

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Trio Motion Technology
Motion Coordinator

Technical Reference Manual

Sixth Edition • 2004
Revision 4

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

This manual applies to systems based on the *Motion Coordinator* MC202, Euro 205 and MC216 with system software version 1.52, or higher. Or MC206 and MC224 with system software version 1.62, or higher.

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

Revision 6.4 April 2004

Copyright (C) 2000-2004 Trio Motion Technology Ltd.

All Rights Reserved

UK

Trio Motion Technology Ltd.
Shannon Way
Tewkesbury
GL20 8ND
United Kingdom

Phone: +44 (0)1684 292333
Fax: +44 (0)1684 297929

USA

Trio Motion Technology LLC.
1000 Gamma Drive, Suite 206
Pittsburgh
PA 15238,
USA

Phone: +1 412 968 9744
Fax: +1 412 968 9746

⚠ SAFETY WARNING ⚠

During the installation or use of a control system, users of Trio products must ensure there is no possibility of injury to any person, or damage to machinery.

Control systems, especially during installation, can malfunction or behave unexpectedly. Bearing this in mind, users must ensure that even in the event of a malfunction or unexpected behaviour the safety of an operator or programmer is never compromised.

CONTENTS

INTRODUCTION 1-1
Setup and Programming 1-4
Products 1-5
System Building 1-11
System Examples 1-12
Features and Typical Applications 1-15
The Trio Motion Technology Website 1-16

HARDWARE OVERVIEW 2-1
Motion Coordinator MC202 2-3
Connections to the MC202 2-4
Serial Port Connections 2-7
MC202 - Feature Summary 2-8
Motion Coordinator Euro205(X) 2-10
Axis Configuration 2-11
Connections to the Euro 205(X) 2-12
Euro 205(X) Backplane Connector 2-13
Euro205 - Feature Summary 2-19
Euro205X - Feature Summary 2-20
Motion Coordinator MC206 2-22
Connections to the MC206 2-25
Amplifier Enable (Watchdog) Relay Output 2-28
Reference Encoder Input 2-29
Universal Serial Bus 2-29
MC206 - Stepper Outputs / Encoder Inputs 2-30
MC206 - Feature Summary 2-31
Motion Coordinator MC216 2-33
Connections to the MC216 2-35
MC216 - Feature Summary 2-40
Motion Coordinator MC224 2-41
Connections to the MC224 2-43
MC224 - Feature Summary 2-48

INSTALLATION 3-1
Motion Coordinator MC202 3-3
Motion Coordinator Euro 205 3-5
Motion Coordinator MC216 3-6
Motion Coordinator MC206 3-8

EMC Considerations 3-9
Installation Requirements to Ensure Conformance 3-11
Motion Coordinator MC202 3-11
Motion Coordinator Euro 205 3-11
Motion Coordinator MC206 3-12
Motion Coordinator MC216 3-13

DAUGHTER BOARDS 4-1
Fitting and Handling Daughter Boards 4-4
MC216 + Axis Expander Slot Sequence 4-6
Fitting Daughter Boards to the Euro205 4-7
Fitting Daughter Boards to the MC206 4-9
Servo Encoder Daughter Board 4-11
Servo Resolver Daughter Board 4-14
Reference Encoder Daughter Board 4-16
Stepper Daughter Board 4-18
Stepper Encoder Daughter 4-20
Hardware PSWITCH Daughter Board 4-23
Analog Output Daughter Board 4-25
SSI Servo Encoder Daughter Board 4-26
Differential Stepper Daughter Board 4-29
CAN Daughter Board 4-31
SERCOS Daughter Board 4-32
SLM Daughter Board 4-33
USB Daughter Board 4-35
Ethernet Daughter Board 4-36
Profibus Daughter Board 4-37

EXPANSION MODULES 5-1
Input/Output Modules 5-3
CAN 16-I/O Module (P315) 5-3
CAN Analog Inputs Module (P325) 5-11
Operator Interfaces 5-15
Membrane Keypad (P503) 5-16
Mini-Membrane Keypad (P502) 5-19
Programming the Membrane Keypad 5-20
Keypad KEY ON - KEY OFF Mode 5-24
FO-VFKB Fibre Optic Keypad/Display Interface 5-26
FO-VFKB Keypad Interface 5-26
Communications Adaptors 5-28
Serial to RS485 Adapter: (P348) 5-28
P349 - Serial to RS485 adapter to an MC202: 5-30

P435 - Serial to Fibre-Optic Adapter 5-31

SYSTEM SETUP AND DIAGNOSTICS 6-1
Preliminary Concepts: 6-3
System Setup 6-3
Preliminary checks 6-4
Checking Communications and System Configuration 6-4
Input/Output Connections: 6-6
Setting Servo Gains 6-8
Proportional Gain 6-10
Integral gain 6-10
Derivative gain 6-10
Output Velocity Gain 6-11
Velocity Feed Forward Gain 6-11
Diagnostic Checklists 6-13

PROGRAMMING 7-1
What is a program? 7-3
Sequence 7-4
Selection 7-5
Iteration 7-5
Controller Functions 7-7
. 7-7
Identifiers 7-8
Expressions 7-9
Parameters 7-11
Command Line Interface 7-15

TRIO BASIC COMMANDS 8-1
Motion and Axis Commands 8-11
Input / Output Commands 8-45
Program Loops and Structures 8-65
System Parameters and Commands 8-78
Process Parameters and Commands 8-129
Mathematical Operations and Commands 8-133
Constants 8-150
Axis Parameters 8-152

PROGRAMMING EXAMPLES 9-1
Example Programs 9-3

SUPPORT SOFTWARE 10-1
Motion Perfect 2 10-3
System Requirements: 10-4
New Features for *Motion* Perfect 1 Users 10-5
Connecting *Motion* Perfect to a controller 10-6
Motion Perfect 2 Projects 10-7
The Project Check Window 10-7
Project Check Options 10-8
The *Motion* Perfect Desktop 10-11
Main Menu 10-12
Controller Menu 10-13
Controller Configuration 10-14
Feature Enable 10-15
Loading New System Software 10-17
Flashstick support 10-19
Lock / Unlock 10-20
Motion Perfect Tools 10-21
Terminal 10-22
Axis Parameters 10-23
Oscilloscope 10-26
Keypad Emulation 10-35
Table / VR Editor 10-37
Jog Axes 10-38
Digital IO Status 10-42
Analogue Input Viewer 10-44
Linking to External Tools 10-45
Control Panel 10-47
Creating and Running a program 10-52
The *Motion* Perfect Editor 10-53
Editor Menus 10-56
Program Debugger 10-58
Running Programs 10-61
Making programs run automatically 10-62
Set Powerup Mode 10-62
Storing Programs in the Flash EPROM 10-63
Configuring The *Motion* Perfect 2 Desktop 10-64
Communications 10-64
Changing Comms Port Parameters 10-65
Editor Options 10-67
General Options 10-67
Saving the Desktop Layout 10-68
Running *Motion* Perfect 2 Without a Controller 10-69

MC Simulation 10-69
CAD2Motion 10-72
DocMaker 10-73
Project Autoloader 10-74
Files 10-75
. 10-75
Running the program 10-76
Script Commands 10-77
Script File 10-82

FIBRE-OPTIC NETWORK 11-1
General Description 11-3
Connection of Network 11-4
Network Programming 11-6
Examples of network programming 11-8
Network Specification 11-12

USING THE TRIO ACTIVEX CONTROL 12-1
Introduction 12-3
Requirements 12-3
Installation of the ActiveX Component 12-3
Using the Component 12-3
Connection Commands 12-4
Properties 12-7
Motion Commands 12-9
Process Control Commands 12-18
Variable Commands 12-19
Input / Output Commands 12-22
General commands 12-28
Events 12-30
TrioPC status 12-31

COMMUNICATIONS PROTOCOLS 13-1
MODBUS RTU 13-3
Introduction 13-3
Initialisation and Set-up 13-3
Modbus Technical Reference 13-4
Glossary 13-7
Profibus 13-8
Installation and Set-up 13-8

DeviceNet 13-13
Installation and Set-up 13-13
DeviceNet Information 13-13
Connection Types Implemented 13-14
DeviceNet Objects Implemented 13-14
MC Object 13-20
DeviceNet Status LEDs 13-25
Ethernet 13-26
Default IP Address 13-26
The Subnet Mask 13-27
Connecting to the Trio Ethernet Daughter Board 13-28

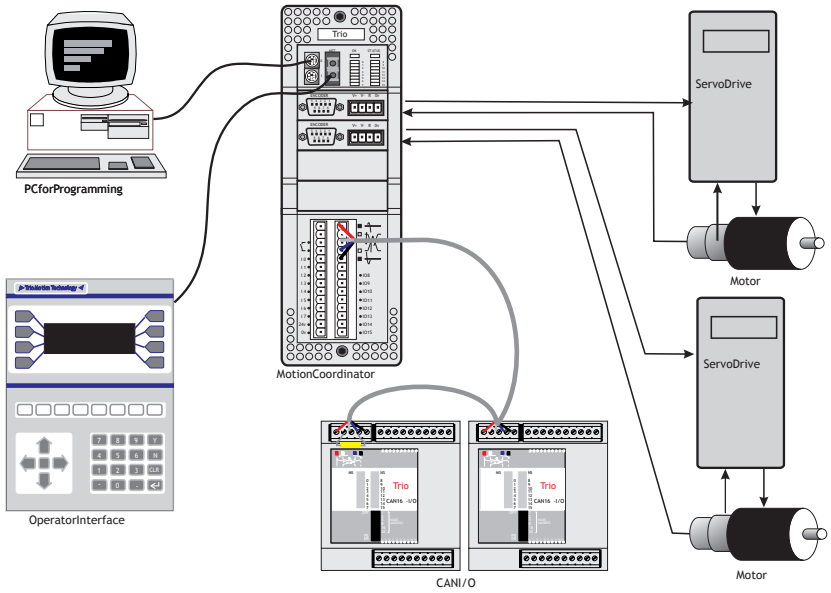
REFERENCE 1-1
ATYPE 1-3
AXISSTATUS / ERRORMASK 1-4
CONTROL 1-5
Communications Ports 1-5
Communications Errors 1-6
MTYPE 1-7
NETSTAT 1-7
Data Formats and Floating-Point Operations 1-8
Single-Precision Floating Point Format 1-8
Product Codes 1-9

CHAPTER

1

INTRODUCTION

Trio Motion Technology's range of *Motion* Coordinator products are 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 stand-alone programs or commands can be sent from an external computer.

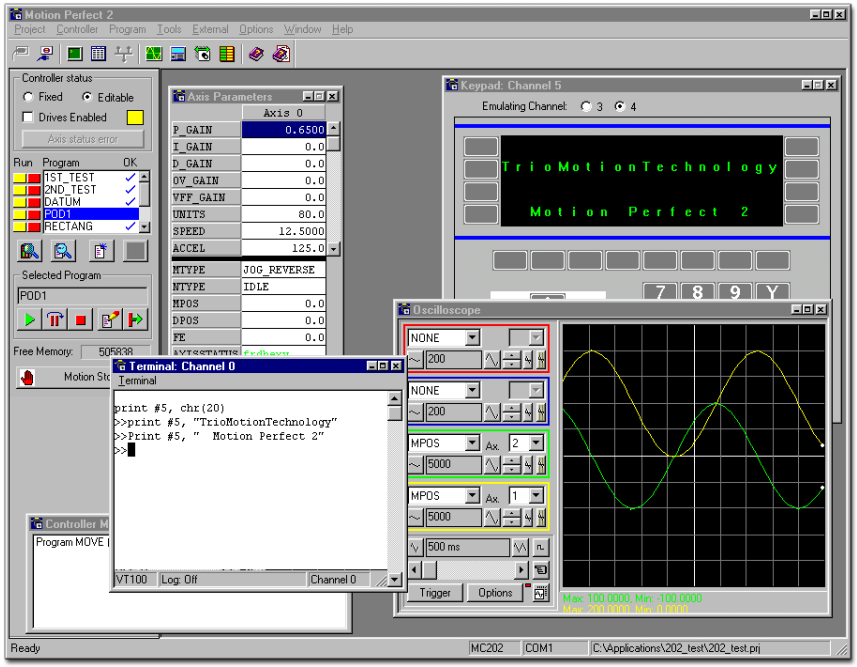


Typical System Configuration

The *Motion* Coordinator 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.

Setup and Programming

To program the *Motion* Coordinator a PC is connected via an RS-232 or USB serial link. The dedicated *Motion* Perfect program is normally used to provide a wide range of programming facilities, on an IBM PC or 100% compatible microcomputer running Windows.



***Motion* Perfect**

Once connected to the *Motion* Coordinator, the user has direct access to *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 unsynchronised multi axis moves as well as the control of the digital I/O.

The *Motion* Coordinator range currently consists of the MC202, MC206, Euro 205X, MC216 and MC224. These controllers feature multi-tasking BASIC. Multiple *Trio* BASIC programs can be constructed and run simultaneously to make programming complex applications much easier.

Products

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

Motion Coordinator Master Controllers

Product Code	Name	Description
P165	**MC202**	Compact, low cost DIN-rail mounting module features 1 1/2 servo or 2 stepper axes. 4 Opto-isolated Inputs and 4 Opto-isolated I/O channels are built in. Multi-tasking *Trio* BASIC. I/O expansion is via CAN bus.
P135	**MC206**	Low cost, high performance DIN-rail mounting controller for 1-6 axes. 8 Opto-isolated Inputs and 8 Opto-isolated Input/Output channels,1 Opto-isolated analogue input, USB and memory stick socket are built in. Multi-tasking *Trio* BASIC. I/O expansion is via CAN bus.
P150	**Euro 205**	For OEM applications, Trio offer a 3U Eurocard format controller featuring 4 onboard axes plus the option for a further axis via a standard Trio daughter board. 16 Opto-isolated Inputs and 8 Opto-isolated Output channels are built in. Multi-tasking *Trio* BASIC. I/O expansion is via CAN bus.
P140	**MC216**	Flexible high 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.
P170	**MC224**	Flexible high performance master controller for 1-24 axes. 8 Opto-isolated Inputs and 8 Opto-isolated Input/Output channels,2 Opto-isolated analogue inputs, USB and memory stick socket are built in. Multi-tasking *Trio* BASIC. I/O expansion is via CAN bus.

Daughter Boards



The Daughter Board concept is a one of the key features which give the *Motion* Coordinator system enormous flexibility in its configuration.

The Daughter Boards provide the interface to many types of Servo or Stepper Axes, plus a number of advanced communications options as well.

There are 13 types of daughter boards currently available:

Product Code:	Name	Description
P200	**Servo Encoder**	**+/- 10v Output, Differential Encoder Input plus Hardware Registration Input** The servo encoder daughter board provides the interface to a DC or Brushless servo motor fitted with an encoder or encoder emulation.
P210	**Servo Resolver**	**+/- 10v Output, Resolver Input plus Hardware Registration Input** The servo resolver 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.
P220	**Reference Encoder**	**Differential Encoder Input plus Hardware Registration Input** The encoder daughter board provides an encoder input without any servo feedback facility for measurement, registration and synchronization functions on conveyors, drums, flying shears, etc.
P230	**Stepper**	**Open-collector Step, Direction, Boost and Enable outputs** 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.

Product Code:	Name	Description
P240	Stepper Encoder	**Open-collector Step, Direction, Boost and Enable outputs plus Differential Encoder Input.** 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.
P242	Hardware Pswitch	**Differential Encoder Input plus Hardware Position Switch Outputs** 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 PWITCH command which is implemented in the system software and allows outputs to be switched ON and OFF over defined position sectors. The Hardware PSWITCH daughter board performs the position comparison in electronic hardware on the daughter board. This allows the pulses generated to be very accurately timed.
P260	Analogue Output	**+/- 10v Output with direct DAC control** 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.
P270	SSI Absolute Servo	**+/- 10v Output, Differential Encoder Input plus Hardware Registration Input** The servo resolver daughter board provides the interface to a DC or Brushless servo motor fitted with an absolute encoder fitted with a Synchronous Serial Interface (SSI).
P280	Differential Stepper	**Differential Line Driver outputs for Step, Direction, Boost and Enable, plus Hardware Registration Input** 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.

Product Code:	Name	Description
P290	CAN	**Digital Link to CANBus drives** The combination of the motherboard CAN channel and the ability to add up to 4 CAN daughter boards allows the MC216 to potentially control up to 5 CAN channels.
P291*	SERCOS	**Digital Link to SERCOS drives** The SERCOS daughter board provids digital control to appropriate servo drives via a Fibre Optic loop. Up to 8 axes can be connected to each P291, which allows the MC224 to control up to 24 axes.
P292	SLM (Drivelink)	**Digital Link to SLM drives** The SLM daughter board is aimed at providing digital control channels for servo drives utilising the SLM protocol.
P295	USB Interface	**Universal Serial bus interface for high-speed PC communications** The USB daughter board provides a very high speed interface between the motion coordinator and a host PC fitted with a USB port. Support for this high speed interface is included in Trio's MotionPerfect 2 application and software libraries are in development to allow developers to support the interface within their own programs.
P296*	Ethernet	**Ethernet LAN Interface** The Ethernet daughter board enables the Motion Coordinator to be connected to a local factory or office network. Supports TCP/IP using the 10Base-T physical layer.
P297	Profibus	**Profibus Fieldbus Interface** With the Profibus Daughter Board and appropriate software on the *Motion* Coordinator, it is possible to connect to external devices using the Profibus protocol.

***** (Available for use with MC206 and MC224 only)

Custom Daughter Boards

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

I/O Expansion options

Prod. Code	Name	Description
P315	**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.
P325	**CAN Analog Inputs**	DIN Rail mounted +/- 10v Analog Input module provides 8 opto-isolated channels.
P300	**Axis Expander**	Expansion module provides housing for up to 4 additional axis daughter boards. Up to 3 connect to the MC216



P315 - CAN-16 I/O



P325 - CAN-8 Analog Inputs



MC216 Shown with P300 Axis Expander

Operator Interfaces

Name	Name	Description
P502	Mini-Membrane Keypad	Compact operator keypad/display
P503	Membrane Keypad	High performance general purpose operator key-pad/display



System Building

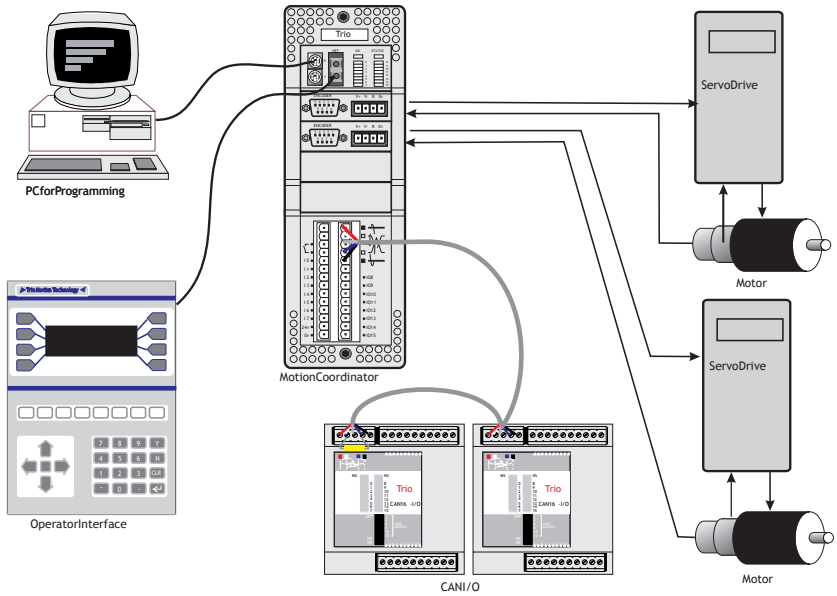
The modules and boards may be mixed within the system rules:

1) Every system must start with one *Motion* Coordinator master unit as this contains the processor and logic power supply for the system.

