughter Slot 1	1	1 Mhz
Daughter Slot 2	2	1 Mhz
Daughter Slot 3	3	1 Mhz

In addition to using the CAN command to control CAN channels, Trio is introducing specific protocol functions into the system software. These functions are dedicated software modules which interface to particular devices.

The Motion Coordinator CAN hardware uses the Siemens 81C91 CAN interface chip. This chip can be programmed at a register level using the CAN command if necessary. To program in this way it is necessary to obtain a copy of the chip data sheet.

The CAN command provides access to 8 separate functions:

CAN(channel#,function#,...)

Channel# The channel number is in the range -1 to 3 and specifies the hardware channel

Function# There are 8 CAN functions 0..7:

0	Read 81C91 Register:	val=CAN(channel#,0,register#)
1	Write 81C91 Register:	CAN(channel#,1,register#,value#)
2	Initialise Baudrate:	CAN(channel#,2,baudrate)
3	Check if message received	val=CAN(channel#,3,message#)
4	Set transmit request	CAN(channel#,4,message#)
5	Initialise message	CAN(channel#,5,message#,identifier,length)
6	Read message	CAN(channel#,6,message#,variable#)
7	Write message	CAN(channel#,7,message#,byte0,byte1...)

Notes: Register# is 81C91 register#

Baudrate: 0=1Mhz, 1=500kHz, 2=250kHz etc

The 81C91 has 16 message buffers(0..15). The message# is which message buffer is required to be used.

"Identifier" is the CAN identifier.

Variable# is the number of the global variable to start loading the data into. The function will load a sequence of n+1 variables. The first variable holds the identifier. The subsequent values the data bytes.

### CAN\_ADDRESS

- Type: Axis Parameter (MC204/MC216 Only)
- Description: The CAN\_ADDRESS axis parameter is used when control is being made of remote servo drives with CAN communications. The CAN\_ADDRESS holds the address of the remote servo drive.

### CAN\_ENABLE

- Type: Axis Parameter (MC204/MC216 Only)
- Description: The CAN\_ENABLE axis parameter is used when control is being made of the remote servo drives with CAN communications. The CAN\_ENABLE is used to control the enable on the remote servo drive.

### CANCEL

- Type: Motion Command
- Alternate Format: CA
- Description: Cancels a move on an axis. Velocity profiled moves (FORWARD, REVERSE, MOVE, MOVEABS, MOVECIRC, MHELICAL, MOVEMODIFY ) will be ramped down at the programmed deceleration rate then terminated. Other move types will be **immediately** terminated. See also RAPIDSTOP.
- CANCEL(1) clears a buffered move, leaving the current executing movement intact.*
- Note: Cancel will only cancel the presently executing move. If further moves are buffered they will then be loaded.
- Example: FORWARD  
WA(10000)  
CANCEL' stop forward move after 10 seconds
- Example 2: MOVE(1000)  
MOVEABS(3000)  
' now change your mind: move to 4000 not 3000  
CANCEL(1)  
MOVEABS(4000)  
' MOVEMODIFY would be better for this !

### CANIO\_ADDRESS

- Type: System Parameter (MC204/MC216 Only)
- Description: The CANIO\_ADDRESS holds the address used to identify the Motion Coordinator when using the Trio CAN I/O networking. The value is held in flash eeprom in the controller and for most systems does not need to be set from the default value of 32. The value may be set to a different value in the range 32..47 *but in this case the Motion Coordinator will not connect to CAN-I/O modules following reset or EX.*



### CANIO\_ENABLE

Type: System Parameter (MC204/MC216 Only)

Description: The CANIO\_ENABLE should be set OFF to completely disable use of the built-in CAN interface by the system software. This allows users to program their own protocols in BASIC using the CAN command. The default is ON.

### CANIO\_STATUS

Type: System Parameter (MC204/MC216 Only)

Description: A bitwise system parameter:

- Bit 0 set indicates an error from the I/O module 0,3,6 or 9
- Bit 1 set indicates an error from the I/O module 1,4,7 or 10
- Bit 2 set indicates an error from the I/O module 2,5,8 or 11
- Bit 3 set indicates an error from the I/O module 12,13,14 or 15
- Bit 4 should be set to re-initialise the CANIO network
- Bit 5 is set when initialisation is complete

### CHECKSUM

Type: System Parameter (READ ONLY)

Description: The checksum parameter holds the checksum for the battery backed RAM. On power up the checksum is recalculated and compared with the previously held value.

### 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.

Note: TRIO BASIC 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:

```
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      12.3456      2.0000
0.0000      0.0000      0.0000
```

**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.

Example: CLOSE\_WIN=10.5' Register window closes at 10.5mm

**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	Retransmit buffer overrun on Network channel
2	RX structure error on Network channel
3	TX structure error on Network channel
4	Error RS232 port A
5	Error RS232 port A
6	Error RS232 port A
7	Error RS232 port A
8	Error RS485 port
9	Error RS485 port
10	Error RS485 port
11	Error RS485 port
12	Error RS232 port B
13	Error RS232 port B
14	Error RS232 port B
15	Error RS232 port B
16	Error FO Network port
17	Error FO Network port
18	Error FO Network port
19	Error FO Network port

**COMPILE**

Type: System Command

Description: Forces compilation (to intermediate code) 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.

### CONNECT(ratio , driving axis)

Type: Command

Alternate Format: CO(ratio, driving axis)

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

Parameters:

*ratio:*

This parameter holds the number 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:*

The driving axis parameter defines the axis that the base axis will be connected to. In the case of the MC204 the driving axis parameter is valid between 0 and 3 inclusive. For the MC216 the parameter is valid between 0 and 15 inclusive.

Note:

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.

Example:

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. An encoder daughter board monitors the encoder pulses from the conveyor and forms axis 1.

```
SERVO AXIS(1)=OFF'   This axis is used to monitor the conveyor
SERVO=ON
CONNECT(0.25,1)
```

### CONTROL

Type: System Parameter (READ ONLY)

Description: The Control parameter returns the type of Motion Coordinator in the system:

MC1:( Not covered by this manual)	1
MC2:	2
EURO 1: (Not covered by this manual)	3
MC204:	204
MC216:	216

### COPY

Type: System Command

Description: Makes a copy of an existing program in memory under a new name

Example: >>COPY "prog" "newprog"

Note:

Motion Perfect users should use the "Copy program..." function under the "Program" menu.

### **COS(expression)**

Type: Function

Description: Returns the COSINE of an expression. Will work for any value. Input values are in radians.

Parameters:

*Expression:* Any valid TRIO BASIC expression.

Example: 

```
>>PRINT COS(0)[3]
1.000
>>
```

### **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. The creep speed must always be a positive value. When given a DATUM move the axis will move at the programmed SPEED until the datum input DATUM\_IN goes low. The axis will then ramp the speed down and start a move in the reversed direction at the CREEP speed until the datum input goes high.

The creep speed is entered in units/sec programmed using the unit conversion factor. For example, if the unit conversion factor is set to the number of encoder edges/inch the speed is programmed in INCHES/SEC.

Example: 

```
BASE(2)
CREEP=10
SPEED=500
DATUM(4)

CREEP AXIS(1)=10
SPEED AXIS(1)=500
DATUM(4) AXIS(1)
```

### **CURSOR**

Type: Command

Description: The CURSOR command is used in a print statement to position the cursor on the Trio membrane keypad and mini-membrane keypad. CURSOR(0), CURSOR(20), CURSOR(40),CURSOR(60) are the start of the 4 lines of the 4 line display. CURSOR(0) and CURSOR(20) are the start of the 2 line display.

Example: 

```
PRINT#3,CURSOR(60);">Bottom line";
```

## DAC

Type: Axis Parameter

Description: Writing to this axis parameter when SERVO=OFF allows the user to force a specified voltage on a servo axis. The range of values that DAC can take is:

DAC=-2048 corresponds to a voltage of 10V  
to  
DAC=2047 corresponds to a voltage of -10v

**Note: The SERVO DAUGHTER BOARD hardware inverts the signal compared to the number.**

Example: To force a square wave of amplitude +/-5V and period of approximately 500ms on axis 0.

```

square:
        WDOG=ON
        SERVO AXIS(0)=OFF
        DAC AXIS(0)=1024
        WA(250)
        DAC AXIS(0)=-1024
        WA(250)
        GOTO square
    
```

## DAC\_OUT

Type: Axis Parameter (Read Only)

Description: The axis DAC is the electronics hardware used to output +/-10volts to the servo amplifier when using a servo daughter board. The DAC\_OUT parameter allows the value being used to be read back. The value put on the DAC comes from 2 potential sources: If the axis parameter SERVO is set OFF then the axis parameter DAC is written to the axis hardware. If the SERVO parameter is ON then a value calculated using the servo algorithm is placed on the DAC. Either case can be read back using DAC\_OUT. Values returned will be in the range -2048 to 2047.

Example: 

```
>>PRINT DAC_OUT AXIS(8)
288.0000
>>
```

## DATE

Type: System Parameter (MC2/MC216 Only)

Description: Returns/Sets the current date held by the MC2/MC216's real time clock.

Example: >>DATE=20:10:98

This sets the date to 20th October 1998. On the MC2 the date 20th October 2000 is set using:

```
>>DATE=20:10:00
```

On the MC216 either this OR:

```
>>DATE=20:10:2000
```

Example2: >>PRINT DATE

This prints the number representing the current day. This number is the number of days since 1st January 1900. On the MC2 controller ONLY this number returns to 0 on 1st January 2000. This may represent a year 2000 compliance issue if this command is used on the MC2. Trio has issued a year 2000 compliance statement which describes the year 2000 issue in relation to all Trio products.

Example3:

This example shows how dates in the year 2000 and beyond can be allowed for in an MC2 application program. This type of compensation is *not* required on the MC216.

```
current_date=DATE
IF current_date<=32872 THEN
    'If date is before 1/1/90 then a date greater than
    '2000 is implied, thus add one century to date.
    date_2000=36524+current_date
ELSE
    'Use date as it stands without adjustment.
    date_2000=current_date
ENDIF
```

## DATE\$

Type: Command (MC2/MC216 Only)

Description: Prints the current date as a string to the port.

Example: PRINT #3,DATE\$

This will print the date in format for example: 20/10/98

## DATUM(Sequence)

Type: Command

Description: Performs one of 7 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.

Parameter:

*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.**
  
- 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.

Note:            **The datuming input set with the DATUM\_IN is active low so is set when the input is OFF. This is similar to the FWD,REV and F HOLD inputs which are designed to be "failsafe".**

Example:        DATUM\_IN=10  
                   SPEED=50  
                   CREEP=5  
                   DATUM(5)

## 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 can be in the range 0..31. If DATUM\_IN is set to -1 then no input is used as a datum.

The same input may also be used as a limit input if required.

Example: `DATUM_IN AXIS(0)=28`

**Note:** **Feedhold, forward,reverse,datum and jog inputs are ACTIVE LOW.**

## DAY

Type: System Parameter (MC2/MC216 only)

Description: Returns the current day as a number 0..6, Sunday is 0.

## DAY\$

Type: Command (MC2/MC216 only)

Description: Prints the current day as a string.

## DEC(rate of dec)

Type: Command

Description: Sets the deceleration rate for an axis. Different rates may be set for each axis.

Parameters:

*rate of dec:* The units of the parameter are dependant on the unit conversion factor. As the deceleration factor is entered in UNIT/SEC/SEC. Therefore if the unit conversion factor is set to the number of encoder edges in 1 revolution and the DEC command is issued DEC(10) it will take 2 seconds to decelerate from 20 revs/sec to stand still.

Note: ACC() sets both the acceleration and deceleration rates. DEC() sets only the deceleration rate. Therefore to set the deceleration rate to a different value than the acceleration rate you must use ACC() then DEC().

**Note 2:** **The DEC() command is provided to maintain compatibility with older controllers. The Axis Parameter DECEL provides the same functionality and is preferred.**



**DECEL**

Type: Axis Parameter

Description: The DECEL axis parameter may be used to set or read back the deceleration rate of each axis fitted. The deceleration rate will be returned in units/sec/sec.

Example: `DECEL=100' Set deceleration rate`  
`PRINT " Deceleration rate is ";DECEL;" mm/sec/sec"`

**DEFKEY(key no, keyvalue1 [,keyvalue2..keyvalue11])**

Type: Command

Description: **Under most circumstances this command is not required and. It is recommended that the values of keys are inputted using a GET#4 sequence.** A GET#4 sequence does *not* use the DEFKEY table. In this example a number representing which key has been pressed is put in the variable k:

`GET#4,k`

The DEFKEY command can be used to re-define what numbers are to be put in the variable when a key is pressed on a MEMBRANE keypad or Mini-Membrane keypad interfaced using an FO-VFKB module. To use the DEFKEY table the values are read using GET#3:

`GET#3,k`

The key numbers of the membrane keypad are shown in chapter 5 of this manual. To each of these key numbers is assigned a value by the DEFKEY command that is returned by a GET#3 command.

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54

Parameters:

*key no:* start key number

*keyvalue1:* value returned by start key through a GET or GET#3 command.

*keyvalue2..keyvalue11:*

values returned by successive keys through a GET or GET#3 command.

Example: The command DEFKEY (33,13) would therefore be used to generate 13 when the first key on row 3 of a pad was pressed. Note DEFKEY can only be used to redefine input on channel#3.

**DEFPOS(position1 [,position2[, position3[, position4.....]]])**

Type: Command.

Alternate Format: DP(position1 [,position2[, position3[, position4]]])

Description: Defines the current position as a new absolute value. The command is typically used after a DATUM sequence as these always set a datum of zero. It may however be used at any time, even whilst a move is in progress.

Parameters:

*position1*: Absolute position to set on current base axis in user units.

*position2*: Absolute position to set on the next axis in BASE array in user units.

*position3*: Absolute position to set on the next axis in BASE array in user units.

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

Example:   
 DATUM(5)   
 BASE(2)   
 DATUM(4)   
 BASE(1)   
 WAIT IDLE   
 DEFPOS(-1000,-3500)

The last line defines the current position, reset to (0,0) by the two DATUM statements as (-1000,-3500) in user units.

Note: See also OFFPOS which performs a relative adjustment of position.

Note 2: **Changes to the axis positions made via DEFPOS and OFFPOS are made on the next servo update.** This can potentially cause problems as the BASIC may continue to execute commands after the DEFPOS is complete, but before the next servo update. For example, the following sequence could easily fail to move to the correct absolute position because the DEFPOS will not have been completed when the MOVEABS is loaded.

Example 2:   
 DEFPOS(100)   
 MOVEABS(0)' DEFPOS may not have occurred yet

DEFPOS statements are internally converted into OFFPOS position offsets which are provides an easy way to avoid the problem described:

```
DEFPOS(100)
WAIT UNTIL OFFPOS=0' Ensures DEFPOS is complete before next line
MOVEABS(0)
```

**DEL**

Type: System Command

Alternate Format: RM

Description: Allows the user to delete a program from memory. The command may be used without a program name to delete a currently selected program. Motion Perfect users should use "Delete program..." on Program menu.

Example: >>DEL oldprog

### DEMAND\_EDGES

Type: Axis Parameter (Read Only)

Description: Allows the user to read back the current DPOS in edge units.

Example: `>>PRINT DEMAND_EDGES AXIS(4)`

### DIR

Type: System Command

Alternate Format: LS

Description: Prints a list of all programs in memory, their size and their RUNTYPE.

### DPOS

Type: Axis Parameter (READ ONLY)

Description: The demand position DPOS is the demanded axis position generated by the move commands. Its value may also be adjusted without doing a move by using the DEFPOS() or OFFPOS commands. It is reset to 0 on power up or software reset. The demand position must never be written to directly although a value can be forced to create a step change in position by writing to the ENDMOVE parameter if no moves are currently in progress on the axis.

Example: `>>? DPOS AXIS(10)`

This will return the demand position in user units.

### D\_GAIN

Type: Axis Parameter

Description: The derivative gain is a constant which is multiplied by the change in following error.

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. High values may lead to oscillation. For a derivative term  $K_d$  and a change in following error  $\Delta e$  the contribution to the output signal is:

$$O_d = K_d \times \Delta e$$

Example: `D_GAIN=0.25`

Note: Servo gains have no effect in stepper motor axes.

### EDIT [optional line sequence number]

Type: System Command

**(Motion Perfect users should use the Motion Perfect editor, not the EDIT command)**

Alternate Format: ED [optional line sequence number]

Description: The edit command starts the built in screen editor allowing a program in the controller memory to be modified using a VT100 terminal. The SELECTed program is edited.

The line sequence number may be used to specify where to start editing.

The editor commands are:

Quit Editor - Control K then D

Delete line - Control Y

Cursor Control - Cursor Keys

### EDPROG

Type: System Command used by Motion Perfect editor.

Alternate Format: &

Description: This is a special command that may be used to manipulate the programs on the controller. **It is not normally used except by Motion Perfect.** It has several forms:

&A Print the line numbers of all the label addresses in the currently selected program

&C Print the name of the currently selected program

&D<line> Delete line <line> from the currently selected program

&I<start>,<string> Insert the text <string> in the currently selected program at the line <line>

&K Print the checksum of the system software

&L<start>,<end> Print the lines of the currently selected program between <start> and <end>

&N Print the number of lines in the currently selected program

&R<start>,<string> Replace the line <start> in the currently selected program with the text <string>

&S<start>,<string> Search for the text <string> starting at line <start> in the currently selected program

&Z Print the CRC checksum of the currently selected program. This uses the standard CCITT 16 bit generator polynomial

### ELSE

Type: Command

Description: This command is used as part of a multi-line IF statement. See IF.

### **ENCODER**

Type: Axis Parameter

Description: The ENCODER axis parameter holds a raw copy of the encoder or resolver hardware register. On Servo daughter boards, for example, this is a 12 bit (Modulo 4096) number. On SSI absolute daughter boards the ENCODER register holds a value of the numbers of bits programmed with SSI\_BITS. The MPOS axis measured position is calculated from the ENCODER value automatically allowing for overflows and offsets.

### **ENDIF**

Type: Command

Description: The ENDIF command marks the end of a multi-line IF statement. See IF.

### **ENDMOVE**

Type: Axis Parameter

Description: This parameter holds the position of the end of the current move in user units. It is normally only read back. If SERVO=ON then it is possible to write to ENDMOVE. This will produce a step change in DPOS.

### **EPROM**

Type: Command

Description: Stores the TRIO BASIC programs in the controller in the FLASH EPROM. This information is be retrieved on power up if the the POWER\_UP parameter has been set to 1. (see POWER\_UP and RUNTYPE for further details).

Note: Motion Perfect performs EPROM automatically when the Motion Coordinator is set to "Fixed"

### **ERROR\_AXIS**

Type: System Parameter (READ ONLY)

Description: Returns the number of the axis which caused the enable WDOG relay to open when a following error exceeded its limit.

Example: `>>? ERROR_AXIS`

### **ERROR\_LINE**

Type: Process Parameter (READ ONLY)

Description: Stores the number of the line which caused the last TRIO BASIC error. This value is only valid when the BASICERROR is TRUE. This parameter is held independently for each process.

Example: `>>PRINT ERROR_LINE PROC(14)`

## 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. The default setting is 256. This will trip the enable relay only if a following error condition occurs.

## EX

Type: Command

Description: Software reset. Resets the controller as if it were being powered up again.

Note: 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.

Note2: **When running Motion Perfect executing an EX command will prevent communication between the controller and the PC.** The same effect as an EX can be obtained by using "Reset the controller..." under the "Controller" menu in Motion Perfect.

## EXP(expression)

Type : Function

Description: Returns the exponential value of the expression.

## FALSE

Type : Constant

Description: The constant FALSE takes the numerical value of 0.

Example:

```
test:      res=IN(0) OR IN(2)
           IF res=FALSE THEN
               PRINT "Inputs are off"
           ENDF
```

## FAST\_JOG

Type : Axis Parameter

Description: This parameter holds the input number to be used as the fast jog input. The input can be in the range 0..31. If FAST\_JOG is set to -1 then no input is used for the fast jog. If the FAST\_JOG is asserted then the jog inputs use the axis SPEED for the jog functions, otherwise the JOGSPEED will be used.

Note: **Feedhold, forward,reverse,datum and jog inputs are ACTIVE LOW.**

**FASTDEC**

Type : Axis Parameter

Description: The FASTDEC axis parameter is not currently used in the motion generator program.

**FE**

Type : Axis Parameter (READ ONLY)

Description: This parameter is the position error, which is equal to the demand position(DPOS)-measured position(MPOS). The parameter is returned in user units.

**FE\_LIMIT**

Type : Axis Parameter

Alternate Format: FELIMIT

Description: This is the maximum allowable following error. When exceeded the controller will generate a run time error and always resets the enable (WDOG) relay thus disabling further motor operation. This limit may be used to guard against fault conditions such as mechanical lock-up, loss of encoder feedback, etc. It is returned in USER UNITS. The default value is 2000 encoder edges.

**FERANGE**

Type : Axis Parameter

Description: Following error report range. When the following error exceeds this value on a servo axis the axis has bit 1 in the AXISSTATUS axis parameter set.

**FEGRAD**

Type : Axis Parameter

Description: Following error limit gradient. Specifies the allowable increase in following error per unit increase in velocity profile speed. The parameter is not currently used in the motion generator program.

**FEMIN**

Type : Axis Parameter

Description: Following error limit at zero speed. The parameter is not currently used in the motion generator program.

## FHOLD\_IN

Type : Axis Parameter

Alternate Format: FH\_IN

Description: This parameter holds the input number to be used as a feedhold input. The input can be in the range 0..31. If FHOLD\_IN is set to -1 then no input is used as a feedhold. When the feedhold input is set motion on the specified axis has its speed overridden to the Feedhold speed (FHSPEED) **WITHOUT CANCELLING THE MOVE IN PROGRESS**. This speed is usually zero. When the input is reset any move in progress when the input was set will go back to the programmed speed. Moves which are not speed controlled E.G. CONNECT, CAMBOX, MOVELINK are not affected.

**Note:** Feedhold, forward,reverse,datum and jog inputs are ACTIVE LOW.

## FH\_SPEED

Type : Axis Parameter

Alternate Format: FHSPEED

Description: When the feedhold input is set motion is normally stopped on that axis because the feedhold speed is set to zero (default value). In some cases it may be desirable for the axis to ramp to a known constant speed when the input is set. To do this the FH\_SPEED parameter is set to a non zero value. The value is in user units/sec.

## FLAG(flag no [,value])

Type : Command/Function

Description: The FLAG command is used to set and read a bank of 32 flag bits. The FLAG command can be used with one or two parameters. With one parameter specified the status of the given flag bit is returned. With two parameters specified the given flag is set to the value of the second parameter. The FLAG command is provided to aid compatibility with earlier controllers and is **not recommended for new programs**.

Parameters:

*flag no:* The flag number is a value from 0..31.

*value:* If specified this is the state to set the given flag to i.e. ON or OFF. This can also be written as 1 or 0.

Example 1: FLAG(27,ON)' Set flag bit 27 ON

## FLAGS([value])

Type: Command/Function

Description: Read/Set the FLAGS as a block. The FLAGS command is provided to aid compatibility with earlier controllers and is **not recommended for new programs**. The 32 flag bits can be read with FLAGS and set with FLAGS(value).

Parameters:

*value:* The decimal equivalent of the bit pattern to set the flags to.



**FOR variable=start TO end [STEP increment]**

```

...
block of commands
...
NEXT variable

```

Type: Command

Description: On entering this loop the variable is initialized 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 no STEP is defined then it is assumed to be 1. **The STEP value may be positive or negative.**

If the value of the variable is less than or equal to the end parameter then the block of commands is repeatedly executed until this is so.

Once the variable is greater than the end value the command after the NEXT statement is executed.

**FOR .. NEXT statements can be nested up to 8 deep in each BASIC program.**

Parameters:

*variable:* A valid TRIO BASIC variable. Either a global VR variable or a local variable may be used.

*start:* A valid TRIO BASIC expression.

*end:* A valid TRIO BASIC expression.

*increment:* A valid TRIO BASIC expression.

Example 1: 

```
FOR opnum=10 TO 18
    OP(opnum,ON)
NEXT opnum
```

This loop sets outputs 10 to 18 ON.

Example 2: 

```
loop:FOR dist=5 TO -5 STEP -0.25
    MOVEABS(dist)
    GOSUB pick_up
NEXT dist
```

Example 3: FOR .. NEXT statements may be nested ( up to 8 deep) provided the inner FOR and NEXT commands are both within the outer FOR..NEXT loop:

```
FOR l1=1 TO 8
    FOR l2=1 TO 6
        MOVEABS(l1*100,l2*100)
        GOSUB operation
    NEXT l2
NEXT l1
```

## FORWARD

Type: Command

Alternate Format: FO

Description: Sets continuous forward movement.

Note: The forward motion can only be stopped by issuing a CANCEL, RAPIDSTOP or by hitting the forward, or datum limits.

Example: 

```
start: FORWARD
        'WAIT FOR STOP SIGNAL
        WAIT UNTIL IN(0)=ON
        CANCEL
```

## FRAC(expression)

Type: Function

Description: Returns the fractional part of the expression.

Example: 

```
>>PRINT FRAC(1.234)
0.2340
>>
```

## 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. 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 robots geometry.

**Frame transformations to perform functions such as these need to be compiled from "C" language source and loaded into the controller system software. Contact Trio if you need to do this.**

A machine system can be specified with several different "frames". The currently active FRAME is specified with the FRAME system parameter. The default FRAME is 0 which corresponds to a one-to-one transformation.

Example: 

```
FRAME=1
```

**FREE**

Type: System Parameter (Read Only)

Description: Returns the amount of program memory available for user programs.

Note: 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 Coordinator compiles programs before they are run, this means that approximately twice the memory is required to be able to run a program.

Parameters: None

Example 1: 

```
>>PRINT FREE
47104.0000
>>
```

Example 2: VR(10)=IN AND 255

This line requires 21 bytes of storage in the uncompiled version and 19 in the compiled version:

Uncompiled:

	Byte	Length	Value
	0	1	PREVIOUS LINE LENGTH
	1	1	THIS LINE LENGTH
	2	1	PRECEDING BLANKS
	3	1	VR TOKEN
	4	1	( TOKEN
	5	1	NUMBER TOKEN
	6	2	Digits 1-0
	8	1	END OF NUMBER TOKEN
	9	1	) TOKEN
	10	1	= TOKEN
	11	1	IN TOKEN
	12	1	SPACE TOKEN
	13	1	AND TOKEN
	14	1	SPACE TOKEN
	15	1	NUMBER TOKEN
	16	3	Digits 2-5-5
	19	1	END OF NUMBER TOKEN
	20	1	END OF LINE

Compiled version:

	0	1	VR TOKEN
	1	1	( TOKEN
	2	1	NUMBER TOKEN
	3	4	32 bit floating number
	7	1	END OF EXPRESSION TOKEN
	8	1	) TOKEN
	9	1	ASSIGNMENT TOKEN
	10	1	IN TOKEN
	11	1	NUMBER TOKEN
	12	4	32 bit floating point number
	16	1	END OF NUMBER TOKEN
	17	1	AND TOKEN
	18	1	END OF LINE

## 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. When the limit is hit the controller will ramp down the speed to zero then cancel the move. Bit 9 of the AXISSTATUS register is set when the axis position is greater than the FS\_LIMIT.

Note: **Feedhold, forward,reverse,datum and jog inputs are ACTIVE LOW.**

---

**FWD\_IN**

Type: Axis Parameter (Default=-1)

Description: This parameter holds the input number to be used as a forward limit input. The input can be in the range 0..31. If FWD\_IN is set to -1 then no input is used as a forward limit. When the forward limit input is asserted any forward motion on that axis is stopped.

Example: FWD\_IN=19

**Note:** Feedhold, jog forward,reverse and datum inputs are ACTIVE LOW.

**FWD\_JOG**

Type: Axis Parameter (Default=-1)

Description: This parameter holds the input number to be used as a jog forward input. The input can be in the range 0..31. If FWD\_JOG is set to -1 (default) then no input is used as a forward jog.

Example: FWD\_JOG=7

**Note:** Feedhold, forward,reverse,datum and jog inputs are ACTIVE LOW.

**GET**

Type: Command.

Description: Waits for the arrival of a single character on the default serial port 0. The ASCII value of the character is assigned to the variable specified. The BASIC program will wait until a character is available.

Example: GET k

**GET#n,variable**

Type: Command

Description: Functions as GET but the input device is specified as part of the command. The device specified is valid only for the duration of the command.

Parameters:

*n*: Input device:-  
 0=Serial port A  
 1=Serial port B  
 2=RS485 Port  
 3=Fibre optic port (value returned defined by DEFKEY)  
 4=Fibre optic port (returns raw keycode of key pressed)  
 5=Motion Perfect user channel  
 6=Motion Perfect user channel  
 7=Motion Perfect user channel  
 8=Used for Motion Perfect internal operations  
 9=Used for Motion Perfect internal operations  
 10=Fibre optic network data

*x*: Variable.

Example: `GET#3,k 'Just for this command input taken from fibre optic`

Note: Channels 5 to 9 are logical channels which are superimposed on to Serial Port A by Motion Perfect.

**GOSUB label**

Type: Command

Description: Stores the position of the line after the GOSUB command and then branches to the line specified. Upon reaching the RETURN statement, control is returned to the stored line.

Parameters:

*label*: A valid label that occurs in the program. If the label does not exist an error message will be displayed during structure checking at the beginning of program run time and the program execution halted.

Example:

```
main:      GOSUB routine
           GOTO main

routine:   PRINT "Measured Position=";MPOS;CHR(13);
           RETURN
```

Note: Subroutines on each process can be nested up to 8 deep.

## **GOTO label**

Type:	Command
Description:	Identifies the next line of the program to be executed.
Parameters:	
<i>label:</i>	A valid label that occurs in the program. If the label does not exist an error message will be displayed during structure checking at the beginning of program run time and the program execution halted.
Example:	<pre> loop:      PRINT "Measured Position=";MPOS;CHR(13);            WA(1000)            GOTO loop </pre>
Note:	Labels may be character strings of any length. (The first 15 characters are significant) Alternatively line numbers can be used.

## **HALT**

Type:	System Command
Description:	Halts execution of all programs currently running. The STOP command will stop a specific program.

**IF condition THEN  
          commands  
          ELSE  
          commands  
ENDIF**

Type: Command

Description: The command evaluates the condition and if it is true it executes the commands specified, otherwise the commands are skipped. If the condition is false and an ELSE command sequence is specified then this command sequence is executed.

Parameters:

*condition:* Any logical expression.

*commands:* Any valid trio basic commands including further IF..THEN {ELSE} ENDIF sequences

Note: IF..THEN {ELSE} ENDIF sequences can be nested without limit other than program memory size

Example 1: `IF MPOS>(0.22*VR(0)) THEN GOTO exceeds_length`

Example 2: `IF IN(0)=ON THEN  
          count=count+1  
          PRINT "COUNTS=";VR(1)  
          fail=0  
ELSE  
          fail=fail+1  
ENDIF`

**Note:** For a multi-line IF ..THEN construction there must not be any statement after the THEN keyword. If there is the BASIC will assume it is a single line IF construction. The single line construction should not use ENDIF.

The ELSE sequence is optional. If it is not required the ENDIF is used to mark the end of the conditional block.