2) The MC216 master unit can house up to 4 daughter boards. The Euro 205 and MC206 will accept a single daughter board. These can be of any type.

3) The MC216 can have up to 3 axis expander modules added to house up to 16 daughter boards. 4 being housed in the Master and 4 in each of the axis expanders.

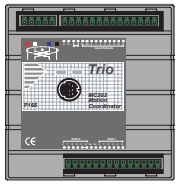
4) Up to 16 CAN-16 I/O and 4 CAN Analog Input modules can be connected to any *Motion* Coordinator.



Typical System Configuration

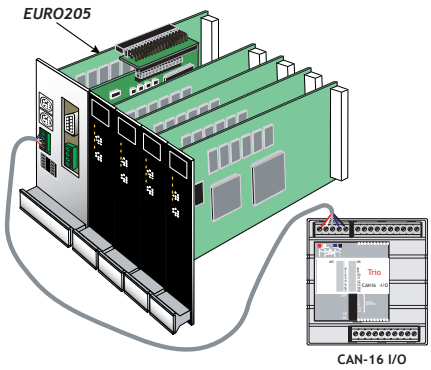
System Examples

Example 1 Simplest Possible System - Single MC202



- 1½ Axis Servo (Servo + reference encoder) or 2 Axis Stepper
- 8 Channels of 24v I/O on board
  - 4 Inputs
  - 4 I/O

Example 2 Euro205, 5 Axis Rackmounted System



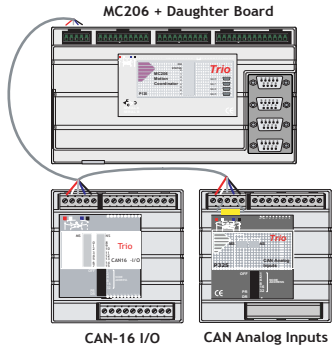
The Euro 205 controller provides a compact rack-mounted controller ideal for volume OEM applications.

The four internal axes are configured as stepper axes and connected via the backplane to third party stepper drives.

An optional P200 Servo Daughter Board provides a fifth axis connected to a servo drive.

- 5 servo axes
- 32 Channels of 24v I/O
  - 8 in + 8 I/O on MC204, plus
  - 16 I/O on CAN-16 I/O expander

Example 3 MC206 - 5½ Axis Servo System

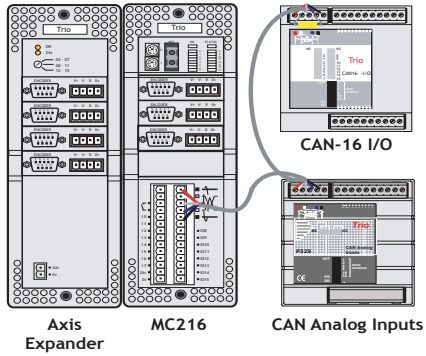


Utilising the four internal axis as servo axes, an optional servo daughter board and the reference encoder input to provide a total of 5 servo axes plus the reference (master) encoder.

The MC206 has 16 channels of 24v I/O as standard, this is expanded up to 32 channels with the addition of a CAN-16 I/O module. 8 Analog Inputs (+/- 10v) are provided by the CAN Analog Inputs module.

- 5 servo / stepper axes
- 32 Channels of 24v I/O
  - 8 in + 8 I/O on MC206, plus
  - 16 I/O on CAN-16 I/O expander
- 8 Analog Input Channels

Example 4 MC216 - 7 Axis System



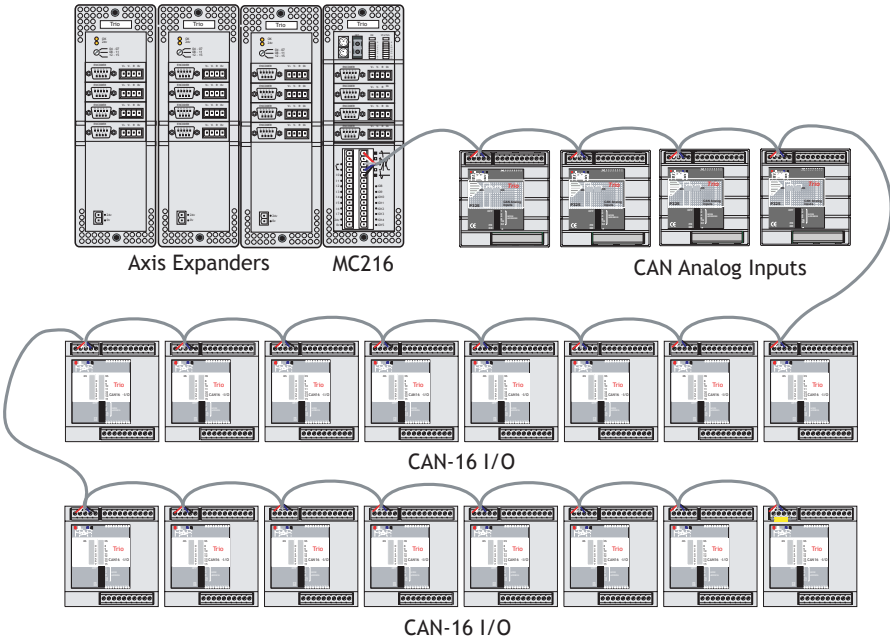
The system comprises an MC216 Master with and additional three axes in the optional Axis Expander module. The axes can be any combination of servo and stepper drives determined by the daughter boards used.

The MC216 has 16 channels of 24v Digital I/O as standard, this is expanded up to 32 with the addition of a CAN-16 I/O module. 8 Analog Inputs (+/- 10v) are provided by the CAN Analog Inputs module.

- 7 axes
  - 4 on MC216 Master, plus
  - 3 on Axis Expander
- 32 Channels of 24v I/O
  - 8 in + 8 I/O on MC204, plus - 16 I/O on CAN-16 I/O expander
- 8 Analog Input Channels

Example 5 MC216 - 16 axis system with maximum I/O expansion

The example system illustrated below shows an MC216 controller with the maximum possible I/O expansion. Applications for a system of this complexity might include collating or packaging machinery where there are multiple operations performed on a product and many I/O are used for sensors etc.



- 16 Axes (Any type)
- 272 Channels of 24v I/O
  - 8 in + 8 I/O on MC216, plus
  - 256 I/O (16 * 16) on the 16 CAN-16 I/O modules
- 32 Analog Input Channels (8 per module)

Features and Typical Applications

Trio BASIC contains accurate motion control functions for the generation of complex movements of various types, including:

- Linear interpolation of up to 24 axes
- 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 (connected via USB or RS-232 serial port)
- Membrane Keypad with Vacuum Fluorescent Display
- Dedicated Operator Panel using the'Modbus' serial protocol.
- 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:

- Cut to length
- Flying shears
- Glue laying
- Web control
- Tension control
- Pick & Place
- Coil winding
- Laser guidance
- Electronic assembly
- Printing
- Collating
- Packaging
- Automotive welding
- Spark erosion
- Drilling
- Milling
- YOUR application

The Trio Motion Technology Website

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



Website Features

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

WWW.TRIOMOTION.COM

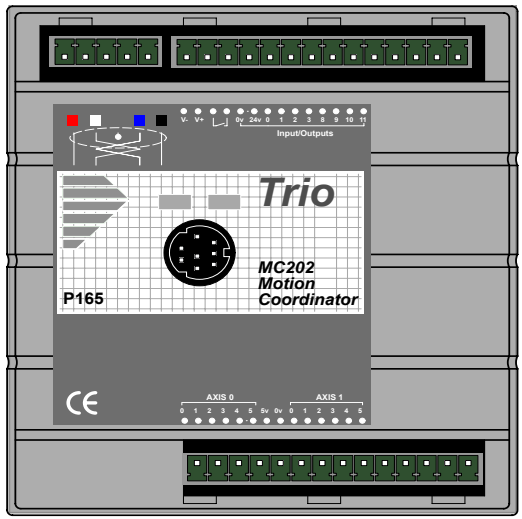
CHAPTER

2

HARDWARE OVERVIEW

Motion Coordinator MC202

Overview



The MC202 is a miniature stepper/servo positioner with the built-in ability to control one servo/stepper axis with additional synchronisation encoder, or two stepper axes.

The MC202 is designed to provide a compact, low cost, easy to use unit for OEM machine builders.

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

Programming

The Multi-tasking ability of the MC202 allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware. On the MC202 up to 3 Trio BASIC programs can be run simultaneously.

I/O Capability

The MC202 has 4 built in 24v inputs and 4 built-in bi-directional input/output 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. The MC202 can have up 256 external Input/Output channels and up to 32 analog input channels connected using DIN rail mounted I/O modules. These units connect to the built-in CAN channel of the MC202

Communications

The MC202 has a built-in RS-232 port. A further serial channel is available at TTL levels. External adapters are available to allow this serial port to be linked to a full-duplex RS485 channel or a Trio membrane keypad on a fibre-optic link.

If the built-in CAN connection is not being use to for I/O modules it may be used for CAN communications.

Connections to the MC202

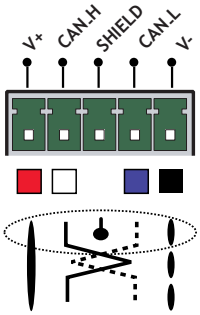
Wiring All wires and cables should be of suitable size and type for the signals carried and the operating environment.

Suitable wires would include multi-core screened cable for encoder feedback and the serial links. The analog output from the MC202 (if used) should be connected to the servo drive via a screened twisted pair. Drive enable outputs and 24v inputs do not have a large current requirement so the choice of wire is not critical.

Connectors

Top 5-Way Connector

This is a 5 way 3.81mm pitch connector. The connector is used both to provide the 24 volt power to the MC202 and provide connections for I/O expansion via Trio's P315 and P325 CAN I/O expanders. 24 volts must be provided as this powers the unit.



This 24 volt input is internally isolated from the I/O 24 volts and the +/-10v voltage output. The MC202 has internal power supply filters for the 24v power supply. This supply should be isolated and independent from the I/O 24v.

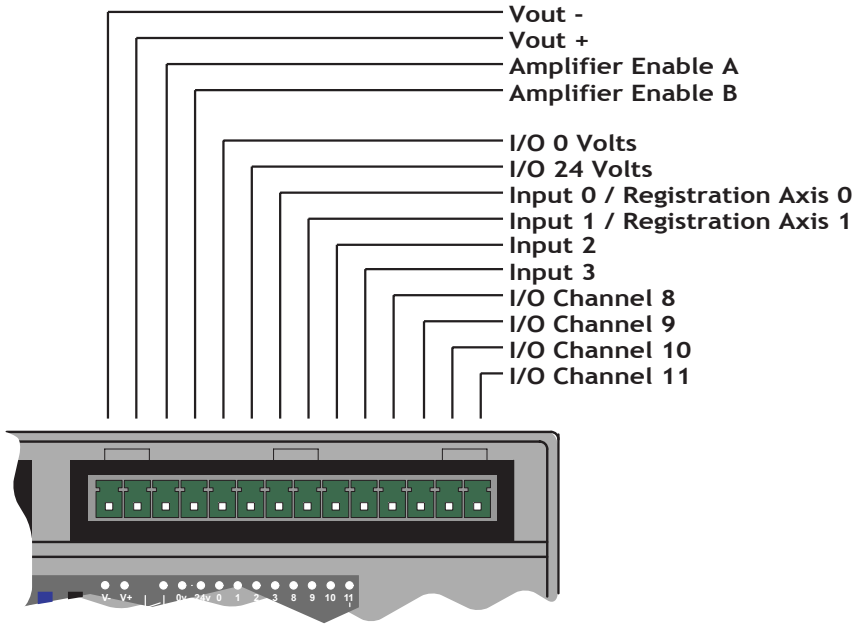
Note: ***The CAN connections are optional, although the CAN Screen must be connected to Earth in all cases.***

The V+(24v) and V- MUST be connected as this powers the MC202.

Power supply: 24V dc, Class 2 transformer or power source.

Top 14 Way Connector:

This is a 14 way 3.81 pitch connector. The connector provides for the +/-10volt analog output, the enable relay contacts, and the I/O connections.



Analog Output

This feature when required is used to drive a servo drive or inverter connected to axis 0. The +/-10 volt analog output is isolated from the power input and the I/O modules of the MC202 and is powered via an internal DC-DC converter. The pair of connections should be connected by a screened cable to the drive input.

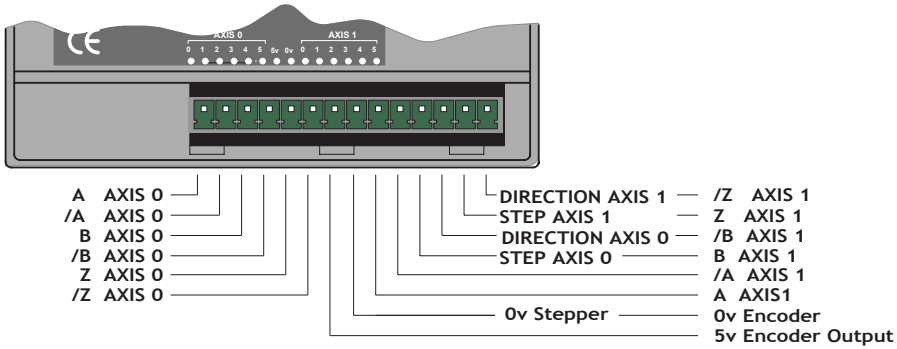
The drive enable pins are used to interlock the MC202 with a servo OR stepper drive and should be used in all cases. The connections are internally connected to a pair of normally open relay contacts which close when the drive is enabled.

I/O Power Inputs

The I/O 0 Volts and I/O 24 Volts are used to power the 24 volt inputs and outputs. The I/O connections are isolated from the module power inputs on the 5-way CAN connector. The I/O 0 Volts connection must be made if any inputs or outputs are used. The I/O 24 Volts is only required to power outputs and may be omitted if none are used. The I/O channels 8 to 11 are bi-directional and can be used

either as an input or an output. They are numbered from 8 to 11 for greater compatibility with other Trio *Motion Coordinator*s. The inputs channels 0 to 3 are not bi-directional. Inputs 0 and input 1 can be used as registration inputs for axes 0 and 1 for use with the **REGIST** command.

Bottom 14 Way Connector:



This is a 14 way 3.81 pitch connector. The connector provides for 2 axes of 5v differential encoder inputs, 2 axes of step and direction encoder inputs and a 5 volt output for powering external encoder only.

Axis 0 Encoder Inputs:

6 pins are used to provide dedicated encoder inputs for Axis 0. If axis 0 is used as a stepper axis these connections are not required. The inputs are 5 volt differential encoder inputs. The inputs must be connected to the encoder via a screened cable and the screen connected to the panel ground.

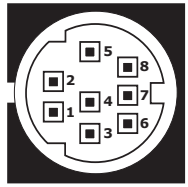
Encoder Power Supply:

2 pins provide a low power output at 5v (150mA maximum). This supply is provided for driving one or two encoders (if current consumption permits). The power supplies should be included within the encoder screened cable. Do not use the 0V as a screen ground point.

Stepper Outputs / Axis 1 Encoder Inputs:

6 pins are used to provide encoder inputs for Axis 1. Alternatively 4 of the pins can be used as step and direction open-collector outputs for axes 0 and 1. The inputs are 5 volt differential encoder inputs. Whatever the use of the pins they must be connected to the stepper drive or encoder using screened cable and the screen connected to the panel ground.

Serial Port Connections



The MC202 features a standard RS-232 serial port for programming (port 0), and an additional TTL-level output (port 1) for external communications adaptors such as the P349 RS-485 Adaptor, or the P435 Serial to Fibre-Optic Adaptor.

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

MC202 Serial Port Connections

Pin	Function	Notes
1	Internal 5v	Do not connect
2	Internal 0v	Do not connect
3	RS232 Transmit	Serial Port 0
4	Ground	
5	RS232 Receive	Serial Port 0
6	External Control Output	Do not connect
7	Externally buffered output	Do not connect
8	Externally buffered input	Do not connect
Pins 1, 2, 6, 7, and 8 are required to connect to an external fibre-optic adapter.		

Serial Cables

Trio recommend the use of their pre-made serial cables (product code P350). 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 9pin serial connector

MC202 - Feature Summary

Size	94 mm x 101 mm x 48mm Overall
Weight	200 g
Operating Temperature	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	1 RS232 Channel: 9600 baud. TTL serial Channel 9600 baud, 1 CAN channel built on to motherboard
Position Resolution	32 bit position count
Interpolation modes	Linear 1-3 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking Trio BASIC system, maximum 3 simultaneous Trio BASIC programs.
Speed Resolution	32 bit. Speed may be changed at any time. Moves may be merged.
Servo Cycle	1ms
Memory	128 Kbytes flash user program memory.
Power Input	24V dc, Class 2 transformer or power source. 18 ... 29V dc at 150mA.
Drive Enable Output	Normally open relay contact rated 24Vdc @ 0.5A..
+/-10volt DAC Output	12 bit resolution, single channel on axis 0 only
Encoder Inputs	2 axes, Differential 5v inputs, 6Mhz maximum edge rate *MC202 can generate 150mA at 5v for encoder power supplies)
Stepper Outputs	2 axes. Step & Direction: Open collector, 24V @ 50mA max. 500khz maximum step rate *
Digital Inputs	4 Opto-isolated 24v inputs, 2 may be used for high speed registration
Digital I/O	4 Opto-isolated bi-directional 24v 250 mA Outputs/ Inputs

Motion Coordinator Euro205(X)



Overview The *Motion Coordinator* Euro205 and Euro205X are Eurocard stepper/servo positioners with the built-in ability to control up to 4 servo or stepper motors in any combination. In addition a single Trio Daughter Board may be fitted to allow the control of a fifth axis or communications channel. The Euro205(X) is designed to provide a powerful yet cost-effective control solution for OEM machine builders who are prepared to mount the unit and provide the power supplies required. 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 Euro205 allows up to 5 Trio BASIC programs (7 for the Euro205X) to be run simultaneously on the controller using pre-emptive multi-tasking.

Programming The Multi-tasking ability of the Euro205(X) allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware.

I/O Capability The Euro205(X) has 16 built in 24v inputs and 8 built-in output 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. 8 status LEDs are available which can be set to display the status of banks of inputs or outputs. The Euro205(X) can have up 256 external Input/Output channels and up to 32 analog input channels connected using DIN rail mounted I/O modules. These units connect to the built-in CAN channel of the Euro205

Communications The Euro205 has two built in RS-232 ports and 2 further serial channels available at TTL levels. External adapters are available to allow one of these serial ports to be linked to a full-duplex RS-485 channel and the second to be used to link to Trio Fibre Optic Network devices. The Euro205X has the RS485 port built-in and so does not need the adapter.

Note Serial port pin-out is changed between the Euro205 and Euro205X.

One of the RS-232 ports or the RS485 port may be configured to run the MODBUS protocol for PLC or HMI interfacing.

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

USB and Profibus daughter boards may be fitted to provide additional communications options and the Euro205X will accept the Ethernet daughter board.

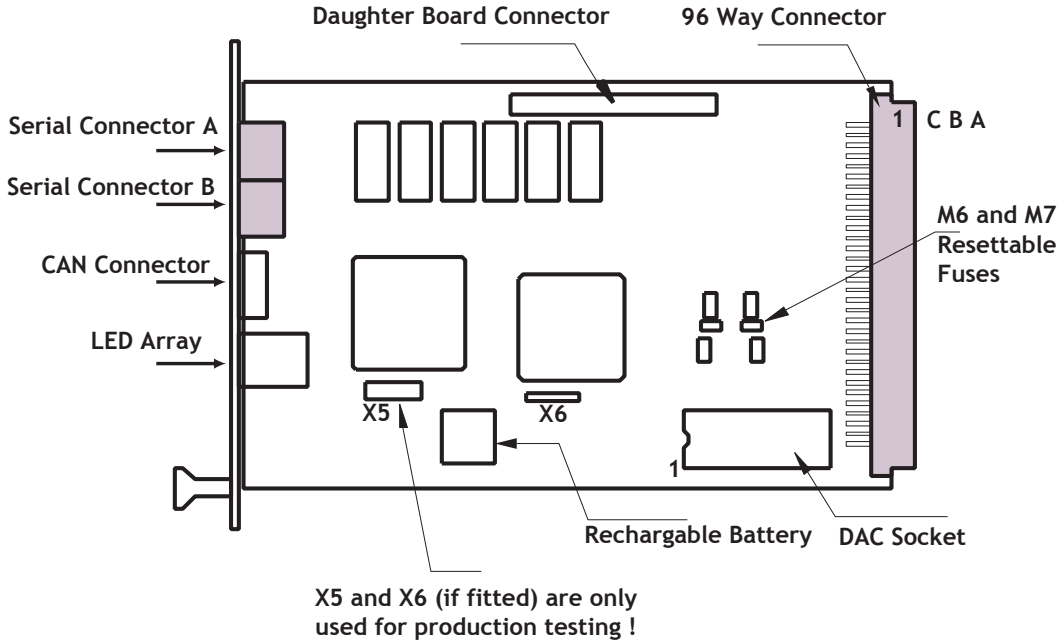
Axis Configuration

The Euro205X may be purchased as a stepper base card (P151), or a servo base card (P156). Additional servo or stepper axes may be specified up to the 4 internal axis limit. In addition axes may be added in the field by the entry of “feature enable codes” into the controller. The codes can be purchased already installed into a new controller or may be ordered for controllers purchased earlier provided that the serial number is given to Trio. Note that both P151 and P156 have a serial number label starting P151. P156's have the Feature Enable Code F4 loaded for axis 0 servo.

The gate array at the heart of the Euro205X design has facilities for 4 servo and 4 stepper axes built into every chip. The “feature enable codes” allow users to purchase only those facilities required for their configuration. Once entered onto the controller, the feature enable codes are stored in permanent flash memory. The feature enable codes are unique for each Euro205X.

The Euro205X features a total of 8 axes in software. Any axes not having a hardware interface can be used as a “virtual” axis.

Connections to the Euro 205(X)

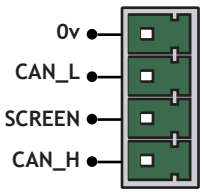


5 volt Power Supply

The minimum connections to the Euro205 are just the 0v and 5v pins. The Euro205(X) is protected against reverse polarity on these pins. Application of more than 5.5 volts will permanently damage the *Motion Coordinator* beyond economic repair. All the 0v and 5v pins are internally connected together. The 0v pins are, in addition, internally connected to the AGND pins. The Euro205(X) has a current consumption of approximately 500mA on the 5v supply. The supply should be filtered and regulated within 5%.

Built-in CAN Connector

The Euro205(X) features a built-in CAN channel. This is primarily intended for Input/Output expansion via Trio's P315 and P325 modules. It may be used for other purposes when I/O expansion is not required.



Euro 205(X) Backplane Connector

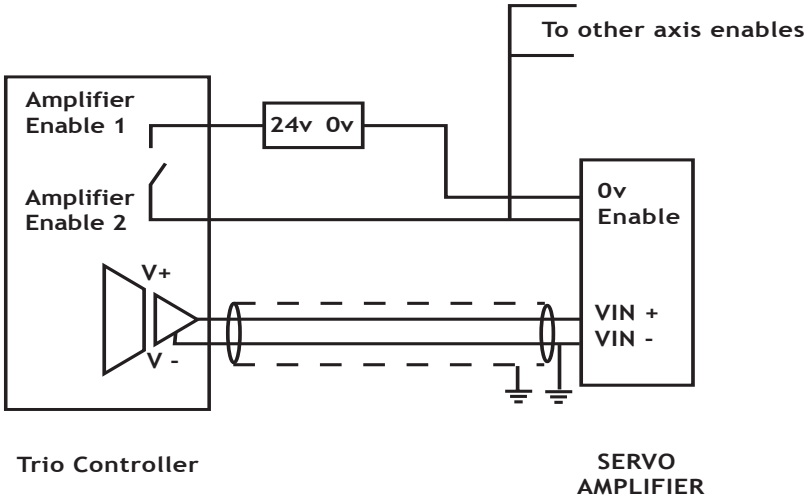
Most connections to the Euro205 and Euro205X are made via the 96 Way DIN41612 backplane Connector. Signals marked * are Euro205X only.

	C	B	A
1	5v	5v	5v
2	5v	5v	5v
3	0v	0v	0v
4	IO GND	OP13	OP10
5	OP9	OP12	OP15
6	OP8	OP11	OP14
7	IO 24v	IN0 / R0*	IN1 / R1*
8	IN2 / R2*	IN3 / R3*	IN4
9	IN5	IN6	IN7
10	IN8	IN9	IN10
11	IN11	IN12	IN13
12	IN14	0v	IN15
13	0v	DIR2	0v
14	STEP1	STEP2	DIR3
15	DIR0	DIR1	STEP3
16	STEP0	FAULT	RESET
17	ENABLE 1	ENABLE (OC)	AIN(0)*
18	BOOST1	BOOST0	ENABLE 2
19	BOOST3	BOOST2	Z3- / BOOST3-*
20	A3- / STEP3-*	B3- / DIR3-*	Z3+ / BOOST3+*
21	A3+ / STEP3+*	B3+ / DIR3+*	Z2- / BOOST2-*
22	A2- / STEP2-*	B2- / DIR2-*	Z2+ / BOOST2+*
23	A2+ / STEP2+*	B2+ / DIR2+*	Z1- / BOOST1-*
24	A1- / STEP1-*	B1- / DIR1-*	Z1+ / BOOST1+*
25	A1+ / STEP1+*	B1+ / DIR1+*	Z0- / BOOST0-*
26	A0- / STEP0-*	B0- / DIR-*	Z0+ / BOOST0+*
27	A0+ / STEP0+*	B0+ / DIR+*	VOUT0
28	VOUT3	VOUT2	VOUT1
29	+12v	+12v	+12v
30	AGND	AGND	AGND
31	-12v	-12v	-12v
32	Earth	Earth	Earth

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 on the Euro205 and a solid-state relay on the Euro205X with normally open contacts. The enable relay 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 hold-up circuit or chain that must be intact before a 3-phase power input is made live.

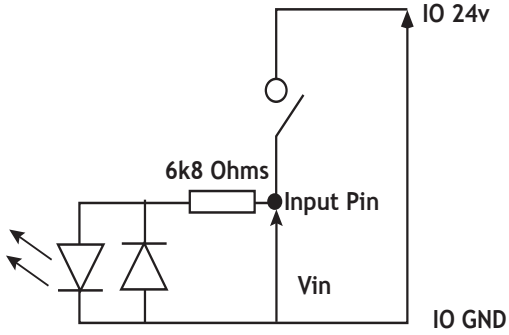


Amplifier Enable Output

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

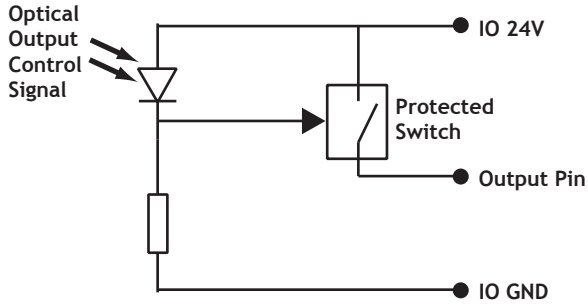
24v Input Channels

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



24v Output Channels

8 output channels are provided. These channels are labelled 8..15 for compatibility with other *Motion Coordinator*s, but are NOT bi-directional as on some *Motion Coordinator*s. Each channel has a protected 24v sourcing output. The output circuit has electronic over-current protection and thermal protection which shuts the output down when the current exceeds 250mA. 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. Up to 256 further Outputs may be added by the addition of CAN-16I/O modules (P315).



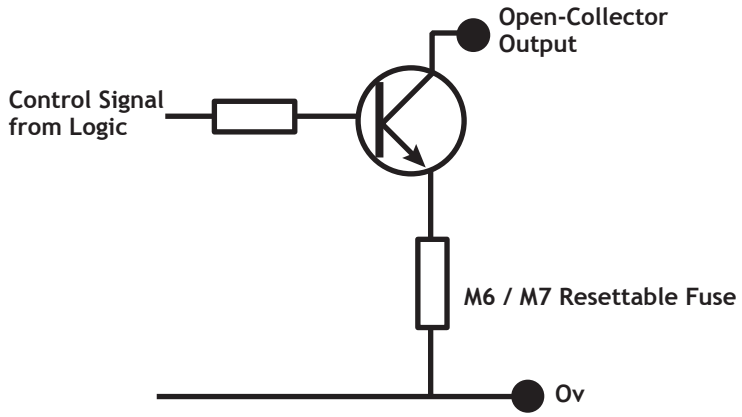
Open Collector Stepper Driver Outputs

The STEP, DIR, BOOST and ENABLE signals use open-collector outputs. These outputs are NOT opto-isolated from the processor logic. The open-collector outputs may be pulled up to any voltage in the range 5v..24v as required but a current limiting resistor MUST be provided externally to the Euro205(X) to limit the current in the output channel. Normally this current limiting resistor is built-in to the stepper amplifier circuit.

On the Euro205 the BOOST output shares a pin with the registration input for that axis. The BOOST output is normally used in stepper motor systems, and the Registration input is used to capture encoder positions. There is not therefore normally a conflict. The BOOST output should not be turned ON on an axis where the registration input is used otherwise the BOOST output will be overloaded.

The BOOST outputs are separate from the registration function on the Euro205X.

The open-collector outputs are protected by 2 resettable fuses. These fuse links will go high impedance if a total of more than 200mA is passed through a group of open-collector outputs. M6 protects STEP0, STEP1, STEP2, STEP3, DIR0, DIR1 and DIR2. M7 protects DIR3, REG(BOOST)0, REG(BOOST)1, REG(BOOST)2, REG(BOOST)3 and ENABLE. If any outputs are overloaded it is necessary to remove the power from the circuit in order to reset the fuses.



Fault and Reset Inputs

Fault is a non-isolated 5 volt input for connection to stepper amplifier fault outputs. Signals connected to this input should NOT exceed 5volts.

The Reset input can be momentarily connected to 0v to reset the Euro205(X). Signals connected to this pin must not exceed 5 volts.

Registration Inputs

The registration inputs are 24 volt non-isolated inputs referenced to 0v. The Euro205(X) can be programmed to capture the position of an encoder axis in hardware when a transition occurs on the registration input.

Differential Encoder Inputs

The encoder inputs on the Euro205 and Euro205X are designed to be directly connected to 5 volt differential output encoders. On the Euro205X the encoder ports are bi-directional so that when axes are set to stepper, the encoder port for that axis becomes a Differential Stepper output.

Voltage Outputs

The Euro205(X) can generate 4 +/-10volt analog outputs which are primarily designed for controlling servo-amplifiers. Euro205 requires a DAC chip (Burr-Brown DAC4814AP) to be fitted in the socket provided, whereas the Euro205X has the DAC chip soldered to the PCB. Note that for servo operation the necessary feature enable codes must have been entered into the Euro205/Euro205X. To use the voltage outputs the +/-12volt supplies must be present.

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

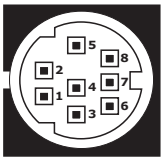
Switches used for the FORWARD/REVERSE/DATUM/FEEDHOLD inputs must be NORMALLY CLOSED type.

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.

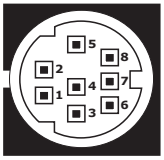
Euro 205 Serial Port Connections

Serial Connector A:



Pin	Function (Euro205)	Function (Euro205X)
1	Internal 5v - P348 supply	Internal 5v - P435 supply
2	Internal 0v - P348 supply	Internal 0v - P435 supply
3	RS232 Transmit (Port 0)	RS232 Transmit (Port 0)
4	0V	0V
5	RS232 Receive (Port 0)	RS232 Receive (Port 0)
6	Output to P348	Internal 5v
7	Input from P348	Output to P435
8	Control to P348	Input from P435
Euro205 only: The external RS485 adapter (P348) is addressed as port #2. Euro205X only: The external Fibre Optic adapter (P435) is on port #4		

Serial Connector B:



Pin	Function (Euro205)	Function (Euro205X)
1	Internal 5v - P435 supply	RS485 Data in A (Port 2)
2	Internal 0v - P435 supply	RS485 Data in B (Port 2)
3	RS232 Transmit (Port 1)	RS232 Transmit (Port 1)
4	0V	0V
5	RS232 Receive (Port 1)	RS232 Receive (Port 1)
6	+5V output	+5V output
7	Output to P435	RS485 Data out Z (Port 2)
8	Input from P435	RS485 Data out Y (Port 2)
Euro205 only: The external Fibre Optic adapter (P435) is on port #4		

Network Interconnection

The Euro 205(X) supports Trio's fibre-optic network with the optional P435 serial to fibre-optic adaptor connected to the appropriate serial connector.

The software for the network supports interconnection of up to 15 nodes in a token-ring network format. The nodes may consist of any combination of compatible master controllers and Trio Membrane Keypads.

Euro205 - Feature Summary

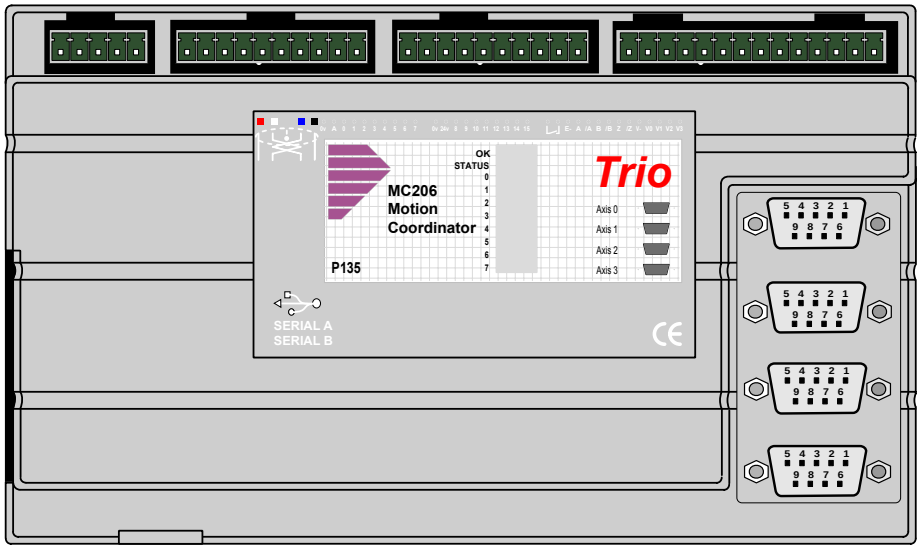
Size	170 mm x 129 mm Overall (160mm x 100 mm PCB) 25mm deep
Weight	170 g
Operating Temp.	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	2 RS232 Channels: 1200, 9600,19200,38400 baud.+ 1 RS485 Channel (using adapter)+ 1 Fibre Optic Channel (using adapter) 1 CAN channel built on to motherboard
Position Resolution	32 bit position count
Interpolation modes	linear 1-5 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking Trio BASIC system, maximum 5 user tasks.
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Servo Cycle	1ms default value
Memory	122 Kbytes battery backed + flash program memory.
Power Input	500mA at 5 V d.c. (+/-12v at 25mA required for DAC output)
Amplifier Enable Output	NO relay contact rated 24Vdc @ 0.5A..
Digital Inputs	16 Opto-isolated 24v inputs
Registration Inputs	4. One per axis shared with Boost outputs.
Encoder Inputs	4 differential 5v inputs, 6MHz maximum edge rate
Stepper Outputs	4 Open collector, step and direction. 5 to 24V. Maximum frequency 500kHz.
Digital Outputs	8 Opto-isolated 24v current sourcing 250 mA outputs

Euro205X - Feature Summary

Size	170 mm x 129 mm Overall (160mm x 100 mm PCB) 25mm deep
Weight	170 g
Operating Temp.	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	2 RS232 Channels: 9600,19200,38400 baud, 1 RS485 Channel + 1 Fibre Optic Channel (using adapter) 1 CAN channel built on to motherboard
Position Resolution	32 bit position count
Interpolation modes	linear 1-5 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking Trio BASIC system, maximum 7 user tasks.
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Servo Cycle	250us, 500us or 1ms (default)
Memory	256 Kbytes battery backed + flash program memory.
Power Input	500mA at 5 V d.c. (+/-12v at 25mA required for DAC output)
Amplifier Enable Output	NO solid-state relay. Maximim load 100mA, maximum voltage 29V.
Analog Inputs	2 x 12 bit 0 to 10V
Digital Inputs	16 Opto-isolated 24v inputs
Registration Inputs	4. One per axis shared with inputs 0 to 3.
Encoder Inputs	4 differential 5V inputs, 6MHz maximum edge rate
Stepper Outputs	4 differential (5V) or open collector (5 to 24V) step & direction outputs. Maximum frequency 500kHz (OC), 2MHz (Differential).
Digital Outputs	8 Opto-isolated 24v current sourcing 250 mA outputs

Motion Coordinator MC206

Overview The MC206 is based on Trio's high-performance 32-bit DSP technology and provides up to 4 axes of servo or stepper control, plus a master encoder axis. Trio uses advanced FPGA techniques to reduce the size and fit 4 axes of stepper and servo circuitry in a compact DIN-rail mounted package. The housing allows for a single daughter board to be mounted internally (with the optional P399 adaptor). This daughter board may provide additional axis control or communications functions.



User programs are written in Trio's established multi-tasking Trio BASIC language using the powerful *Motion* Perfect development software for the PC. Complex motion such as cams, gears, linked axes, and interpolation is made easy with the comprehensive Trio BASIC command set.

I/O Capability The MC206 has 16 opto-isolated digital I/O (8 in, 8 bi-directional). A high-speed hardware registration input is available on each axis for highly accurate control of print and packaging lines. A single 0..10v opto-isolated analog input is built-in.