### IN(input#<,final input>)/IN

- Type: Function.
- Description: Returns the value of digital inputs. If called with no parameters, IN returns the binary sum of the first 24 inputs (if connected). If called with one parameter whose value is less than the highest input channel. It returns the value (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.
- Parameters:
- 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:
- ```
test: WAIT UNTIL IN(4)=ON
      `CONVEYOR IS NOW IN POSITION
      GOSUB place
```
- Example 2: Move to the distance set on a thumbwheel multiplied by a factor. The thumbwheel is connected to inputs 4,5,6,7 and gives output in BCD.
- ```
moveloop: MOVEABS(IN(4,7)*1.5467)
          WAIT IDLE
          GOTO moveloop
```
- Note how the move command is constructed:
- Step 1: IN(4,7) will get a number 0..15  
 Step 2: multiply by 1.5467 to get required distance  
 Step 3: incremental MOVE by this distance
- Note: IN is equivalent to IN(0,23)

### INDEVICE

- Type: Process Parameter
- Description: This parameter specifies the active input device. Specifying an INDEVICE for a process allows the channel number for a program to set for all subsequent GET and KEY, INPUT and LINPUT statements. **(This command is not usually required - Use GET # and KEY # etc. instead)**
- Input device:-
- 0=Serial port A
  - 1=Serial port B
  - 2=RS485 Port
  - 3=Fibre optic port (value returned defined by DEFKEY)
  - 4=Fibre optic port (returns raw keycode of key pressed)
  - 5=Motion Perfect user channel
  - 6=Motion Perfect user channel
  - 7=Motion Perfect user channel
  - 8=Used for Motion Perfect internal operations
  - 9=Used for Motion Perfect internal operations
  - 10=Fibre optic network data
- Example:
- ```
INDEVICE=5
' Get character on channel 5:
GET k
```

## 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. When using Motion Perfect a "Reset the controller.." under the "Controller" menu performs the equivalent of an EX command

## INPUT

Type: Command.

Description: Waits for a 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 inputs may be requested on one line, separated by commas, or on multiple lines, separated by <CR>.

Example1: 

```
INPUT num
PRINT "BATCH COUNT=" ;num[0]
```

On terminal:  
123 <CR>  
BATCH COUNT=123

Example2: 

```
getlen: PRINT ENTER LENGTH AND WIDTH ?";
INPUT VR(11),VR(12)
```

On terminal:  
ENTER LENGTH AND WIDTH ? 1200,1500 <CR>

Note: This command will not work with the serial input device set to 3 or 4, i.e. the fibre optic port, as the received codes are not ASCII 0..9. It is also not possible for a program to use the serial port 0 as the command line process will remove the characters. Programs needing a "terminal" style interface should use one of the channel 6 to channel 7 ports if using Motion Perfect.

## INPUTS0/INPUTS1

Type: System Parameters

Description: The INPUTS0 parameter holds 24 volt Input channels 0..15 as a system parameter. INPUTS1 parameter holds 24 volt Input channels 16..31 as a system parameter. **It is not normally required to read the inputs using this system parameter** (use the IN(x,y) command instead). They are made available in this format to make the input channels available storable by the SCOPE command which can only store parameters.

### INT(expression)

Type: Function

Description: The INT function returns the integer part of a number.

Parameters:

*expression*: Any valid TRIO BASIC expression.

Example: 

```
>>PRINT INT(1.79)
1.0000
>>
```

Note: To round a positive number to the nearest value take the INT function of the number + 0.5

### INVERT\_STEP

Type: Axis Parameter (Default=OFF)

Description: INVERT\_STEP is used to switch a hardware inverter into the stepper pulse output circuit. This can be necessary in for connecting to some stepper amplifiers. 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 amplifiers. Setting INVERT\_STEP=ON effectively makes the RISING edge of the step signal the active edge. INVERT\_STEP should be set if required prior to enabling the controller with WDOG=ON.

Note: **If the setting is incorrect. A stepper motor *may* lose position by one step when changing direction.**

### I\_GAIN

Type: Axis Parameter (Default=0)

Description: The integral gain is a constant which is multiplied by the sum of following errors of all the previous samples. This term may often be set to zero. Adding integral gain to a servo system reduces position error when at rest or moving steadily but it will produce or increase overshoot and may lead to oscillation. For an integral gain  $K_i$  and a sum of position errors  $\sum e$ , the contribution to the output signal is:

$$O_i = K_i \times \sum e$$

Note: Servo gains have no effect on stepper motor axes.

### JOGSPEED

Type: Axis Parameter

Description: Sets the slow jog speed in user units for an axis to run at when performing a slow jog. A slow jog will be performed when a jog input for an axis has been declared and that input is low. The jog will be at the JOGSPEED provided the FAST\_JOG input has not be declared and is set low. Two separate jog inputs are available for each axis FWD\_JOG and REV\_JOG.



## LINPUT

|              |                                                                                                                                                                                                                                                                             |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type:        | Command                                                                                                                                                                                                                                                                     |
| Description: | Waits for an input string and stores the ASCII values of the string in an array of variables starting at a specified <u>numbered variable</u> . The string must be terminated with a carriage return <CR> which is also stored. The string is not echoed by the controller. |
| Parameters:  | None.                                                                                                                                                                                                                                                                       |
| Example:     | <pre>LINPUT VR(0)</pre> <p>Now entering: START&lt;CR&gt;<br/>will give: VR(0)=83 ASCII 'S'<br/>VR(1)=84 ASCII 'T'<br/>VR(2)=65 ASCII 'A'<br/>VR(3)=82 ASCII 'R'<br/>VR(4)=84 ASCII 'T'<br/>VR(5)=13 ASCII carriage return</p>                                               |

## LIST

|                   |                                                                                                                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type:             | Command                                                                                                                                                                                     |
| Alternate Format: | TYPE                                                                                                                                                                                        |
| Description:      | Prints the current SELECTed program or a specified program to channel 0.                                                                                                                    |
| Note:             | LIST is used as an immediate (command line) command only and should not be used in programs. It is also recommended not to use LIST in Motion Perfect but to use the Motion Perfect editor. |

## LN(parameter)

|              |                                                  |
|--------------|--------------------------------------------------|
| Type:        | Function                                         |
| Description: | Returns the natural logarithm of the expression. |
| Parameter:   | Any valid TRIO BASIC expression.                 |

## LOADSYSTEM

Type: System Command

Description: Loads new version of system software:

On the Motion Coordinator SERIES 2 family of controllers the system software is stored in FLASH EPROM. It is copied into RAM when the system is powered up so it can execute faster. The system software can be re-loaded through the serial port 0 into the RAM copy using Motion Perfect. The command STORE is then used to transfer the updated copy of the system software into the FLASH EPROM for use on the next power up.

To re-load the system software you will need:

- 1 - The system software on disk supplied by TRIO in COFF format. (Files have a .OUT suffix, for example A140.OUT)
- 2 - Motion Perfect.

The download sequence:

Run Motion Perfect in the usual way. Under the "Controller" menu select "Load system software...". Select the version of system software to be loaded and follow the on screen instructions. The system file takes around 12 minutes to download. When the download is complete the system performs a checksum prior to asking to confirm that the file is to be loaded into flash eprom. The storing process takes around 10 seconds and must NEVER be interrupted by the power being removed. **If this final stage is interrupted the controller may have to be returned to Trio for re-initialisation.**

Note: MC216,MC2 and MC204 controllers have different system software files. Updates can be obtained from Trio's website at [www.triomotion.com](http://www.triomotion.com)

Note 2: Application programs should be on disk prior to a system software load and MUST be reloaded following a system software load.

## LOCK(*code*)

|              |                                                                                                                                                                                                                                                                                                                                                                     |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type:        | Command                                                                                                                                                                                                                                                                                                                                                             |
| Description: | Prevents program from being viewed or modified by personnel unaware of the security code. The security code is stored in the flash eeprom.                                                                                                                                                                                                                          |
| Note:        | LOCK is always an immediate command and may only be issued when the system is UNLOCKED. The system may be in either a LOCKED or UNLOCKED state. The LOCK comand allows the LOCKED state to be entered. The security code should be remembered as it is required to unlock the system when required. The code number may be any integer and is held in encoded form. |
| Parameters:  |                                                                                                                                                                                                                                                                                                                                                                     |
| <i>code</i>  | Any integer number                                                                                                                                                                                                                                                                                                                                                  |
| Example:     | <pre>&gt;&gt;LOCK(561234)</pre> <p>The program cannot now be modified or seen.</p> <pre>&gt;&gt;UNLOCK(561234)</pre> <p>The system is now unlocked.</p>                                                                                                                                                                                                             |
| Note:        | The LOCK and UNLOCK sequences are not supported at present in Motion Perfect. They should be performed in a disconnected terminal window.                                                                                                                                                                                                                           |

## MARK

|              |                                                                                                                                                                                 |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type:        | Axis Parameter                                                                                                                                                                  |
| Description: | Returns TRUE when a registration event has occurred. This is set to FALSE by the REGIST command and set to true when the register event occurs. When TRUE the REG_POS is valid. |
| Example:     | See REGIST                                                                                                                                                                      |

## MATCH

|              |                                                                                                                                                                      |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type:        | Axis Function                                                                                                                                                        |
| Description: | Instructs the controller to perform a pattern comparison between a stored pattern of registration input transitions and the pattern held following a REGIST command. |
| Note:        | See also REGIST and RECORD                                                                                                                                           |

## **MERGE**

Type: Axis Parameter

Description: This is a software switch which can be used to enable or disable the merging of consecutive moves. With merging enabled, if the next move is already in the buffer the axis will not ramp down to zero speed but load up the following move allowing them to be seamlessly merged. Note that 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:

- 1 - The next move is loaded
- 2 - Axis group must not change on multi-axis moves
- 3 - Velocity profiled moves (MOVE, MOVEABS, MOVECIRC, MHELICAL, REVERSE, FORWARD) cannot be merged with linked moves (CONNECT, MOVELINK, CAMBOX)

Note: When merging multi-axis moves only the base axis MERGE flag needs to be set.

If the moves are short a high deceleration rate must be set to avoid the controller ramping the speed down in anticipation of the end of the buffered move

Example: `MERGE=OFF'` Decelerate at the end of each move  
`MERGE=ON'` Moves will be merged if possible



**MHELICAL(finish 1, finish 2, centre1, centre2, direction, distance 3)**

Type: Command.

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. Finish 1 and centre 1 are on the current BASE axis. Finish 2 and centre 2 are on the following axis. The first 4 distance and the sixth parameter are scaled according to the current unit conversion factor for each axis.

Parameters:

*finish1*: position on BASE axis to finish at.

*finish2*: 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 0 or 1. See MOVECIRC.

*distance3*: The distance to move on the third axis in the BASE array axis in user units

**MICROSTEP**

Type: Axis Parameter.

Description: Sets microstepping mode when using a stepper daughter board. The stepper pulse circuit contains a circuit which places the step pulses more evenly in time by dividing the pulserate by 2 or 16:

|               |         |                  |
|---------------|---------|------------------|
| MICROSTEP=OFF | DEFAULT | 62.5 kHz Maximum |
| MICROSTEP=ON  |         | 500 kHz Maximum  |

The stepper daughter board can generate pulses at up to 62500 Hz with MICROSTEP=OFF (This is the default setting and should be used when the pulserate does not exceed 62500 Hz even if the motor is microstepping) With MICROSTEP=ON the stepper board can generate pulses at up to 500,000 Hz although the pulses are not so evenly spaced in time.

With MICROSTEP=OFF the UNITS parameter should be set to 16 times the number of pulses in a distance parameter. With MICROSTEP=ON the UNITS should be set to 2 times the number.

Example: UNITS AXIS(2)=180\*2' 180 pulses/rev \* 2  
MICROSTEP AXIS(2)=ON

**MOD(expression)**

Type: Function

Description: Returns the modulus of an expression.

Example: 

```
>>PRINT 122 MOD(13)
5.0000
>>
```

**MOTION\_ERROR**

Type: System Parameter

Description: This system parameter has the value 1 when some axis has had a motion error, i.e. following error, and the value 0 when no axis has had a motion error. When there is a motion error then the ERROR\_AXIS contains the number of the first axis to have an error. When any axis has a motion error then the WDOG relay is opened. A motion error can be cleared by resetting the controller with an EX command ("Reset the controller.." under the "Controller" menu in Motion perfect), or by using the DATUM(0) command.

**MOVE(distance1 [,distance2[ ,distance3[ ,distance4...]]])**

Type: Command

Alternate Format: MO()

Description: Incremental move. Axis or axes move at the programmed speed and acceleration to a position specified as an increment from the end of the last specified move.

Note: The MOVE command can interpolate up to 4 axes on MC204, 12 axes on MC2, or 16 axes on MC216. The values specified are scaled using the UNIT CONVERSION FACTOR, axis parameter UNITS. Therefore if, for example, an axis has 4000 encoder edges/mm then UNITS for that axis are 4000. The command MOVE(12.5) would move 12.5 mm. The first parameter in the list is sent to the BASE axis or can be re-directed with the AXIS() command, the second to the next axis in the BASE array, etc. By changing the axis uninterpolated, unsynchronised multi-axis motion can be achieved. Incremental moves can be merged together for profiled continuous path movement. MERGE should be set to ON. In multi-axis systems the speed and acceleration employed for the movement are taken from the first axis in the group.

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 number of parameters can increase to the number of axes on the controller

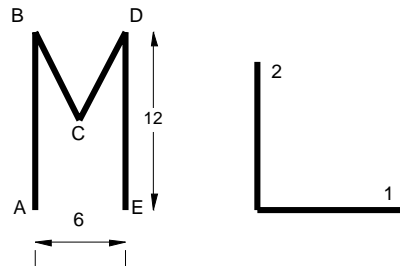
Example 1: A system is working with a unit conversion factor of 1 and has a 1000 line encoder. It is therefore necessary to give the instruction MOVE(40000) to incrementally move 10 turns on the motor. (A 1000 line encoder gives 4000 edges/turn)

Example 2: MOVE(10) AXIS(5)  
MOVE(10) AXIS(4)  
MOVE(10) AXIS(3)

In this example axes 3,4 and 5 are moving independently (without interpolation). Each axis will move at its programmed SPEED etc.

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 no matter what the plot position. The command subroutine for the letter 'M' might be:

```
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
```



**MOVEABS(1 position [, 2 position[, 3 position[, 4 position...]]])**

Type: Command.

Alternate Format: MA()

Description: Absolute position move. Move an axis or axes to position(s) referenced to the zero position.

Parameters:

- 1 position: position to move to on base axis.
- 2 position: position to move to on next axis in BASE array.
- 3 position: position to move to on next axis in BASE array.
- 4 position: position to move to on next axis in BASE array. ...

Note: Command can be used to interpolate up to 4 axes on MC204, up to 12 axes on MC2, and up to 16 axes on MC216. The values specified are scaled using the UNIT CONVERSION FACTOR, axis parameter UNITS. Therefore if, for example, an axis has 4000 encoder edges/mm the UNITS for that axis is 4000. The command MOVEABS(6) would move to a position 6 mm from the zero 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, etc. By changing the BASE axis uninterpolated, unsynchronised multi-axis motion can be achieved. Absolute moves can be merged together for profiled continuous path movement. Axis parameter MERGE should be set to ON. In multi-axis systems the speed, acceleration and deceleration employed for the movement are taken from the BASE AXIS for the group.

Note2: The position of the axis zero positions can be moved by the commands: OFFPOS,DEFPOS,REP\_DIST,REP\_OPTION, and DATUM.

Example 1: 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(20,350)
```

Example 2: A pallet consists of a 6 by 8 grid in which gas canisters are inserted 85mm 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:

```

xloop: FOR x=0 TO 5
yloop:   FOR y=0 TO 7
          'MOVE TO PICK UP POINT:
          MOVEABS(-340,-516.5)
          'PICK UP SUBROUTINE:
          GOSUB pick
          PRINT "MOVE TO POSITION: ";x*6+y+1
          MOVEABS(x*85,y*85)
          'PLACE DOWN SUBROUTINE:
          GOSUB place
        NEXT y
      NEXT x
  
```

**MOVECIRC(finish1, finish2, centre1, centre2, direction)**

Type: Command.

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 incremental from 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 following axis in the BASE array. The first 4 distance parameters are scaled according to the current unit conversion factor for each axis.

Parameters:

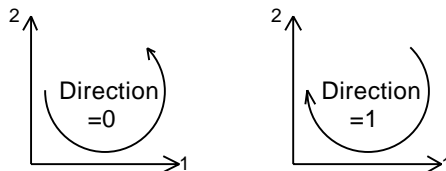
*finish1*: position on BASE axis to finish at.

*finish2*: 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.

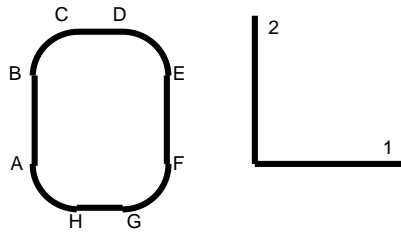


Note: In order for the MOVECIRC() command to be correctly executed, the two axes generating the circular arc must have the same number of encoder pulses/linear axis distance. If this is not the case it is possible to adjust the encoder scales in many cases by adjusting with PP\_STEP.

Note2: 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.

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

|                         |             |
|-------------------------|-------------|
| MOVE(0,6)'              | move A -> B |
| MOVECIRC(3,3,3,0,1)'    | move B -> C |
| MOVE(2,0)'              | move C -> D |
| MOVECIRC(3,-3,0,-3,1)'  | move D -> E |
| MOVE(0,-6)'             | move E -> F |
| MOVECIRC(-3,-3,-3,0,1)' | move F -> G |
| MOVE(-2,0)'             | move G -> H |
| MOVECIRC(-3,3,0,3,1)'   | move H -> A |



**MOVELINK (distance, link distance, link acc, link dec, link axis[, link options] [, link start]).**

Type: Command.

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 of another axis.

Parameters:

1 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.

2 link distance: 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.

3 link acc: positive incremental distance in user units on the input axis over which the base axis accelerates.

4 link dec: positive incremental distance in user units on the input axis over which the base axis decelerates. N.B. 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.

5 link axis: Specifies the axis to "link" to. It should be set to 0..number of axes.

6 link options: Settings:

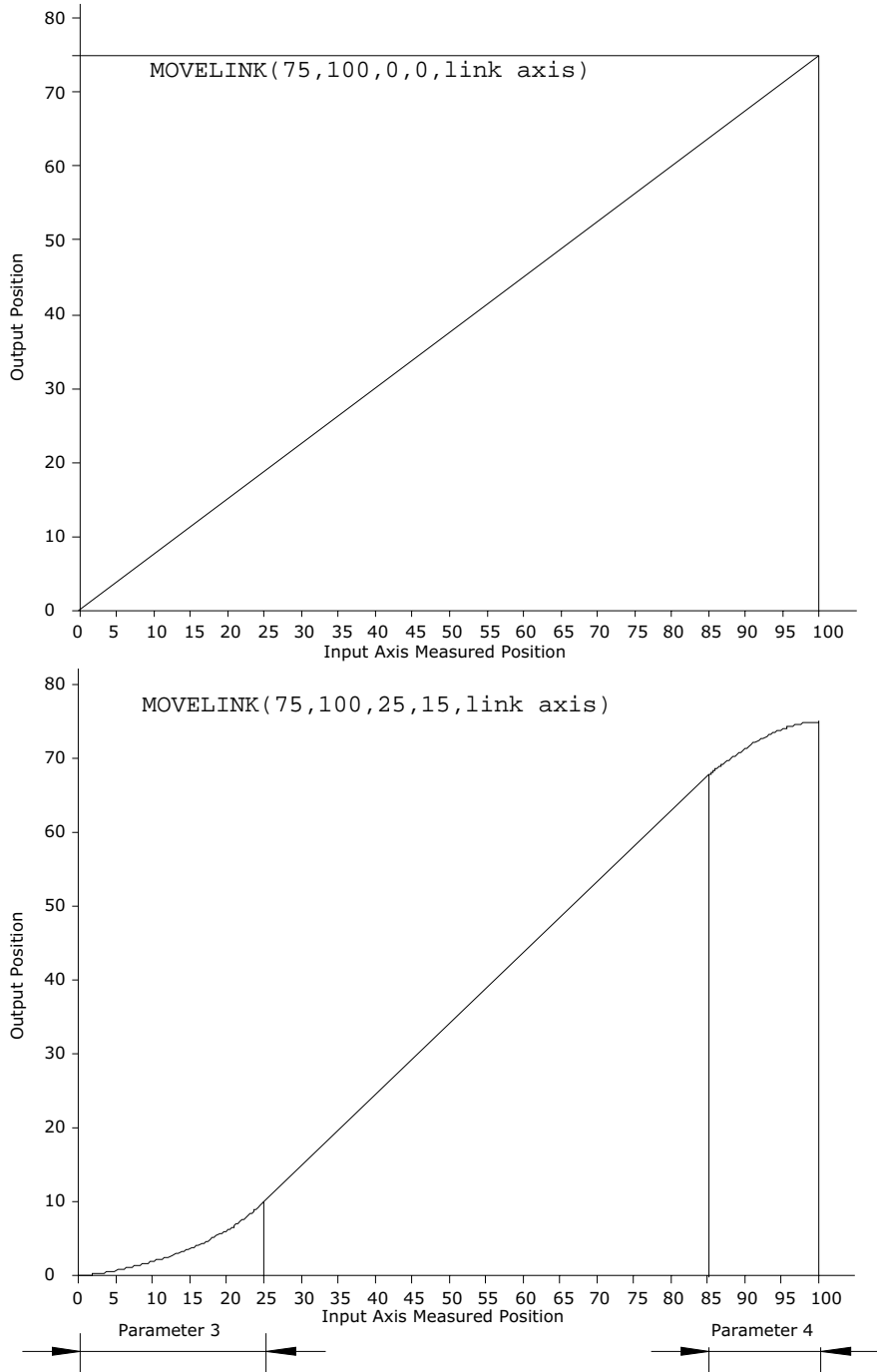
- 1 -link commences exactly when registration event occurs on link axis
- 2 -link commences at an absolute position on link axis (see param 7)
- 4 -MOVELINK repeats automatically and bi-directionally when this bit is set. (This mode can be cleared by setting bit 1 of the REP\_OPTION axis parameter)

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

Note: 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.

**Note: Parameters 6 and 7 are optional.**





Example: A flying shear cuts a roll of paper every 160m whilst moving at the speed of the paper. 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)

```

MOVELINK(0,150,0,0,7)'          wait distance
MOVELINK(0.4,0.8,0.8,0,7)'      accelerate
MOVELINK(0.6,1.0,0,0.8,7)'      match speed then decelerate
WAIT LOADED'                    wait till previous move started
OP(8,ON)'                        activate cutter
MOVELINK(-1,8.2,0.5,0.5,7)'      move back
    
```

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 coasts at the same speed then decelerates to a stop within the 1m stroke. This movement is specified using two separate MOVELINK commands. The program then waits for the the next move buffer to be clear NTYPE=0 . This 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(0.4,0.8,0.8,0,1)' This 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(0.2,0.2,0,0,1)' This is all constant speed
```

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

```
MOVELINK(0.4,0.8,0,0.8,1)' This move is all decel
```

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

```
MOVELINK(1,1.8,0.8,0.8)' Same as 3 moves above
```

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

```
MOVELINK(0.4,0.8,0.8,0)
MOVELINK(0.6,1.0,0,0.8)
```

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

### MOVEMODIFY(absolute position)

Type: Command.

Alternate Format: MM()

Description: This move type changes the absolute end position of the current single axis linear move (MOVE, MOVEABS). If there is no current move or the current move is not a linear move then MOVEMODIFY is loaded as a MOVEABS. See ENDMOVE.

Parameters:

*absolute position*: The absolute position to be set as the new end of move.

Example: 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)'           A long move on conveyor
REGIST(3)'            set up registration
WAIT UNTIL MARK '     MARK will be true when proximity seen
OFFPOS=-REG_POS'      set position where mark seen to 0
MOVEMODIFY(250)'      change move to stop at 250mm
```

### MPE

Type: Command

Description: Sets the type of channel handshaking to be performed on the serial port A. **This is normally only used by the Motion Perfect program**, but can be used for user applications. There are three valid settings:

- 0 No channel handshaking, XON/XOFF 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 XOFF is sent even though there may be enough space in a different channel buffer to receive more characters
- 1 Channel handshaking on, XON/XOFF controlled by the port. When the current output channel is changed, the channel change sequence is sent (<ESC><channel number>). When there is not enough space to store any more characters in the current input channel then XOFF is sent even though there may be enough space in a different channel buffer to receive more characters
- 2 Channel handshaking on, XON/XOFF controller by the channel. When the current output channel is changed, the channel change sequence is sent (<ESC><channel number>). When there is not enough space to store any more characters in the current input buffer, then XOFF is sent for this channel (<XOFF><channel number>) 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.

Parameters:

*channel type*: Any valid TRIO BASIC expression

Example1:        >> MPE(0)  
                  >> PRINT #5,"Hello"  
                  Hello  
                  >>

Example2:        >> MPE(1)  
                  >> PRINT #5,"Hello"  
                  <ESC>5Hello  
                  <ESC>0  
                  >>

## **MPOS**

Type:            Axis Parameter (READ ONLY)

Description:    This parameter is the position of the axis as measured by the encoder or resolver. It is reset to 0 (unless a resolver is fitted) 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.

Example:        WAIT UNTIL MPOS>=1250  
                  SPEED=2.5

## **MSPEED**

Type:            Axis Parameter (READ ONLY)

Description:    The MSPEED represents the change in measured position in user units (per second) in the last servo period. The SERVO\_PERIOD defaults to 1 mSec. It therefore can be used to represent the speed measured. 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.

**MTYPE**

Type: Axis Parameter (Read Only)

Description: This parameter holds the type of move currently being executed.

| <b>MTYPE</b> | <b>Move Type</b> |
|--------------|------------------|
| 0            | Idle (No move)   |
| 1            | MOVE             |
| 2            | MOVEABS          |
| 3            | MHELICAL         |
| 4            | MOVECIRC         |
| 5            | MOVEMODIFY       |
| 10           | FORWARD          |
| 11           | REVERSE          |
| 12           | DATUMING         |
| 13           | CAM              |
| 14           | Forward Jog      |
| 15           | Reverse Jog      |
| 20           | CAMBOX           |
| 21           | CONNECT          |
| 22           | MOVELINK         |

This parameter may be interrogated to determine whether a move has finished or if a transition from 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.

**NAIO**

Type: System Parameter (Read Only)

Description: This parameter returns the number of CAN analog input channels connected on the IO expansion CAN bus. For example an MC216 will return 8 if there is 1 x P325 CAN Module connected as it has 8 analog input channels.

**NETSTAT**

Type: System Parameter

Description: This parameter stores the network error status since the parameter was last cleared by writing to it. The error types reported are:

| <b>Bit Set</b> | <b>Error Type</b> | <b>Value</b> |
|----------------|-------------------|--------------|
| 0              | TX Timeout        | 1            |
| 1              | TX Buffer Error   | 2            |
| 2              | RX CRC Error      | 4            |
| 3              | RX Frame Error    | 8            |

**NEW**

Type: Command

Description: Deletes all the program lines in the controller memory. It also may be used to delete the current TABLE entries.

Note:
 

|              |   |                                                                                    |
|--------------|---|------------------------------------------------------------------------------------|
| NEW          | - | Deletes the currently selected program                                             |
| NEW progname | - | Deletes a particular program                                                       |
| NEW ALL      | - | Deletes all programs in memory                                                     |
| NEW "TABLE"  | - | Delete TABLE (In this case ONLY the program name "TABLE" <b>must</b> be in quotes) |

**NEXT**

Type: Command

Description: Used to mark the end of a FOR..NEXT loop. See FOR.

**NIO**

Type: System Parameter (Read Only)

Description: This parameter returns the number of inputs/outputs fitted to the system, or connected on the IO expansion CAN bus. For example an MC204 will return 16 as there are 16 built- in IO channels in this module. (8 inputs and 8 bi-directional channels)

**NOT(expression)**

Type: Function

Description: The NOT function truncates the number and inverts all the bits of the integer remaining.

Parameters:

expression: Any valid TRIO BASIC expression.

Example:
 

```
>>PRINT 7 AND NOT(1.5)
6.0000
>>
```

**NTYPE**

Type: Axis Parameter (Read Only)

Description: This parameter holds the type of the next buffered move. The values held are as for MTYPE. If no move is buffered zero will be returned. The NTYPE parameter is read only but the NTYPE can be cleared using CANCEL(1)

**OFF**

Type: Constant

Description: OFF returns the value 0

Example:
 

```
IF IN(56)=OFF THEN GOTO label
```

 Branch if input 56 is off.

## OFFPOS

- Type: Axis Parameter
- Description: The OFFPOS parameter allows the demand position to be offset by any value without affecting motion. This effectively adjusts the zero position of the axis. Values loaded into the OFFPOS axis parameter are reset to 0 by the system as they are loaded.
- Example: Define the current demand position as zero:
- ```
OFFPOS=-DPOS
WAIT UNTIL OFFPOS=0' wait until applied
```
- This is equivalent to DEFPOS(0)
- Note: **The OFFPOS adjustment is executed on the next servo period. Several BASIC instructions may occur prior to the next servo period. Care must be taken to ensure these instructions do not assume the position shift has occurred.**

## ON

- Type: Constant
- Description: ON returns the value 1.
- Example: `OP (lever,ON)` This sets the output named lever to ON.

## ON expression GOSUB label[,label[,...]]

- Type: Command
- Description: 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. If the value of the expression is less than 1 or greater than the number of labels then an error occurs. Once the label is selected a GOSUB is performed.
- Example:
- ```
REPEAT
GET #3,char
UNTIL 1<=char and char<=3
ON char GOSUB mover,stopper,change
```

## ON expression GOTO label[,label[,...]]

- Type: Command
- Description: 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. If the value of the expression is less than 1 or greater than the number of labels then an error occurs. Once the label is selected a GOTO is performed.
- Example:
- ```
REPEAT
GET #3,char
UNTIL 1<=char and char<=3
ON char GOTO mover,stopper,change
```

**OP([output no,] value)]**

Type: Command/Function.

Description: Sets output(s) and allows the state of the first 24 outputs to be read back. The command has three different forms depending on the number of parameters. A single output channel may be set with the 2 parameter command. The first parameter is the channel number 8-95 and the second is the value to be set 0 or 1.

If the command is used with 1 parameter the parameter is used to simultaneously set the first 24 outputs with the binary pattern of the number. If the command is used with no parameters the first 24 outputs are read back. This allows multiple outputs to be set without corrupting others which are not to be changed.

Note: The first 8 outputs (0 to 7) do not physically exist on the MC204/MC-2/MC216 so if they are written to nothing will happen and if they are read back they will always return 0.

Parameters:

*output no:* Output number to set.

*value:* Output value to be set. 0/1 for 2 parameter command, decimal equivalent of binary number to set on outputs for one parameter command

Example 1: `OP(44,1)`  
 This is equivalent to `OP(44,ON)`

Example 2: `OP (34*256)`  
 This sets the bit pattern 10010 on the first 5 physical outputs, outputs 13-31 would be cleared. Note how the bit pattern is shifted 8 bits by multiplying by 256 to set the first available outputs as outputs 0 to 7 do not exist.

Example 3: `read_op: VR(0)=OP  
 `SET OUTPUTS 8..15 ON SIMULTANEOUSLY  
 VR(0)=VR(0) AND 65280  
 OP(VR(0))`

Note how this example can also be written:

```

`SET OUTPUTS 8..15 ON SIMULTANEOUSLY
OP(OP AND 65280)
    
```

**OPEN\_WIN**

Type: Axis Parameter

Alternate Format: OW

Description: This parameter defines the position before which registration marks will be ignored if windowing is specified by the REGIST() command.

**Expression1 OR expression2**

Type: Logical and bitwise operator

Description: This performs an OR function between corresponding bits of the integer part of two valid TRIO BASIC expressions. The OR function between two bits is defined as follows:

OR	0	1
0	0	1
1	1	1

Parameters:

*Expression1:* Any valid TRIO BASIC expression

*Expression2:* Any valid TRIO BASIC expression

Example 1: `IF KEY OR IN(0)=ON THEN GOTO label`

Example 2: `result=10 OR (2.1*9)`

The BASIC 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

**OUTDEVICE**

Type: Process Parameter

Description: The value in this parameter determines the serial output device for the PRINT command. The channel numbers are the same as described in INDEVICE.



## OUTLIMIT

- Type: Axis Parameter
- Description: The output limit restricts the voltage output from a daughter board to a lower value than the maximum which is the default. The voltage output generated by a 12 bit DAC. The output values are normally therefore limited to +2047..-2048.
- Example: `OUTLIMIT AXIS(0)=1023`
- The above will limit the voltage output to a  $\pm 5V$  output range. This will apply to the DAC command if SERVO=OFF or to the voltage output by the servo if SERVO=ON.

## OV\_GAIN

- Type: Axis Parameter
- Description: The output velocity gain is a constant which is multiplied by the change in measured position. Default value is 0. Adding 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  $Kov$  and change in position  $\Delta Pm$ , the contribution to the output signal is:
- $$Oov = Kov \times \Delta Pm$$
- Note: Servo gains have no effect on stepper motor axes.

## PI

- Type: Constant
- Description: PI is the constant equal to 3.1416
- Example: `circum=100`  
`PRINT "Radius=";circum/(2*PI)`

## PMOVE

- Type: Process Parameter
- Modifier: PROC
- Description: Returns 1 if the process move buffer is occupied, and 0 if it is empty. When one of the Motion Coordinator processes encounters a movement command the process loads the movement requirements into the "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 interrupt occurs the periodic 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.

## POWER\_UP

- Type: Flash EPROM stored system variable
- Description: This parameter is used to select the location of programs on power up or software reset. 2 Values are valid:
- 0 - Use program in battery backed RAM
  - 1 - Copy program from FLASH EPROM into RAM
- Programs are individually selected to be run at power up with the RUNTYPE command
- Note: POWER\_UP is always an immediate command and therefore not included in programs.

## PP\_STEP

- Type: Axis parameter
- Description: This parameter allows the incoming raw encoder counts to be multiplied by any **integer** value. This can be used to match encoders to high resolution microstepping motors for position verification or for moving along circular arcs on machines where the number of encoder edges/distance do not match on the axes. PP\_STEP may be negative. Using a negative number will reverse the encoder count.
- Example: A microstepping motor has 20000 steps/rev. The motion coordinator is working in MICROSTEP=ON mode so will internally process 40000 counts/rev. A 2500 pulse encoder is to be connected. This will generate 10000 edge counts/rev. A multiplication factor of 4 is therefore required to convert the 10000 counts/rev to match the 40000 counts/rev of the motor.
- ```
PP_STEP AXIS(3)=4
```
- Example 2: An X-Y machine has encoders which give 50 edges/mm in the X axis (Axis 0) and 75 edges/mm in the Y axis (Axis 1). Circular arc interpolation is required between the axes. This requires that the interpolating axes have the same number of encoder counts/distance. It is not possible to multiply the X axis counts by 1.5 as the PP\_STEP parameter must be an integer. Both X and Y axes must therefore be set to give 150 edges/mm:
- ```
PP_STEP AXIS(0)=3  
PP_STEP AXIS(1)=2  
UNITS AXIS(0)=150  
UNITS AXIS(1)=150
```

## PRINT

Type: Command.

Description: The PRINT command allows the BASIC program to output a series of characters to either the serial ports or to the fibre optic port (if fitted). The PRINT command can output parameters, fixed ascii strings, and single ascii characters. Multiple items to be printed can be put on the same PRINT line provided they are separated by a comma or semi-colon. The comma and semi-colon are used to control the format of strings to be output.

Example 1: `PRINT "CAPITALS and lower case CAN BE PRINTED"`

Example 2: `>>PRINT 123.45,VR(1)`

Suppose `VR(1)=6` and `variab=1.5`: print output will be:

```
123.4500      4.5000
>>
```

Note how the comma separator forces the next item to be printed into the next tab column. The width of the field in which a number is printed can be set with the use of `[w,x]` after the number to be printed. Where `w`=width of column and `x`=number of decimal places.

For example:

```
PRINT VR(1)[4,1];variab[6,2]

6.0  1.50
```

Note that the numbers are right justified in the field with any unused leading characters being filled with spaces. If the number is too big then the field will be filled with asterisks to signify that there was not sufficient space to display the number. The maximum field width allowable is 127.

Example 3: `length: PRINT "DISTANCE=";mpos`

```
DISTANCE=123.0000
```

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. The PRINT command prints variables with 4 digits after the decimal point. The number of decimal places printed can be set by use of `[x]` after the item to be printed. Where `x` is the number of decimal places from 1..4:

```
params: PRINT "DISTANCE=";mpos[0];" SPEED=";v[2];

DISTANCE=123 SPEED=12.34
```

Example 4: `15 PRINT "ITEM ";total" OF ";limit;CHR(13);`

The `CHR(x)` command is used to send individual ASCII characters which are referred to by number. 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 linefeed so the line above would be printed on top of itself if it were the only print statement in a program.

`PRINT CHR(x)`; is equivalent to `PUT(x)` in some other BASICs.

Note: The PRINT statements are normally transmitted to serial port 0. They can be redirected to other output ports by using `PRINT#`.

## **PRINT#n,**

Type: Command

Description: This performs the same function as PRINT but the serial output device is specified as part of the command. The device is selected for the duration of the PRINT# command only. When execution is complete the output device reverts back to that specified by the common parameter OUTDEVICE.

Parameters:

n: Output device:-

- 0 RS-232 port A
- 1 RS-232 port B
- 2 RS-485 port
- 3 Fibre optic port
- 4 Fibre optic port duplicate
- 5 RS-232 port A - channel 5
- 6 RS-232 port A - channel 6
- 7 RS-232 port A - channel 7
- 8 RS-232 port A - channel 8 - reserved for use by MotionPerfect
- 9 RS-232 port A - channel 6 - reserved for use by MotionPerfect

10..24 send text string to fibre optic network node 1..15

Example: `PRINT#10,"SPEED=";SPEED[6.1];`

## **PROC(expression)**

Type: Process Modifier

Description: Allows a process parameter from a particular process to be read or set.

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

## **PROCESS**

Type: System Command

Description: Displays the running status and process number for each current process.

## **PROCMOVE**

Type: Process Parameter (Read only)

Description: This process parameters returns the bitmap of which processes have moves pending.

Example: `' if process 3 has loaded its move then inform the  
' operator  
IF PROCMOVE AND 8 THEN PRINT "Move started"`

**PROCNUMBER**

Type: Process Parameter

Description: Returns the process which it is running on to a BASIC program. This is normally required when multiple copies of a program are running on different processes.

Example: `MOVE (length) AXIS (PROCNUMBER)`

**PSWITCH(*sw,en,[,axis,opno,opst,setpos,rspos]*)**

Type: Command

Description: The PSWITCH command allows an output to be fired when a predefined position is reached, and to go OFF when a second position is reached. There are 8 position switches each of which can be assigned to any axis, and can be assigned ON/OFF positions and OUTPUT numbers.

The command can be used with 2, all 7 parameters. With only 2 parameters a given switch can be enabled or disabled.

Parameters:

*sw*: The switch number in the range 0..15

*en*: Switch enable -  
     1 or ON to enable software PSWITCH  
     0 or OFF to disable software PSWITCH  
     3 to enable hardware PSWITCH (on PSWITCH daughter board)  
     2 to disable hardware PSWITCH

*axis*: Axis number which is to provide the position input in the range 0..number of axes on the controller. For a hardware PSWITCH it should be set to the axis slot number.

*opno*: Selects the physical output to set, should be in range 8..31. For a hardware PSWITCH it should be set to 0..3.

*opst*: Selects the state to set the output to, if 1 then output set ON else set it OFF

*setpos*: The position at which output is set, in user units

*rspos*: 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.

*sw* - The switch number can be any one we chose that is not in use so for the purpose of this example we will use number 0.

*en* - The switch must be enabled to work, therefore this must be set to 1.

*axis* - We are told that the shaft is controlled by axis 0, thus axis is set to 0.

*opno* - We are told that output 11 is the one to fire, so set opno to 11.

*opst* - When the output is set it should be on so set to 1.

*setpos* - The output is to fire at 80° after TDC hence the set position is 80 as we are working in degrees.

*rspos* - 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 BASIC 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.

**Note:** After switching the PSWITCH off, the output may remain ON if the state was ON when the PSWITCH was switched off. The OP() command can be used to force an output OFF:

```
PSWITCH(2,OFF) '   Switch OFF pswitch controlling OP 14
OP(14,OFF)
```

### **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:  $O_p = K_p \times e$
Note:	P_GAIN may be fractional values. The default value is 1.0. Servo gains have no effect on stepper motor axes.
Example:	P_GAIN AXIS(11)=0.25

### **RAPIDSTOP**

Type:	Command
Alternate Format:	RS
Description:	Rapid Stop. The RAPIDSTOP command cancels the currently executing move on all axes. Velocity profiled move types (MOVE, MOVEABS, MOVEMODIFY, FORWARD, REVERSE, MOVECIRC, MHELICAL) will be ramped down. Others will be immediately cancelled. The NEXT move buffers and the process buffers are NOT cleared.

### **READPACKET**

Type:	Command
Description:	Readpacket is used to transmit numbers from an external computer into the global variables of the Motion Coordinator over a serial communications port. The data is transmitted from the PC in binary format with a CRC checksum.
Parameters:	
<i>Port Number</i>	This value should be 0 or 1
<i>Variable Number</i>	This value tells the Motion Coordinator where to start setting the variables in the VR() global memory array.
<i>Number of Vars.</i>	The number of variables to download
<i>Format</i>	The number format for the numbers being downloaded

### **RECORD**

Type:	Command
Description:	The RECORD command is part of the pattern recognition system built into the Motion Coordinator. Following the recording of a sequence of transitions the RECORD command is used to:  <ol style="list-style-type: none"> <li>1 - Reduce the number of transitions to a number defined by the programmer</li> <li>2 - Store the transition pattern for subsequent comparison with MATCH</li> </ol>
Note:	See the MATCH command for an example of a complete recognition sequence.

## REG\_MATCH

- Type: Axis Parameter
- Description: Indicates to a programmer the quality of the fit of a RECORD / MATCH sequence. A value of 1 is returned if the fit is 100%.
- Note: See the MATCH command for an example of a complete recognition sequence.

## REG\_POS

- Type: Axis Parameter
- Alternate Format: RPOS
- Description: Stores the position at which a registration mark was seen on each axis in user units. See REGIST() for more details.
- 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)



## REGIST(mode,{distance})

Type:	Command
Description:	The regist command captures an axis position when it sees the registration input or the z mark on the encoder. The capture is carried out by hardware so software delays do not affect the accuracy of the position capture. The capture is initiated by executing the REGIST() command. If the input or z mark is seen as specified by the mode with in the specified window the MARK parameter is set TRUE and the position is stored in REG_POS.
Parameters:	
<i>mode</i> :	Determines the position to capture:  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 - Sets pattern recognition mode
<i>distance</i> :	The distance parameter is used for the pattern recognition mode ONLY, and specifies the distance over which to record transitions,  Add 256 to the above <b>mode</b> values to apply inclusive windowing function:  When the windowing function is applied signals will be ignored if the axis measured position is not in the range:  Greater than OPEN_WIN and Less than CLOSE_WIN  Add 768 to the above values to apply exclusive windowing function:  When the windowing function is applied signals will be ignored if the axis measured position is not in the range:  Less than OPEN_WIN or Greater than CLOSE_WIN
Note:	The REGIST command must be re-issued for each position capture.
Example:	<pre>catch:    REGIST(3+256)           WAIT UNTIL MARK           PRINT "Registration Input Seen at: ";REG_POS</pre>

## 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.
Example:	To change the speed to a slower value 5mm from the end of a move.  <pre>start:    SPEED=10           MOVE ( 45)           WAIT UNTIL REMAIN&lt;5           SPEED=1           WAIT IDLE</pre>

## RENAME

- Type: System Command
- Description: Renames a program in the Motion Coordinator directory.
- Example: >>RENAME car voiture
- Note: Motion Perfect users should use "Rename Program..." under the "Program" menu to perform a RENAME command.

## REPDIST

- Type: Axis Parameter
- Description: The repeat distance contains the allowable range of movement for an axis before the position count overflows or underflows. For example, when an axis executes a FORWARD move the demand and measured position will continually increase. When the measured position reaches the REPDIST twice that distance is subtracted to ensure that the axis always stays in the range -REPEAT DISTANCE to +REPEAT DISTANCE (Assuming REP\_OPTION=OFF). The Motion Coordinator will adjust its absolute position without affecting the move in progress or the servo algorithm.

## REP\_OPTION

- Type: Axis Parameter
- Description: Bit 0 of the the REPDIST parameter can be applied in two ways. In the default setting (REP\_OPTION=OFF) operation is selected in the range -REPEAT DISTANCE to +REPEAT DISTANCE. In some circumstances it is more convenient for the axis positions to be specified from 0 to +REPEAT DISTANCE. (REP\_OPTION=ON)
- REP\_OPTION bit 1 is set ON to switch OFF the automatic repeat option of the CAMBOX function. When the system software has set the option OFF it automatically clears bit 1 of REP\_OPTION.

## REPEAT commands UNTIL condition

- Type: Command
- Description: The REPEAT..UNTIL construct allows a block of commands to be continuously repeated until a condition becomes TRUE. 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.

```
cycle:  SPEED=1000
        REPEAT
        MOVE(100)
        WAIT IDLE
        WA(500)
        UNTIL IN(4)=ON
```

## RESET

- Type: Command
- Description: Sets the value of all the local named variables of a BASIC process to 0.

## RETURN

Type: Command

Description: Instructs BASIC to return from a subroutine. Execution continues at the line following the GOSUB instruction.

Note: Subroutines on each process can be nested up to 8 deep.

Example:

```
' calculate in subroutine:
      GOSUB calc
      PRINT "Returned from subroutine"
      STOP

calc:  x=y+z/2
      RETURN
```

## REV\_IN

Type: Axis Parameter

Description: This parameter holds the input number to be used as a reverse limit input. The input should be in the range 0..31. If REV\_IN is set to -1 then no input is used as a reverse limit. When the reverse limit input is asserted moves going in the reverse direction will be cancelled. The axis status bit 5 will also be set.

**Note: Feedhold, forward,reverse and datum inputs are ACTIVE LOW.**

## REV\_JOG

Type: Axis Parameter

Description: This parameter holds the input number to be used as a reverse jog input. The input should be in the range 0..31. If REV\_JOG is set to -1 then no input is used as a reverse jog. When the reverse jog input is asserted then the axis is moved forward at the JOGSPEED or axis SPEED depending on the status of the FAST\_JOG input.

**Note: Feedhold, forward,reverse and datum inputs are ACTIVE LOW.**

## REVERSE

Type: Motion Command

Alternate Format: RE

Description: Sets continuous reverse movement on the specified or base axis.

Parameters: None.

Note: The reverse motion can only be stopped by issuing a CANCEL or by hitting the reverse, inhibit or datum limits.

Example:

```
back:  REVERSE
      'Wait for stop signal:
      WAIT UNTIL IN(0)=ON
      CANCEL
```

## **RS\_LIMIT**

Type: Axis Parameter

Alternate Format: RSLIMIT

Description: An end of travel software 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 reverse travel limit in user units. When the limit is hit the controller will ramp down the speed to zero then cancel the move. Bit 10 in the axis status parameter is set when the axis is in the RS\_LIMIT

## **RUN**

Type: Command

Description: Runs a program on the controller.

Parameter: A program name, and process number may optionally be specified.

Note: Execution continues until:

There are no more lines to execute

or HALT is typed at the command line. This stops all programs

or STOP "name" is typed at the command line. This stops single program

**RUN may be included in a program to run another program:**

```
RUN "cycle"
```

Example: RUN(this will run currently selected program)

Example 2: RUN "sausage" (this will run the named program)

Example 3: RUN "sausage",3 (run named program on a particular process)

## **RUN\_ERROR**

Type: Process Parameter

Modifier: PROC

Description: Contains the number of the last BASIC error that occurred on the specified process.

Example: 

```
>>? RUN_ERROR PROC(5)
9.0000
>>
```

---

**RUNTYPE**

Type: System Command

Description: Sets whether program is run automatically at power up, and which process it is to run on. The current status of each programs RUNTYPE is displayed when a DIR command is performed.

Parameters:

*program name* Can be in inverted commas or without

*autorun* 1 to run automatically, 0 for manual running

*<process number>* optional to force process number

Example: >>RUNTYPE progname,1,10

- Sets program "progname" to run automatically on power up on process 10

>>RUNTYPE "progname",0

- Sets program "progname" to manual running

Note: To set the RUNTYPE using Motion Perfect select the "Set Power-up mode" option in the "Program" menu.

**Note 2:** **The RUNTYPE information is stored into the flash EPROM only when an EPROM command is performed.**

## SCOPE

Type: System Command

Description: The SCOPE command is used to program the system to automatically store up to 4 parameters every sample period. The sample period can be any multiple of the servo period. The data stored is put in the TABLE data structure. It may then be read back to a PC and displayed on the *Motion Perfect* Oscilloscope or stored to a file for further analysis using the "Create TABLE file" option under the "File" menu. Motion Perfect uses the SCOPE command when running the Oscilloscope function.

Parameters:

*ON/OFF control* Set ON or OFF to control the SCOPE function. OFF implies that the scope data is not ready. ON implies that the scope data is loaded correctly and is ready to run when the TRIGGER command is executed. 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

*P0* first parameter to store

*P1* optional second parameter to store

*P2* optional third parameter to store

*P3* optional fourth parameter to store

Example 1: `SCOPE(ON,10,0,1000,MPOS AXIS(5), DPOS AXIS(5))`

This example programs the SCOPE facility to store away the MPOS axis 5 and DPOS axis 5 every 10 milliseconds. The MPOS will be stored in table values 0..499, the DPOS in table values 500 to 999. Note that the SCOPE facility will wrap around and start storing at the beginning again unless stopped. The sampling does not start until the TRIGGER command is executed.

Example 2: `SCOPE(OFF)`

## SCOPE\_POS

Type: System Parameter

Description: Returns the current TABLE position at which the SCOPE function is currently storing its first parameter.

## SELECT

- Type: System Command
- Description: Selects the current active program for editing, running, listing etc. SELECT makes a new program if the name entered is not a current program.
- When a program is SELECTed the commands EDIT, RUN, LIST, NEW etc assume that the SELECTed program is the one to operate with unless a program is specified as in for example: RUN progname
- When a program is selected any previously selected program is compiled.
- Note: The SELECTed program cannot be changed when programs are running.
- Note 2: Motion Perfect automatically SELECTs programs when you click on their entry in the list in the control panel.**

## SEND(*n,type,data1[,data2]*)

- Type : Command
- Description: Outputs a fibre-optic network message of a specified type to a given node.
- Parameters:
- n*: Number from 10 to 24 defining the destination node.
- type*: Message type:
- 1 - Direct variable transfer
  - 2 - Keypad offset
- data1*: If message type 1, data1 is the numbered variable number, 0..250, on the destination Motion Coordinator to modify. If message type 2: data1 is the number of nodes from the keypad that the key characters are to be sent. In the range 10..24, where 10 is the next node and 24 is the fifteenth node away from the keypad.
- data2*: Only used if message is type 1. In this case it contains the value to change the specified variable to.
- Example 1: Two Motion Coordinators are fibre-optic networked together. One is acting under instruction from the other. Instructions are given by setting variable 100 to different values and the receiving Motion Coordinator jumping to a subroutine determined by the variable value.
- The program on the controlling Motion Coordinator would have the following send routine:
- ```
SEND(10,1,100,value)' Set vr(100) on dest. to value
```

Example 2: Any network containing membrane keypad(s) must initialise the keypads first to tell them where to send their output and to set them into network mode. To do this a keypad offset message is sent to the membrane keypad. Consider a network with four nodes. Three Motion Coordinators and one membrane keypad connected as follows:

MCa ---> MCB---> MCC ---> Keypad ---> (ring made back to MCa)

MCa is to initialise the keypad (Offset of 0,1,2 from MCa) but MCC is to receive the keypad output (Offset of 0,1,2 from Keypad to MCC) .

SEND(10+2,2,10+2)

## 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 a voltage dependent on the DAC parameter. When SERVO=ON the axis hardware will output a voltage dependent on the gain settings and the following error.

SERVO is also used on stepper axes with position verification. If SERVO=ON the system software will compare the difference between the DPOS and MPOS (FE) on the axis with the FE\_LIMIT. If the difference exceeds the limit the following error bit is set in the AXISSTATUS register, the enable relay is forced OFF and the servo is set OFF. If the SERVO=OFF on a stepper verification axis the FE is not compared with the FE\_LIMIT.

Example: SERVO AXIS(0)=ON' Axis 0 is under servo control  
SERVO AXIS(1)=OFF' Axis 1 is run open loop

Note: Stepper axes with position verification need consideration also of VERIFY and PP\_STEP.

## SERVO\_PERIOD

Type: System Parameter

Description: This parameter allows the controller servo period to be specified in seconds to a value other than 0.001 seconds - the default however many axes are functioning. The default value is suitable for the large majority of systems. Servo periods below 1 millisecond are of little practical benefit for servo motor systems whose time constants are substantially above 1 millisecond.

Note: The servo period must not be adjusted if there are any stepper axes being controlled.

**Note 2: The servo period can only be adjusted in a small range. Too small values prevents the system software from functioning. Too large values allow the hardware watchdog to drop out.**



**SETCOM(baudrate, databits, stopbits, parity,[port number], [XON/XOFF switch])**

Type: Command

Description: Permits the serial communications parameters to be set by the user. By default the controller set the RS232-C port to 9600 baud, 7 data bits, 2 stop bits and even parity.

Parameters:

*baudrate:* 1200, 2400,4800, 9600,19200 or 38400

*databits:* 7or 8

*stopbits:* 1or 2

*parity:* 0=none, 1=odd, 2=even

*port number:* 0 or 1 (If not specified this parameter defaults to 0)

*xon/xoff switch:* This switch is available on serial port B (Channel #1) ONLY. 1=active,0=off

**Note:** **On power up the controllers always set the serial ports to 9600,7,2,even,XON/XOFF active.**

**SGN(expression)**

Type: Function

Description: The SGN function returns the SIGN of a number.

|                   |              |
|-------------------|--------------|
| Positive non-zero | : returns 1  |
| Zero              | : returns 0  |
| Negative          | : returns -1 |

Parameters:

*expression:* Any valid TRIO BASIC expression.

Example: 

```
>>PRINT SGN(-1.2)
-1.0000
>>
```

**SIN(expression)**

Type: Function

Description: Returns the SINE of an expression. This is valid for any value in expressed in radians.

Parameters:

*expression:* Any valid TRIO BASIC expression.

Example: 

```
>>PRINT SIN(0)
0.0000
>>
```

### SP(speed)

Type: Command - **Use the SPEED axis parameter for new applications.**

Description: This format is only provided to simplify compatibility with earlier controllers. Sets demand speed of the current or base axis.

### SPEED

Type: Axis Parameter

Description: The SPEED axis parameter can be used to set/read back the demand speed axis parameter. The speed is returned in units/s. The demand speed is the speed ramped up to during the movement commands MOVE, MOVEABS, MOVECIRC, FORWARD, REVERSE, MHELICAL and MOVEMODIFY.

Example: 

```
SPEED=1000
PRINT "Speed Set=" ;SPEED
```

### SQR(number)

Type: Function

Description: Returns the square root of a number. Parameters:

number: Any valid TRIO BASIC number or variable.

Example: 

```
>>PRINT SQR(4)
2.0000
>>
```

### SRAMP

Type: Axis Parameter

Description: This parameter stores the s-ramp factor. This controls the amount of rounding applied to trapezoidal profiles. 0 sets no rounding. 10 maximum rounding. Using S ramps increases the time required for the movement to complete. S RAMP can be used with MOVE, MOVEABS, MOVECIR, MHELICAL, FORWARD, REVERSE and MOVEMODIFY move types.

**Note: The S ramp factor should not be changed while a move is in progress.**

### SSI\_BITS

Type: Axis Parameter

Description: This parameter is only used with the SSI Absolute daughter board. It is used to set the number of data bits to be clocked out of the encoder by the axis hardware. The maximum permitted value is 24. The default value is 0 which will cause no data to be clocked from the SSI encoder, users MUST therefore set a value depending on the encoder type.

Example: 

```
SSI_BITS AXIS(3)=12
SSI_BITS AXIS(7)=21
```

## STEP

Type: Command

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: Command

Description: Steps one line in a program. If no parameters are supplied then the program running on the process the STEPLINE command is executed on will be stepped, unless this is the COMMAND LINE process, in which case the currently SELECTed program will be stepped. If the "program" is specified then all occurrences of this program will be stepped, if there is no copy of the program running then one will be started on the next available process. If the process is specified as well then only the copy of the specified program running on the specified process will be stepped, if there is no copy of the program running on this process then one will be started on it.

Parameters:

*{Program name}*

*{Process number}*

Example: >>STEPLINE "conveyor"

Example 2: >>STEPLINE "maths" ,2

## STOP

Type: Command

Description: Stops one program at the current line. A particular program may be specified or the selected program will be assumed.

Example 1: >>STOP progname

Example 2: 

```
`DO NOT EXECUTE SUBROUTINE AT label
      STOP
label: PRINT var
      RETURN
```

## STORE

- Type: System Command
- Description: Stores an update to the system software into FLASH EPROM. This should only be necessary following loading an update to the system software supplied by TRIO. See also LOADSYSTEM.
- Warning:** **If the controller power is removed during a STORE sequence can lead to the controller having to be returned for re-initialization.**
- Note: Use of STORE and LOADSYSTEM is automated for Motion Perfect users by the "Load system software..." option in the "Controller" menu.

## TABLE(address [, data1..data20])

- Type: Command
- Description: The TABLE command is used to load and read back the internal cam table. This table has a fixed maximum table length of 16000 points. Issuing the TABLE command or running it as a program line must be done before table points are used by a CAM or CAMBOX command. The table values are floating point and can therefore be fractional.
- The command has two forms:
- (i) With 2 or more parameters the TABLE command defines a sequence of values, the first value is the first table position.
  - (ii) If a single parameter is specified the table value at that entry is returned. As the table can be written to and read from, it may be used to hold information as an alternative to variables.
- The values in the table may only be read if a value of THAT NUMBER OR GREATER has been specified.** For example, if the value of table position 1000 has been specified e.g. TABLE(1000,1234) then TABLE(1001) will produce an error message. The highest TABLE which has been loaded can be read using the TSIZE parameter.
- The table entries are automatically battery backed. It is recommended to set the values inside a program if FLASH Eprom storage is required. It is not normally required to delete the table but if this necessary the DEL command can be used:
- ```
>>DEL "TABLE"
```
- Parameters:
- address:* location in the table at which to store a value or to read a value from if only this parameter is specified.
- data1..data20:* the value to store in the given location and at subsequent locations if more than one data parameter is used.

Example 1:           TABLE(100,0,120,250,370,470,530,550)

This loads the internal table:

Table Entry:	Value:
100	0
101	120
102	250
103	370
104	470
105	530
106	550

Example 2:           >>PRINT TABLE(1000)

**Note:**           **The SCOPE function of Motion Perfect uses the table as a data area. The range used can be set in the scope "Options..." screen. Care should be taken not to use a data area in use by the Oscilloscope function.**

**TABLEVALUES(first table number, last required table number, format)**

Type:                Command

Description:       Returns a list of table points starting at the number specified. There is only one format supported at the moment, and that is comma delimited text.

Parameters

*address:*           Number of the first point to be returned

*number of points:* Total number of points to be returned

*format:*            Format for the list

**Note:**           **TABLEVALUES is provided mainly for Motion Perfect to allow for fast access to banks of TABLE values.**

**TAN(expression)**

Type:                Mathematical Function

Description:       Returns the TANGENT of an expression. This is valid for any value expressed in radians. Parameters:

Expression:        Any valid TRIO BASIC expression.

Example:            >>PRINT TAN(0.5)  
0.5463  
>>

## THEN

Type: Command

Description: Forms part of an IF expression. See IF for further information.

Example:

```
IF MARK THEN
    offset=REG_POS
ELSE
    offset=0
ENDIF
```

**Note:** Comments should not be placed after a THEN statement

## TICKS

Type: Process Parameter

Description: The current count of the process clock ticks is stored in this parameter. The process parameter is a 32 bit counter which is DECREMENTED on each servo cycle. It can therefore be used to measure cycles times, add time delays, etc. The ticks parameter can be written to and read.

Example:

```
delay:    TICKS=3000
          OP(9,ON)
test:     IF TICKS<=0 THEN OP(9,OFF) ELSE GOTO test
```

Note: TICKS is held independently for each process.

## TIME\$

Type: Command (MC2/MC216 only)

Description: Prints the current time as defined by the real time clock as a string in 24hr format.

Example:

```
>>? TIME$
14:39:02
>>
```

## TIME

Type: System Parameter (MC2 only/MC216)

Description: Returns the time from the real time clock. The time returned is the number of seconds since midnight 24:00 hours.

## TO

Type: Program Structure

Description: Precedes the end value of a FOR..NEXT loop.

Example:

```
FOR x=10 TO 0 STEP -1
    PRINT x
NEXT x
```

## TRANS\_DPOS

- Type: Axis Parameter
- Description: 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. For some machinery configurations it can be useful to install a frame transformation which is not 1:1, these are typically machine such as robotic arms or machines with parasitic motions on the axes. Frame transformations have to be specially written in the "C" language and downloaded into the controller. It is essential to contact Trio in order to install frame transformations.
- Note: See also FRAME

## TRANSITIONS

- Type: Axis Parameter
- Description: Records the number of register input transitions in a REGIST(5,length) sequence
- Note: See also REGIST, RECORD, MATCH

## TRIGGER

- Type: System Command
- Description: Starts a previously set up SCOPE command
- Note: Motion Perfect uses TRIGGER automatically for its oscilloscope function.

## TROFF

- Type: System Command
- Description: Suspends the trace facility at the current line and resumes normal program execution. A program name can be specified or the selected program will be assumed.
- Example: >>TROFF "lines"

## **TRON**

- Type: System Command
- Description: The trace on command suspends a programs execution at the current line. The program can then be single stepped, executing one line at a time, using the STEPLINE command.
- Note: Program execution may be restarted without single stepping using TROFF.
- The trace mode may be halted by issuing a STOP or HALT command.
- Motion Perfect* highlights lines containing TRON in its editor and debugger.
- Example: 

```
TRON
MOVE(0,10)
MOVE(10,0)
TROFF
MOVE(0,-10)
MOVE(-10,0)
```

## **TRUE**

- Type: Constant
- Description: The constant TRUE takes the numerical value of -1.
- Example: 

```
test:    t=IN(0) AND IN(2)
          IF t=TRUE THEN
              PRINT "Inputs are on"
          ENDIF
```

## **TSIZE**

- Type: System Parameter
- Description: Returns one more than the highest currently defined table value.
- Example: 

```
>>TABLE(1000,3400)
>>PRINT TSIZE
1000.0000
```
- Note: TSIZE can be reset using >>DEL "TABLE"



## UNITS

- Type: Axis Parameter
- Description: The unit conversion factor sets the number of encoder edges/stepper pulses in a user unit. The motion commands to set speeds, accelerations and moves use the UNITS parameter to allow values to be entered in more convenient units E.G.: mm for a move or mm/sec for a speed.
- Note: Units may be any positive value but it is recommended to design systems with an integer number of encoder pulses/user unit.
- 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. One rev is equal to 5mm therefore there are  $4000/5 = 800$  edges/mm so:
- ```
>>UNITS=1000*4/5
```
- Example 2: A stepper motor has 180 pulses/rev and is being used with MICROSTEP=OFF  
To program in revolutions the unit conversion factor will be:
- ```
>>UNITS=180*16
```
- Note: Users with stepper axes should also read the MICROSTEP command when choosing UNITS.**

## UNTIL

- Type: Program Structure
- Description: Defines the end of a REPEAT ..UNTIL multi-line loop, or part of a WAIT UNTIL loop. After the UNTIL statement is a condition which decides if program flow continues on the next line or at the REPEAT statement
- Example:
- ```
' loop loading a CAMBOX move each time Input 0 comes on, 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
```

## VERIFY

- Type: Axis Parameter
- Description: The verify axis parameter is used to select different modes of operation on a stepper encoder axis.
- VERIFY=OFF  
Encoder count circuit is connected to the STEP and DIRECTION hardware signals so that these are counted as if they were encoder signals. This is particularly useful for registration as the registration circuit can therefore function on a stepper axis.
- VERIFY=ON  
Encoder circuit is connected to external A,B, Z signals
- Example: VERIFY AXIS(3)=ON

## VERSION

Type: System Parameter

Description: Returns the version number of the BASIC installed on the Motion Coordinator.

Example: 

```
>>? VERSION
1.4000
>>
```

## VFF\_GAIN

Type: Axis Parameter

Description: The velocity feed forward gain is a constant which is multiplied by the change in demand position. Adding velocity feed forward gain to a system decreases the following error during a move by increasing the output proportionally with the speed. For a velocity feed forward term  $Kvff$  and change in position  $\Delta Pd$ , the contribution to the output signal is:

$$Ovff = Kvff \times \Delta Pd$$

Note: Servo gains have no effect on stepper motor axes.

## VIEW

Type: System Command

Description: Lists the currently selected program in tokenised and internal compiled format.

Example: For the following program:

```
VR(10)=IN AND 255
```

the view command will give the following output:

```
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:
```

## VP\_SPEED

Type: Axis Parameter

Alternate Format: VPSPEED

Description: The velocity profile speed is an internal speed which is ramped up and down as the movement is velocity profiled. It is reported in user units/sec.

Example: 

```
' Wait until at command speed:
MOVE(100)
WAIT UNTIL SPEED=VP_SPEED
```

## VR(expression)

Type: Command

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. There are 251 variable locations which are accessed as variables 0 to 250.

The numbered variables are used for several purposes in Trio BASIC. **If these requirements are not necessary it is better to use a named variable:**

The numbered variables are BATTERY BACKED and are not cleared between power ups. - 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.**

The numbered variables can be changed by remote controllers on the TRIO Fibre Optic Network.

The numbered variables can be used for the LINPUT, READPACKET and CAN commands.

Example 1: ' put value 1.2555 into VR() variable 15. Note local variable ' 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:    ` MOVE TO FIRST PUT DOWN POSITION:
          MOVEABS(115)
          FOR VR(0)=101 TO 110
            IF VR(VR(0))=0 THEN GOSUB load
            ` 200 IS SPACING BETWEEN POSITIONS:
            MOVE(200)
          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
          RETURN
```

Note: The variables are battery-backed so the program here could be designed to store the state of the machine when the power is off. It would of course be necessary to provide a means of resetting completely following manual intervention.