The I/O count can be expanded using Trio CANbus digital and analog modules to provide a further 256 digital 24v I/O channels and 32 +/-10v analog inputs.

Communications

The MC206 offers wide communications capability with two RS-232 serial ports, an RS-485 port, a TTL serial port, a USB port and a CAN channel as standard. Adapters are available to convert the TTL port to Trio's fibre-optic network.

RS-232 port #1, or RS-485 port #2 may be configured to run the MODBUS protocol for PLC or HMI interfacing.

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

USB and Profibus daughter boards may be fitted to provide additional communications options.

Removable Storage

A memory stick used with the MC206 allows a simple means of transferring programs without a PC connection. Offering the OEM easy machine replication and servicing.
The memory stick is ordered separately from the MC206 if required (order code: P398).



Writing Programs to FlashStick

EPROM(1) Writes programs to FlashStick and clears EPROM request bit on FlashStick

EPROM(2) Writes programs to FlashStick and sets EPROM request bit on FlashStick

Checking Programs on FlashStick:

DIR F Lists a directory of FlashStick and prints EPROM request status

Reading Programs from FlashStick:

Programs will be copied on power up or **EX** from the FlashStick into the Motion Coordinator if a FlashStick is inserted and the controller is not locked.

If a FlashStick is inserted and the controller is unlocked the action taken depends on the **POWER_UP** system variable in the controller and the EPROM request bit on the FlashStick:

	EPROM Request OFF	EPROM Request ON
POWER_UP=0	Programs will be transferred from the FlashStick to the controller ram on power up.	Programs will be transferred from the FlashStick to the controller ram on power up. The programs will then be written to internal controller EPROM and the POWER_UP system variable set to 1.
POWER_UP=1	Programs will be transferred from the FlashStick to the controller ram on power up.	Programs will be transferred from the FlashStick to the controller ram on power up. The programs will then be written to internal controller EPROM.

Notes: The programs on the FlashStick have auto/manual run settings in the same way as programs in internal flash EPROM do.

In all cases if programs transferred from the FlashStick are set to autorun this occurs in a similar way

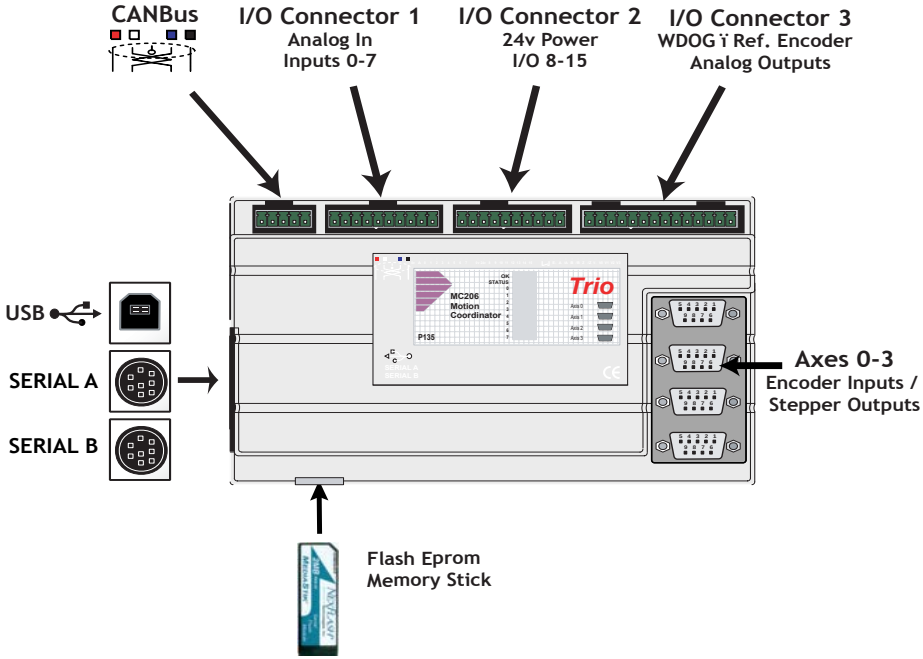
Methods of Use A machinery builder could use a FlashStick with programs loaded and the EPROM request bit set to install programs on to a series of machines.

If an application update is sent out on a FlashStick to a machinery customer the EPROM request bit could be left OFF [Using `EPROM(1)`]. In this way if the update does not do what is required the old program will be copied from internal EPROM when the FlashStick is removed.

Axis Configuration The MC206 initialises 8 axes in its software: up to four real axes of servo or stepper are built in, one daughter board axis may optionally be used, an encoder follower input axis is also built in to every MC206. Any axis of the 8 not used in hardware can be used as a virtual axis for camming, on-the-fly registration adjustments, and linking motion.

The MC206 can be a servo or stepper controller by using a “feature enable code”. All necessary DAC and stepper circuitry is installed and ready for use. The controller can be supplied with any combination of servo or stepper axes pre-enabled, and has been designed to allow the user to upgrade at a later date by entering additional feature enable codes. Up to 4 axes can be enabled on the MC206.

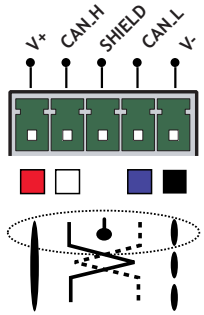
Connections to the MC206



5-Way Connector

This is a 5 way 3.81mm pitch connector. The connector is used both to provide the 24 volt power to the MC206 and provide connections for I/O expansion via Trio's P315 and P325 CAN I/O expanders. 24 volts must be provided as this powers the unit.

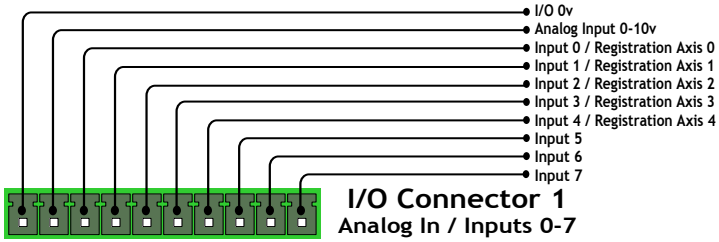
This 24 volt input is internally isolated from the I/O 24 volts and the +/-10v voltage outputs.



Note: ***The 24v (V+) and 0v (V-) MUST be connected as they power the MC206. The Shield MUST also be connected as it provides the EMC screen for the Motion Coordinator. The CAN connections are optional.***

Power supply: 24V dc, Class 2 transformer or power source.

I/O Connector 1



Analog Input

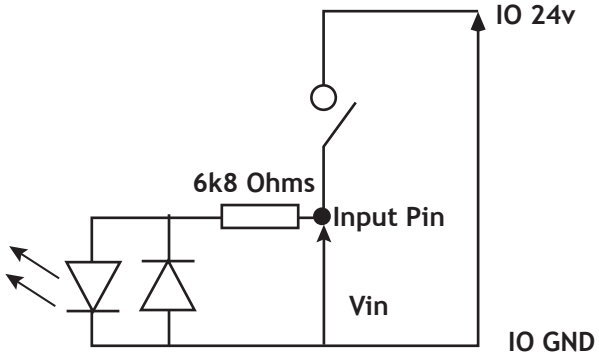
The MC206 provides a single 0-10v, 10-Bit analog input as standard. The analog input resolution is fixed and will return values 0..1023 to the system parameter **AIN0**. The input works “single ended” and is referenced to the IOGND pin alongside the AIN. Power must be applied to the 24v I/O power input in order for the analog input to function.

It is possible to add further analog inputs via the P325 CAN Analog modules. Each P325 module provides a further 8 single ended +/- 10v inputs with a 12-Bit resolution.

24v Input Channels

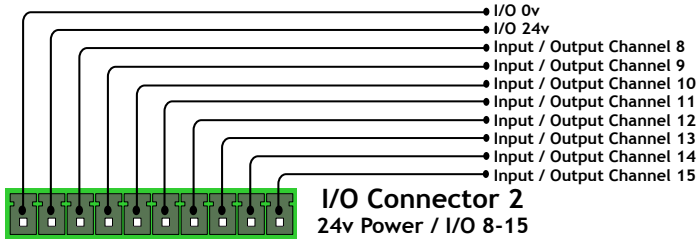
The MC206 has 8 dedicated 24v Input channels and 8 bi-directional 24v I/O channels built into the master unit. A further 256 inputs can be provided by the addition of CAN-16 I/O modules. The dedicated input channels are labelled channels 0..7.

Inputs 0 to 4 can be used as registration inputs for axes 0 to 4 for use with the **REGIST** command..



MC206 24v Input Channel

I/O Connector 2



I/O Power Inputs

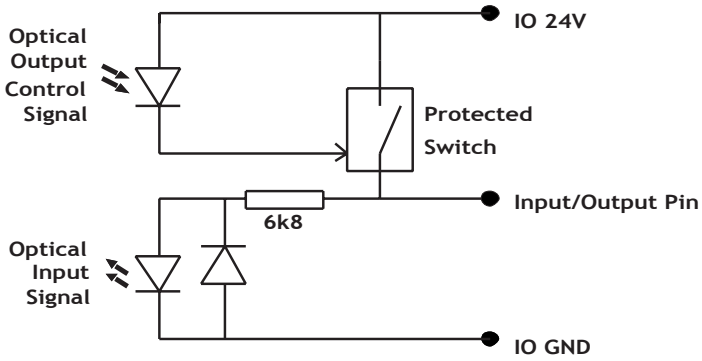
The I/O 0 Volts and I/O 24 Volts are used to power the 24 volt inputs and outputs.

The I/O connections are isolated from the module power inputs. The I/O channels are bi-directional and can be used either as an input or an output.

24v I/O Channels

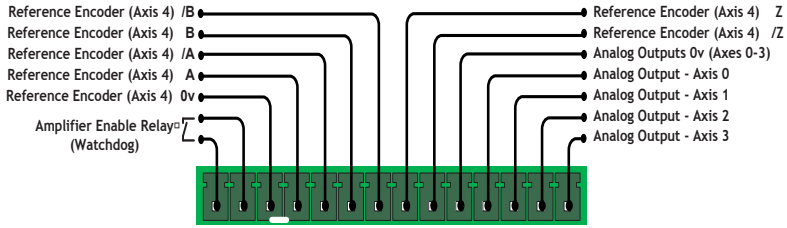
Input/output channels 8..15 are bi-directional. The inputs have a protected 24v sourcing output connected to the same pin. If the output is unused it may be used as an input in the program. The input circuitry is the same as on the dedicated inputs. The output circuit has electronic over-current protection and thermal protection which shuts the output down when the current exceeds 250mA.

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



MC206 24v I/O Channel

I/O Connector 3

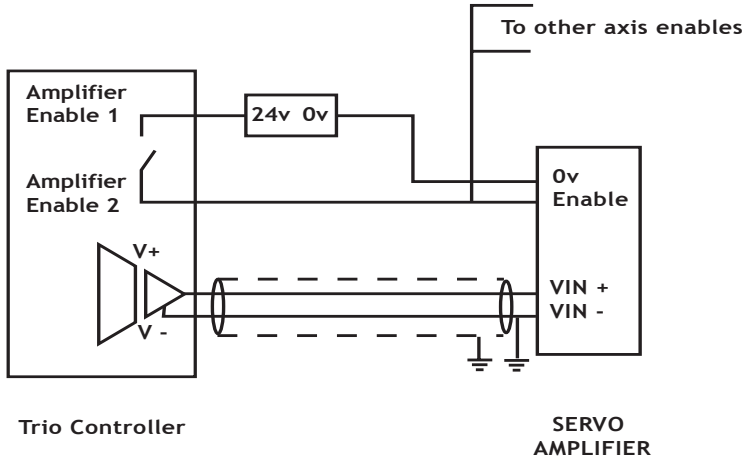


I/O Connector 3
WDOG / Ref. Enc. / Analog Outputs

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 hold-up circuit or chain that must be intact before a 3-phase power input is made live.



Amplifier Enable Output

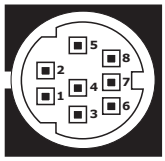
Note: ***ALL STEPPER AND SERVO AMPLIFIERS MUST BE INHIBITED WHEN THE AMPLIFIER ENABLE OUTPUT IS OPEN CIRCUIT***

Reference Encoder Input

A reference encoder - axis 4, provides an encoder input facility for measurement, registration and synchronization functions on conveyors, drums, flying shears, etc. The encoder port features high speed differential receiver inputs.

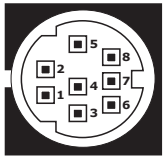
MC206 Serial Connections

Serial Connector A:



Pin	Function	Note
1	Internal 5v	
2	Internal 0v	
3	RS232 Transmit	Serial Port #0
4	RS232 GND	
5	RS232 Receive	
6	+5v output	
7	Externally buffered output (TTL)	For fibre-optic adaptor.
8	Externally buffered input (TTL)	
Note: Port 0 is the default programming port for connection to the PC running *Motion* Perfect.		

Serial Connector B:



Pin	Function		Note
1	RS485 Data In	A	Serial Port #2
2	$\overline{\text{RS485 Data In}}$	B	
3	RS232 Transmit		Serial Port #1
4	RS232 GND		
5	RS232 Receive		
6	Internal 5v		Serial Port #2
7	$\overline{\text{RS485 Data Out}}$	Z	
8	RS485 Data Out	Y	

Universal Serial Bus



The USB port provides a high-speed Universal Serial Bus link to a PC or other device supporting USB.

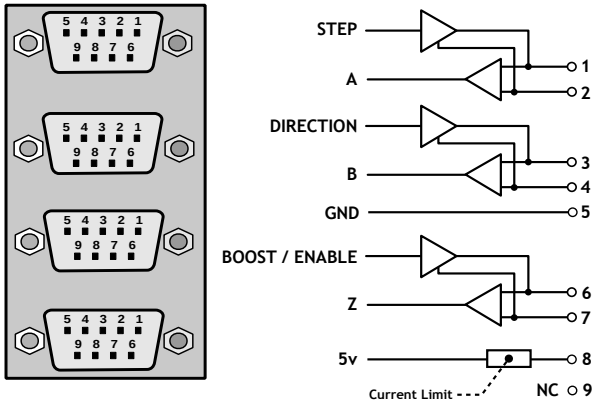
This port can be used for a high speed connection to *Motion* Perfect, or to a user program on the PC via Trio's ActiveX component.

MC206 - Stepper Outputs / Encoder Inputs

The MC206 controller is designed to support any combination of servo and stepper motors on the standard controller hardware. Each of the first four axes (0-3) can be enabled as either servo or stepper according to the users requirements.

The function of the 9-pin 'D' connectors will be dependant on the specific axis configuration which has been defined. If the axis is setup as a servo, the connector will provide the encoder input. If the axis is configured as a stepper, the connector provides differential outputs for step/direction and boost/enable signals.

The encoder port also provides a current-limited 5v output capable of powering most encoders. This simplifies wiring and eliminates external power supplies.



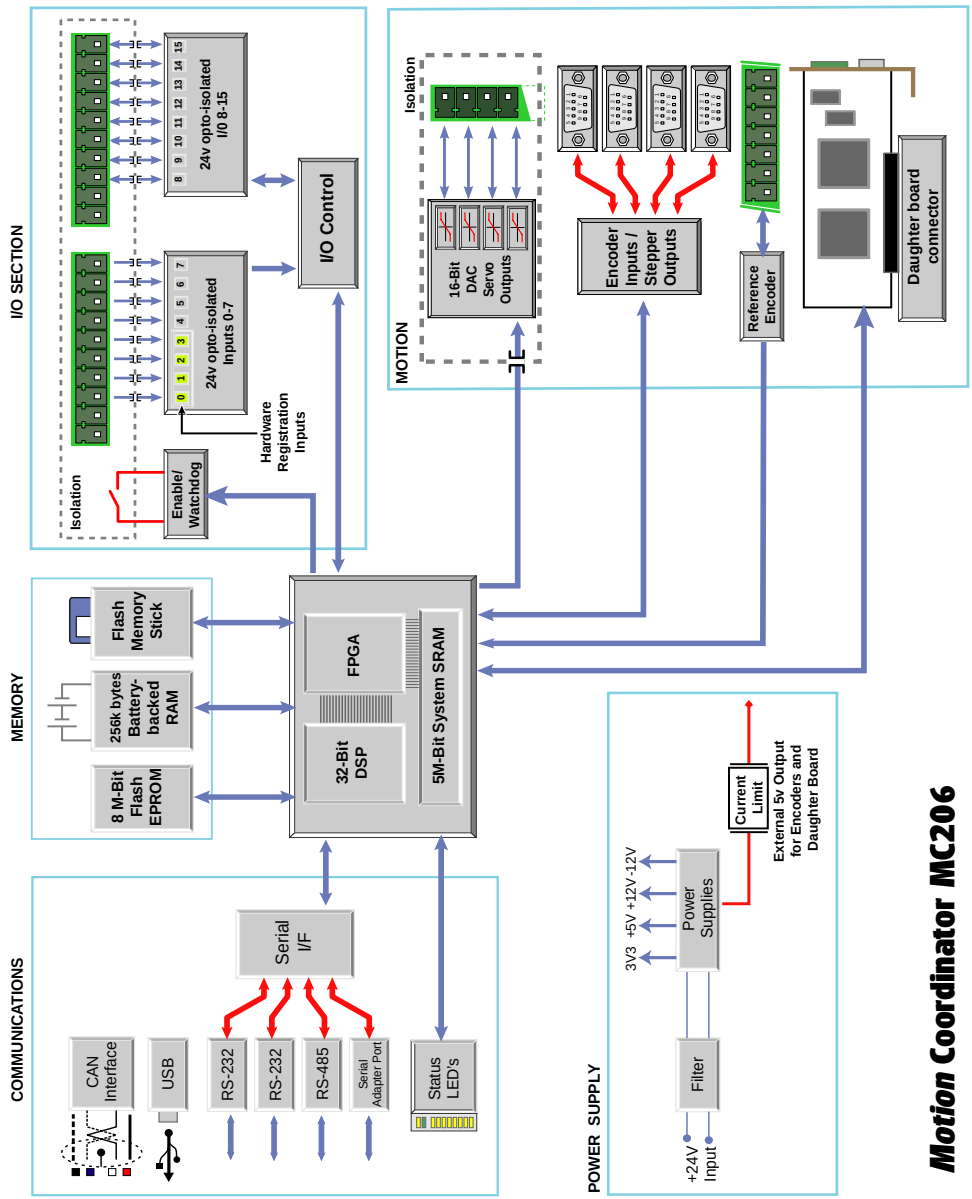
Once the axes have been enabled, using the **FEATURE_ENABLE** function, the controller will enable the appropriate hardware for each axis and the connector will function as below.

Pin	Servo Axis	Stepper Axis
1	Enc. A	Step +
2	Enc. /A	Step -
3	Enc. B	Direction +
4	Enc. /B	Direction -
5	GND	GND
6	Enc. Z	Boost +
7	Enc. /Z	Boost -
8	5v	5v
9	Not Connected	Not Connected
shell	Screen	Screen

MC206 - Feature Summary

Size	107mm(H) x 182mm(W) x 53mm(D)
Mounting	DIN- rail mounting box
Weight	600 g
Operating Temp.	0 - 45 degrees C
Control Inputs	Forward Limit, Reverse Limit, Datum Input, Feedhold Input.
Communication Ports	(2) RS-232 1200-38400 baud, (1) RS-485, (1) Serial adapter port, (1) USB 12Mbit, (1) CAN channel 1 Mbaud
Position Resolution	32 bit position count
Interpolation modes	linear 1-5 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Programming	Multi-tasking Trio BASIC system, maximum 8 tasks.
Speed Resolution	32 bits. Speed/Accel/Decel may be changed at any time. Moves may be merged for continuous motion.
Servo Cycle	250 µs minimum for all axes, 1 mS default
Memory	256k battery-backed user memory. Entire contents may be flashed to EPROM.
Memory Stick (Nexflash MediaStik)	Flash memory for program and project transfers to other MC206 controllers.
Power Input	24V dc, Class 2 transformer or power source. 18 ... 29V dc at 300mA.
Amplifier Enable Output	NO relay contact rated 24Vdc at 250mA.
Digital Inputs	8 Opto-isolated 24V inputs
Digital I/O	8 Opto-isolated 24V current sourcing 250 mA outputs/ inputs
Encoder Inputs	5 differential 5V inputs, 6MHz maximum edge rate
Encoder Power Output	5v at 150mA total for 4 encoders plus daughter board
Analog Input	Isolated 10bit 0-10Vdc
Analog Outputs	4 Isolated 16-bit +/- 10V
Stepper Outputs	4 Differential Step / Direction outputs 500kHz Max Rate
LED Indicators	OK, Status and 8 user programmable I/O status
Connections	Screw Terminals, 9-pin D-Shell, 8-pin mini-DIN and USB

MC206 - Schematic



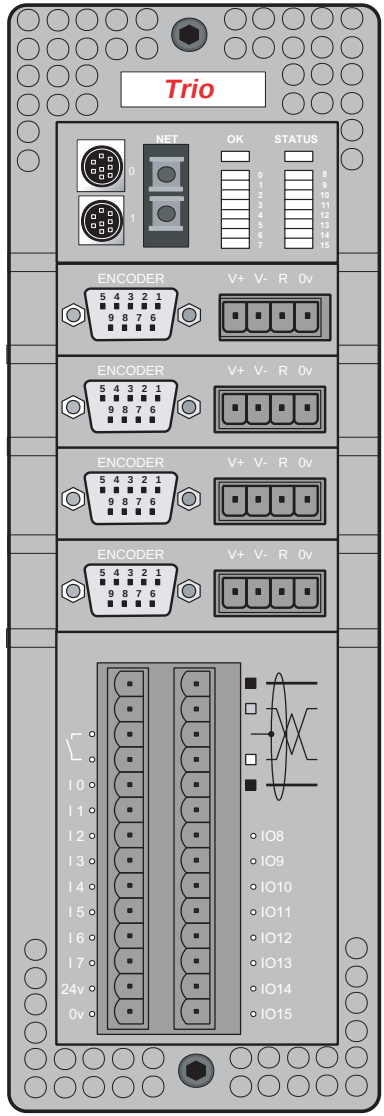
Motion Coordinator MC216

Overview

The *Motion Coordinator* MC216 is a 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 using a PC running the *Motion* Perfect application software, and then may be set to run “standalone” if an external computer is not required for the final system.

The Multi-tasking version of Trio BASIC for the MC216 allows up to 14 Trio BASIC programs to be run simultaneously on the controller using pre-emptive multi-tasking.



Programming 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.

I/O Capability The MC216 has 8 built in 24v inputs and 8 bi-directional I/O channels. These may be used for system interaction or may be defined to be used by the controller for end of travel limits, 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 512 external Input/Output channels connected using DIN rail mounted CAN 16-I/O modules. These units connect to the built-in CAN channel.

Communications 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.

One of the RS-232 ports may be configured to run the MODBUS protocol for PLC or HMI interfacing.

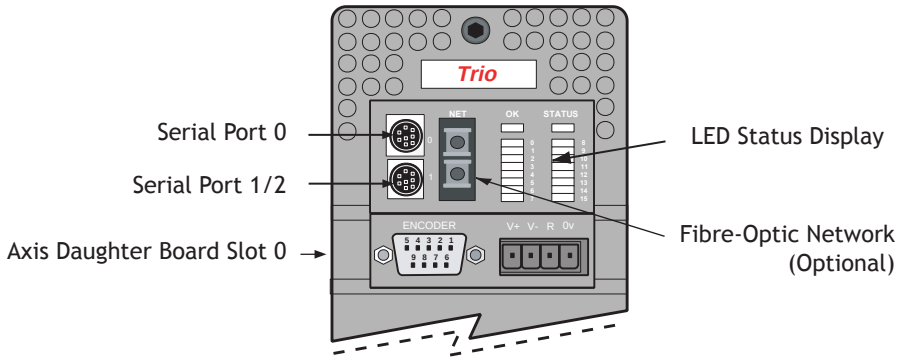
If the built-in CAN channel is not used for connecting I/O modules, it may optionally be used for CAN communications.

USB and Profibus daughter boards may be fitted to provide additional communications options.

Axis Positioning Functions The motion control generation software receives instructions to move an axis or axes from the Trio BASIC language which is running concurrently on the same processor. The motion generation software provides control during operation to ensure smooth, coordinated 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.

Connections to the MC216



MC216 upper front panel

MC216 Serial Connections

The MC216 features two serial connectors. Both ports feature a standard RS-232 serial interface. The lower connector (port 1) also has connections for RS-485 multi-drop (addresses as port 2).

Port 0 is the default connection between the *Motion Coordinator* and the host PC running *Motion* Perfect for programming. *Motion* Perfect may also be connected via USB with the optional (P295) USB Daughter Board

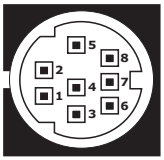
Serial Cables

Trio recommend the use of their pre-made serial cables (product code P350). If cables need to be made to connect to a PC serial port the following connections are required:



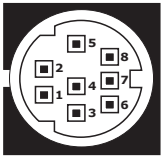
Motion Coordinator to 'AT' style PC with 9pin serial connector

Serial Connector A:



Serial Connector 0:		
Pin	Function	Notes
1	Internal 5v	
2	Internal 0v	
3	RS232 Transmit	Serial Port #0
4	RS232 Ground	
5	RS232 Receive	Serial Port #0
6	Externally buffered output	Serial Port #2
7	Externally buffered input	
8	Externally buffered control	
On MC216 RS485 can be provided via external adapter connected to pins 1,2,6-8		

Serial Connector B:



Serial Connector 1		
Pin	Function	Note
1	Internal 5v	
2	Internal 0v	
3	RS232 Transmit	Serial Port #1
4	RS232 Ground	
5	RS232 Receive	Serial Port #1
6	RS232 +5v Output	
7	Externally buffered output 2	Serial Port #3 / #4
8	Externally buffered input 2	
On the MC216 pins 1,2,7 and 8 can be used to interface a fibre optic adapter.		

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

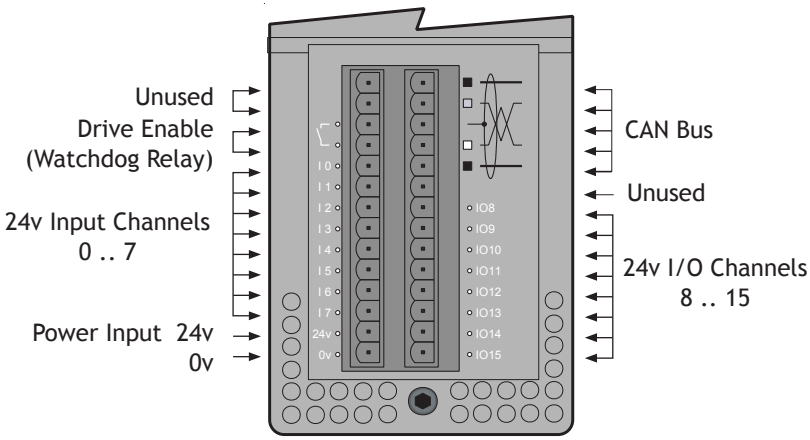
Network Interconnection



The optional Fibre-Optic Network connection is designed for communication between the master modules and membrane keypads on large machines, or small workgroups of machines.

The software for the network supports interconnection of up to 15 nodes in a token-ring network format. The nodes may consist of any combination of compatible master controllers and Trio Membrane Keypads.

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



MC216 lower front panel

24v Power Supply Input

The MC216 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.

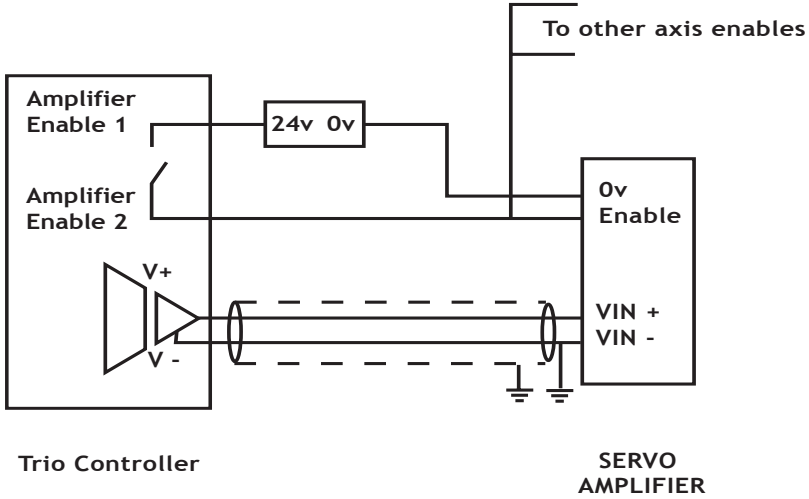
24V dc, Class 2 transformer or power source.

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 hold-up circuit or chain that must be intact before a 3-phase power input is made live.

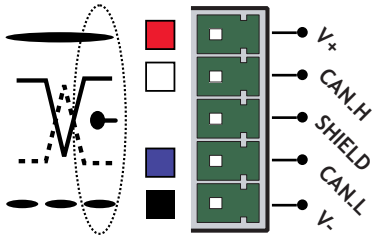
Note: ***ALL STEPPER AND SERVO AMPLIFIERS MUST BE INHIBITED WHEN THE AMPLIFIER ENABLE OUTPUT IS OPEN CIRCUIT***



Amplifier Enable Output

CAN Bus:

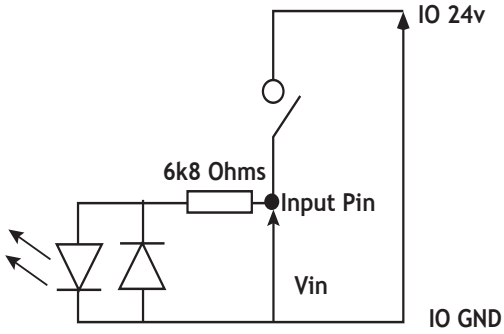
The MC216 features a built-in CAN channel. This is primarily intended for Input/Output expansion via Trio's P315 and P325 modules. It may be used for other purposes when I/O expansion is not required.



24v Input Channels

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

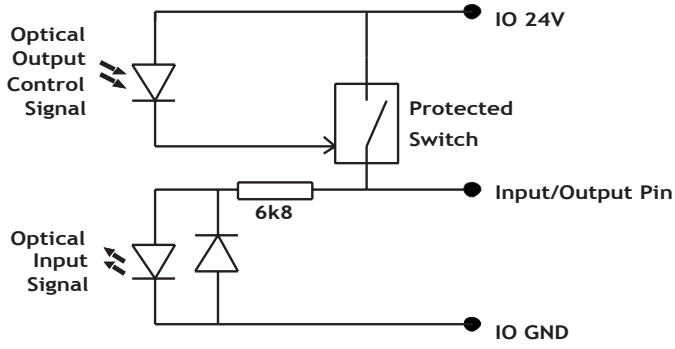
All of the 24v input channels have the same circuit although 8 on the master unit have 24v Output channels connected to the same pin. These bi-directional channels may be used for Input or Output to suit the application. If the channel is to be used as an Input the output should not be switched on in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



24v I/O 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.

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



MC216 - Feature Summary

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: up to 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	24V dc, transformer or power source. 18..29V dc at 500mA.
Encoder Power Output	5v at 600mA total for 4 encoders via daughter boards. Maximum 150mA per daughter board.
Amplifier Enable Output	NO relay contact rated 24Vdc @ 0.5A..
Digital Inputs	8 Opto-isolated 24V inputs.
Digital Outputs	8 Opto-isolated 24V current sourcing 250 mA outputs/inputs.
LED Indicators	OK, Status and 16 I/O status.
Connections	Screw Terminals and 2x 8-pin mini-DIN.

Motion Coordinator MC224

Overview

The *Motion Coordinator* MC224 is Trio's most powerful modular servo control positioner with the ability to control servo or stepper motors in any combination by the insertion of “Axis Daughter Boards” to suit the application. Up to 24 axes can be controlled by means of Digital Bus links (e.g. SERCOS) or 16 axes via traditional analog command / encoder feedback. 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 using a PC running the *Motion* Perfect application software, and then may be set to run “standalone” if an external computer is not required for the final system.

The Multi-tasking version of Trio BASIC for the MC224 allows up to 14 Trio BASIC programs to be run simultaneously on the controller using pre-emptive multi-tasking.

Programming The Multi-tasking ability of the MC224 allows parts of a complex application to be developed, tested and run independently, although the tasks can share data and motion control hardware.

I/O Capability The MC224 has 8 built in 24v inputs and 8 bi-directional I/O channels. These may be used for system interaction or may be defined to be used by the controller for end of travel limits, 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 MC224 can have up 512 external Input/Output channels connected using DIN rail mounted CAN 16-I/O modules. These units connect to the built-in CAN channel.

Communications The MC224 has two built in RS-232 ports and one built in duplex RS-485 channel for simple factory communication systems. The TRIO fibre optic network system can be added with an optional adapter cable.

One of the RS-232 ports or the RS485 port may be configured to run the MODBUS protocol for PLC or HMI interfacing.

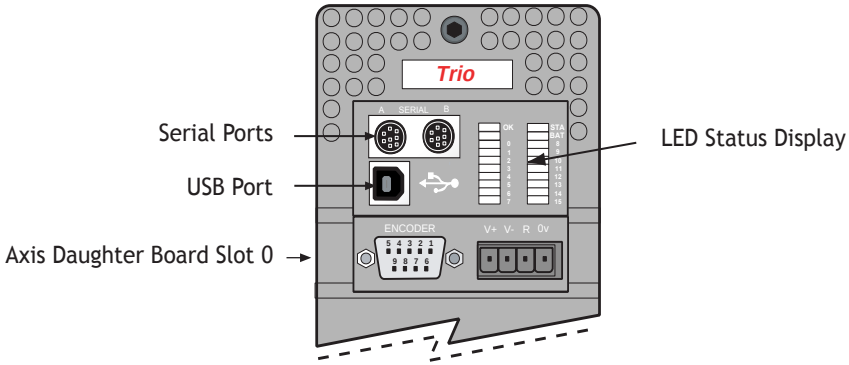
If the built-in CAN channel is not used for connecting I/O modules, it may optionally be used for CAN communications. e.g. DeviceNet, CANopen etc.

Ethernet, CANbus and Profibus daughter boards may be fitted to provide additional communications options.

Axis Positioning Functions The motion control generation software receives instructions to move an axis or axes from the Trio BASIC language which is running concurrently on the same processor. The motion generation software provides control during operation to ensure smooth, coordinated movements with the velocity profiled as specified by the controlling program. Linear interpolation may be performed 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.

Connections to the MC224



MC224 upper front panel

MC224 Serial Connections

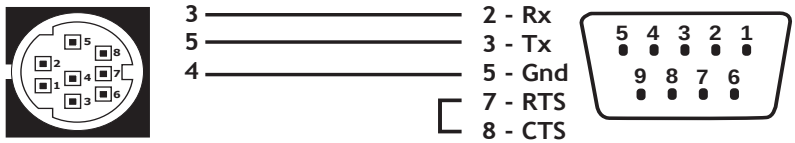
The MC224 features two serial connectors. Both connectors have a standard RS-232 serial interface. In addition, the left-hand connector (skt A) has connections for the Fibre-Optic Adapter and skt B contains the RS-485 multi-drop terminals (addressed as port 2).

Below the serial port connectors is the built in USB port.

Port 0 is the default connection between the *Motion Coordinator* and the host PC running *Motion* Perfect for programming. As an option, *Motion* Perfect may be connected via USB.

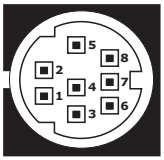
Serial Cables

Trio recommend the use of their pre-made serial cables (product code P350). If cables need to be made to connect to a PC serial port the following connections are required:



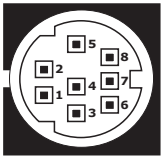
Motion Coordinator to 'AT' style PC with 9pin serial connector

Serial Connector A:



Serial Connector 0:		
Pin	Function	Notes
1	Internal 5v	
2	Internal 0v	
3	RS232 Transmit	Serial Port #0
4	RS232 Ground	
5	RS232 Receive	Serial Port #0
6	+5V Output	Serial Port #3 / #4
7	Externally buffered output (TTL)	
8	Externally buffered input (TTL)	
On the MC224 pins 1,2,7 and 8 can be used to interface a fibre optic adapter. Note: Port 0 is the default programming port for connection to the PC running *Motion Perfect*.		

Serial Connector B:



Serial Connector 1		
Pin	Function	Note
1	RS485 Data in A	Serial Port #2
2	RS485 Data in B	
3	RS232 Transmit	Serial Port #1
4	RS232 Ground	
5	RS232 Receive	Serial Port #1
6	+5v Output	
7	RS485 Data out Z	Serial Port #2
8	RS485 Data out Y	

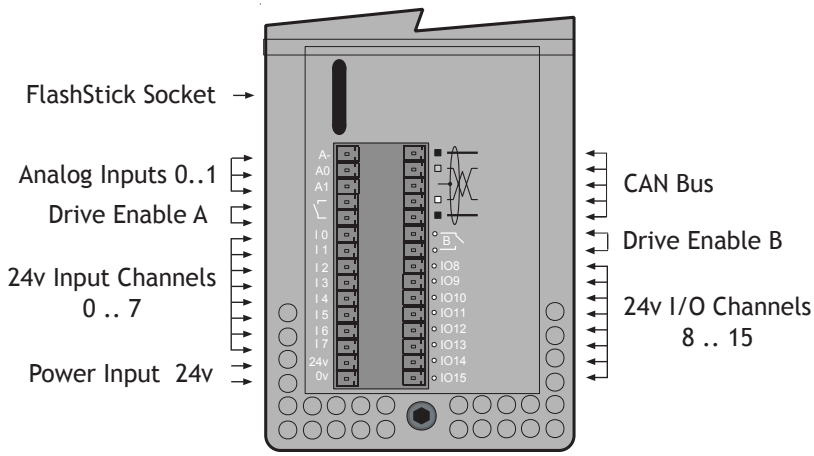
There is no hardware handshake on the serial ports. An XON\XOFF protocol is used.

Network Interconnection

The optional Fibre-Optic Network connection is designed for communication between the master modules and membrane keypads on large machines, or small workgroups of machines.

The software for the network supports interconnection of up to 15 nodes in a token-ring network format. The nodes may consist of any combination of compatible master controllers and Trio Membrane Keypads.