Example 3: loop: 'Assign VR(65) to VR(0) multiplied by  
'Axis 1 measured position  
VR(65)=VR(0)\*MPOS AXIS(1)  
PRINT VR(65)

### **WA(time)**

Type: Command

Description: Holds up program execution for the number of milliseconds specified in the parameter.

Parameters:

*time*: The number of milliseconds to wait for.

Example: 

```
OP(11,OFF)
WA(2000)
OP(17,ON) This turns output 17 off 2 seconds after switching
output 11 off.
```

### **WAIT IDLE**

Type: Command

Description: Suspends program execution until the base axis has finished executing its current move and any further buffered move.

Note: This does not necessarily imply that the axis is stationary in a servo motor system.

Example: 

```
MOVE(100)
WAIT IDLE
PRINT "Move Done"
```

### **WAIT LOADED**

Type: Command

Description: Suspends program execution until the base axis has no moves buffered ahead other than the currently executing move

Note: This is useful for activating events at the beginning of a move, or at the end of a move when multiple moves are buffered together.

Example: 

```
' Switch output 45 ON at start of MOVE(350) and OFF at the end

MOVE(100)
MOVE(350)
WAIT LOADED
OP(45,ON)
MOVE(200)
WAIT LOADED
OP(45,OFF)
```

### WAIT UNTIL condition

Type: Command

Description: Repeatedly evaluates the condition until it is true then program execution continues.

Parameters:

*condition:* A valid TRIO BASIC logic expression.

Example 1: 

```
WAIT UNTIL MPOS AXIS(0)>150
MOVE(100) AXIS(7)
```

In this example the program waits until the measured position on axis 0 exceeds 150 then starts a movement on axis 7.

Example 2: The expressions evaluated can be as complex as you like provided they follow the TRIO BASIC syntax, for example:

```
WAIT UNTIL DPOS AXIS(2)<=0 OR IN(1)=ON
```

This waits until demand position of axis 2 is less than or equal to 0 or input 1 is on.

### WDOG

Type: System Parameter

Description: Controls the WDOG relay contact used for enabling external amplifiers. The WDOG=ON command MUST be issued in a program prior to executing moves. It may then be switched ON and OFF under program control. If however a following error condition exists on any axis the system software will override the watchdog contact OFF.

Example: 

```
WDOG=ON
```

Note: WDOG=ON / WDOG=OFF is issued automatically by Motion Perfect when the "Drives Enable" button is clicked on the control panel

### WEND

Type: Program Structure

Description: Marks the end of a WHILE..WEND loop. See WHILE

Note: WHILE..WEND loop can be nested without limit other than program size.

**WHILE condition**

Type: Command

Description: The commands contained in the WHILE..WEND loop are continuously executed until the condition becomes FALSE. Execution then continues after the WEND.Parameters:

condition: Any valid logical TRIO BASIC expression

Example: 

```
WHILE IN(12)=OFF
    MOVE(200)
    WAIT IDLE
    OP(10,OFF)
    MOVE(-200)
    WAIT IDLE
    OP(10,ON)
WEND
```

**expression1 XOR expression2**

Type: Logical and bitwise operator

Description: This performs an XOR function between corresponding bits of the integer part of two valid TRIO BASIC expressions. It may therefore be used as either a bitwise or logical condition.

The XOR function between two bits is defined as follows:

|     |   |   |
|-----|---|---|
| XOR | 0 | 1 |
| 0   | 0 | 1 |
| 1   | 1 | 0 |

Parameters:

*Expression1:* Any valid TRIO BASIC expression

*Expression2:* Any valid TRIO BASIC expression

Example: `a=10 XOR (2.1*9)`

The BASIC 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 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.
```

**' (comment field)**

Type: Command

Description: A single ' is used to mark a line as being a comment only with no execution significance.

Note: The REM command of other BASICs is replaced by '. Like REM statements ' must be at the beginning of the line or statement or at the end of a line. 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.**

Example: 

```
'PROGRAM TO ROTATE WHEEL
turns=10
'turns CONTAINS THE NUMBER OF TURNS ON FLYWHEEL REQD
MOVE(turns)
```

**: (statement separator)**

Type: Command

Description: The colon is used to separate BASIC statements on a multi-statement line. The only limit to the number of statements on a line is the maximum of 79 characters per line.

Example: 

```
PRINT "THIS LINE":GET low:PRINT "DOES THREE THINGS!"
```

Note: 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 following statements will not be executed:

```
PRINT "Hello":GOTO Routine:PRINT "Goodbye"
```

Goodbye will not be printed.

Similarly because subroutine calls return to the following LINE.

The code:

```
PRINT "Hello":GOSUB sub:PRINT "Goodbye"
```

This will print "Hello" only.





## 9.0 Example Programs

### 9.1 Example Subroutine 1 - Fetching an Integer Value from the Membrane Keypad

#### 9.1.1 Description

The subroutine "getval" fetches an integer value from the membrane keypad in variable "num". The routine prints the number on the display bottom line at cursor position 70, although this can be set to other values. Only the number keys, the "-" key, the "CLR" key and the ENTER key are used. Other keys are ignored.

#### 9.1.2 Program Listing

' Demonstrate integer number entry via Membrane Keypad:

```
getnum:   pos=40
          num=0
          PRINT#4,CHR(20);
          REPEAT
            PRINT#4,CURSOR(pos);num[6,0];
            GET#4,k
            IF k=69 THEN GOTO getnum
            IF k>=59 AND k<=61 THEN k=k-7
            IF k>=66 AND k<=68 THEN k=k-17
            IF k=71 THEN k=48
            IF k>47 AND k<58 THEN
              k=k-48
              num=num*10+k
            ENDIF
          UNTIL k=73
RETURN
```

### 9.2 Example Subroutine 2 - Fetching a Real Value from the Membrane Keypad

#### 9.2.1 Description

This similar routine also fetches a number from the membrane keypad, but this number can have up to 2 decimal places. Note how this example uses the emulated keypad from Motion Perfect.

#### 9.2.2 Program Listing

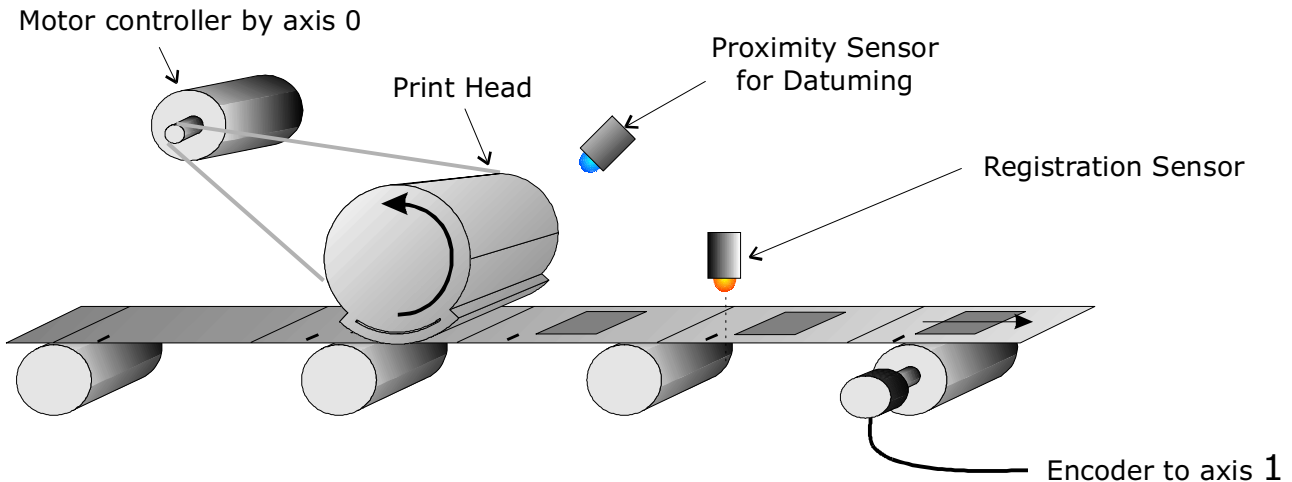
```
getnum:   pos=40
          dpoint=0
          num=0
          negative=1
          PRINT#5,CHR(20);
          REPEAT
            PRINT#5,CURSOR(pos);num*negative[8,2];
            GET#5,k
            IF k=72 AND dpoint=0 THEN dpoint=1
            IF k=70 THEN negative=-negative
            IF k=69 THEN GOTO getnum
            IF k>=59 AND k<=61 THEN k=k-7
            IF k>=66 AND k<=68 THEN k=k-17
            IF k=71 THEN k=48
            IF k>47 AND k<58 THEN
              k=k-48
              IF dpoint>0 THEN
                dpoint=dpoint/10
                IF dpoint>=0.01 THEN num=num+k*dpoint
              ELSE
                num=num*10+k
              ENDIF
            ENDIF
          UNTIL k=73
          num=num*negative
RETURN
```

## 9.3 Example Program 3 - Rotating Printhead with Registration

### 9.3.1 Description

A rotating printhead prints a number on a conveyor mounted product. During printing the print head must be synchronized with the conveyor. The print position must be registered to be relative to a registration mark.

The program achieves the motion profile by:



- 1 Making a synchronization gearbox connection between the conveyor and the print head with CONNECT. This will let the printer make a print on the conveyor. The distance between the prints will be the peripheral distance of the printhead rotation.
- 2 Datuming and setting the repeat distance REPDIST for the print head rotation so that it has an absolute position of zero every time it touches the conveyor.
- 3 Superimposing a movement onto the printhead. This movement has two functions: To adjust the printer position to keep the system in register and to adjust by the difference between the peripheral distance of the printer and the registration marks on the conveyor.
- 4 The superimposed movement is run on axis 3 of the controller (an "imaginary" axis) and the movement summed with the ADDAX command. The move is performed when the printer is going "over the top" of its stroke.

### 9.3.2 Program Listing

```

' Example Program:
' Rotating Head with Registration:

start: GOSUB initial

loop:  WAIT UNTIL MPOS>100'           Wait 100mm past print
      IF MARK THEN
          ' mark seen during last cycle: Limit adjust to 10mm
          r_adj=REG_POS*0.5'         Apply 50% of error
          IF ABS(r_adj)>10 THEN r_adj=SGN(r_adj)*10
          OP(8,OFF)
      ELSE
          ' mark not seen last cycle: Set Zero adjust
          r_adj=0
          OP(8,ON)'                 light "no register" warning lamp
      ENDIF

      BASE(3)'                     Correction on axis 3
      MOVELINK(75-r_adj,150,25,25,1)' Move linked to conveyor
      WAIT IDLE
      BASE(0)
      REGIST(3)
      GOTO loop

initial:

      BASE(0)'                     Setup axis 0
      UNITS=20'                   Edges/mm
      P_GAIN=50
      REP_DIST=200'               200mm=180 degrees
      SERVO=ON

      BASE(1)'                     Setup axis 1
      SERVO=OFF
      UNITS=15'                   Edges/mm on conveyor

      BASE(3)'                     Setup axis 3
      UNITS=20'                   Match axis 0 units

' Datum axis 0 keeping printer in sync with paper:

      WDOG=ON
      BASE(0)
      CONNECT(20/15,1)
      WAIT UNTIL IN(0)=ON'       Wait for prox
      DEFPOS(-150)
      ADDAX(3)'                   Add moves on axis 3

      RETURN

```

## 9.4 Example Program 4 - Motion Coordinator SERIES 2 programs sharing data

### 9.4.1 Description

These two programs run multi-tasking on the SERIES 2 controller. The Motion Cycle program performs a movement. The Operator Interface program communicates with a membrane keypad to control the Motion Cycle program. In this simple example of multi-tasking the two tasks communicate via two global variables.

```
VR(start)      -   This holds the start/stop signal
VR(length)     -   This holds the movement length
```

### 9.4.2 Operator Interface Program

```
'
'
' Motion Coordinator SERIES 2 Demonstration Program
'
'
start: GOSUB initial

loop:
  PRINT #3,CHR(20);CHR(14);
  PRINT #3,CURSOR(0);">LENGTH:";in_length[0];
  IF VR(start)=ON THEN
    PRINT #3,CURSOR(15);"STOP<";
  ELSE
    PRINT #3,CURSOR(15);" RUN<";
  ENDIF

  GET #3,kp
  PRINT kp
  IF kp=53 THEN GOSUB input_length
  IF kp=54 THEN VR(start)=1-VR(start)

GOTO loop

input_length:
  PRINT #3,CURSOR(60);"New Length:";
  GOSUB getval
  IF num>=smallest_len THEN
    in_length=num
  ELSE
    PRINT #3,CURSOR(60);"Min. Length=200mm";
    WA(1000)
  ENDIF

RETURN
```

```

getnum:  pos=40
         num=0
         PRINT#4,CHR(20);
         REPEAT
             PRINT#4,CURSOR(pos);num[6,0];
             GET#4,k
             IF k=69 THEN GOTO getnum
             IF k>=59 AND k<=61 THEN k=k-7
             IF k>=66 AND k<=68 THEN k=k-17
             IF k=71 THEN k=48
             IF k>47 AND k<58 THEN
                 k=k-48
                 num=num*10+k
             ENDIF
         UNTIL k=73
RETURN

```

```

initial:
    PRINT #3,CHR(20);CHR(14);
    PRINT #3,CURSOR(0);
    PRINT #3," Demonstration of "
    PRINT #3," Motion Coordinator"
    PRINT #3," SERIES 2          "
    WA(3000)

    ' Set Global Variable Pointers:

    start=0
    length=1

    ' Set any none zero local variables:

    in_length=VR(length)
    VR(start)=0

    RUN "cycle"

RETURN

```

### 9.4.3 Motion Cycle Program

```

'
' Motion Cycle for SERIES 2 demonstration program:
'
'
    GOSUB setvar
    GOSUB initial

loop:  WAIT UNTIL VR(start)=ON
       MOVE(VR(length)
       MOVEABS(0)
       GOTO loop

'

```

initial:

```
WDOG=ON
WA(100)

BASE(0)
P_GAIN=0.8
FE_LIMIT=1000
SERVO=ON
ACCEL=1000000
DECEL=100000
SPEED=10000
```

RETURN

setvar:

```
' Define Global Variable Pointers:
start=0
length=1
```

RETURN

## 10.0 Motion Perfect

### 10.1 What is *Motion Perfect*?

*Motion Perfect* is Trio's PC "Windows" based motion application development environment. It provides a full suite of tools to facilitate rapid application development and commissioning of *Motion Coordinator* MC216, MC204 and MC2 controlled servo systems.

*Motion Perfect* is designed to run on an IBM compatible PC, with Microsoft Windows 3.1 or higher.

### 10.2 What can *Motion Perfect* do for me ?

Having installed *Motion Perfect*, you can:

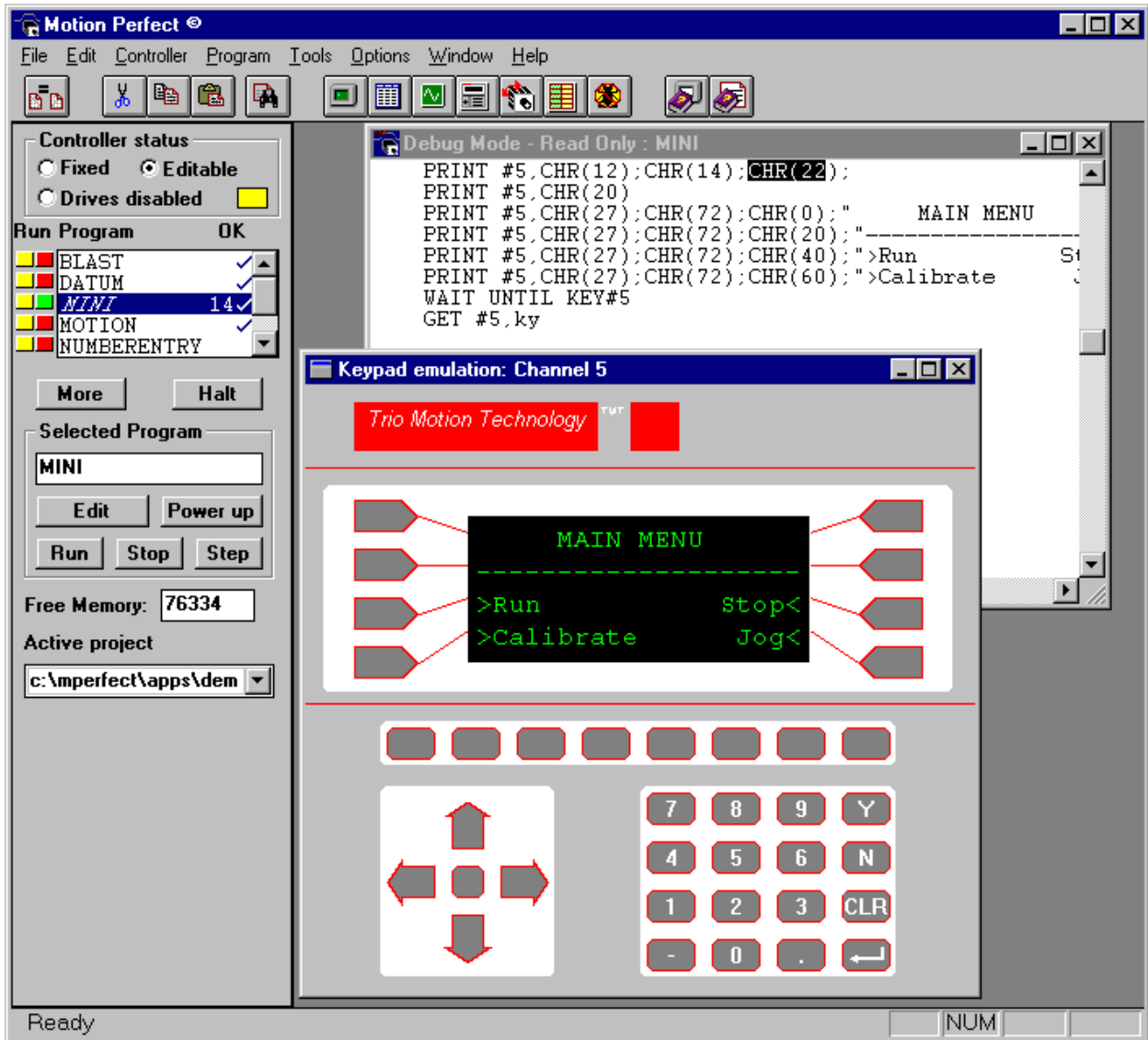
1. Create, copy, rename, delete, edit, run and debug programs on the controller.
2. Use the project manager to maintain a consistent copy of your application programs on the PC.
3. Use the control panel, full program directory, axis parameters window, and IO status to see what activities are being performed by the controller, and to check and change the controllers status.
4. Use the keypad emulator to develop keypad driven applications without requiring an actual Membrane Keypad.
5. Use the program debugger, axis parameters, software oscilloscope and jog axes windows to commission your servo system.

It is possible to open several windows on the *Motion Perfect* desktop, which will then run concurrently. Hence a user could be stepping through a program displayed in an edit window, whilst checking the programs output and entering input characters via a separate terminal window, and monitoring and updating the axis parameters and IO status in their respective windows.

### 10.3 *Motion Perfect* PC Requirements:

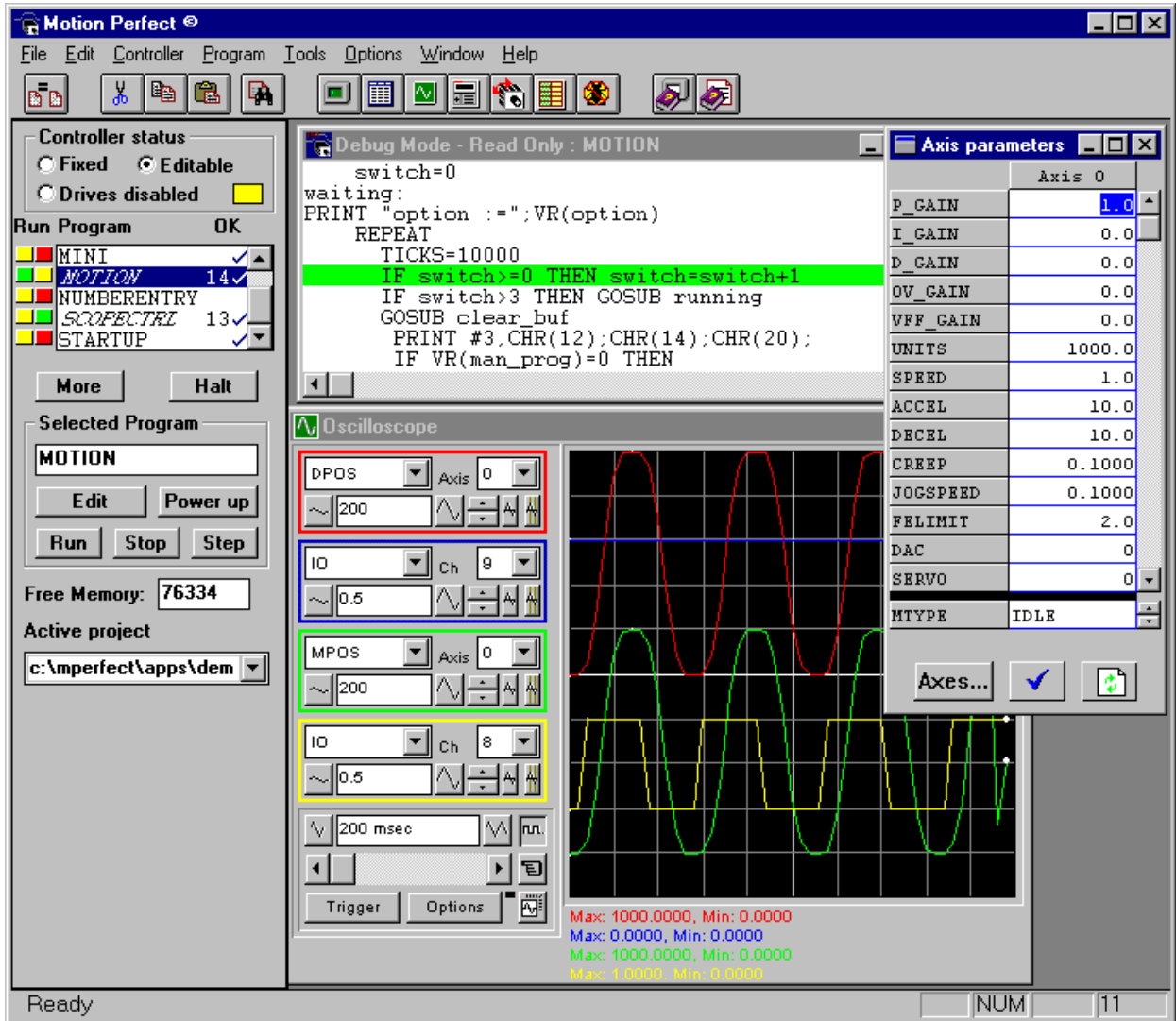
The following equipment is required to be able to run *Motion Perfect*:

1. *Motion Coordinator* SERIES MC2 or MC204 with operating system version 1.16 or greater.
2. IBM PC or 100% compatible running Microsoft Windows 3.1 or Higher
3. An enhanced serial communications port (UART 16550)
4. 80486 or higher processor.
5. 2.3MB of Hard disk space.
6. VGA display, SVGA 800x600 or higher resolution recommended.
7. Mouse or Trackball



**Testing a program using the membrane keypad emulation**





The program 'MOTION' is being stepped, and the program 'SCOPECTRL' is being run whilst observing axis 0 parameters using the axis parameter and oscilloscope windows.

#### 10.4 Obtaining *Motion Perfect*

The latest version of *Motion Perfect* is supplied with this manual, otherwise it can be obtained by downloading a copy from Trio's internet web site at

<http://www.triomotion.com/>

If you do not have access to the internet then contact Trio and request a copy via post.

#### 10.5 Installing *Motion Perfect*

*Motion Perfect* is supplied on a 3.5 inch disk with all new Motion Coordinator SERIES 2 controllers (the disk can be found in the pocket at the front of this manual). This disk contains an installation program which must be used install *Motion Perfect* on your PC hard drive.

Turn on your PC and run Windows (type WIN at the DOS prompt if windows does not start automatically),

##### **For Windows<sup>tm</sup> 3.1 users:**

From the Windows File Manager, select 'RUN' from the start menu

##### **For Windows<sup>tm</sup> 95 users:**

Click on the **Start** button and select the Run option

At the 'Open' box enter the following command line: **A:\INSTALL**

The installation program will prompt you for a directory in which to install *Motion Perfect*, (by default the program will choose C:\MPERFECT - this is recommended). Once you have chosen the directory to use the necessary files will be installed onto your hard disk.

If you have previously installed *Motion Perfect* you will be prompted to overwrite any matching files.

The MPERFECT folder created will contain the following items :

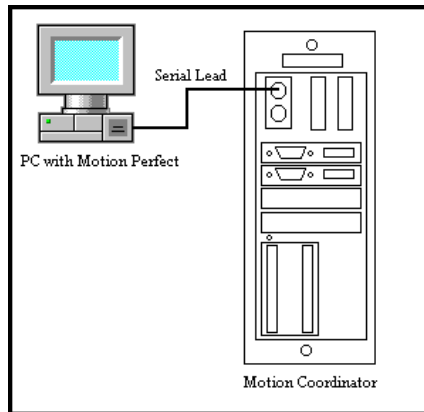
|               |   |                                                                      |
|---------------|---|----------------------------------------------------------------------|
| MP . EXE      | - | This is the <i>Motion Perfect</i> executable program file.           |
| MP . HLP      | - | <i>Motion Perfect</i> help file.                                     |
| BASIC . HLP   | - | Help file describing the Trio BASIC on the Motion Coordinator SERIES |
| Axxx . OUT    | - | This is the latest Motion Coordinator System software update file    |
| LOAD . EXE    | - | DOS program for loading system software.                             |
| APPS          | - | This folder contains 2 simple application project examples.          |
| README . TXT  | - | Introductory readme file.                                            |
| INSTALL . EXE | - | The installation executable file.                                    |
| UNSTALL . EXE | - | An un-install executable file.                                       |

For information about running *Motion Perfect* see the 'Running *Motion Perfect*' section below

## 10.6 Connecting *Motion Perfect* to your *Motion Coordinator*

*Motion Perfect* is an on-line environment which mean it must be connected to a Trio controller to use its advanced features. The PC should be connected to the controller using the RS232 serial lead ( supplied by Trio) between a free COM port on the PC and the controller's Serial Port A.

To run *Motion Perfect* on your PC:



### **Windows 95**

- Click the Start button
- Select the 'Programs' Group
- Select '*Motion Perfect*'
- Click on the 'MP' application item

### **Windows 3.xx**

- From the Program Manager's 'File' menu 'Run...' menu option, and enter "c:\MPERFECT\MP.EXE"
- or double-click on the *Motion Perfect* icon in its program group window.

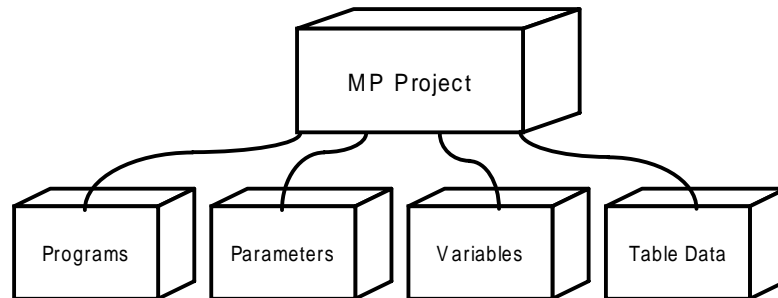
*Motion Perfect* will display its introductory splash screen whilst looking for any controllers connected to the PC. The messages 'Searching for a valid controller', followed by 'Controller found on port COMx' (where x is the number of the PC COM port) should be seen.

If any other message is seen, such as 'No controllers found' or 'Invalid version', then see the 'Problems' section below. Clear this dialog by pressing the 'enter' key, or clicking the 'Ok' button.



## 10.7 Motion Perfect 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 required for a single motion application. It provides a single 'container' for this information on the PC, enabling effective version control to be maintained, and providing a mechanism for verifying the application programs on the controller.



Whenever *Motion Perfect* is run it will only enable its tool suite if it has successfully opened an existing project or created a new one, and verified the programs on the controller are the same as those in the project on the PC. If either of these criteria are not met, *Motion Perfect* can only be run in the 'disconnected mode' - see 'Running *Motion Perfect* in the Disconnected Mode' section for further information.

*Motion Perfect* holds the project in a folder (sub-directory), and performs its verification process by checking the "CRC" value of the programs in the project with those on the controller.

## 10.8 Motion Perfect Project Manager

The project manager works behind the scenes automatically maintaining consistency between the programs on the controller and the files on the PC. Whenever you edit a program in the *Motion Perfect* editor, it changes BOTH copies of the program. This avoids the slow process of up and down loading of programs and ensures that there is always a backup of changes you make. As programs are created, copied or erased on the controller using the *Motion Perfect* tools, the project is updated so that the files and programs are always consistent.

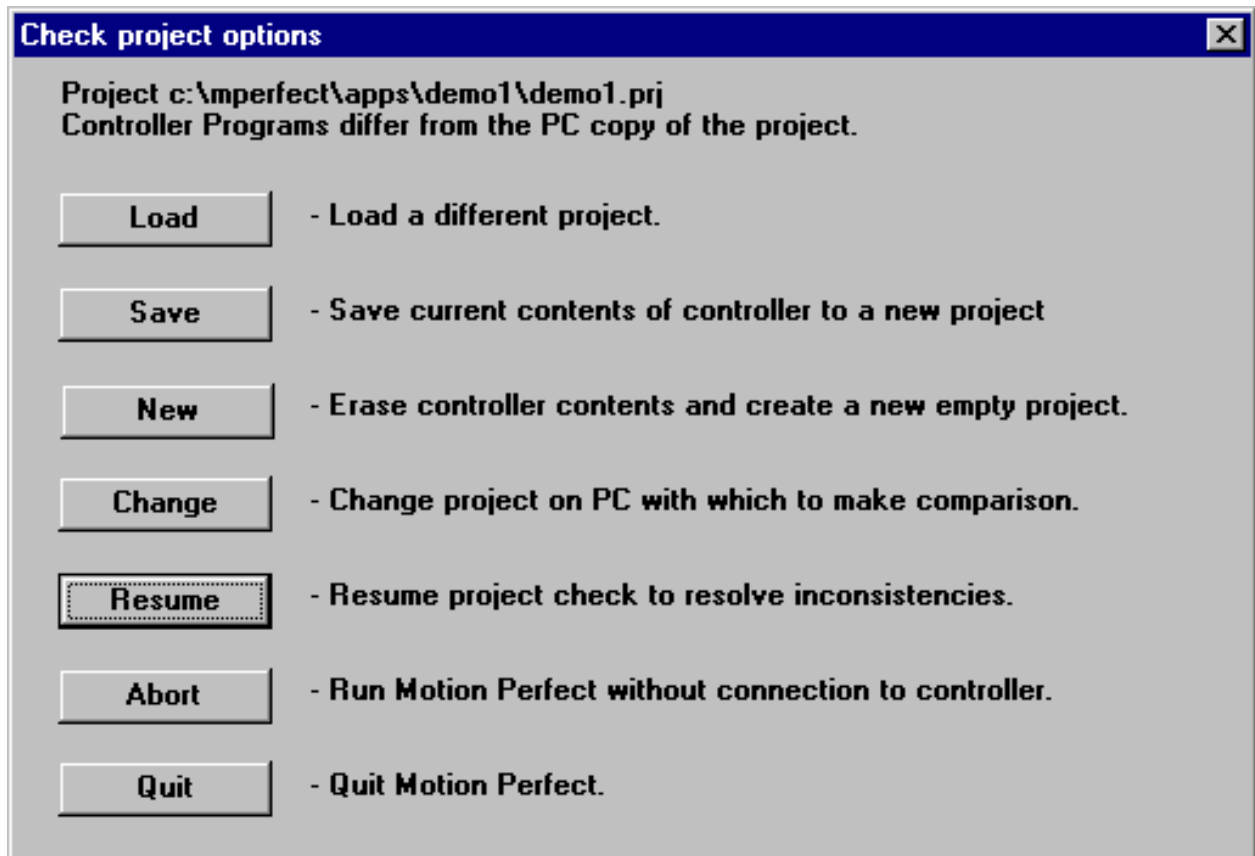
If the project - controller verification process is successful when *Motion Perfect* is first run (or after any inconsistencies have been resolved by the user) then a backup copy of the PC project will be created. This backup copy can be loaded if the controller version of the programs become corrupted for some reason during your development session. This option is found under the 'File' 'Revert to backup' menu option. It should be noted that the backup project file is overwritten EACH time a project is opened, hence if you open a new project during a development session the new project's backup copy will be overwritten. In addition, you can force *Motion Perfect* to check the two copies are identical at any time by running the 'Check project' facility, found under the 'File' menu.

When *Motion Perfect* starts up it always performs a consistency check between the programs on the controller (if any), and its current PC project file. If these differ the 'Check project options' dialog will be displayed, enabling the user to determine how to resolve these inconsistencies.

If you do not see this 'Check project options' dialog when running *Motion Perfect* for the first time, then either *Motion Perfect* failed to connect to a controller or, *Motion Perfect* has been run previously on your PC and the programs on the controller are the same as those in the PC project.

When the controller and computer versions match the 'Check Project' dialog will be displayed with the final message 'Project consistent'. In this case, click the 'Ok' button and the full *Motion Perfect* desktop will open.

From the 'Check project options' dialog, your course of action will now depend upon whether this is the first time the controller has been used, or whether it is installed in an existing servo system (and so there are already programs on the controller).



### Running Motion Perfect on a Controller being used for the First Time :

If this is the first time the controller has been used with *Motion Perfect*, and you do not have any programs on the controller, click the check project options 'New' button, and then click the 'Yes' button when asked

### New Project Dialog

The 'New Project' dialog will open enabling you to enter the required name of the new project, and specify where on your PC ( in which sub-directory) you want to locate this project. Type a suitable project name in the 'Project name:' text box, and then use the 'Directory' list box to move to the sub-directory on your PC where you want to locate the project. This is achieved by scrolling through the list of directories until the required name is seen, and then double clicking over this name. Repeat this process until the required sub-directory is found.

If you wish to create a new sub-directory on your PC, then move to the required parent directory location, enter the new sub-directory name in the 'Project name:' text box, and then click the 'Create directory' button. The pull down 'Drives' list box can be used to select a different drive if required. The full directory path name will be displayed under the 'Project Path' label.

After selecting the required directory path and entering the new project name, click the 'OK' button. If you enter a project path and name which already exists, then when the 'OK' button is clicked a dialog will appear asking whether to overwrite the existing project :

It is best to answer 'No' to this question, which will return you to the 'New project' dialog, and then enter a different project name (otherwise the original maybe lost).

Having successfully entered the new project path and file name, the 'Check project' dialog will then be displayed, with empty 'Controller Program' and 'Project Program' list boxes and a 'Project consistent' message. Click the 'Ok' button to continue. You have now created a new project on your PC, the *Motion Perfect* desktop will open, and all its facilities will now be available.

### **Running Motion Perfect on a Controller Installed in Existing System :**

If your controller already contains application programs ( perhaps because it is installed in an existing servo system, or you have previously developed programs on the controller using), then click the Check project options 'Save' button to save these programs to your PC.

The 'Save project as' dialog will appear enabling you to enter a project name and path on your PC where the controller programs will be saved. See the 'New Project Dialog' section for further information about how to use this dialog and enter the required details, since the save and new project dialogs work in the same way.

After selecting the project path and entering its name, click this dialog's 'OK' button, the 'Check project' dialog will appear and *Motion Perfect* will save all the programs from the controller to this new project on the PC. The 'Check project' dialog shall display progress messages as the programs are saved, and a 'Project consistent' message when they have all been successfully saved to the PC project file. Click the 'Ok' button to continue. You have now created a new project on your PC, the *Motion Perfect* desktop will open, and all its facilities will now be available.

### **10.9 Check Project Options Dialog**

The 'Check project options' dialog is displayed whenever *Motion Perfect* performs a project - controller consistency check and finds a difference between the PC project and the programs on the controller. The dialog enables the user to select the required option to resolve the discrepancy. The options available include -

**Load** - load a different project.

Select a new project on the PC and load this project onto the controller.

**Save** - save current contents of controller to a new project.

Create a new project on the PC and save the programs on the controller to this project.

**New** - erase controller contents and create a new empty project.

Create a new empty project on the PC and delete all the programs on the controller.

**Change** - change project on PC with which to make comparison.

Select a different project on the PC with which to perform the project files/controller programs consistency check.

**Resume** - resume project check to resolve inconsistencies.

Continue with the 'Check Project' dialog, enabling the user to resolve each individual program inconsistency (by loading the project version onto the controller, or saving the controller version to the project).

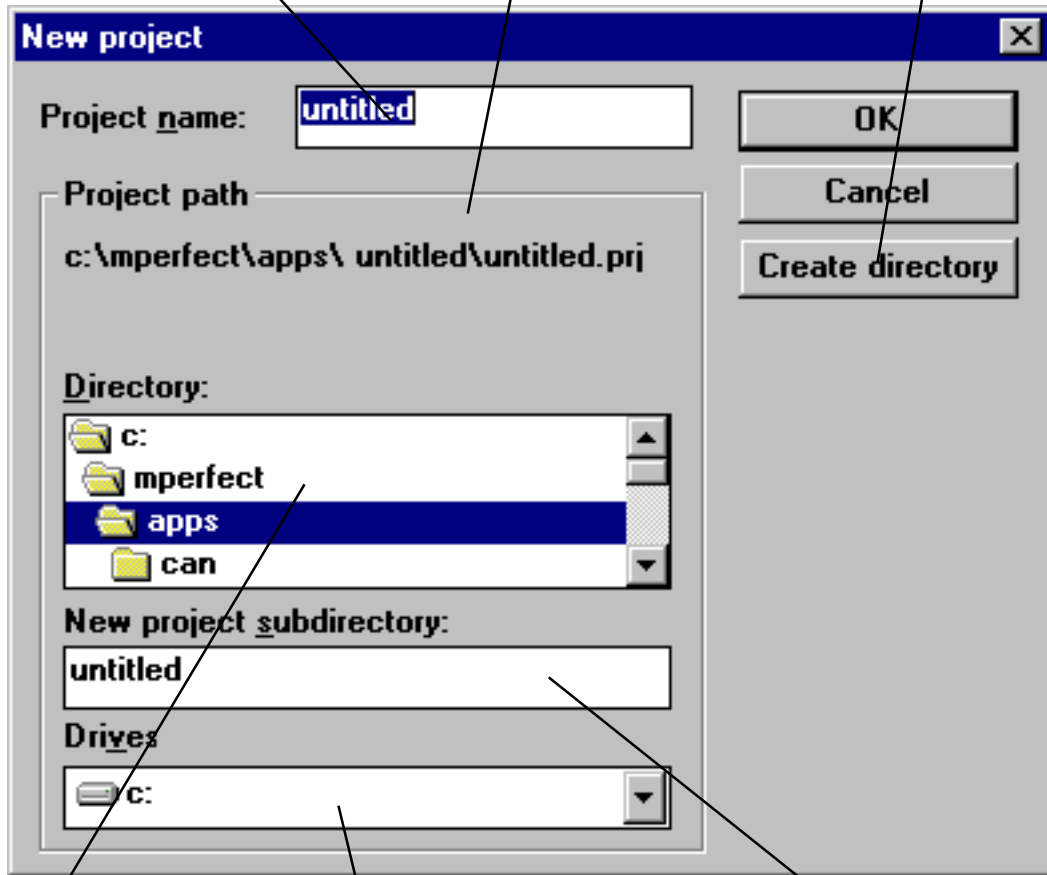
**Abort** - Run *Motion Perfect* without connection to controller.

**Quit** - Quit *Motion Perfect*

Enter required project name here

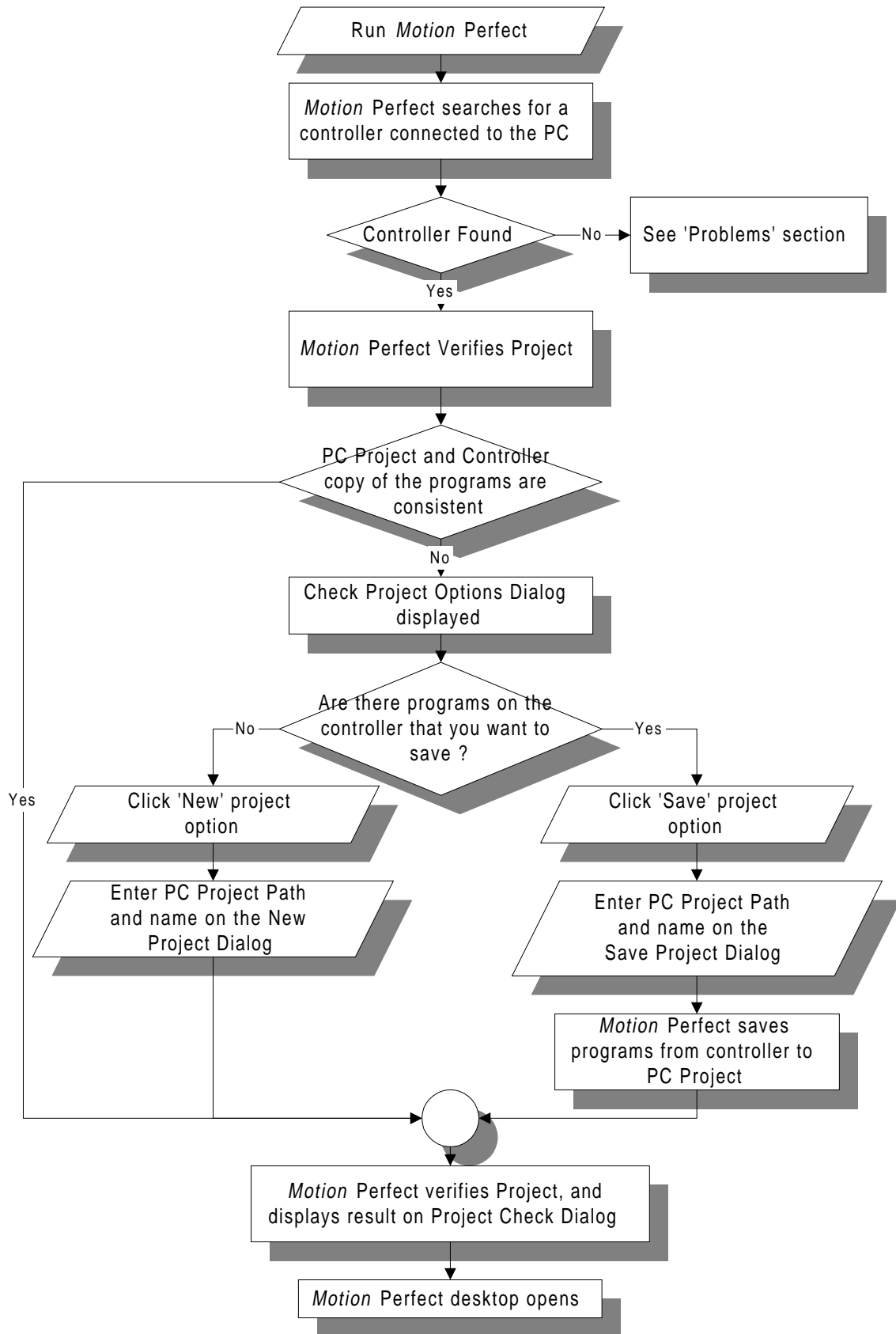
PC Projects name and path appears here

Click this button to create the new sub directory named in the Project name textbox.



Use this Directory list box and this Drives list box to find the required PC folder in which to locate the project.

New projects sub-directory name appears here.

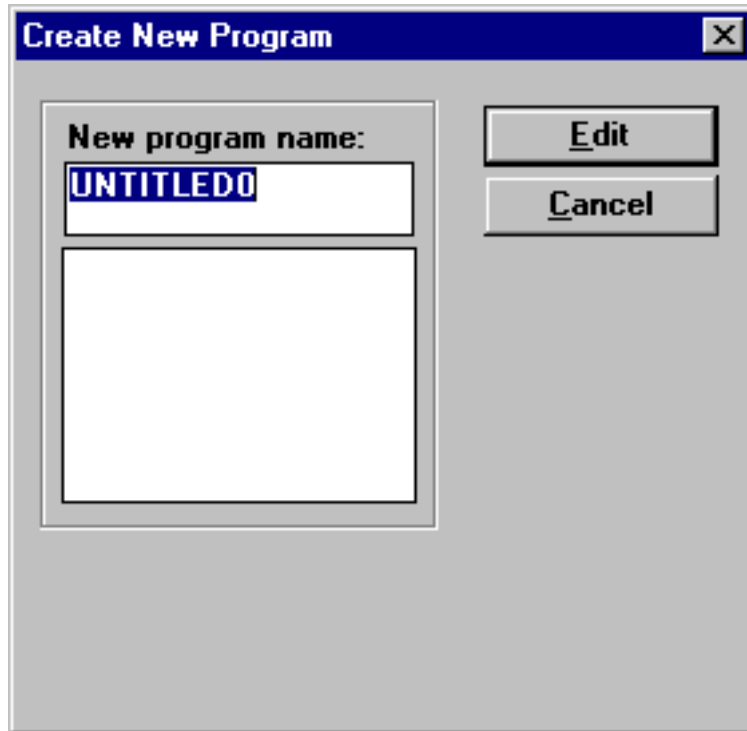




### 10.10 A simple *Motion* Perfect Session

This section describes how to create, edit and debug a program after having opened a project as described above. **See also section 7.1.2**

Select the menu option 'Program' 'New', and the following dialog will be displayed :