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



MC224 lower front panel

24v Power Supply Input

The MC224 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.

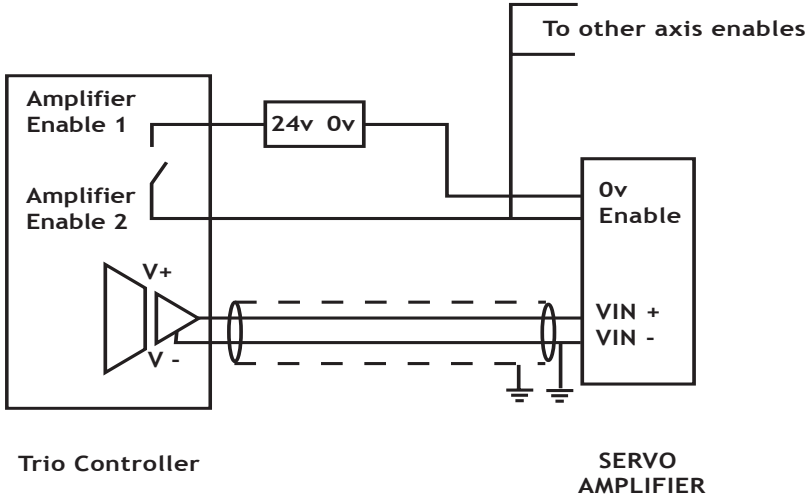
Note: ***24V dc, Class 2 transformer or power source.***

Amplifier Enable (Watchdog) Relay Outputs

Two internal relay contacts are available to enable external amplifiers when the controller has powered up correctly and the system and application software is ready. The amplifier enables A and B are solid-state relays with an ON resistance of 25 ohms at 100mA. The enable relay will be open circuit if there is no power on the controller OR a 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 hold-up circuit or chain that must be intact before a 3-phase power input is made live.

Note: ***ALL STEPPER AND SERVO AMPLIFIERS MUST BE INHIBITED WHEN THE AMPLIFIER ENABLE OUTPUTS ARE OPEN CIRCUIT***

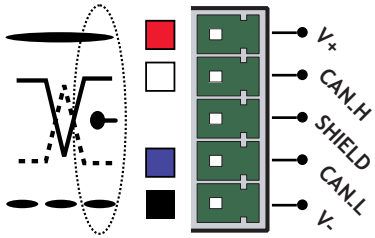


Amplifier Enable Output

CAN Bus:

The MC224 features a built-in CAN channel. This is primarily intended for Input/Output expansion via Trio's P315 and P325 modules. It may be used for other purposes when I/O expansion is not required.

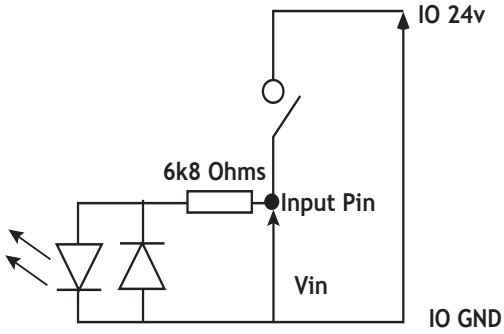
The CANbus port is electrically equivalent to a ***DeviceNet*** node.



24v Input Channels

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

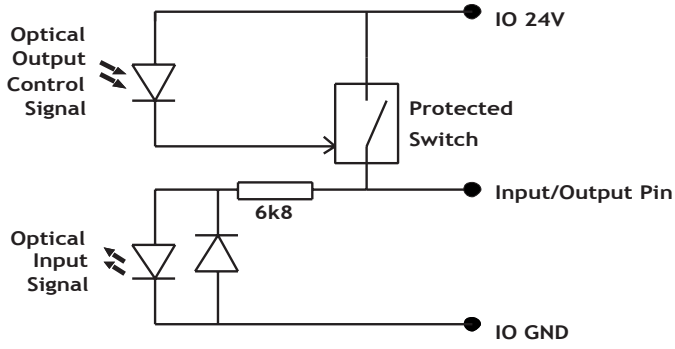
All of the 24v input channels have the same circuit although 8 on the master unit have 24v Output channels connected to the same pin. These bi-directional channels may be used for Input or Output to suit the application. If the channel is to be used as an Input the output should not be switched on in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



24v I/O 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.

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



MC224 - Feature Summary

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: up to 38400 baud. (1) RS485 channel built in. (1) Serial adapter port. CANbus port. (***DeviceNet*** compatible)
Memory	1 Mbyte battery backed RAM. 1 Mbyte TABLE memory. Flash EPROM program storage.
Memory Stick	Socket for plug-in "Nexflash Mediastick". Used for storing programs and/or data.
Programming	Multi-tasking TRIO BASIC system, maximum 14 user tasks.
Servo Cycle	Programmable: 1 ms, 500 usec or 250 usec.
Position Resolution	32 bit position count.
Speed Resolution	32 bits. Speed may be changed at any time. Moves may be merged.
Interpolation modes	Linear 1-24 axes, circular, helical, CAM Profiles, speed control, electronic gearboxes.
Power Input	24V dc, Class 2 transformer or power source. 18..29V dc at 450mA typical. Maximum: 800 mA + digital output current.
Encoder Power Output	5v at 600mA total for 4 encoders via daughter boards. Maximum 150mA per daughter board.
Amplifier Enable Output	2 N.O. solid-state relays rated 24V ac/dc nominal. Maximum load 100mA. Maximum voltage 29V.
Digital Inputs	16 Opto-isolated 24V inputs. (Upper 8 inputs shared with output terminals)
Digital Outputs	8 Opto-isolated 24V outputs. Current sourcing (PNP) 250 mA. (max. 1A per bank of 8)
Analog Inputs	2 x 12-bit 0 to 10V.

CHAPTER

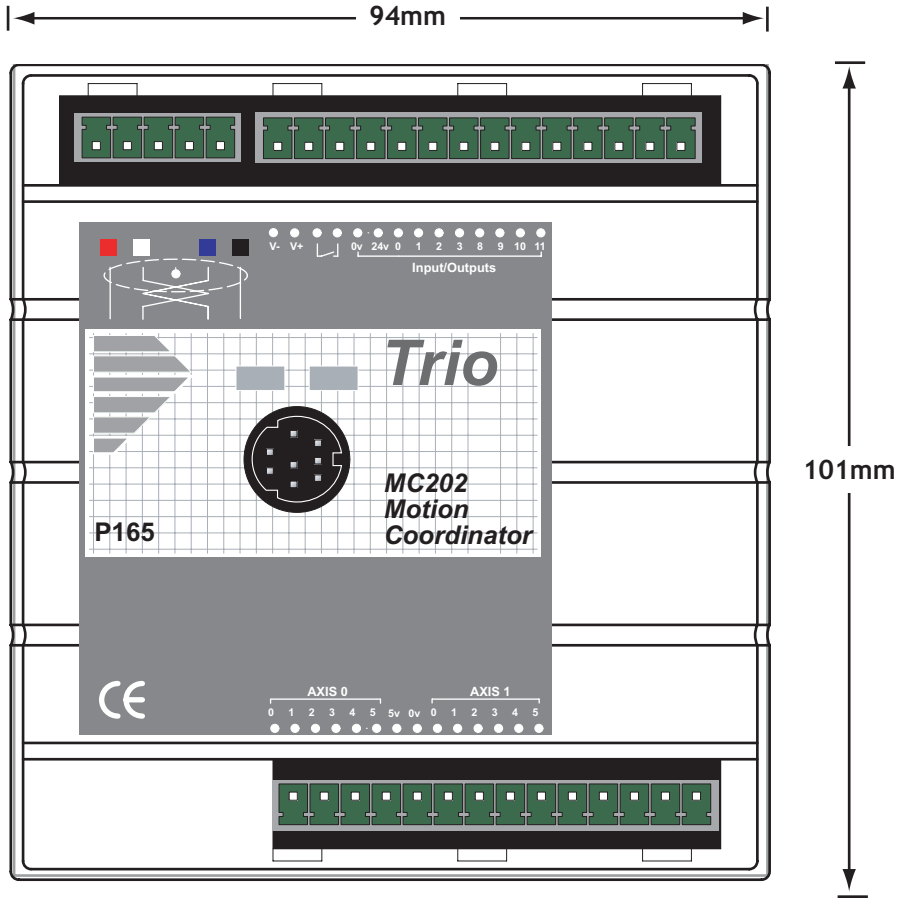
3

INSTALLATION

Motion Coordinator MC202

Packaging

The *Motion Coordinator* MC202 is a DIN rail mounting product. The module is held by a spring clip on its bottom edge on to the DIN rail. 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 around the unit. The dimensions are as shown below.



Including the disconnect terminal, the unit is 48mm deep.
When the serial connector is fitted, around 60mm more depth is required.

Connection To Other Trio Products

The MC202 may be connected to other *Motion Coordinator* modules on the CAN bus only.

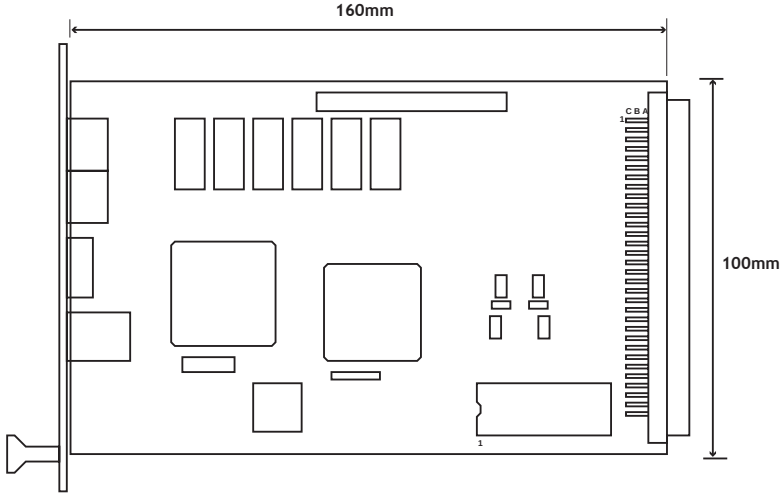
Environmental Considerations

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.

Motion Coordinator Euro 205

Packaging

The *Motion Coordinator* Euro205 is a 3U Euro rack mounting product. The PCB dimensions are 160mm long x 100mm height. The connectors and front panel overhang these dimensions.



Rack Mounting

The Euro205 should be mounted in a shielded metal rack. 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 around the unit.

Connection To Other TRIO Products

The Euro205 may be connected to other *Motion Coordinator*s on the CAN bus or fibre-optic network only.

Environmental Considerations

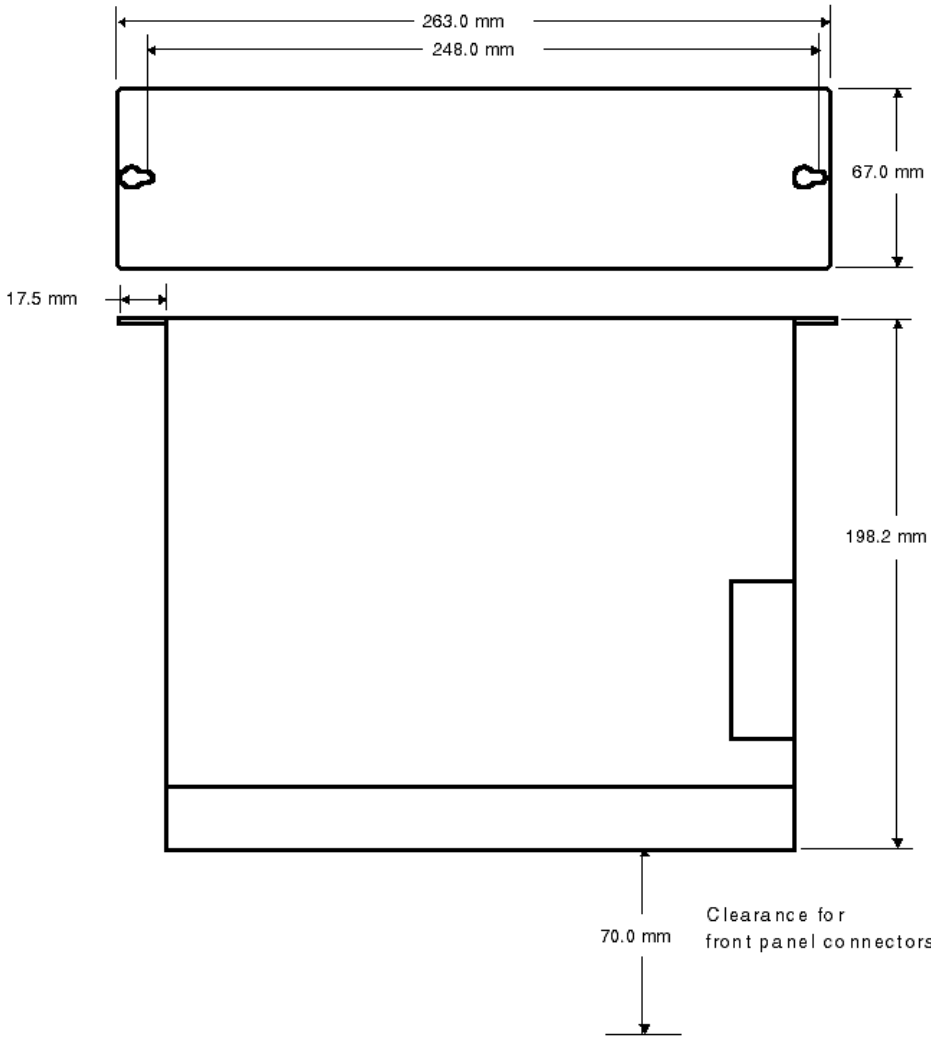
Avoid violent shocks to, or vibration of, the Euro205 whilst in use or storage.

EMC Considerations

The Euro205 requires external power supplies, a card housing and suitable back-plane connections to be supplied and fitted by the machine builder before it can be used. For this reason the Euro205 is not considered by Trio to be a product for which we can offer qualification to known EMC standards. It is therefore only suitable to be taken into service on machines where the whole system fulfils the necessary EMC requirements.

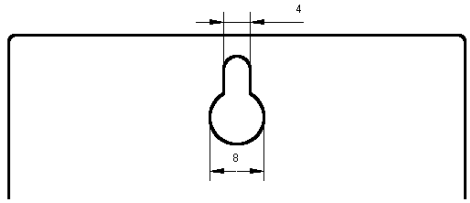
Motion Coordinator MC216

The *Motion Coordinator* MC216 modules are supplied with the same exterior packaging which has been specifically designed for ease of use and flexibility of mounting. The MC216 dimensions are given below.



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.



Module Interconnection

Modules to be connected should be mounted on 70mm centres at the same height. The MC216 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.***

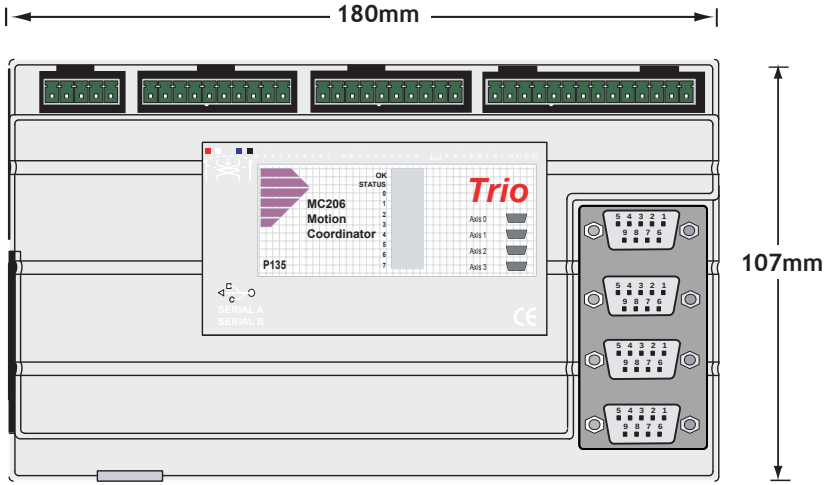
Environmental Considerations

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.

Motion Coordinator MC206

Packaging

The *Motion Coordinator* MC206 is a DIN rail mounting product. The module is held by spring clips on its bottom edge on to the DIN rail. 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 around the unit. The dimensions are as shown below.



Including the disconnect terminal, the unit is 48mm deep.
When the serial connector is fitted, around 60mm more depth is required.

Connection To Other TRIO Products

The MC206 may be connected to other *Motion Coordinator*s on the CAN bus or fibre-optic network only.

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.

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.

1) Mount noise generators such as contactors, solenoid coils and relays as far away as possible from the modules.
2) Where possible use solid-state contactors and relays.
3) Fit suppressors across coils and contacts.
4) Route heavy current power and motor cables away from signal and data cables.
5) Ensure all the modules have a secure earth connection.
6) 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 advice on specific controllers, see the installation requirements later in this chapter. 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.

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:

1) Equipment must be constructed to ensure that any electromagnetic disturbance it generates allows radio and telecommunications equipment and other apparatus to function as intended.
2) Equipment must be constructed with an inherent level of immunity to externally generated electromagnetic disturbances.

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

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

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

Testing Standards

For the purposes of testing a typical system configuration 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:

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.

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.

Installation Requirements to Ensure Conformance

Motion Coordinator MC202

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:

1) The MC202 modules should be earthed via its EARTH connection of the CAN connector.

2) 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.

3) Screened cables should be used for encoder, stepper and registration input feedback signals and for the demand voltage from the controller to the servo amplifier if relevant. The demand voltage wiring must be less than 1m long and preferably as short as possible. The screen 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 earth.

4) 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.

5) The MC202 should not be handled whilst the 24 volt power is connected.

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

Motion Coordinator Euro 205

The Euro205 should be mounted in a screened metal card cage. The power supplies should be generated by regulated filtered supplies.

The following requirements should also be adhered to:

1) The Euro205's front panel should be earthed.

2) 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.

3) Screened cables should be used for encoder, stepper and registration input feedback signals and for the demand voltage from the controller to the servo amplifier if relevant. The demand voltage wiring must be less than 1m long and preferably as short as possible. The screen 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 earth.

4) 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.

The Euro205 should not be handled whilst the power is connected.

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

Motion Coordinator MC206

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:

1) The MC206 modules should be earthed via its EARTH connection of the CAN connector.

2) 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.

3) Screened cables should be used for encoder, stepper and registration input feedback signals and for the demand voltage from the controller to the servo amplifier if relevant. The demand voltage wiring must be less than 1m long and preferably as short as possible. The screen 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 earth.

4) 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.

5) The MC206 should not be handled whilst the 24 volt power is connected.

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

Motion Coordinator MC216

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:

1) 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.
2) 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.
3) 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.
4) 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.

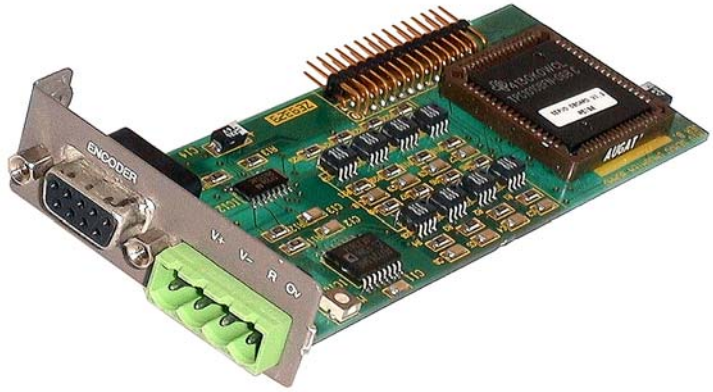
As well as following these guidelines, any installation instructions for other products in the system must be observed.

CHAPTER

4

DAUGHTER BOARDS

Description The axis daughter boards give the *Motion Coordinator* system enormous flexibility in its configuration.



The number of daughter boards which may be fitted is dependant on the controller type.

The MC216 and MC224 master units can hold up to 4 daughter boards in any combination. Up to 12 further daughter boards can be connected using optional Axis Expander units (4 daughter boards per unit). Communication Daughter Boards P290, P291, P292, P295, P296 and P297 can only be fitted in the master unit.

The Euro 205 has a single daughter board slot which can be used to provide a fifth axis in addition to the four axes (maximum) on board.

The MC206 can use a single daughter board to provide a sixth axis in addition to the five axes (maximum) which are available on the standard controller.

The *Motion Coordinator* recognises the type of axis daughter board or communications daughter board fitted in each slot. For example, a slot with a stepper daughter board fitted will return its **ATYPE** axis parameter as 1:

```
>>Print atype
1.0000
```

Also slot 0 fitted with a USB daughter board, will return its **COMMSTYPE** parameter as 21:

```
>>Print COMMSTYPE SLOT(0)
21.0000
```

There are 15 types of daughter boards currently available:

Product Code:	Description	ATYPE/ COMMS TYPE
P200	Servo Encoder	2
P210	Servo Resolver	5
P220	Reference Encoder	3
P230	Stepper	1
P240	Stepper Encoder	4
P242	Hardware Pswitch	10
P260	Analog Output	6
P270	SSI Absolute Servo	7
P280	Differential Stepper	4
P290	CAN	8
P291	SERCOS *	24
P292	SLM (Drivelink)	22
P295	USB Interface	21
P296	Ethernet *	25
P297	Profibus	23

Note: For volume / OEM applications, Trio can produce custom daughter boards for specific customer applications where required.

Daughter boards marked * are for use with MC206 and MC224 only.

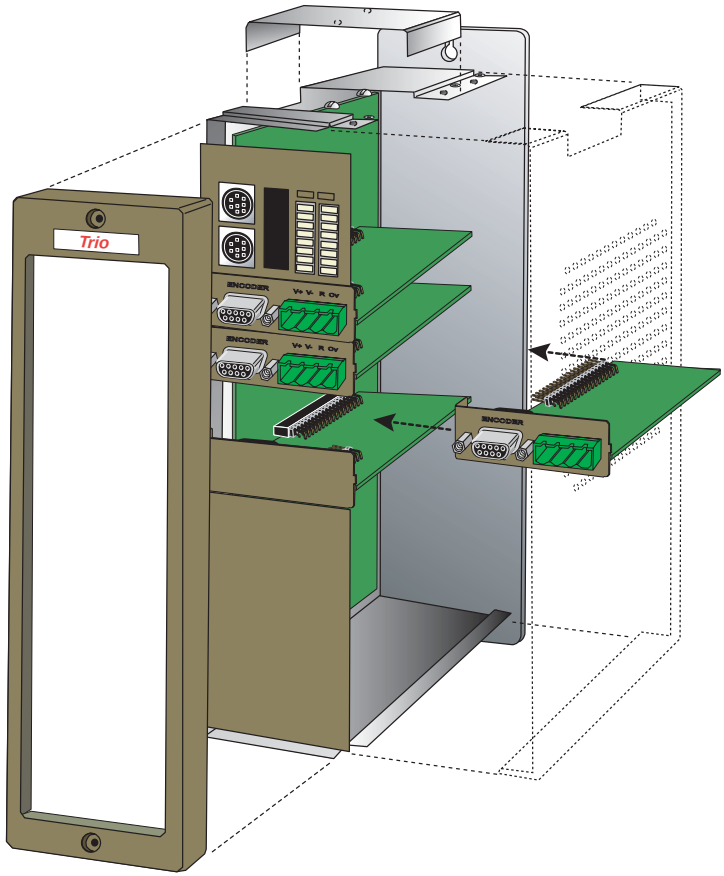
Fitting and Handling Daughter Boards

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*.

Fitting Daughter Boards to the MC216 / Axis Expander

- Check there is no power on the module
- Unscrew the 2 Allen screws which secure the front moulding cover and remove the cover
- Unscrew the top ribbon cable bus cover. If the ribbon cable is connected, remove this by pushing the retaining levers outwards.
- Unscrew the 8 screws which secure the right hand side cover of the module
- 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.



MC216 + Axis Expander Slot Sequence

The slot number of any daughter board is fixed by its position in the MC216/ MC224 module. For standard axis daughter boards, this also fixes the axis number as shown.

Location	Axis
Slot 0 (Top)	Axis 0
Slot 1	Axis 1
Slot 2	Axis 2
Slot 3 (Bottom)	Axis 3

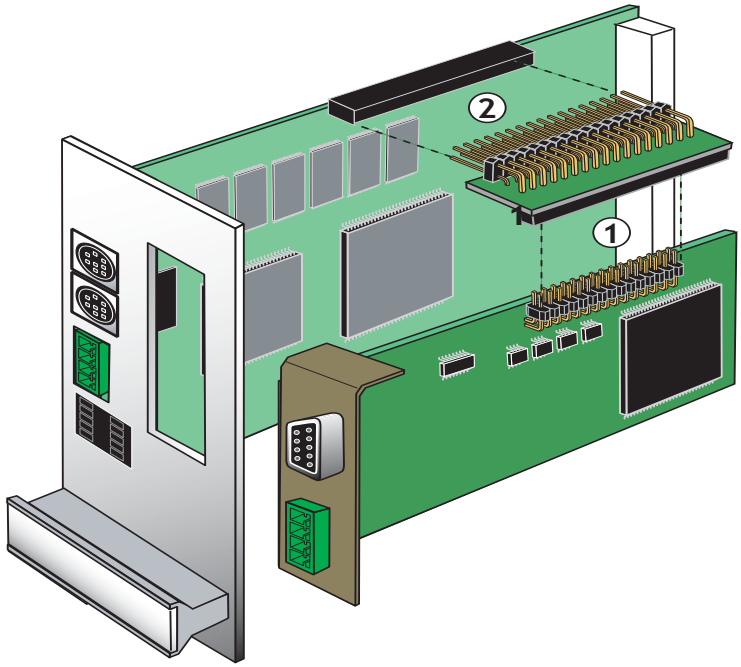
The axis number of any axis daughter board in the Axis Expander module is fixed by its position and the setting of the front panel selector switch on the Axis Expander.

Location	4-7*	8-11*	12-15**
Slot 0 (Top)	Axis 4	Axis 8	Axis 12
Slot 1	Axis 5	Axis 9	Axis 13
Slot 2	Axis 6	Axis 10	Axis 14
Slot 3 (Bottom)	Axis 7	Axis 11	Axis 15

Where digital drive communications daughter boards are used, the axis number is set by software and may not necessarily be the same as the slot number. In any case, with CANopen, SLM and SERCOS, multiple axes can be controlled via a single daughter board.

Fitting Daughter Boards to the Euro205

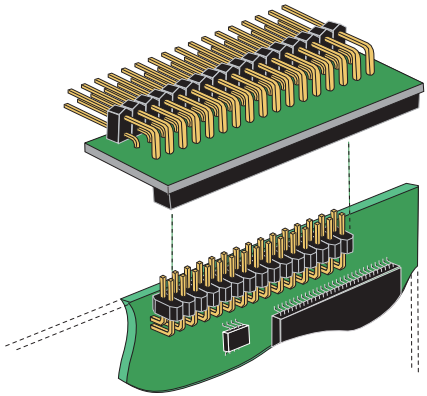
The *Motion Coordinator* Euro205 can use a single daughter board with an optional P445 Euro 205 Daughter Board Mounting Kit, which consists of a replacement (double width) front panel and 90° header connector.



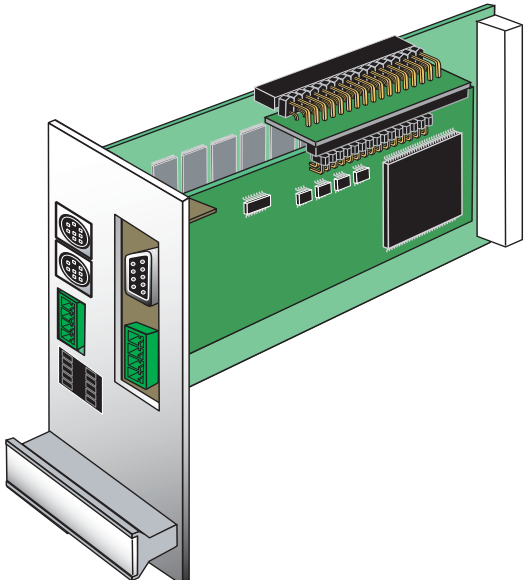
Installation Procedure

- Ensure that anti-static precautions are taken when handling the Euro205 board.
- Check there is no power on the module
- Remove the standard front panel by undoing the retaining screws from the rear of the PCB.
- Insert the new daughter board into the bottom of the adaptor, ensuring that the pins are inserted into the holes nearest the front of the Eurocard as shown in the following diagram (#1).

Note: Some daughter boards, such as the CANBus and Profibus communications boards, feature a longer connector and so the adaptor features a full-length socket to accommodate these. On an axis Daughter Board with the standard (shorter) connector the board should be inserted into the holes nearest the edge of the adaptor, as shown:



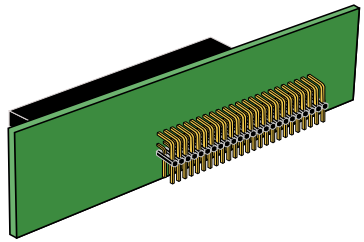
- Place the Euro205 PCB down on a firm flat surface and insert the adaptor pins into the daughter board socket (#2)
- Fit the new front panel, being careful to line up the mounting bracket with the front panel of the daughter board.
- Insert the retaining screw into the front panel bracket to hold the daughter board in place.



Euro205 shown with Daughter Board in place

Fitting Daughter Boards to the MC206

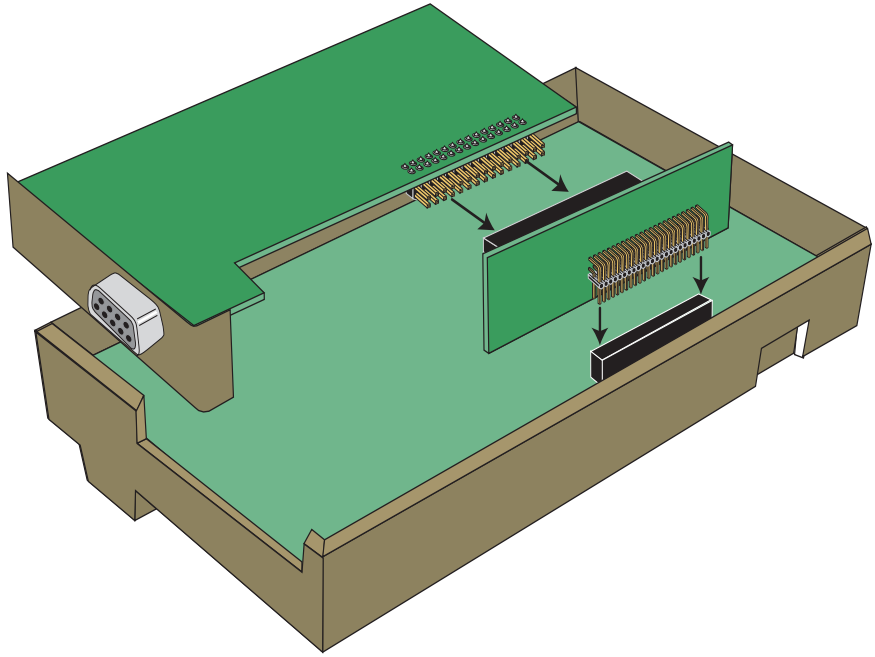
The *Motion Coordinator* MC206 can use a single daughter board. This is fitted internally, and connected via the optional P399 MC206 Daughter Board Adaptor (shown right).



The axis daughter boards are supplied in an anti-static bag or box. This should be used to hold the daughter board at all times when not installed in the *Motion Coordinator*.

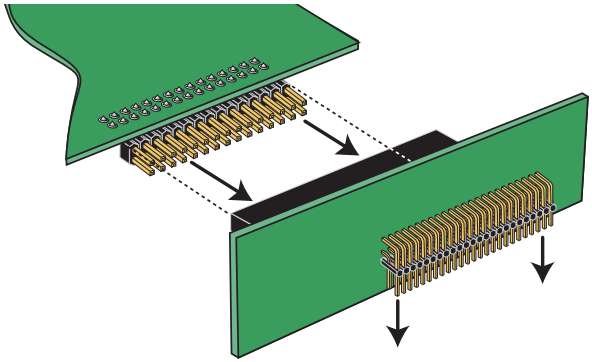
Installation Procedure

- Check there is no power on the module
- Turn the module face-down and carefully remove the back moulding. To remove the back moulding, a thin bladed small screwdriver needs to be inserted between the back and front mouldings to release in sequence each of the five latches which locate the top edge of the back on to the front.
- Ensure that anti-static precautions are taken when handling the MC206 board.
- Connect the daughter board to the P399 adaptor, and insert the adaptor into the connector on the MC206 circuit board, as shown below

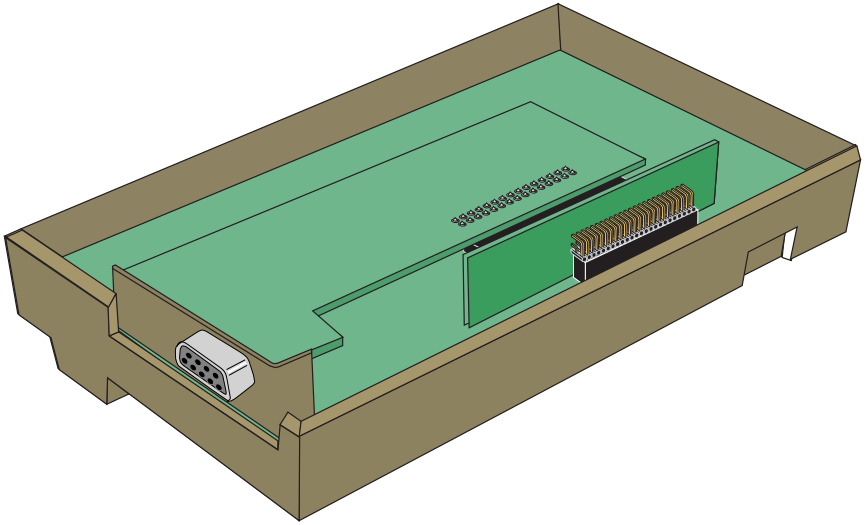


Rear of MC206 showing P399 daughter board adaptor

Note: Some daughter boards, such as the CANBus and Profibus communications boards, feature a longer connector and so the adaptor features a full-length socket to accommodate these. On an axis Daughter Board with the standard (shorter) connector the board should be inserted into the holes nearest the edge of the adaptor, as shown:



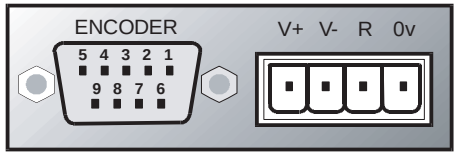
Alignment of MC206 daughter board adaptor



Rear of MC206 with Daughter Board In place

Servo Encoder Daughter Board

Trio Product Code P200



Description: The servo daughter board provides the interface to a DC or Brushless servo motor fitted with an encoder or encoder emulation.

ATYPE parameter for Servo Encoder Daughter Board = 2

Analog Output

The Servo Encoder 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.

Encoder Inputs

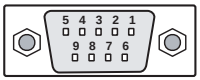
The encoder port provides differential line receiver inputs capable of counting up to 1MHz edge rate. These inputs are opto-isolated to maximise the noise immunity of the system. The encoder port also provides a convenient 5v output for powering the encoder, simplifying wiring and eliminating external supplies.

Registration Function

The Servo Encoder Daughter 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.

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.



Pin.	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	Registration Input 5v Input pin
shell	Screen

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. Screened cable should be used for the voltage output connection and the screen connected to the metal case or backplate. Do not connect screens to the registration 0V terminal.

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 function can be switched on or off with the SERVO axis parameter:

```
>>SERVO =ON
```

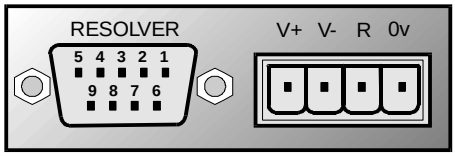
When the servo function is OFF the value in the axis parameter DAC is written to the 12 bit digital to analog converter:

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

Note: To maintain compatibility with earlier products, the daughter board hardware inverts the voltage relative to the DAC parameter.

Servo Resolver Daughter Board

Trio Product Code P210



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

The resolver 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 a12 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.

Connections: **Voltage Output**

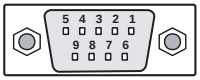
The +/-10v output voltage to drive external servo amplifiers is generated between the V+ and V- pins. The voltage output is isolated and floating but if multiple servo daughter boards are fitted it should be noted that the boards share a common power supply for generation of the +/-10v. This means that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

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

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



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

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

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 analog converter.

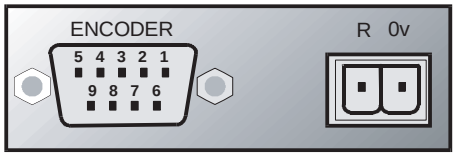
Therefore:

`>>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

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.

Reference Encoder Daughter Board

Trio Product Code P220



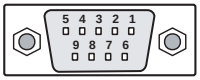
Description: **ATYPE** parameter for Encoder Daughter Board = 3

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.

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.



Pin.	function
1	channel A true
2	channel A complement
3	channel B true
4	channel B complement
5	0V
6	marker (Z) true

Pin.	function
7	marker (Z) complement
8	+5V (150 mA max) - This output is short circuit protected
9	Registration 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.

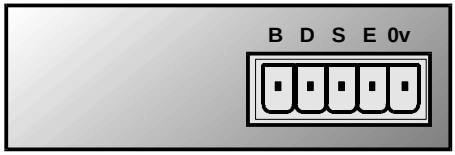
If the encoder does not have a marker pulse, pins 6 and 7 may be left unconnected.

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.

Stepper Daughter Board

Trio Product Code P230



Description `ATYPE` parameter for Stepper Daughter Board = 1

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.

Specifications **Maximum Output Frequency**

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

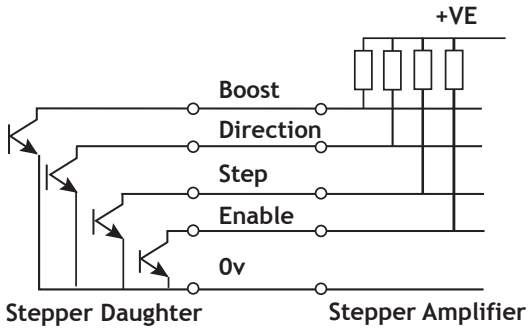
In microstep mode the stepper daughter can generate pulses at up to 250 KHz. See MICROSTEP command.