*Motion* Perfect selects the program name UNTITLED0 by default, but this can be changed by typing the required name in the 'New program name' textbox. So type the name MOTION, and click the 'Edit' button. An empty editor window will open, with the title 'Edit : MOTION'. Type a simple program, such as that in the diagram. Type the program in lower case, and note how when you return from the current edit line *Motion* Perfect updates the program on the controller, and replaces the Trio BASIC statement keywords with their tokenized versions in upper case.

The program can now be run, stepped and stopped - all without closing the editor window. The pull down menu options under 'Program' can be used, but it is easier to use the control panel.

Alongside the MOTION program name in the directory list box there will be a red and yellow button. The red button shows that the program is stopped and if it is pressed, the program will start to run. The directory list box entry will appear in italics, the controller process number on which the program is running will appear alongside the program name, and the red button will be replaced with a green one. The editor title will change to 'Debug Mode - Read Only : MOTION' and you will be unable to edit the program whilst it is running. If the green button is clicked, the program will be stopped.

If the yellow button is clicked, the program will start stepping, the run/stop button will then turn yellow and will execute one line of the program each time the button is clicked. A green bar will appear in the editor window, highlighting the line of the program about to be executed.

If there are any compilation errors in the program when an attempt is made to run (or compile it using the 'Program' 'Compile' menu option) a dialog will appear briefly describing the error, and the line number it is on. Resolve the error, and repeat the process.

## 10.11 **Motion Perfects Tools**

This manual describes the key Motion Perfect Tools:

- Axis Parameters
- Controller configuration
- Editor
- Debugger
- Full Program Directory
- IO Window
- Jog Axes
- Keypad Emulator
- Oscilloscope
- Terminal Windows

Online help is also available inside *Motion Perfect* which describes how to use each of the *Motion Perfect* tools.

### 10.11.1 **Axis Parameters**

The Axis Parameter window allows the user to set and read back the axis parameter settings. This window has been designed to have the look and feel of a normal Windows spreadsheet. 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 are read-only, as these values are set by the system software of the *Motion Coordinator* SERIES 2 as it processes the BASIC motion commands and monitors the status of the external inputs.

When the parameter window is shrunk by the user, the size of the bank 2 will be reduced first and then the size of the bank 1. When the parameter window is stretched by the user, bank 1 will grow first and then bank 2.

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 so that the user can shrink the window to be able to view other windows whilst 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, as is usual for all Windows spreadsheets. 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.

When the user changes the UNITS parameter for any axis, all the data for this axis is reread as many of the parameters, such as the SPEED, ACCEL, MPOS, etc., are adjusted by this factor to be shown in user units.

The AXISSTATUS parameter displays status bits. These are highlighted in red if the bit is set.

### **Column Headers**

This row of buttons shows the number of the axis for which the parameters are being displayed. If this button is clicked then the whole column of data is selected.

**Bank 1 Row Headers**

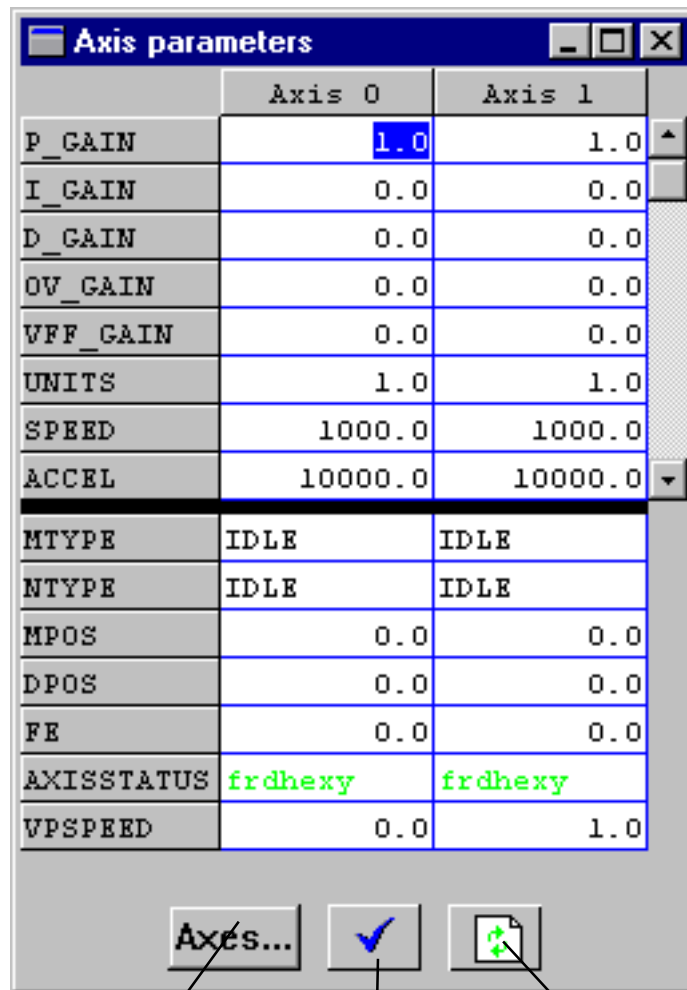
This column of buttons shows the names of the read/write parameters for which the data is displayed for the different axes. If the button is clicked then the whole row of data is selected.

**Bank 2 Row Headers**

This column of buttons shows the names of the read only parameters for which the data is displayed for the different axes. If the button is clicked then the whole row of data is selected.

Bank 1 Data - contains read/write parameter values.

Bank 2 Data - This bank contains the values of the read only parameters.



Click this button to open the dialog used to define which axes to Display.

Click this button to close the window.

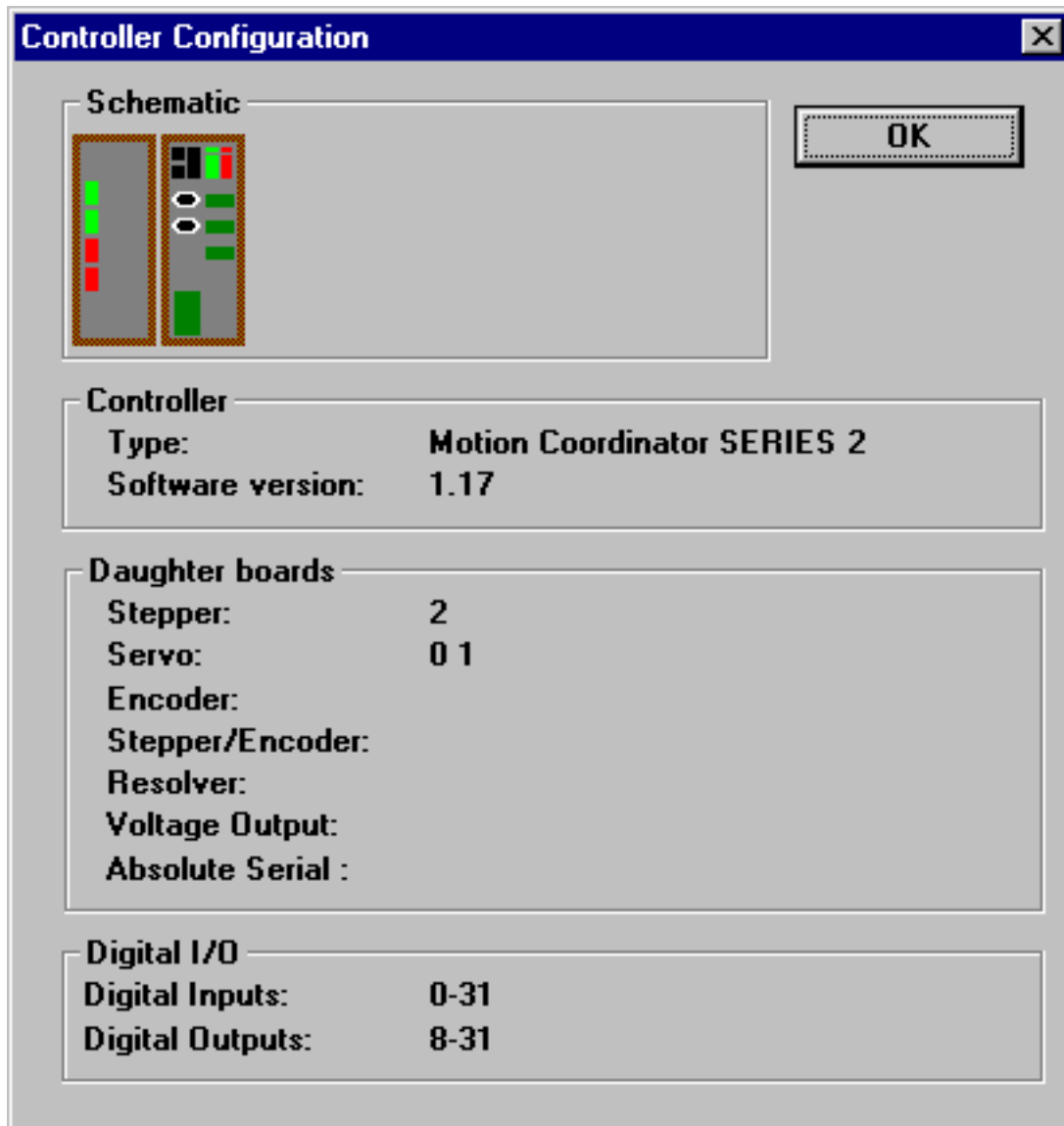
Click this button to refresh the displayed values.

### Axis Selector

This shows a dialog that allows the user to select the axes for which the data will be displayed. By default the selected axes will be defined by the axes set by the last Create Startup, Jog Axes window or Axes Parameters window.

### 10.11.2 Controller Configuration

This screen displays a mimic of the controller with details of options fitted and software version.



### 10.11.3 Editor

#### Overview

The *Motion Perfect* editor is a fully featured windows style editor. As the caret is moved off the current line, any changes made to this line are sent to the controller, which performs syntax checking, tokenizes the line (all recognized Trio Basic keywords, if in lower case, are converted to upper case), and returns the tokenized result. This tokenized reply then replaces the line of text in the editor window. When a edit window is closed, the project file is updated with the newly modified program.

#### Opening an Editor Window:

**Existing programs can be edited by opening an edit window using one of the following routes :**

1. Selecting the 'Program' pull-down menu option, followed by 'Edit' and then selecting the required program using the dialog which appears.
2. Pressing the 'Edit' button on the control panel which will open an edit window on the currently selected program.
3. Double-clicking with the mouse over the required program in the program list box on the control panel.

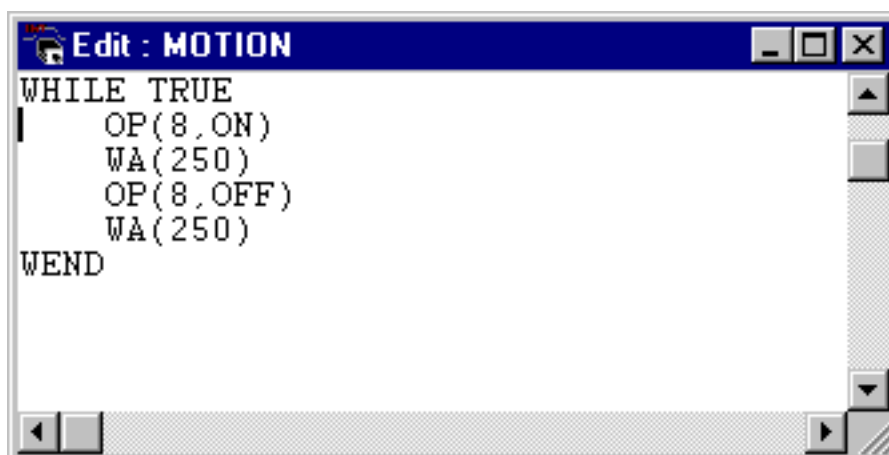
**Alternatively a new program can be created on the controller, and an edit window opened for it :**

1. By selecting the pull-down menu 'Program' option, followed by 'New'. This will give a program called 'untitled<x>' where x is the next available number to create a unique program name in the project. This name can be changed before opening the edit window, by clicking the mouse in the program name text box, and typing the new name required, then press the 'EDIT' button on the dialog to open the window
2. Selecting the 'Program' pull-down option, followed by 'Edit', and then editing the program name as required.

It is not possible to open a new edit window whilst programs are running on the controller.

When opening an edit window, *Motion Perfect* performs a CRC check on the versions of the program on the controller and in the project. If the CRCs are different, the user is warned and advised to perform a 'check project' to reveal further information as to the problem.

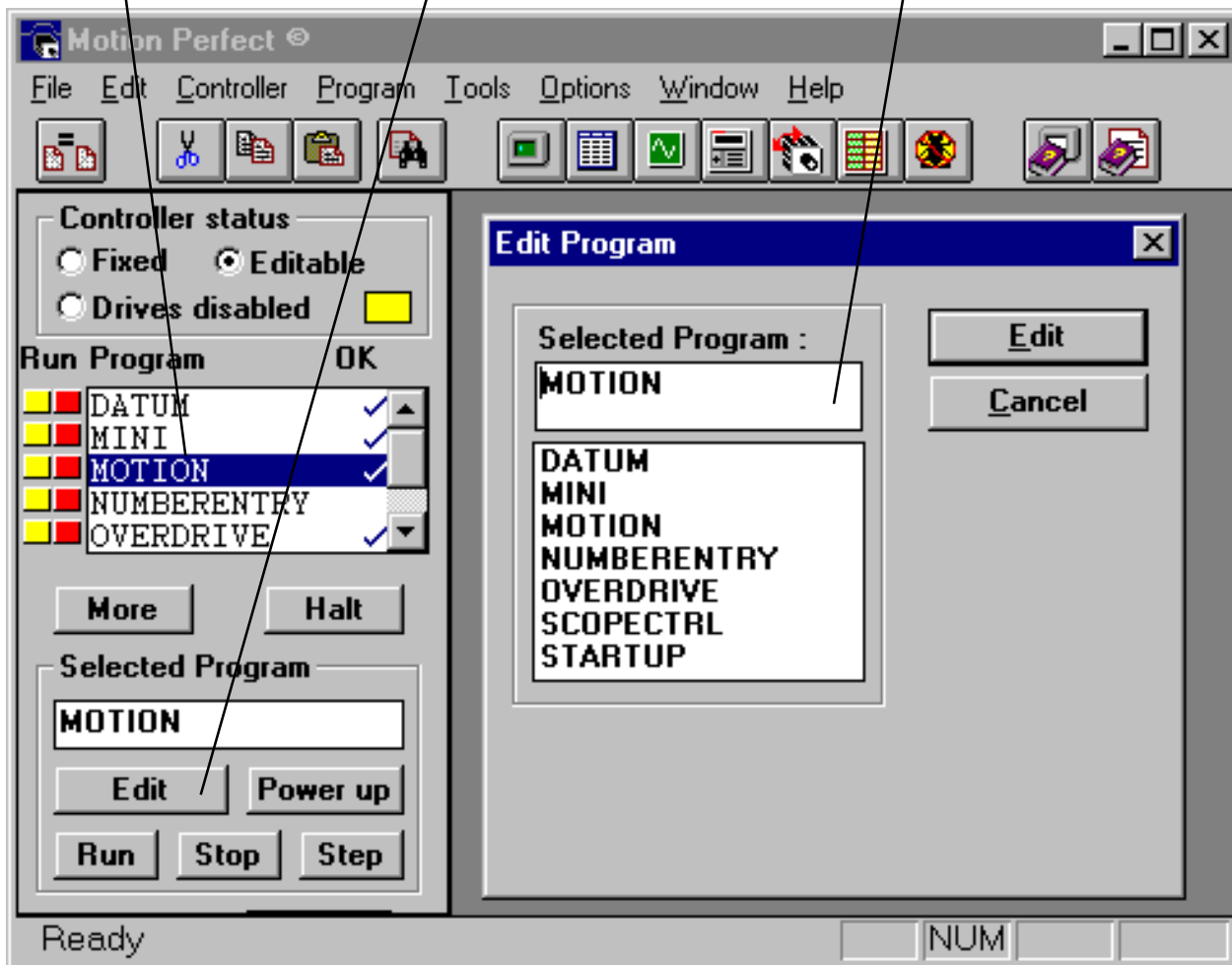
An edit window such as that shown will be opened :



Double-click here

Click this Edit button to open edit window for currently selected program

Select the 'Program' Edit' menu option to open this dialog, then select the required program and click the 'Edit' button on this dialog.



### Cut, Copy and Paste Operations:

The editor possesses windows style cut/copy/paste facilities which maybe mouse or keyboard driven.

To use :

1. Select the text, and cut or copy it to the clipboard.
2. Move the cursor to the insert point, and paste.

### To select an area of text :

Using the mouse :

Move the mouse with the left mouse button held down, and then select 'Edit' followed by 'Cut' to select and remove the text from the display, or 'Copy' to select the text.

To continue an existing selection in the display, before pressing the left mouse button and moving the mouse, press and hold the shift key on the keyboard.

Using the keyboard :

Move the caret to the start point, and then move it using the cursor keys whilst holding down the 'shift' key.

The area of text which has been selected is highlighted (shown as white text on a black background).

**To Cut or copy the selected text :**

The selected text can be 'cut' out, in which case it is removed from the display and copied to the clipboard. Or it can be 'copied' to the clipboard, in which case the original text is unchanged. Once the required text has been selected, it is copied or cut to the clipboard by:

Using the pull-down menu 'Edit' option followed by 'Cut' or 'Copy'.

Using the keyboard keys <Ctrl><c> to copy or <Ctrl><x> to cut.

**To paste the selected text:**

Now the required text is on the clipboard, it is pasted (inserted) into the display at the current caret position by:

Using the pull-down menu 'Edit' option followed by 'Paste'.

Using the keyboard keys <Ctrl><v>.

It is possible to copy text to/from other windows applications (such as a Microsoft Word document) since the windows clipboard is used to copy the selected to/from. Simply use the cut/copy/paste as usual in each application.

**Find and Replace Operations**

**Find :**

The program in the current edit window can be searched for a specific text string, and, if found, the caret is moved to this line. When the 'Edit' 'Find' menu option is selected, the 'Find' dialog is created. This enables the search string to be entered, along with whether the search is to be case sensitive, and the search direction

In the example, a case sensitive search down the program from the carets current location is being made for the word 'GOSUB'.

It is possible to continue editing the current program whilst the 'Find' dialog is still on the *Motion Perfect* display, by simply returning focus to the edit window. The 'Find' dialog remains on the display until the cancel button is pressed.

**Replace:**

The program in the current edit window can be searched for a defined text string, and, if found, the search string can be replaced with a second defined text string.

When the 'Edit' 'Replace' menu option is selected, the 'Replace' dialog is created. This enables the search string to be entered, and the replace string.

In the example, a non case sensitive search is being made for the label `initial_function`, to be replaced by the label `welcome_function`.

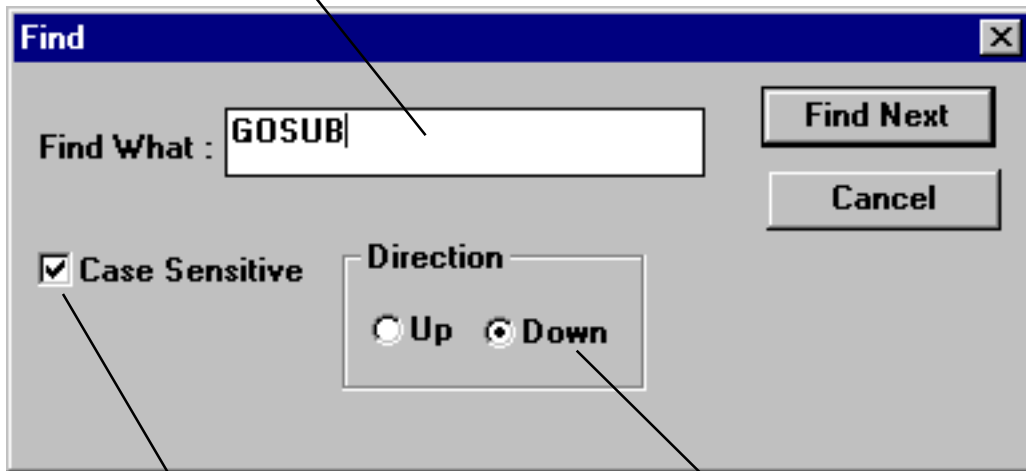
The buttons on the dialog initiate :

**FindNext** - a simple search operation for the search string, or

**Replace** - a search and replace operation, or

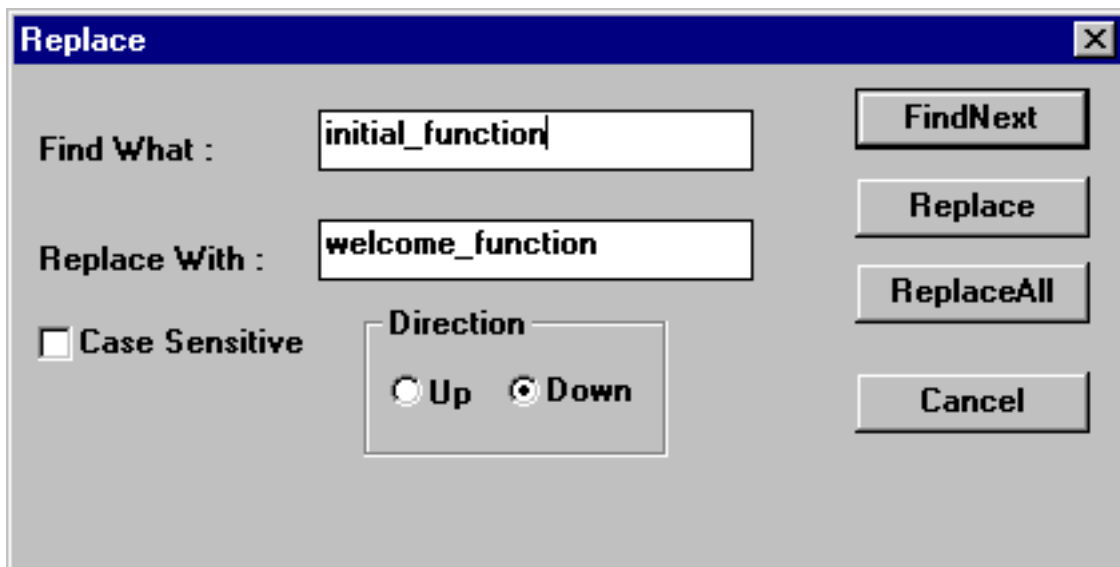
**ReplaceAll** - a search and replace all occurrences operation. (In this case, all occurrences of the search string are replaced from the current caret position to the top or end of the program, depending upon the search direction.

Enter the string to be found here



Tick this box to make the search case sensitive

Determines whether to search from the current caret position backwards towards the top of the line ('Up' box checked), or forwards towards the end of the program ('Down' box checked).



It is possible to continue editing the current program whilst the 'Replace' dialog is still on the *MotionPerfect* display, by simply returning focus to the edit window. The 'Replace' dialog remains on the display until the cancel button is pressed.



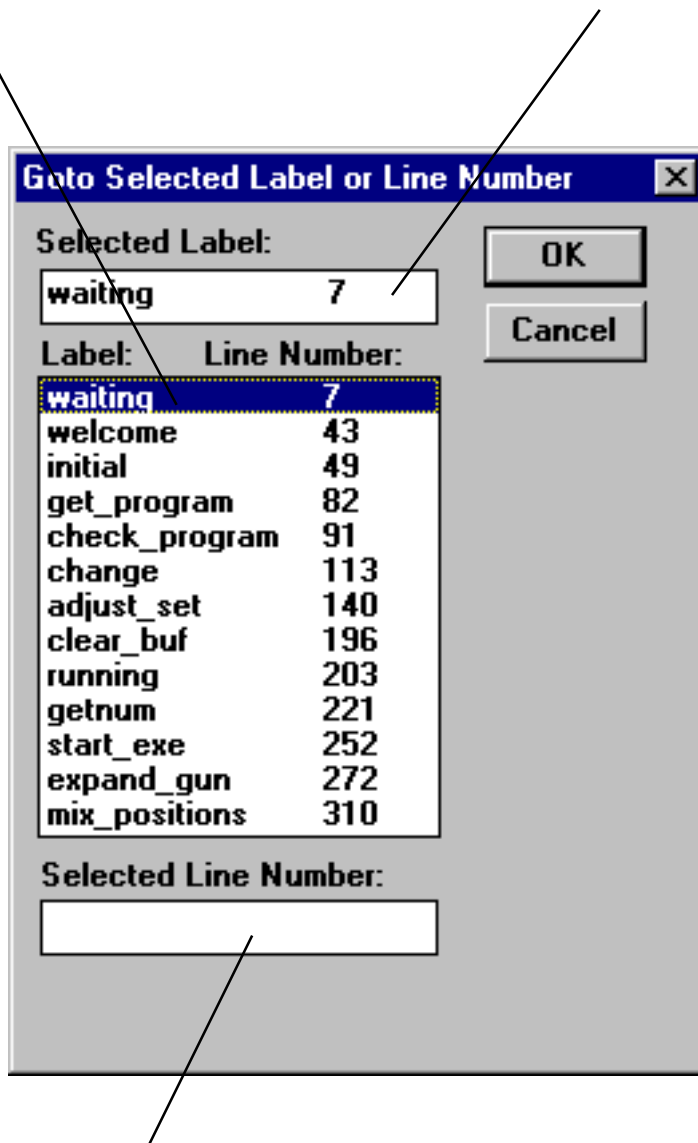
**List all program labels, and jump to specific label or line number:**

A list of all labels in the program in the current edit window can be displayed, and a specific label, or line number selected and the caret moved to this position within the program. When the 'Edit' 'Goto' menu option is selected, the 'Goto' dialog is created. This displays a list of all the labels in the program in the current edit window. A specific label can be selected by clicking with the mouse over the required label in the list box, and this label is then displayed in the selected label text box at the top of the dialog. If the 'OK' button is now pressed, the caret is moved to this label's line within the edit window.

Alternatively, a specific line number can be selected by entering the value in the line number text box (the lower text box on the dialog). If the 'OK' button is now pressed, the caret is moved to the line whose number is within the edit window.

Click required label in this list box. A list of all the labels in the program, along with their line numbers, appear in this list box. If a label is clicked, it will appear in the selected label text box.

The name and line number of the selected label appears in this text box. The caret will be moved to this line number in the edit window if the OK button is pressed.



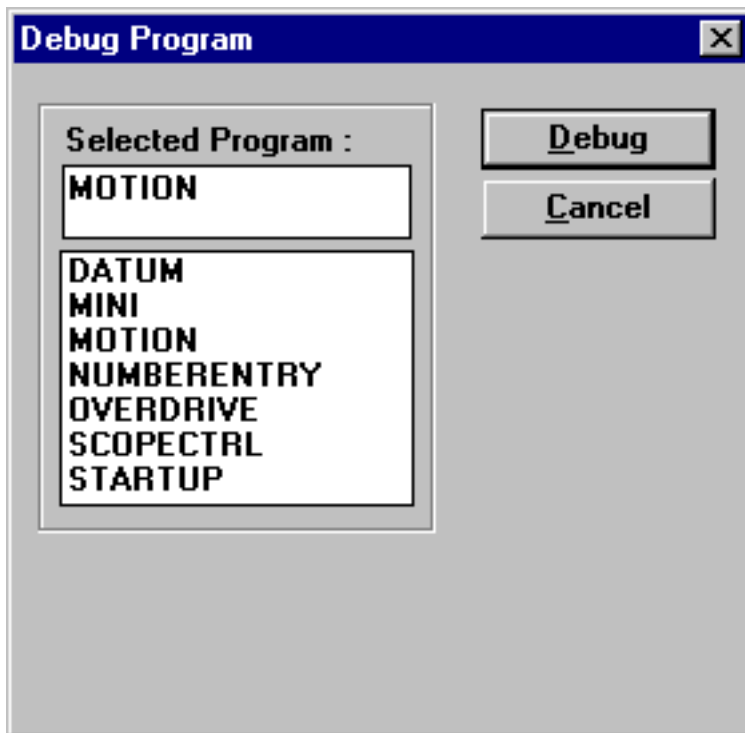
Enter required line number here The caret will be moved to this line number if the OK button is pressed.

#### 10.11.4 The *MotionPerfect* debugger

The *Motion Perfect* debugger enables the user to run application programs on the *Motion Coordinator* in the trace mode, executing one line at a time (known as stepping) whilst viewing the line in a debug window. It is also possible to set breakpoints in the program, so that when the program is run, it continues running until a breakpoint is reached, where it stops and the line of code highlighted in the debug window.

An edit window enters the 'Debug Mode - Read Only' when programs are running on the *Motion Coordinator*. Breakpoints are therefore set in the edit window, and the code viewed in the same window in debug mode when the program is running. It is possible to open a debug window, using the 'Tools' 'Program debugger' menu option, whilst there are programs running.

#### Opening a Debug Window



To open a debug window, select the pull-down menu 'Tools' option followed by 'Program debugger', which will open the 'Debug Program' dialog as shown below :

Select the program required by clicking the left mouse button whilst the cursor is over the name of the required program in the list box, the name of this program will appear in the 'Selected Program' text box.

If there are no programs running on the *Motion Coordinator* then a normal edit window will open, otherwise a 'Debug Mode - Read Only' window will open. This will revert to an edit window when all the programs running on the *Motion Coordinator* have been halted. It is not possible to open more than one edit or debug window per program.

### 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.

Press the 'Step' button on the control panel if the required program is currently selected.

Use the pull-down menu 'Program' 'Start program stepping' option, and then select the required program and a process number if required ( this can be left blank).

The latter (pull down menu option) must be used if it is necessary to define the process on which to step the program.

The program line about to be executed 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' functional key if the program required is currently selected on the *Motion Coordinator*.

```

Debug Mode - Read Only : MOTION
g2n=TABLE(g2p+1)
g3n=TABLE(g3p+1)
g4n=TABLE(g4p+1)
TRON
IF (g1n<=g2n) AND (g1n<=g3n) AND (g1n<=g4n) THEN
  ' gun 1 next:
  g1p=g1p+1
  gun1=1-gun1
  pos=g1n
  ' check others are not the same
  IF g2n=pos THEN
    g2p=g2p+1
    gun2=1-gun2
  ENDIF
  IF g3n=pos THEN
    g3p=g3p+1
    gun3=1-gun3
  ENDIF
  IF g4n=pos THEN
    g4p=g4p+1
    gun4=1-gun4
  ENDIF

```

### **Breakpoints**

A program continues running until it encounters a breakpoint (the *Motion Coordinator* TRON instruction). Breakpoints are set by moving the caret to the required line in an edit session window, and then selecting the 'Program' 'Toggle Breakpoint' menu option, or pressing the keyboard <Ctrl><B> short-cut key combination. This inserts a breakpoint at the current line in the program, indicated by highlighting the line with a red background. Breakpoint lines contain the 'TRON' *Motion Coordinator* command.

The breakpoint command toggles a breakpoint, hence the same menu option or key-strokes remove a breakpoint if there is already one on the line. All breakpoints can be removed from a program by selecting the 'Program' 'Clear all breakpoints' menu option.

It is not possible to add or remove breakpoints whilst any programs are running.

### **Running To a Breakpoint**

#### **Running a Program:**

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,

Pressing the 'Run' button on the control panel, or by using the keyboard <F5> function key, if it is the currently selected program on the *Motion Coordinator*.

Selecting the 'Program' 'Run' menu option.

The latter (pull down menu option) should be used if it is necessary to define the process on which to run the program, or if more than one copy of the program is to be run ( on several processes on the *Motion Coordinator*.)

#### **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).

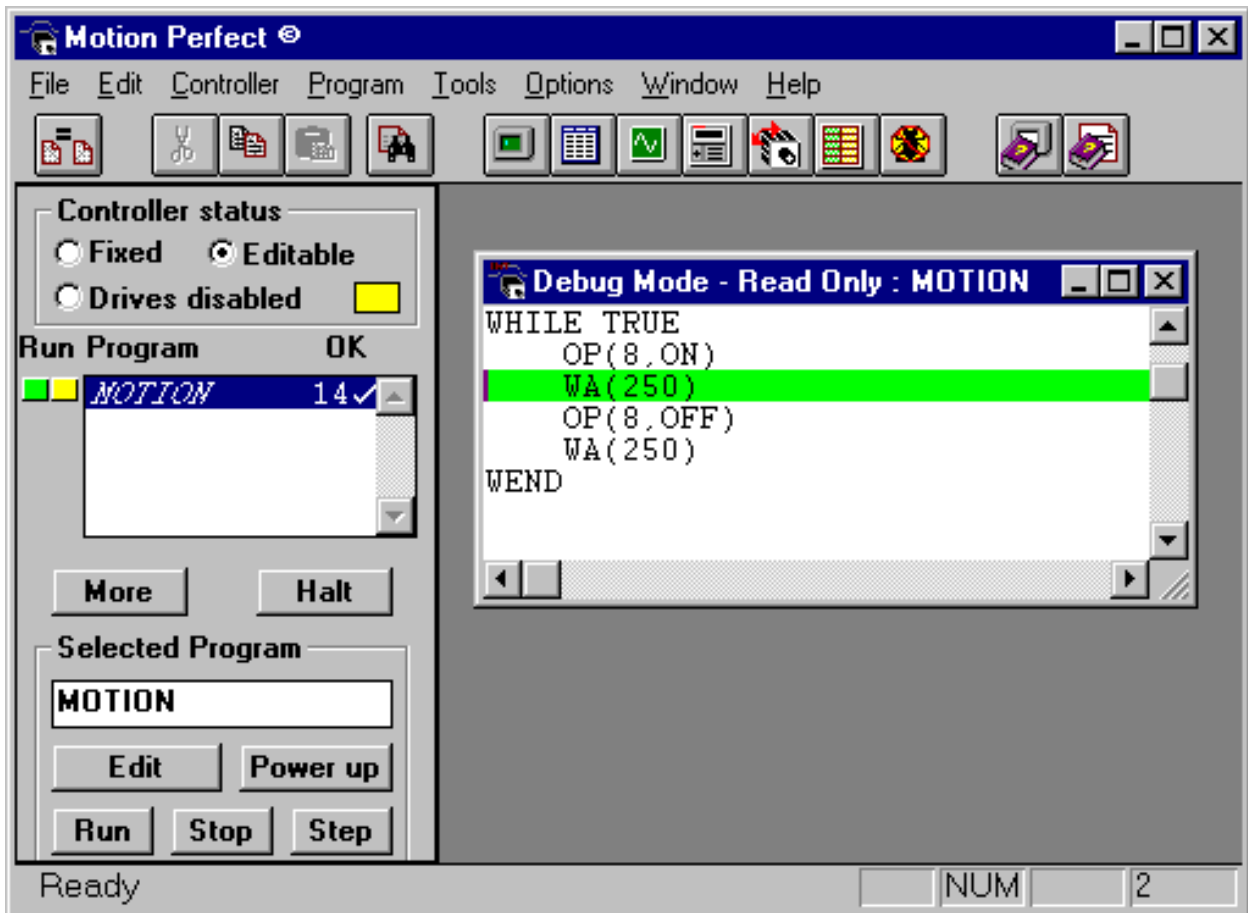
Use the 'Program' 'Stop' menu option.

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**

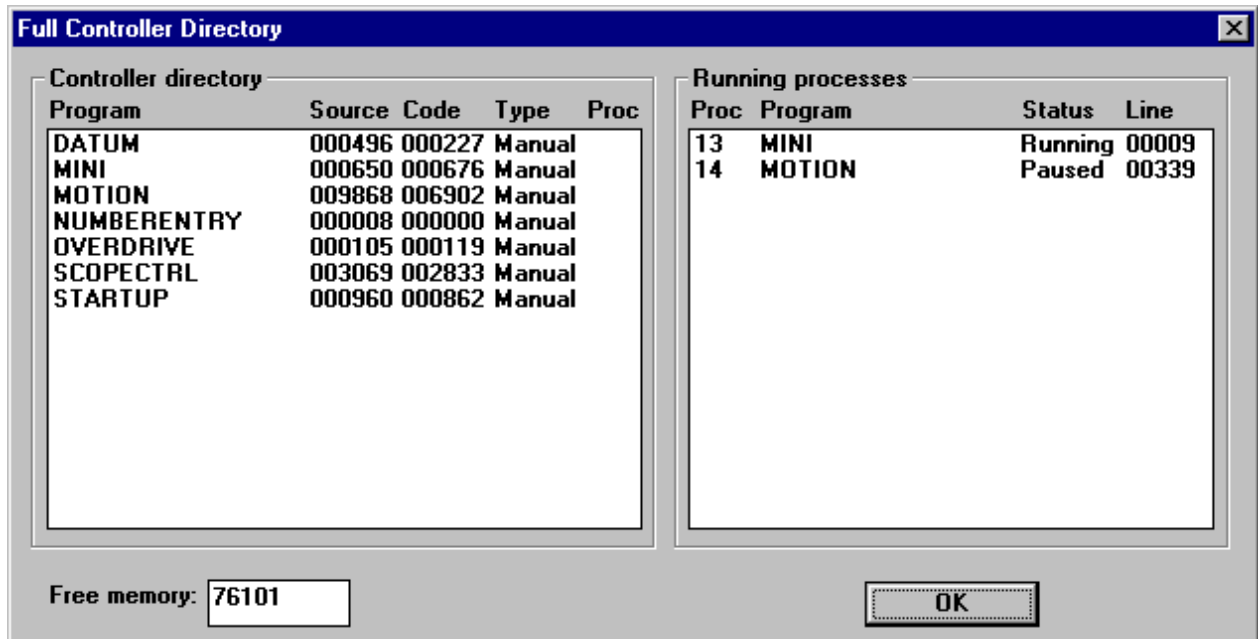
A running program can enter trace (stepping) mode by pressing the yellow button alongside the required program name in the list box on the control panel, or the 'Step' button if the required program is currently selected on the *Motion Coordinator*.





### 10.11.5 Full Program Directory

The full program directory dialog can be opened by selecting the 'Program' 'Full Directory' pull down menu option. This window dynamically shows details of all programs on the Motion Coordinator, and details of all running processes.



### 10.11.6 IO Status Window

This window allows the user to view the status of all the IO channels and toggle the status of the output channels.

#### 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*.

#### Input Bank

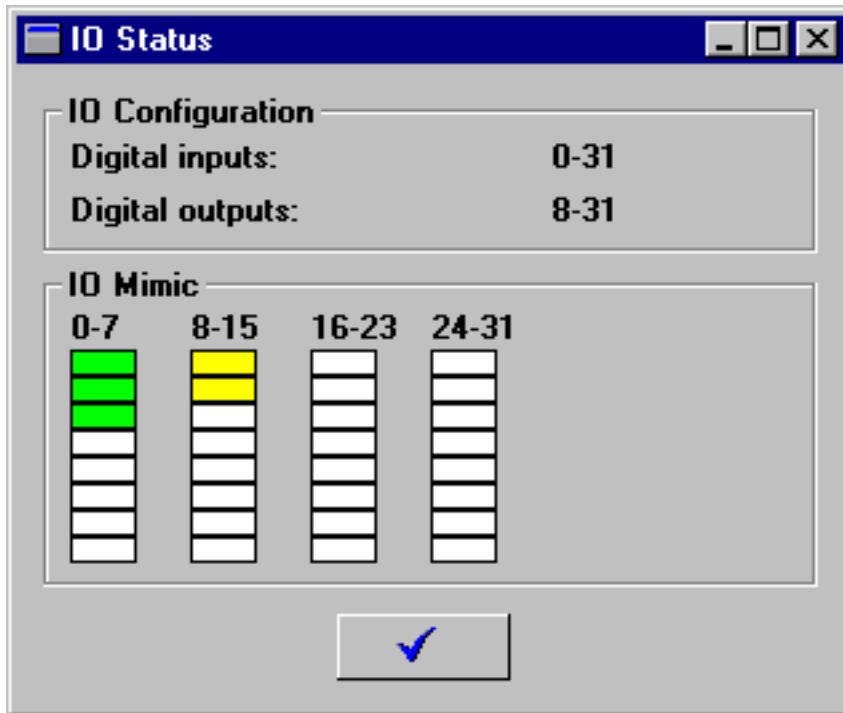
This first bank of 8 LEDs shows the status of the dedicated input channels. If an LED is green then the corresponding input is ON. If an LED is white then the corresponding input is OFF.

#### Input/Output Banks

These banks of LEDs show the status the bi-directional IO channels. If an LED is yellow then the corresponding input is ON. If an LED is white then the corresponding input is OFF. Under normal conditions the input status mimics the output status, except:

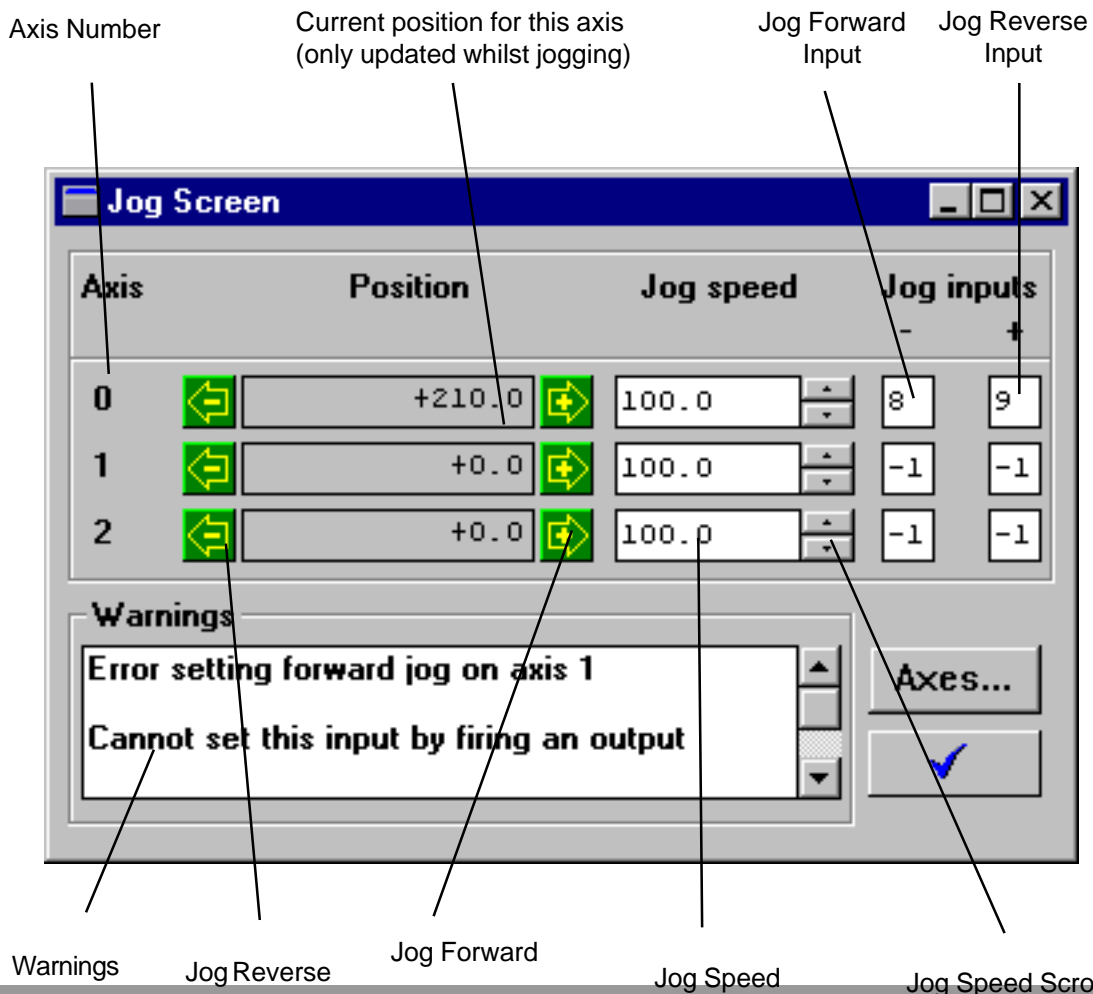
1. If this input is connected to an external 24V then it may be on without the corresponding output being on.
2. If the output chip detects an overcurrent situation, then the output chip will shut down and so the outputs will not be driven, even though they may be turned on.

If the LED is clicked with the mouse the status corresponding output channel is toggled, i.e. if the LED is white then the output will be turned on, if the LED is yellow then the output channel will be turned off.



### 10.11.7 Jog Axes Window

This window allows the user to move the axes on the *Motion Coordinator Series 2* when debugging a system:





## Introduction

This window takes advantage of the bi-directional IO 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 (normally closed) 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 CREEP axis parameter. This window also 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, 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 "Forward 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\_FWD output is turned off, thus enabling the forward jog function.

## Reverse Jog Input

This is the input which will be associated with the reverse jog function. This 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. This input is expected to be ON for the reverse 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.

## Forward Jog Input

This is the input which will be associated with the forward jog function. This 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. This input is expected to be ON for the forward jog function to be disabled and OFF for the forward 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 FWD\_JOG axis parameter is set.

### JogSpeed

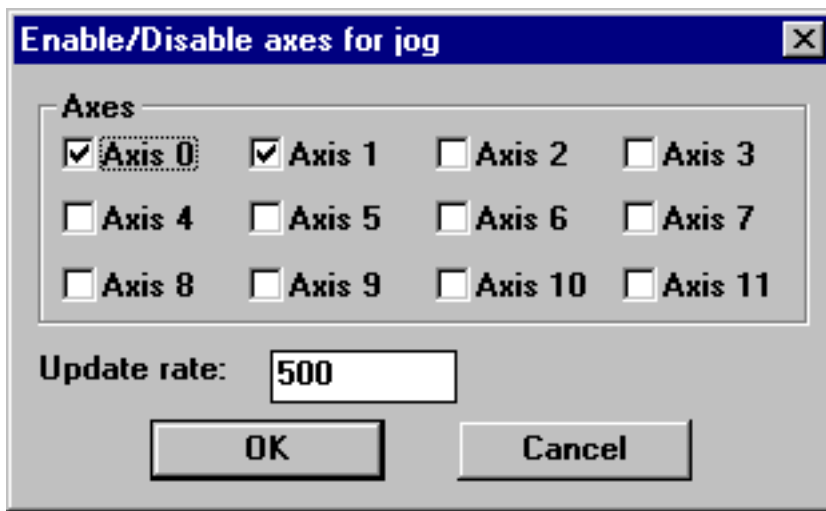
This is the speed at which the jog will be performed. This window limits this value to the range 0 to the demand speed for this axis, where the demand speed is given by the SPEED axis parameter. This value can be changed by writing directly to this control or using the jog speed control.

### Jog Speed Scroll

This scroll bar changes the jog speed up or down in increments of 1 unit per second

### Warnings

This shows the status of the last jog request.



### 10.11.8 Keypad Emulator

This window emulates a Trio Membrane Keypad. This keypad is connected to one of the "logical" communications channels on the serial port A. The user must specify the channel on which to run the emulation. If the specified channel is already in use, either by another keypad or a terminal window, then the window will not be able to open.

In the BASIC program the channel definition for the PRINT, GET, INPUT, LINPUT commands that are associated with the Trio FO Keypad must be changed from #4 or #3 to the channel that corresponds with the channel selected for the emulation. (Often #5) It is recommended that the channel assignment be made through a variable, so when time comes to run the program on the real machine, only one change will be required.

The Trio mini-membrane is effectively a subset of the full membrane keypad. It is therefore possible to emulate a mini-membrane by using just those keys and display lines available on the mini.

#### Channel Number

This dialog allows the user to select one of the valid async communications channel

#### Emulate #4 Codes

When the *Motion Coordinator Series 2* reads characters from the Trio Keypad from channel 3, these characters are translated using the DEFKEY translation table (see the BASIC help file). The normal operation of the keypad emulation returns the characters as if they were read with the DEFKEY translation. Alternatively, the *Motion Coordinator Series 2* can read the characters returned directly from the Trio FO Keypad using channel 4. If the emulate #4 codes is turned on in this window then the keypad emulation will return the untranslated characters. Note that it is only possible to emulate the default DEFKEY table.

#### *Keypad 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.

#### *Keypad function key 1*

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' in the QWERTY area of the PC keyboard.

#### *Keypad function key 2*

This is the keypad function key 2. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '2' in the QWERTY area of the PC keyboard.

#### *Keypad function key 3*

This is the keypad function key 3. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '3' in the QWERTY area of the PC keyboard.

#### *Keypad function key 4*

This is the keypad function key 4. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '4' in the QWERTY area of the PC keyboard.

#### *Keypad function key 5*

This is the keypad function key 5. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '5' in the QWERTY area of the PC keyboard.

#### *Keypad function key 6*

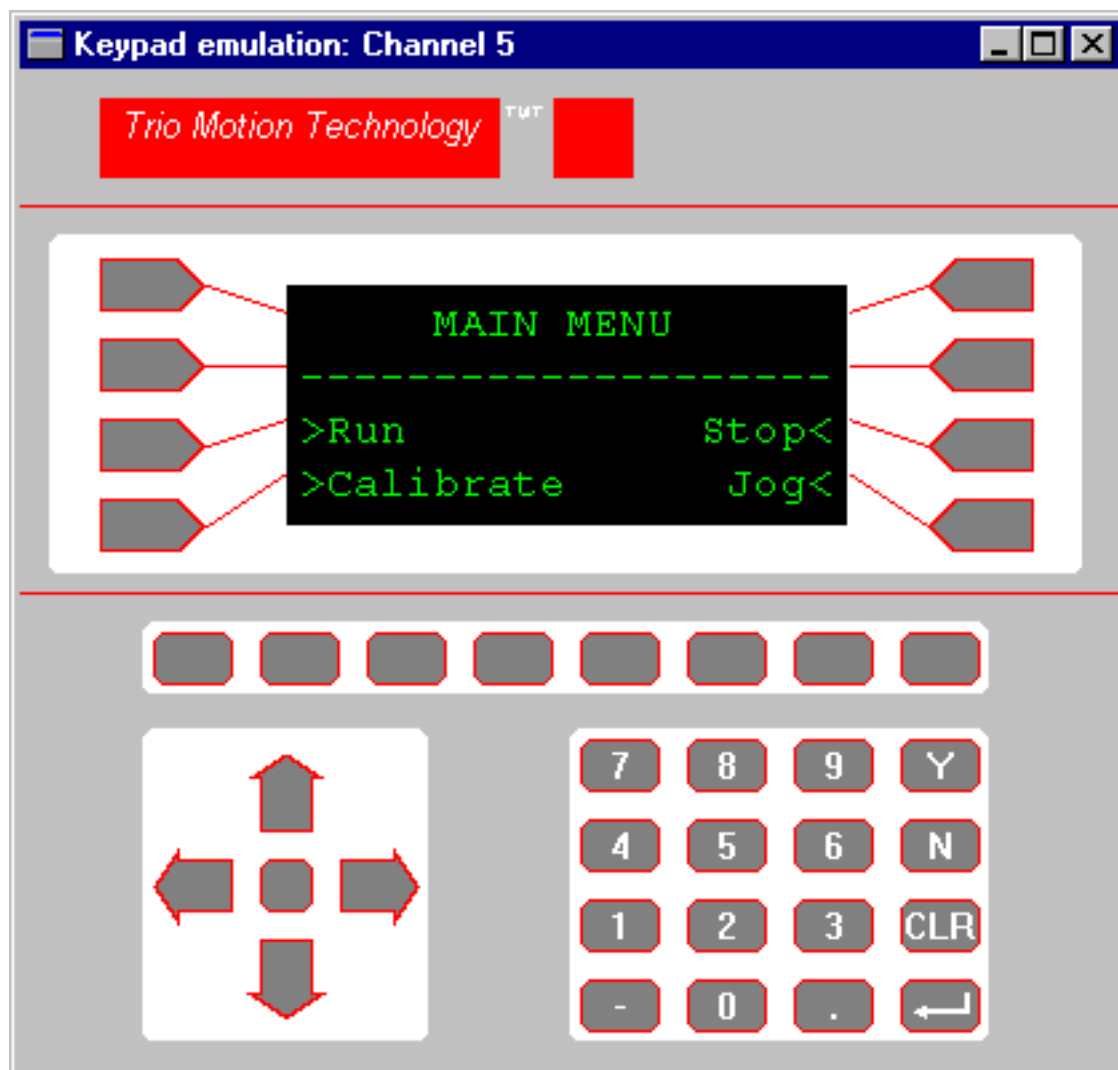
This is the keypad function key 6. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '6' in the QWERTY area of the PC keyboard.

#### *Keypad function key 7*

This is the keypad function key 7. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '7' in the QWERTY area of the PC keyboard.

#### *Keypad function key 8*

This is the keypad function key 8. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '8' in the QWERTY area of the PC keyboard.



*Keypad number key*

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.

*Keypad Y key*

This is the keypad 'Y' key. This is usually used to respond YES to some question on the display. It can be pressed by clicking the mouse over it or using the 'Y' in the QWERTY area of the PC keyboard.

*Keypad N key*

This is the keypad 'N' key. This is usually used to respond NO to some question on the display. It can be pressed by clicking the mouse over it or using the 'N' in the QWERTY area of the PC keyboard.

*Keypad 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.

*Keypad 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.

*Keypad - 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.

*Keypad . 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.

*Keypad up arrow*

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 up arrow key of the PC keyboard.

*Keypad down arrow*

This is the keypad down 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 down arrow key of the PC keyboard.

*Keypad left arrow*

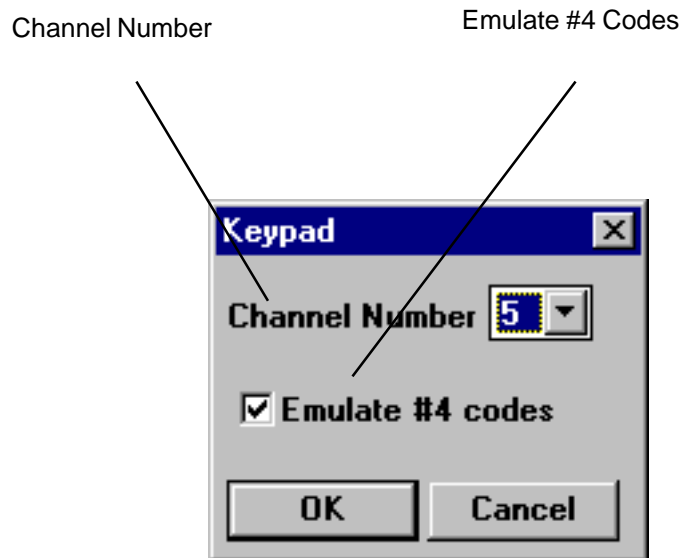
This is the keypad left arrow key. This is usually used to select between options on the display. It can be pressed by clicking the mouse over it or using the left arrow key of the PC keyboard.

*Keypad right arrow*

This is the keypad right 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 right arrow key of the PC keyboard.

*Keypad centre button*

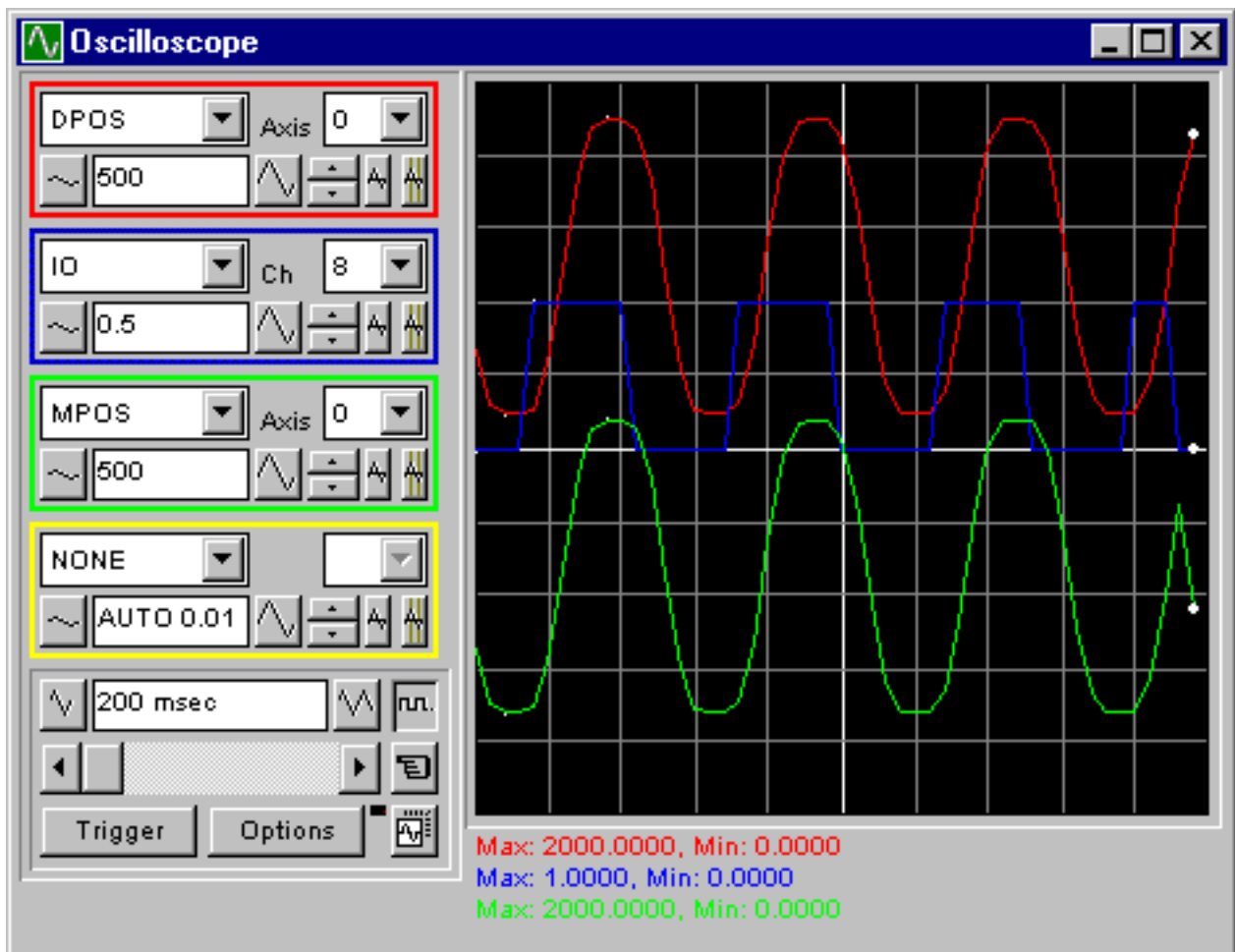
This is the keypad centre key. It can only be pressed by clicking the mouse over it.



### 10.11.9 Oscilloscope

#### Introduction

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 scope 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 recording data when it encounters a TRIGGER instruction in a program running on the controller. In manual mode it starts recording data immediately.



Software Oscilloscope - Main Display

#### Quick Start

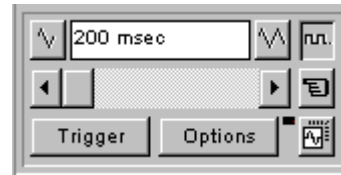
Select the required parameters, axes, and vertical scale per scope channel, then select the required time base and continuous/one-shot and manual/program triggering mode, and press the Trigger button.

## Oscilloscope Controls

The scope controls are organised into two main types; the channel-specific controls and the general controls which apply to the whole of the scope function



Channel Specific Block



General Controls

### Channel-Specific Controls:

Each scope channel has the following channel specific controls organised in each of four 'channel control blocks' surrounded by a coloured border which indicates the colour of this channels trace on the display. There are parameter list box / axis list box / vertical scale up-down buttons/ vertical offset scrollbar/ vertical offset reset button and cursor bars on-off button controls per scope channel.

#### Parameter



The parameters which the scope 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 analog 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 dialog.

If the scope 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 analog input parameter. The list box label switches between being blank if the scope channel is not in use, 'axis' if an axis parameter has been selected, or 'ch' if a channel parameter has been selected.

**Vertical Scaling**



The scope vertical scale (units per grid division on the display) are selected per channel, and these can be configured in either automatic or manual mode.

In automatic mode the scope calculates the most appropriate scale when it has finished running, prior to displaying the trace. Hence if the scope is running with continuous triggering, it will initially be unable to select a suitable vertical scale. It must be halted and re-started, or used in the manual scaling mode. In manual mode the user selects the scale per grid division.

The vertical scale is changed by pressing the up/down scale buttons either side of the current scale text box (left hand side button decreases the scale, and the right hand side button increases the scale value.) To return to the automatic scaling mode, continue pressing the left hand side button (decreasing the scale value) until the word 'AUTO' appears in the current scale text box.

**Channel Trace Vertical Offset**



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.

The offset value remains for a channel until the vertical offset reset button is pressed, or the scrollbar is used to return the trace to its original position.

**Vertical Offset Reset**

The vertical offset value applied using the vertical offset scrollbars can be cleared by pressing this vertical offset reset button.

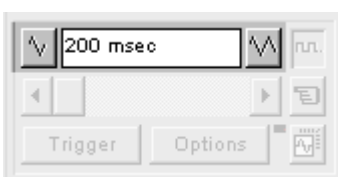
**Measurement Cursor Bars**



After the scope has finished running, and has displayed a trace, cursor bars can be enabled. These are displayed as two vertical bars, of the same colour as the channel trace, and initially located at the maximum and minimum trace location points. The values these represent are shown below the scope display, and again the text is of the same colour as the channel the values represent.

The cursor bars are enabled/disabled by pressing the cursor button which toggles alternately displaying and removing the cursor bars. The bars can then be moved by positioning the mouse cursor over the required bar, and holding down the left mouse button, and dragging the bar to the required position. The respective maximum or minimum value shown below the display is updated as the bar is dragged along with the value of the trace at the current bar position.

When the cursor bars are disabled, the maximum and minimum points are indicated by a single white pixel on the trace.





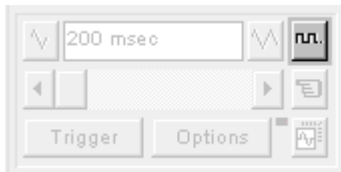
The required time base is selected using the up/down scale buttons either side of the current time base scale text box (left hand side button decreases the scale, and the right hand side button increases the scale value.) The value selected is the time per grid division on the display. If the time base 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 scope has finished running and a trace has been displayed, the time base scale can 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 scope 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 scope 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.


**One-shot/Continuous Trigger Mode**



The one-shot/continuous trigger mode button toggles between these two modes.

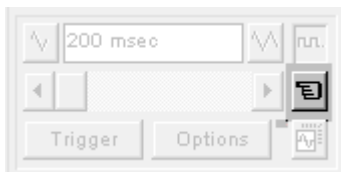
 Button Raised = One Shot Trigger Mode :

In one-shot mode, the scope runs until it has been triggered and one set of data recorded by the controller, retrieved and displayed.

 Button Pressed = Continuous Trigger Mode :

In continuous mode the scope continues running, retrieving data from the controller each time it is re-triggered and new data is recorded. The scope continues to run until the trigger button is pressed for a second time to halt the scope.

**Manual/Program Trigger Mode**



The manual/program trigger mode button toggles between these two modes. When pressed, the scope is set to trigger in the program mode, and two program listings can be seen on the button. When raised, the scope is set to the manual trigger mode, and a pointing hand can be seen on the button.



Button Raised = Manual Trigger Mode :

In manual mode, the controller is triggered, and starts to record data immediately the scope trigger button is pressed.



Button Depressed = Program Trigger Mode :

In program mode the scope 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 scope and displayed. The scope 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 scope is enabled. If it is manual mode then the controller immediately commences recording data. If it is in program mode then the controller waits until it encounters a trigger command in a running program.

After the trigger button has been pressed, the text on the button changes to 'Halt' whilst the scope is running. If the scope is in the one-shot mode, then after the data has been recorded and plotted on the display, the trigger button text returns to 'Trigger', indicating that the operation has been completed.

The scope can be halted at any time when it is running, and the trigger button is displaying the 'Halt' text, by pressing this button.

### *Reset Scope Configuration*

The current scope configuration ( the state of all the controls) is saved when the scope window is closed, and retrieved when the scope window is next opened. This removes the need to re-set each individual control every time the scope window is opened.

The configuration reset button (located at the bottom right hand side of the scope control panel) can be pressed to reset the scope configuration, clearing all controls to their default values.

### *Status Indicator*

The status indicator is located in between the options and configuration reset buttons. This lamp changes colour according to the current status of the scope, as follows :

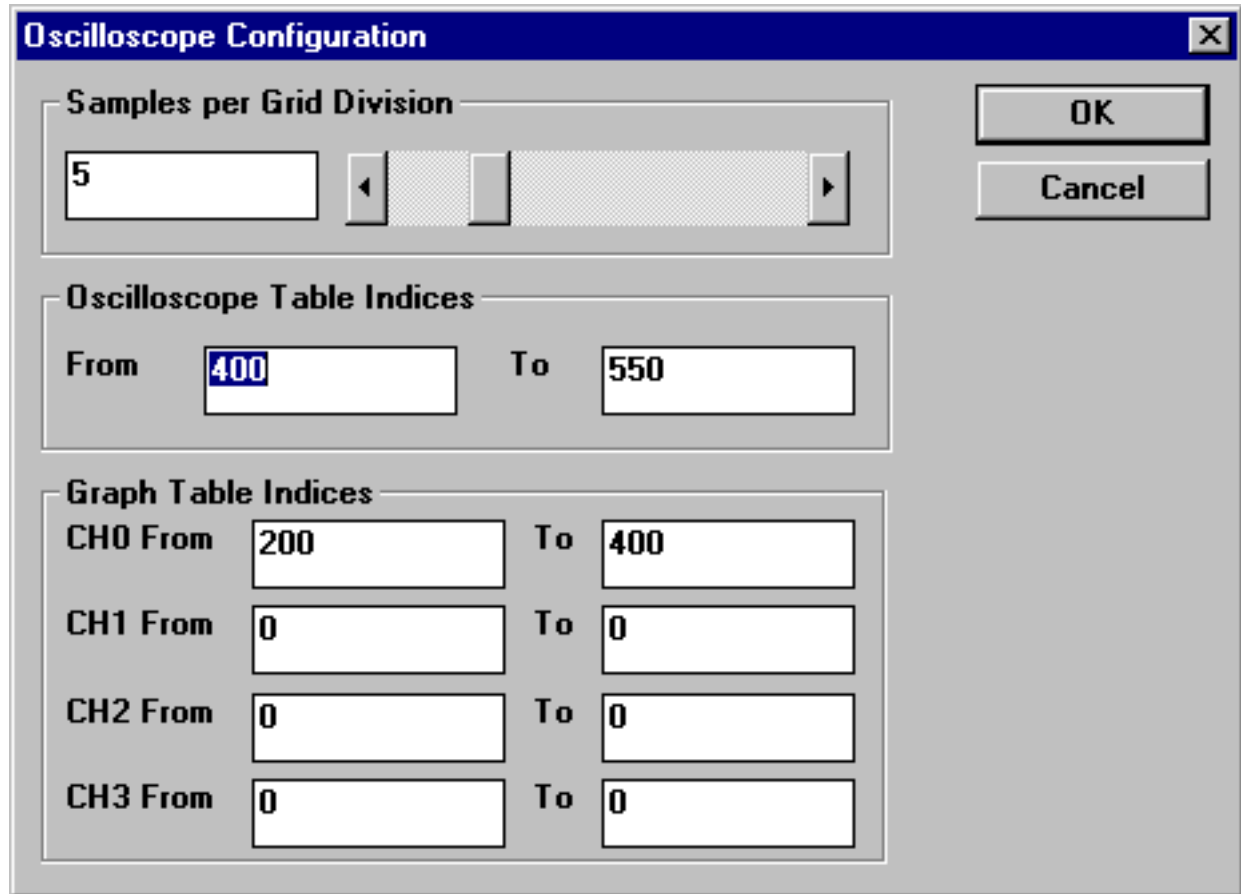
Red - scope stopped.

Black - polling controller waiting for it to complete recording the required data.

Yellow - retrieving data from the controller.

**Advanced Oscilloscope Configuration Options**

When the options button is pressed the advanced oscilloscope configuration settings dialog is displayed, as shown below. Click the mouse button over the various controls to reveal further information.



***Samples per division***

The scope defaults to recording five points per horizontal (time base) grid division. This value can be adjusted using the adjacent scrollbar.

To achieve the fastest scope 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 msec 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.

***Scope Table Values***

The controller records the required parameter data values in the controller "TABLE" data structure prior to uploading these values to the scope. By default, the lowest scope 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 scope table value can be reset.

The lower scope table value is adjusted by setting focus to this text box and typing in the new value. The upper scope 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 replaced by the scope.

### **Graph Table Indices**

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 analog inputs are being recorded, then the fastest scope resolution (sample rate) is the number of analog channels in msec ( ie 2 analog 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.

There is a maximum table size on the controller, hence it is not possible to enter table channel values in excess of this value; nor enter a lower scope table value, or increase the samples per grid division to a value which causes the upper scope table value to exceed the controller maximum table value. 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 scope automatically resets the number of samples per grid division.

### **General Information**

#### **Displaying Controller Table Points:**

If the scope 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. The motion parameter trace does not move.

#### **Data Upload from the controller to the scope**

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 scope 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 scope 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 scope controls -**

Whilst the scope is running all the scope controls except the trigger button are disabled. Hence, if it is necessary to change the time base or vertical scale, the scope 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 scope. 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 scope due to the time taken to upload the data via the serial link.

### 10.11.10 Terminal windows

The terminal window provides a direct connection to the *Motion Coordinator Series 2*. Most of the functions that must be performed during the installation, programming and commissioning of a system with a *Motion Coordinator Series 2* have been automated by the options available in the *MotionPerfect* menu options. However, if direct intervention is required the terminal window may be used. Up to 4 terminal windows can be opened simultaneously over the single serial port. They can be used for:

Issuing direct commands to the BASIC interpreter: For example:

```
>>? MPOS AXIS(9)
```

Only channel 0 can be used for issuing commands.

Channels 5,6 and 7 can be used to provide input/output windows to programs running on the Motion Coordinator.

If *Motion Perfect* is not "connected" to the controller, then the terminal window may be used to talk to the controller in a 'No *Motion Perfect* connection' mode. In this mode the terminal window has complete control over the serial link.

#### *Channel number*

This combobox allows the user to select one of the valid async communication channels.

#### *VT100 emulation*

The *Motion Coordinator Series 2* expects to talk to a terminal that accepts the DEC VT100 terminal protocol. This sets the terminal window to emulate this terminal.

#### *ASCII emulation*

This mode will echo the ASCII description for the non printing characters received. Additionally the CR and LF will cause the corresponding action.

#### **No *Motion Perfect* Connection**

In this mode *MotionPerfect* talks directly to the controller without the normal channel control. As *MotionPerfect* has no valid controller, the user must supply the information about which port the controller is connected to.

Once connected the terminal window emulates a VT100 Terminal.

## 10.12 Hints and Tips For Using *Motion Perfect*

### Multiple windows opened on the *Motion Perfect* desktop

It should be remembered that it is possible to open several windows on the *Motion Perfect* desktop, which will run concurrently. For example, it is possible to have the axis parameter window to update and monitor these motion parameters, whilst stepping through a program displayed in an edit window, with the keypad emulator and terminal windows open to enter information to the program and display its responses. Also, up to four terminal windows could be opened at any time, and then the programs running on the controller can print messages and receive inputs from the separate terminal windows.

### Program Editing and Debugging

*Motion Perfect* provides facilities for all program handling operations, such as edit/delete/rename/new/select/ and copy. These should be used as opposed to entering Trio BASIC system commands via the terminal window - if the latter method is used *Motion Perfect* is unaware of the changes made. It is then necessary to perform a "Check Project" to resolve the inconsistencies.

You should also avoid executing a controller soft reset (the 'EX' command) in the terminal window, since this will crash *Motion Perfect*. Use the 'Controller' 'Reset the controller' menu option.

Remember that you do NOT need to close an edit window to run a program. It saves time not to.

It is better to open an edit session for each program you want to see BEFORE running any programs. If there are programs already running then it is not possible to open an edit session, but you can open a debug session using the 'Tools' 'Program debugger' menu option.

Running programs can be set to trace (single-stepping) and back again by clicking the left hand square yellow box alongside each program. This allows you to see where a program is currently executing. The right hand box normally used to set a program running or to stop one. It is yellow when the program is tracing and may be clicked to step a line.

Avoid turning the power on and off or removing the serial lead when using *Motion Perfect*. If you do so a communications error message will appear, and *Motion Perfect* enter the disconnected mode.

You can force the *Motion Perfect* Project Manager to check the PC project-controller copies of the programs are identical at any time by running the 'Check project' facility, found under the 'File' menu.

### Running *Motion Perfect* in the Disconnected Mode

*Motion Perfect* can be run in a 'disconnected mode' if it is unable to find a controller and open a valid project. This may occur if it does not find any controllers connected to the PC, the controller it finds has a system software version prior to 1.16, or if the project consistency check fails and the user selects the 'Abort' option.

In this mode all the project related facilities are disabled, and only the terminal ( VT100 emulation ), the load system software and communications setup options, and the online help are available. The terminal can be opened and an attempt made to establish communications with the controller. If the controllers line mode '>>' prompt is returned when the PC 'enter' key is pressed, then the controller can be interrogated using the Trio BASIC system commands (see the Trio BASIC online help for further information). If it has a pre version 1.16 system software, then the latest version can be loaded (see the 'Loading a new system software version' for further information.)

### The Project Backup File

The project manager makes a backup copy of the programs within a project when the project is FIRST opened, ( and the project - controller verification process has successfully completed, and any inconsistencies have been resolved by the user). This backup copy can be loaded if the controller version of the programs become corrupted during your development session.

If the controller stops responding during a development session, and you re-connect to the same project, then it is likely the controller - PC project file will be inconsistent. In this case the 'Check project options' dialog will be displayed, and if the current project is re-opened then the backup copy of the project will be OVERWRITTEN. Hence it is necessary to determine which copy of the programs you wish to use before re-connecting *Motion Perfect*.

If you want to investigate the PC Project-controller program versions further then open a terminal in the disconnected mode to select and list the programs on the controller. Any PC based editor or word processor can be used to examine the PC backup project file copy of the programs. In this way, the location of the uncorrupted or latest version of the programs can be identified.

Re-connect to the controller after having established why communications were originally lost. If the controller - PC project files are inconsistent, the 'Check project options' dialog will be displayed. Now click either the 'Resume' button on the 'Project check options' to resolve each individual program inconsistency, or click the 'Load' different project button to reload the backup project.

It should be noted that the PC project copy of a program is updated during a *Motion Perfect* development session whenever an edit session for this program is closed.

### **Resume Project Check to resolve PC Project vs Controller Program Inconsistencies**

Click the 'Resume' project check option on the 'Check project options' dialog, and observe the progress messages displayed by the 'Check project' dialog. If the controller-PC project copies of a program differ, the following message will be displayed :

The copy of the program <your program name> on disk differs from the one on the controller.  
We recommend that you save the file into the project.  
Do you wish to save the copy on the controller or load the copy from the project ?

Click the 'Load' option to load the PC copy of the program onto the controller, or click the 'Save' option to save the copy of the program on the controller into the PC project file.  
When all the program inconsistencies have been resolved the 'Project consistent' message will be displayed. Click the 'Ok' button and the *Motion Perfect* desktop will open.

### **Retrieving the Backup Project**

If you want to retrieve and reload the backup copy of a project click the 'Load' a different project button on the 'Check project options'. The 'Load project' dialog will open, and should be used to select the required backup project. Use the 'Drives' and 'Directories' list boxes to find the required backup project file, whose name and path will be displayed under the 'Project path' label. Now click the 'OK' button. A message box will appear with the following text:

This will erase the actual contents of the controller and load the project <your path><your project name> in its place.

Are you sure?

Click the 'Yes' button, and *Motion Perfect* will display the 'Check project' dialog as it loads all the programs from the project file to the controller. When this has completed successfully, the 'Project consistent' message will be displayed. Click the 'Ok' button and the *Motion Perfect* desktop will open. The project should now be saved under a different name ( as *Motion Perfect* has opened the backup project file.). Hence, select the 'File' 'Save project as...' option and enter a new project name. See the 'New Project Dialog' section for further information about how to use this dialog and enter the required details, since the 'save project as' and 'new project' dialogs work in the same way.

If you want to abandon changes made during a development session and reload the backup copy made at the start of the session, then select the 'File' 'Revert to backup' menu option.

## **10.13 Problems and frequently asked questions**

### ***Motion Perfect can't find my Motion Coordinator:***

Is the *Motion Coordinator* powered on ?

Is the serial lead connected between the *Motion Coordinator* serial port A and the PC ?

Have *Motion Perfect's* baudrate, parity or stopbits been adjusted ?

Try switching the *Motion Coordinator* on with a *Motion Perfect* terminal running in disconnected mode first.

### ***Motion Perfect found my Motion Coordinator, but reported incorrect version number ?***

*Motion Perfect* requires system software version 1.16 or higher on the *Motion Coordinator* to run.

If the controller has a system software version between 1.02 and 1.15, then enter *Motion Perfect* disconnected, and use the load system software option to load the latest system software (which can be found in the MPERFECT folder created when *Motion Perfect* was installed on your hard disk).

If the existing system software is less than 1.02 you can use the DOS program LOAD.EXE.

See the 'Loading a new System Software Version' section.

### ***Motion Perfect cannot find my correctly powered up and connected Motion Coordinator ?***

The *Motion Coordinator* may have run a very comms intensive program following power-on. If so open the terminal in the disconnected mode, and halt the program(s). Then try to connect.

### ***Motion Perfect unexpectedly times out reporting : 'Communications Error' 'Communications with the controller have been lost' :***

The *Motion Perfect* commands are executed by the command line process on the controller. This command line is available using the terminal screen on channel 0. This means that if you type in a command on the command line which will detain the command line ( eg GET, LINPUT, INPUT, WAIT etc) then the *Motion Perfect* commands may not be processed, causing a communications error and disconnecting *Motion Perfect*. Run the commands using a #5, #6 or #7 terminal channel instead to avoid the problem.

You will also get this message if you switch the controller off without disconnecting *Motion Perfect*.

### ***When I use Motion Perfect, I encounter serial comms problems.***

Serial port on PC *Motion Perfect* uses the standard windows serial communications driver.

Communication errors can occur with some computers or when other programs are running. This problem can be recognised by the appearance of a message "Hardware receive error" in a *Motion Perfect* warning window. To avoid these potential problems:

Use a PC with a serial FIFO (Most modern PCs have this)

Under Window95 you may need to switch the serial FIFO on: This can be done by going to the System Control Panel. Under Device Manager, double click Ports then select the communications port you are using. Under Properties... Select "Port Settings" then "Advanced" and click the box marked 16550 FIFO. Then select the default buffer size.

On portable PCs it may be necessary to switch off power saving features and remove desk accessories such as battery life displays.

Ensure there are no TSR programs or mouse drivers which use the serial port currently installed.

If a hardware receive error occurs, a message dialog will appear to inform you, and *Motion Perfect* disconnects from the controller.

### ***Do Motion Perfect Oscilloscope and CAM movements interfere with each other ?***

The *Motion Perfect* oscilloscope uses the TABLE which is also used by the CAM and CAMBOX command. Care should be taken when using both to set the area of table used by *Motion Perfect* to an area NOT used by your application program. The area used is defined in the oscilloscope "options" page.



**10.14 Loading a new system software version onto the *Motion Coordinator*.**

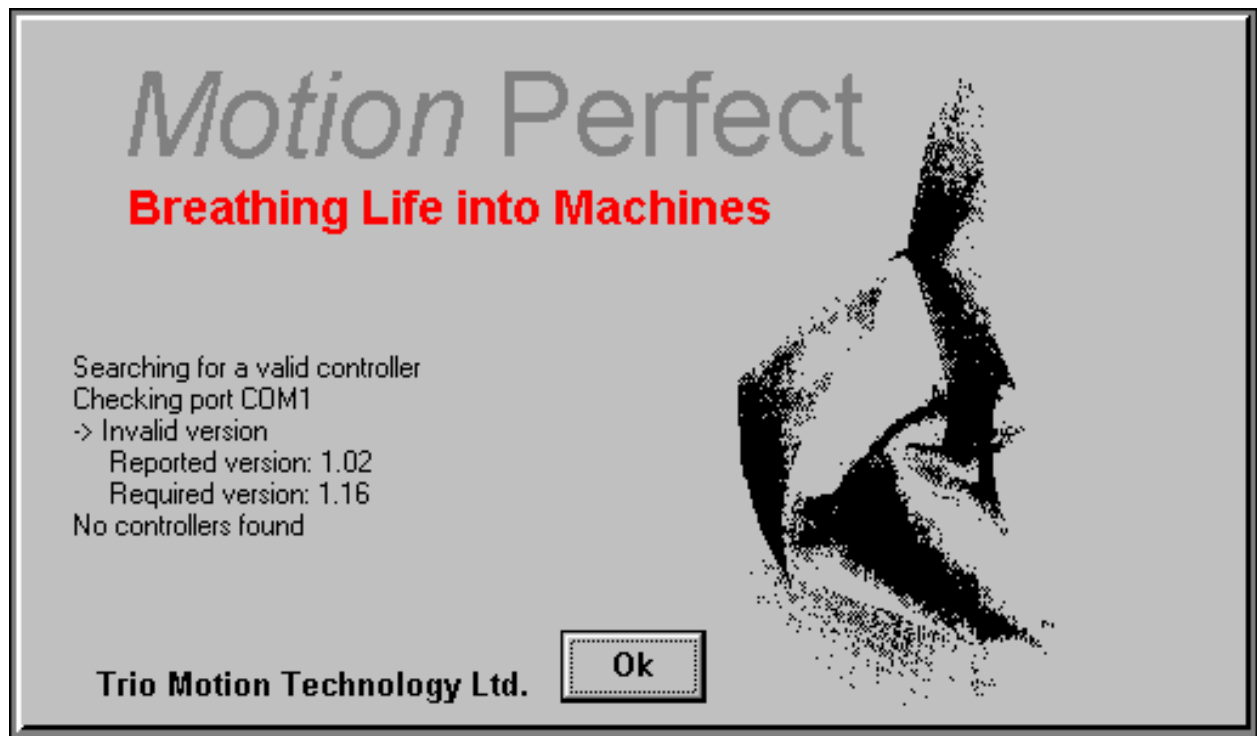
*Motion Perfect* requires system software version 1.16 or higher on the controller to run. The latest version is supplied on the *Motion Perfect* installation disk and can be loaded onto the controller if an update is required. If the controller's current system software version is between 1.02 and 1.15 then *Motion Perfect* can be used to perform the download, otherwise if the version is less than 1.02 the DOS 'LOAD.EXE' program should be used.

**Loading New System Software Using *Motion Perfect***

If the controller has a system software version between 1.02 and 1.15, then enter *Motion Perfect* in the disconnected mode, and use the load system software option to load the latest system software (which can be found in the MPERFECT folder created when *Motion Perfect* was installed on your hard disk).

It is very important to save any existing applications programs in the controller prior to loading the system software. You will have to use MCSETUP to do this at this point as *Motion Perfect* will be unable to do so until after the update.

On running *Motion Perfect* you will be warned by the following text on the introductory splash screen: The actual reported version may differ from the picture below.



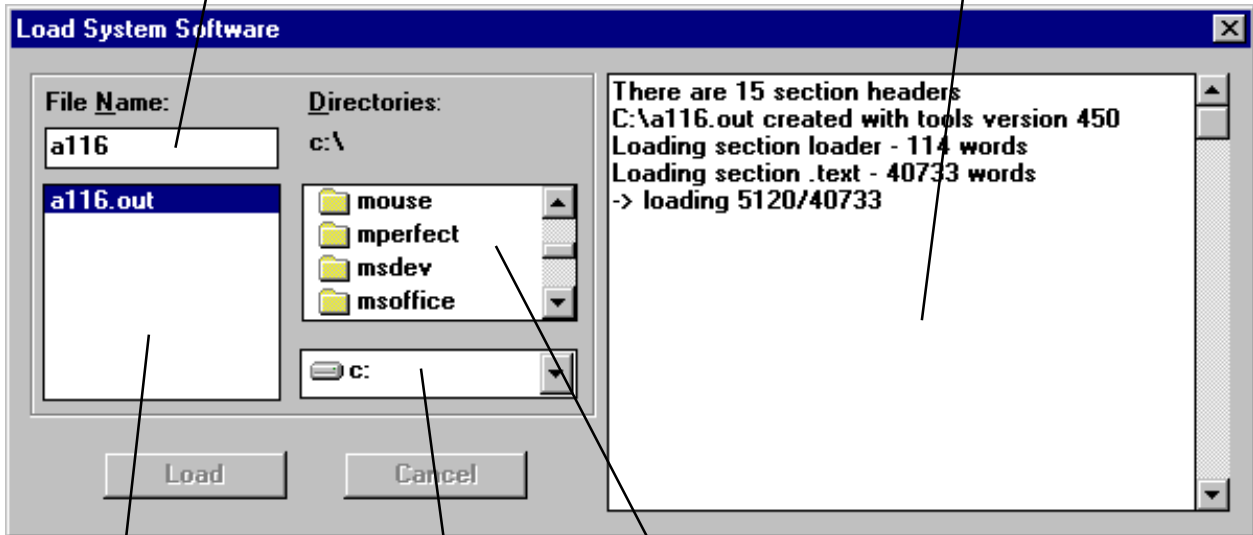
Click OK and enter *Motion Perfect* in the disconnected mode. Under the 'Controller' menu select 'Load System Software', and the 'Load System Software' dialog will appear:

Select the system software file Axxx.OUT (where xxx is the latest version number, MC204 users need Bxxx.OUT) using the file selector. (You can move up a directory level by double clicking on [..]) Read the warnings carefully before continuing.

The download will take approximately 10 minutes, after which *Motion Perfect* will perform a CRC check to verify the software on the controller has been downloaded successfully. If it has, a dialog will appear confirming this and asking whether the user wants to write the system software to EPROM.

Selected Axxx.OUT file name appears here

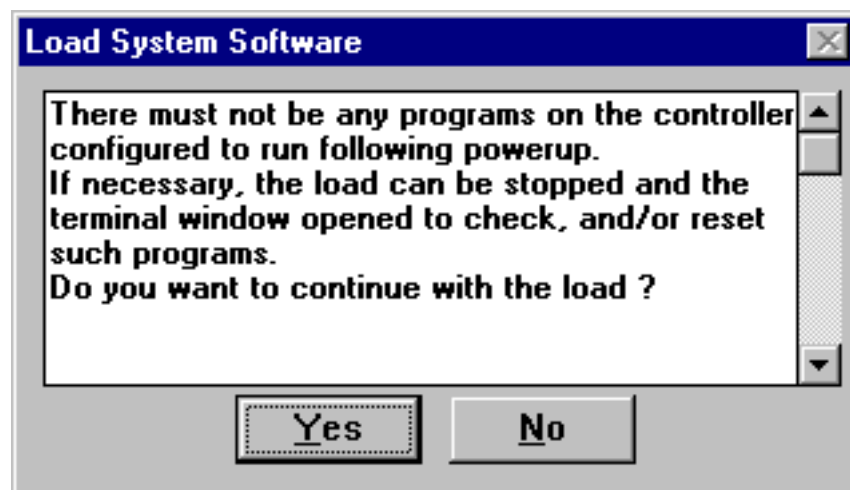
Progress messages will appear here



Axxx.OUT files found displayed here

Use this Drives List box and this Directory list box to locate the required A.OUT file

Click the 'Yes' button to initiate this process. Once the system software has been loaded, *Motion Perfect* will try to connect to the controller again, and if successful it will attempt to verify the existing project on the controller. As the system software has just changed, the best option would be to clear the controller, ready to reload the programs that were saved to disk before the new system software was loaded. Hence select the 'New' option from the 'Check project options' dialog which appears.



If this is a previously installed system then you must reload the stored programs from disk. To do this, use the Load File option from the File menu to reload the files one by one. As these files are loaded they will be copied into the project directory that was specified during the initialisation of *Motion Perfect* and loaded onto the controller as well.

If this is a new system, then programs must be created using the New program option of the Program menu. This option will ask you for a program name which will be used to identify the program on the controller and on the PC as well. Once programs exist in the *Motion Perfect* project, then their names will appear on the directory list on the control panel.

To create a new program the 'New program' option of the Program menu must be used.

### **Loading new system software version using the DOS loader**

If your Motion Coordinator has a system software version prior to 1.02 you will need to use the DOS loader program LOAD.EXE to update the SERIES 2 system software.

Use MCSETUP to save any existing applications programs in the controller prior to loading the system software, and then type the following at the DOS command prompt:

```
LOAD Axxx.OUT
```

Where xxx is the latest version number. After updating the system software, run *Motion Perfect*, and use the 'Load file' option under the 'File' menu to reload the old programs previously saved to the PC.

If you encounter an error when loading the new system software, cycle the power to the controller and repeat the process, this time typing the following at the DOS command prompt :

```
LOAD Axxx.OUT -v
```

Monitor the messages displayed on the screen, and if the load is again unsuccessful then contact Trio and quote the error message displayed.

### **Further Information**

For further information use the on-line help provided with *Motion Perfect*. This contains full details about how to use all the facilities provided by *Motion Perfect*, and the Trio Basic language.



## 11.0 The TRIO Fibre Optic Network

### 11.1 General Description

The TRIO fibre optic network has been designed to link up to fifteen Motion Coordinator modules and membrane keypads. Any number of either type of module can be on the network up to the maximum of fifteen but at least one must be a Motion Coordinator.

To use the network facilities version 1.00 or higher of *Motion Coordinator* SERIES 1 or SERIES 2 BASIC is required together with a keypad, if required, of version 2.01 or higher.

The physical connection of the network takes the form of a ring. The interconnections between nodes being made with fibre optic cable.

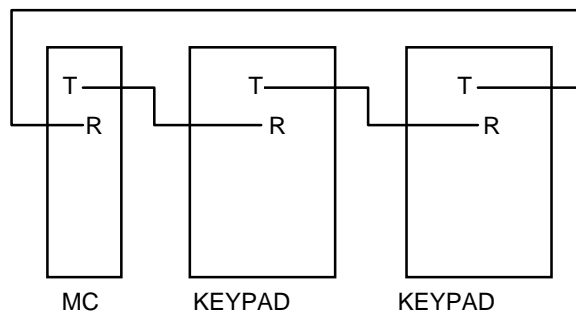


Fig 11.1 Example of Network Connection

There are three fixed types of message that can be transmitted round the network. These are:

- (i) Character transfer - a string of characters is transmitted to a specified node e.g. a message to a display. This is performed with a PRINT# command. Reception of characters to a Motion Coordinator is carried out with the GET# command.
- (ii) Direct variable transfer - a specified variable on a given Motion Coordinator node can be modified by another Motion Coordinator node independent of the BASIC program running on the receiving node. This is achieved with the SEND command.
- (iii) Keypad offset - a special message sent to a membrane keypad node to set it into network mode and to tell it where on the network to direct its key presses to. This message type is also transmitted by the SEND command.

Information is transmitted at 38400 baud with one start and one stop bit per character.

### 11.2 Connection of Network

All the nodes have to be connected to form a ring. Single core fibre optic cable is used terminated with Hewlett Packard "Versatile Link" connectors. The fibre optic modules on the Motion Coordinator and membrane keypad are colour coded as follows:

Grey or Black - Transmitter  
Blue - Receiver.

To form the ring the transmit of a node is connected to the receive of the next node, i.e. grey/black to blue. The transmit of the final node is connected to the receive of the first thus completing the ring.

The order in which nodes are connected is not critical but the performance of the network can be affected by rearranging the order of the nodes. This is especially true if there are a large number of nodes in the network and a large amount of data being passed between two nodes. If these two nodes are not adjacent in the ring any nodes between them will be tied up retransmitting messages and will not be able to transmit as priority is given to retransmission of messages. This can be a particular problem if one of the nodes retransmitting is a membrane keypad as this can cause poor response to key presses.

The standard cable lengths supplied by TRIO for network interconnections are:

1m, 5m, 10m and 30m.

### 11.2.1 Network Node Addressing

Nodes on the network are not given absolute addresses. Instead, when a message is sent it is labelled with the address of a destination node which defines the number of nodes along the ring from the sender that the message must travel. The addresses are in the range 10, denoting a message for the next node along the ring, to 24, denoting a message for the fifteenth node along the ring from the sender.

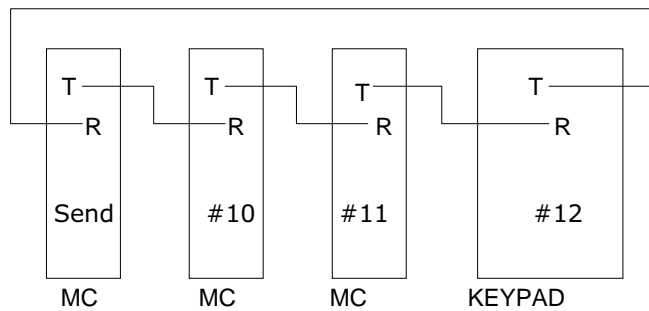


Fig 11.2 Node Addressing Example

The example in fig 11.2 shows the destination node addresses for a network with respect to the transmitting Motion Coordinator. If a message was sent with the address 13 then the message would come back to the sender. Likewise if address 14 was used the message would completely traverse the ring once and finish up at node #10. If address 18 was used then the message would go twice round the ring and finish up at node #10.

## 11.3 Network Programming

### 11.3.1 BASIC Commands

The transmission and reception of messages on the network is performed by four BASIC commands.

#### 11.3.1.1 GET#n,VR(x)

|              |                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type:        | Command.                                                                                                                                                                                                               |
| Description: | Waits for the arrival of a single character on the the specified input device. The ASCII value of the received character is stored in the chosen variable. The characters received are held in a 256 character buffer. |
| Parameters:  |                                                                                                                                                                                                                        |
| n:           | Number to specify how the returned value generated.<br>3 - value returned is defined by DEFKEY<br>4 - value returned is character received                                                                             |
| x:           | Variable number in the range 0 to 250.                                                                                                                                                                                 |

**11.3.1.2 KEY#n**

Type:                   Function.

Description:           Returns TRUE or FALSE depending on whether a character has been received on the specified input device or not. The character is not read nor the receive state reset.

Parameters:

n:                        Number to specify serial input device.  
                           3 - network input from fibre optic port via DEFKEY table  
                           4 - network input from fibre optic port

**11.3.1.3 PRINT#n,**

Type:                   Command.

Description:           The PRINT# command allows the BASIC program to output a series of characters to the specified output device. The PRINT# command can output parameters, variables, fixed ASCII strings and single ASCII characters. Multiple items to be printed can be put on the same PRINT line provided they are separated by a comma or semi-colon. The comma and semi-colon are used to control the format of strings to be output.

Parameters:

n:                        Number from 10 to 24 to specify number of nodes from transmitting node the message must be sent. See 11.2.1 for details of network node addressing.

**11.3.1.4 SEND(n,type,data1[,data2])**

Type:                   Command.

Description:           Outputs a network message of a specified type to a given node.

Parameters:

n:                        Number from 10 to 24 to specify number of nodes from transmitting node the message must be sent. See 11.2.1 for details of network node addressing.

type:                   Message type:  
                           1 - Direct variable transfer  
                           2 - Keypad offset

data1:                  If type=1 data1 is the variable number on the destination node to modify.  
                           If type=2 data1 is in the range 10..24 to specify the number of nodes from the keypad that the key characters are sent.

data2:                  If type=1 data2 is the value to change the specified variable to.  
                           If type=2 data2 is not used.

### 11.3.2 Examples of network programming

Consider a four axis machine which is 30m long. The operator has need to make machine adjustments at either end of the machine, thus requiring two stations for operator input. This can be achieved using a Motion Coordinator, four Servo Daughter Boards and two membrane keypads. The two keypads and the Motion Coordinator will be networked together so input from both keypads is given to the Motion Coordinator and any resulting change in machine parameters is displayed at either end of the machine.

The network should be connected as shown in fig 11.3.

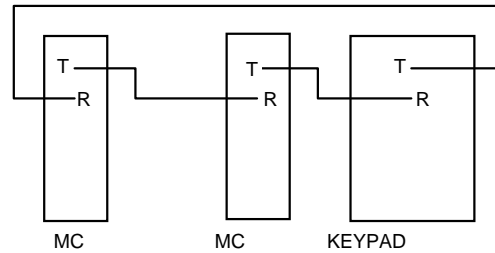


Fig 11.3 Network Example 1

The program could look something like this:

```
length=2
SEND(10,2,11)' Set offset on first keypad to 2, i.e. send key press to MC
SEND(11,2,10)' Set offset on second keypad to 1, i.e. send key press to MC
PRINT #10,CHR(12);CHR(14);CHR(20)' Clear first display & make cursor invisible
PRINT #11,CHR(12);CHR(14);CHR(20)' Clear second display & make cursor invisible
PRINT #10,"SPEED:":PRINT #10,"LENGTH:"' setup display
PRINT #11,"SPEED:":PRINT #11,"LENGTH:"

REPEAT
    IF KEY#3 THEN GOSUB read_key ELSE GOSUB do_motion
UNTIL FALSE

read_key: GET #3,k
    IF k>9 THEN GOTO not_num
    a=a*10+k
    IF a>9999 THEN a=0
    PRINT #10,CHR(27);CHR(72);CHR(7);a[6,0]
    PRINT #11,CHR(27);CHR(72);CHR(7);a[6,0]
not_num: IF (k=10)&(VR(length)<10000) THEN VR(length)=VR(length)+0.1
    IF (k=11)&(VR(length)>0.1) THEN VR(length)=VR(length)-0.1
    PRINT #10,CURSOR(28);VR(length)[8,1]
    PRINT #11,CURSOR(28);VR(length)[8,1]
RETURN

do_motion: 'Main motion routine
```



Consider a five axis machine with one membrane keypad. The axes can be divided into two distinct blocks. Two axes are concerned with the feeding of the material and the other three with the cutting and packing of the material. To get five axes of control we could use two Motion Coordinators Series 1 and five Servo Daughter Boards. Our network will consist of the two Motion Coordinators and the membrane keypad. It should be connected as shown in fig 11.4.

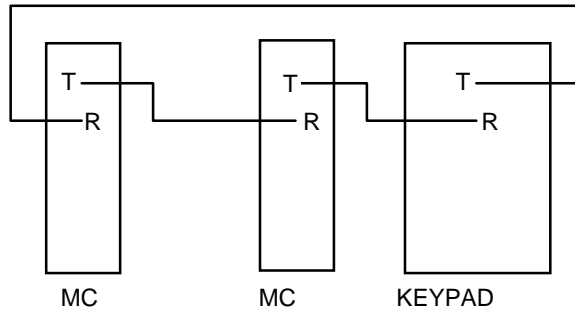


Fig 11.4 Network Example 2

As there are now two Motion Coordinators in the network it is possible and probably necessary to have a program on each.

For the purposes of this example the three axis node will act as the master and issue instructions to the two axis node. The three axis node will also receive all input from the keypad.

The program for the three axis node could be as follows:

```

init:      SEND(11,2,10)'      Set offset on keypad
          PRINT #11,CHR(12);CHR(14);"INITIALISING....."
          PRINT #10,"I";'      Send command to two axis node to initialise
          GOSUB initax'      Initialisation routine
          WAIT UNTIL VR(200)=99'Two axis node signals ready by setting variable 200
          PRINT #11,CHR(12);CHR(14);"MACHINE READY"

          'Get required parameters
setlen:    PRINT #11,"FEED LENGTH:";VR(0)[6.1];
          GET #3,VR(100)
          IF VR(100)=13 THEN GOTO setsp
          IF (VR(100)=10)AND(VR(0)<1000000) THEN VR(0)=VR(0)+0.1
          IF (VR(100)=11)AND(VR(0)>0.1) THEN VR(0)=VR(0)-0.1
          PRINT#11,CHR(27);CHR(72);CHR(32);VR(0)[6.1];
          GOTO setlen

setsp:     PRINT #11,"SPEED:";VR(1)[2];
          GET #3,VR(100)
          IF VR(100)=13 THEN GOTO initnet
          IF (VR(100)=10)AND(VR(1)<100) THEN VR(1)=VR(1)+0.01
          IF (VR(100)=11)AND(VR(1)>0.01) THEN VR(1)=VR(1)-0.01
          PRINT #11,CHR(27);CHR(72);CHR(46);VR(1)[2];
          GOTO setsp

initnet:   'Update variable values on two axis node
          SEND(10,1,0,VR(0))
          SEND(10,1,1,VR(1))

start:     PRINT #11,CHR(12);CHR(14);"PRESS START TO RUN"
readkey:   WAIT UNTIL KEY#3
          GET #3,VR(100)
          IF VR(100)<>20 THEN GOTO readkey
          PRINT #10,"G";
          WAIT UNTIL VR(200)=999
    
```

```
moves: 'Main motion routine
```

```
initax: 'Initialisation routine
```

The program on the two axis node could be as follows:

```
readinit: IF NOT(KEY#4) THEN GOTO readinit
          GET #4, VR(100)
          IF VR(100)<>73 THEN GOTO readinit
          GOSUB initax
          SEND(11,1,200,99)
```

```
readsp:  IF NOT(KEY#4) THEN GOTO readsp
          SPEED=VR(1)
          GET#4,VR(100)
          IF VR(100)<>71 THEN GOTO readsp
          SEND(11,1,200,999)
```

```
moves:  'Main motion routine
```

```
initax:  'Initialisation routine
```

The above example highlights a couple of useful points:

(i) The Motion Coordinator that is not controlling the membrane keypad must not transmit anything to or via the keypad until the keypad has been set into network mode by issuing the SEND command to give its keypad offset. If this restriction is not observed the message sent by the Motion Coordinator will be lost or corrupted. This example uses a simple form of handshaking by setting variables and sending characters to prevent this situation but another alternative is for the controlling Motion Coordinator to issue the SEND command on the first line of its program and for all other Motion Coordinators in the network to have a delay at the start of their program to allow time for the membrane keypads to be initialised.

(ii) When using the GET# command it is a good idea to test for the buffer being empty with the KEY# command, especially if reading input from a keypad, because it is possible for the buffer to overflow with unpredictable results if it is not emptied by reading from it with the GET# command.

## 11.4 Network Specification

### 11.4.1 Message Format

Each message is constructed of a header character, the message, a crc check (on direct variable transfers only) and an end of message character.

|                   | Bit 7 | Bit 6                             | Bit 5 | Bit 4 | Bit 3 | Bit 2          | Bit 1 | Bit 0 |
|-------------------|-------|-----------------------------------|-------|-------|-------|----------------|-------|-------|
| Header            | 1     | <-Address of Receiver>            |       |       |       | <Message Type> |       |       |
| Message (n bytes) | 0     | <-----Character Information-----> |       |       |       |                |       |       |
| CRC (byte 0)      | 0     | <-----Bits 6..0 of CRC----->      |       |       |       |                |       |       |
| CRC (byte 1)      | 0     | <-----Bits 14..8 of CRC----->     |       |       |       |                |       |       |
| End of Message    | 1     | <-Address of Receiver>            |       |       |       | 0              | 0     | 0     |

Note:

(i) Address of receiver is in the range 1 (for next node) to 15 (15th node on network). This is different to the node addressing used by BASIC and is for internal use only.

(ii) The number of message bytes (n) is determined by the message type as described below

(iii) CRC bytes are only used for direct variable transfers (message type 2)

### 11.4.2 Network Protocol

The sender places the message on the ring and assumes it arrives at its destination. Any packet checking must be done by the user because the nodes have no knowledge of the size of the net. Each successive node decrements the address by 1 and if the result is greater than zero it retransmits the message with the decremented address in the header and end of message. If the result is zero then the message is for that node and is processed as necessary. The retransmission is invisible to the user.

### 11.4.3 Message Types

The message types described here are for internal use only and are different to those used by the SEND command.

- 0 - End of message character.
  
- 1 - Character transfer. Message can be 1 byte long, e.g. a key press from a keypad, or several bytes long, e.g. PRINT statement.
  
- 2 - Direct variable transfer. The value of the variable specified is modified directly, without needing to be processed by the BASIC. Message length is 2 bytes (14 bits) for the variable number and 10 bytes (32 bit integer + 32 bit fraction) for the variable value. Ignored by membrane keypads.
  
- 3 - Keypad offset. Message is 1 byte long and tells the keypad the number of nodes along the ring to send key presses.

### 11.4.4 Network Buffers

The transmit and receive on all nodes is buffered. As far as the user is concerned the buffers in the membrane keypad do not exist as all the necessary housekeeping is performed by the keypad itself. The only concern to the user is the receive buffer on the Motion Coordinator. This is 256 bytes long and is used to store the message characters only. The header and end of message bytes are stripped off as they are received so it is not necessary to do this through the BASIC.

#### **11.4.5 Network Status**

The NETSTAT common parameter shows any errors that have occurred on the network since the last time this parameter was cleared. The errors are displayed as follows:

| <u>Bit</u> | <u>Error Type</u> | <u>Value</u> |
|------------|-------------------|--------------|
| 0          | TX Timeout        | 1            |
| 1          | TX Buffer Error   | 2            |
| 2          | RX CRC Error      | 4            |
| 3          | RX Frame Error    | 8            |

TX Timeout - shows that a problem has occurred while trying to put information into the transmit buffer.

TX Buffer Error - indicates an error on transmission from the buffer. This part of the operation is invisible to the user and if this error ever occurs please contact TRIO for further advice.

RX CRC Error - the received variable has become corrupt. The error flag is set but the variable isn't.

RX Frame Error - an incomplete or corrupt message has been received.