Connections

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

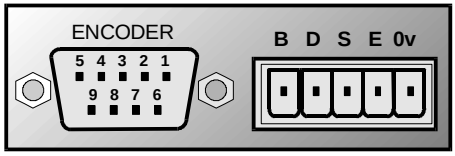
Each of the connections B, D, S, and E is an open-collector output. The outputs can be pulled up to 24v externally and can drive up to 100 mA. Overcurrent protection is provided on the outputs.

Circuit


+VE
Boost
Direction
Step
Enable
0v
Stepper Daughter
Stepper Amplifier

Stepper Encoder Daughter

Trio Product Code P240



Description: **ATYPE** parameter for Stepper Encoder Daughter Board = 4

The stepper daughter board with position verification has all the features of the standard 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.

Connections: **Stepper Output**

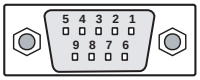
Each of the connections B, D, S, and E is an open-collector output. The outputs can be pulled up to 24v externally and can drive up to 100 mA. Overcurrent protection is provided on the outputs.

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

The circuit is identical to that of the P230 Stepper Daughter Board

Reference Encoder Input

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.



Pin.	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	Registration 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.

Choosing an encoder for position verification:

Stepper encoder daughter boards generate fewer pulses than the number of steps the controller considers the axis to be moving. The daughter board divides the number of steps the controller according to the axis parameter **MICROSTEP**:

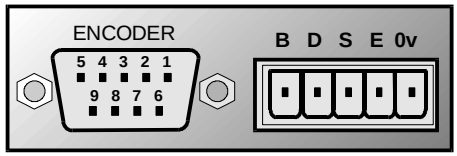
MICROSTEP=OFF	a factor of 16 (default mode)
MICROSTEP=ON	a factor of 2

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

Hardware PSWITCH Daughter Board

Trio Product Code P242



Description: The Hardware PSWITCH Daughter Board is recognised with ATYPE of 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. The daughter board encoder input does NOT have registration facilities. The encoder edge rate for hardware PSWITCH functioning is limited to 500kHz.

Note: PP_STEP = -1 cannot be used to reverse the count direction on this card.

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:

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

Software Considerations:

Each of the 4 channels is programmed individually with a PSWITCH command.

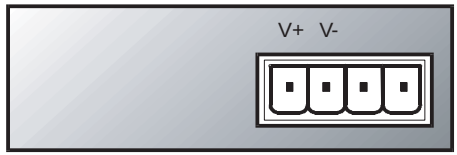
Syntax: **PSWITCH(sw,en,axis,opno,opst,setpos,rspos)**

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
rspos:	The position at which output is reset, in user units

For full details, see the PSWITCH description in the Trio BASIC reference section.

Analog Output Daughter Board

Trio Product Code P260



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

The Analog 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.

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 analog converter.

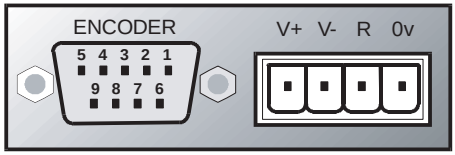
Therefore:

>>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.

SSI Servo Encoder Daughter Board

Trio Product Code P270



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

The SSI daughter board provides the interface to a DC or Brushless servo motor fitted with an absolute SSI Gray Code 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 programmable so that any number of bits from 1 to 24 can be clocked into the position registers.

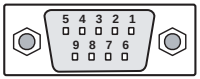
If the registration function is to be used then a minimum of 12 bits is required for correct operation.

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.

Connections: **SSI Connections:**

The SSI 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.



Pin.	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	Registration Input 5v Input pin
shell	protective ground

The SSI 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: 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 multi-turn 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 analog converter.

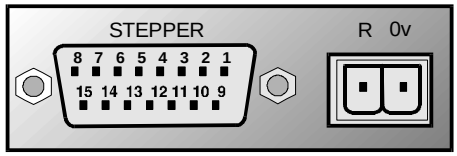
Therefore:

>>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.***

Differential Stepper Daughter Board

Trio Product Code P280



Description: **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.

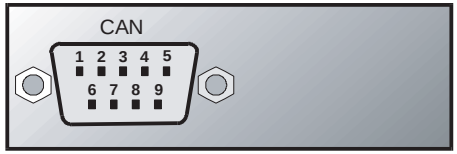
A 2 pin terminal connector is provided for the registration input.

Software Considerations

The differential daughter board uses the same axis gate array as a stepper encoder daughter board as is therefore recognised by the *Motion Coordinator* 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.

CAN Daughter Board

Trio Product Code P290



Description: **ATYPE** parameter for CAN Daughter Board = 8

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.

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:

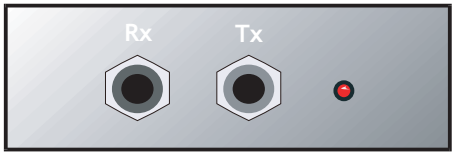
Pin	Use
1	no connection
2	CAN_L
3	CAN_GND
4	no connection
5	Shield
6	no connection
7	CAN_H
8	no connection
9	no connection

No terminating resistor is fitted internally to the CAN daughter board. A 120 ohm resistor must be fitted between pins 2 and 7 if at the end of the CAN network.

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, such as that for INFRANOR SMT-BD CANbus drives and some CANopen drives.

SERCOS Daughter Board

Trio Product Code P291



Description: **ATYPE** parameter for CAN Daughter Board = 24

The SERCOS daughter board is designed to control up to 8 servo amplifiers using the standard SERCOS fibre-optic ring. It has the benefit of full isolation from the drives and greatly reduces the wiring required.

SERCOS Connections SERCOS is connected by 1mm polymer or glass fibre optic cable terminated with 9mm FSMA connectors. The connectors on the daughter board and the drives in the network are colour-coded and the ring is completed by connecting light-grey (Tx) to dark-grey (Rx) in a series loop.

LED Indicator The red LED lights to indicate that the ring is open or there is excessive distortion in the signal.

SERCOS Interface **SE**rial **R**ealtime **CO**mmunications **S**ystem

IEC61491 / EN61491

SERCON 816 ASIC

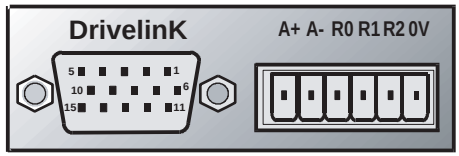
2, 4, 8 and 16 Mbit/sec selectable by software

Software settable intensity

Up to 8 axes per ring at 250usec servo update

SLM Daughter Board

Trio Product Code P292



Description: **ATYPE** parameter for SLM Daughter Board = 22

The SLM daughter board is aimed at providing digital control channels for servo drives utilising the SLM protocol of Control Techniques. This daughter board has the necessary circuits for controlling between 1 and 3 axes which can be individual drives or a single drive using the CT Multiax concept.

SLM Connections The SLM connector is a 15 way high-density D-type socket.

Pin	Use
1	Com axis 0
2	/Com axis 0
3	Hardware enable
4	0V external
5	24V output to supply SLM
6	Com axis 1
7	/Com axis 1
8	no connection
9	no connection
10	no connection
11	24V output to supply SLM
12	0V external
13	Com axis 2
14	/Com axis 2
15	Earth / Shield

Connect the cable screen to the plug shell (screen/cable clamp) as well as pin 15.

A 6-way terminal is provided for power and registration inputs.

Pin	Function
A+	24V supply for SLM
A-	0V supply for SLM
R0	Axis 0 24V registration capture input
R1	Axis 1 24V registration capture input
R2	Axis 2 24V registration capture input
0V	0V return for registration capture intputs

A 24 volt power source must be applied to A+ and A- to supply the SLM modules. This may be the same supply as is used for the Motion Coordinator.

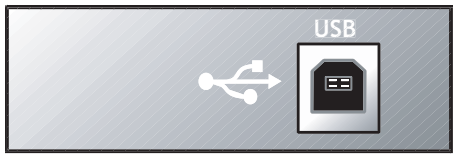
The power requirements are:

Supply voltage, 24V d.c. +/-10%.

Current consumption, 400mA in-rush, 100mA typical under normal running conditions.

USB Daughter Board

Trio Product Code P295



Description: **ATYPE** parameter for USB Daughter Board =21

The USB daughter board provides a very high speed Universal Serial Bus link between the *Motion Coordinator* and a host PC fitted with a USB port.

Support for this high speed interface is included in Trio's *Motion* Perfect 2 application and an ActiveX (OCX) software library is available which allows developers to include direct access to the *Motion Coordinator* within their own programs. Active X controls are supported by the majority of PC development platforms, including Visual Basic, Delphi, Visual C++, C++ Builder etc.

Ethernet Daughter Board

Trio Product Code P296



Description: **ATYPE** parameter for Ethernet Daughter Board =25

The Ethernet daughter board provides a high speed link between the *Motion Coordinator* and a host PC via a factory or office local area network (LAN).

An industry standard Telnet terminal can be used to access the Motion Coordinator's command line over Ethernet.

The daughter board can run the Modbus TCP protocol as a slave, Thus allowing a Modbus master to read and write integer or floating point values to the global VRs and to access the I/O.

Software Support Support for Ethernet is included in Trio's *Motion* Perfect 2 application and an ActiveX (OCX) software library is available which allows developers to include direct access to the *Motion Coordinator* within their own programs. Active X controls are supported by the majority of PC development platforms, including Visual Basic, Delphi, Visual C++, C++ Builder etc.

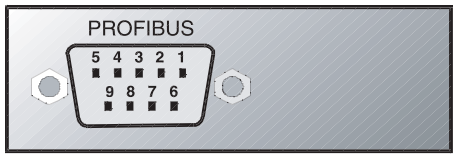
Ethernet Interface Physical Layer: 10 Base-T

Connector: RJ-45

Protocols: Industry standard RFC compliant TCP/IP stack and protocol support for HTTP server, TCP, UDP, ARP, ICMP and IP.

Profibus Daughter Board

Trio Product Code P297



Description: **ATYPE** parameter for Profibus Daughter Board =23

PROFIBUS is a fieldbus system, which is in widespread use all over the world. With the P297 Profibus Daughter Board and appropriate software on the *Motion Coordinator*, it is possible to connect to a variety of third party Master devices using the Profibus DP protocol.

Connections:

Pin	Use
1	Not Connected
2	Not Connected
3	Data Line B
4	Request To Send
5	0v
6	+5v Output
7	Not Connected
8	Data Line A
9	Not Connected

Example: A example Trio BASIC Profibus driver for cyclic data transfer is available from the Trio Web Site **www.triomotion.com** (in the **DOWNLOADS** section, under **Application Notes**).

This program sets up the SPC3 chip for transfer of 16 integers from the master and 16 integers to the master on a cyclic basis as determined by master unit.

CHAPTER

5

EXPANSION MODULES

Input/Output Modules

General Description

Trio can supply a range of Input/Output Modules and Operator Interface Units. The *Motion Coordinator* controllers allow for I/O expansion by having a CAN interface. This allows the I/O modules to form a network up to 100m in length. The operator interface units all communicate with 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:	Product Code:
CAN 16-I/O Module	P315
CAN Analog Input Module	P325
Membrane Keypad	P503
Mini-Membrane Keypad	P502
Application Specific Keypad using FO-VFKB	P504

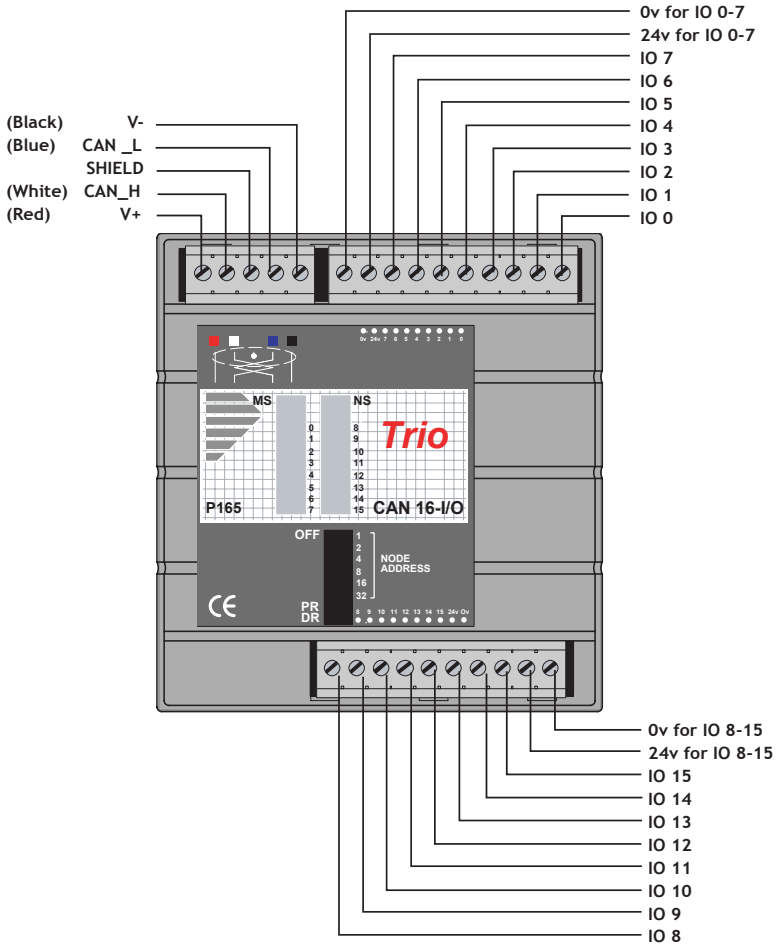
CAN 16-I/O Module (P315)

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

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

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

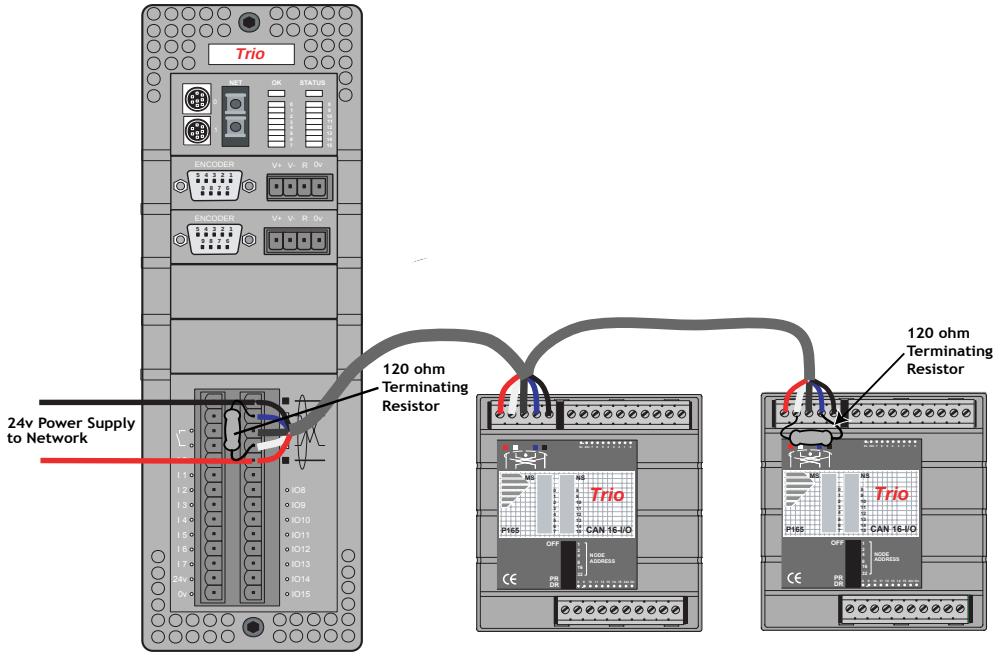
The CAN 16-I/O Module may also be used as an I/O expander for Lenze drives with an appropriate CAN interface.



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

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

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.

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

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

DIP Switch Settings

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

Alternative connection protocols

The DIP switches can be set up to allow for different protocols to be used, enabling the Trio I/O module to be used with other manufacturers devices. At present the only other protocol supported is that utilised by LENZE drives.

The DIP switch marked “PR” selects the protocol to be used. Switched right it selects the TRIO protocol, switched left it selects the module to act as a LENZE drive expansion I/O.

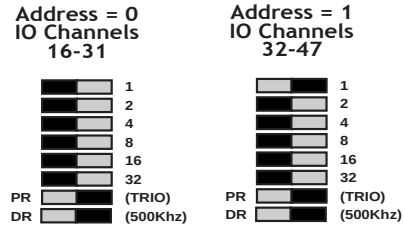
TRIO Protocol:

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

The top 6 DIP switches on the CAN 16-I/O set the module address. Only addresses 0 - 15 are valid for CAN 16-I/O modules.

The switch marked DR sets the CAN Bus communications rate to 125kHz or 500kHz. Only 500Khz is valid with the TRIO protocol.

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



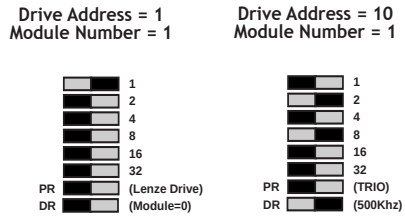
Note: ***The I/O Channels referred to above start at 16. This is because the numbering sequence starts with channels 0 - 15, which are on the Motion Coordinator master unit itself.***

LENZE Drive Protocol:

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

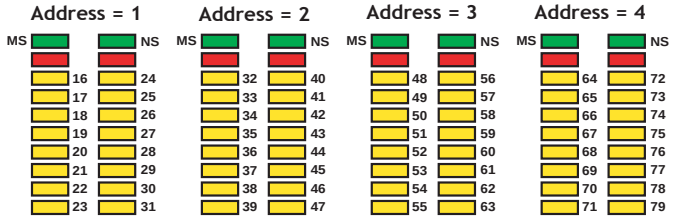
The top 6 DIP switches are used to set the drive number. This should be set to a to 1..63. If the drive number is set to 0, the module will transmit to drive 1.

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.



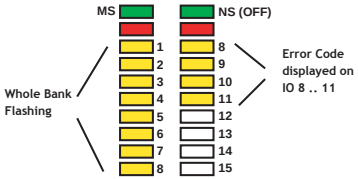
LED Indicators

When NS is ON LEDs marked 0 - 15 represent the input channels 0 - 15 of the module. The actual input as seen by the *Motion Coordinator* software will depends on the I/O modules address:



Error Codes:

When an error occurs on a CAN I/O module, the fault code is represented by a binary number displayed on the leds.



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

Software Interfacing

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

CANIO_STATUS, CANIO_ADDRESS and CANIO_ENABLE

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

- Inputs 0 - 31 ONLY are available for use with system parameters which specify an input, such as **FWD_IN**, **REV_IN**, **DATUM_IN** etc.
- 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.

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 *Motion Coordinator* will report the wrong number when questioned:

```
>>? NIO
```

If this is not as expected check:

- Terminating 120 Ohm Network Resistors fitted?
- 24Volt Power to each IO bank required?
- 24Volt Power to Network?
- DIP switches in sequence starting 0,1,2...?
- System Software Version 1.40 (or higher)?
- *Motion Coordinator* CANIO_ADDRESS=32?

Specification

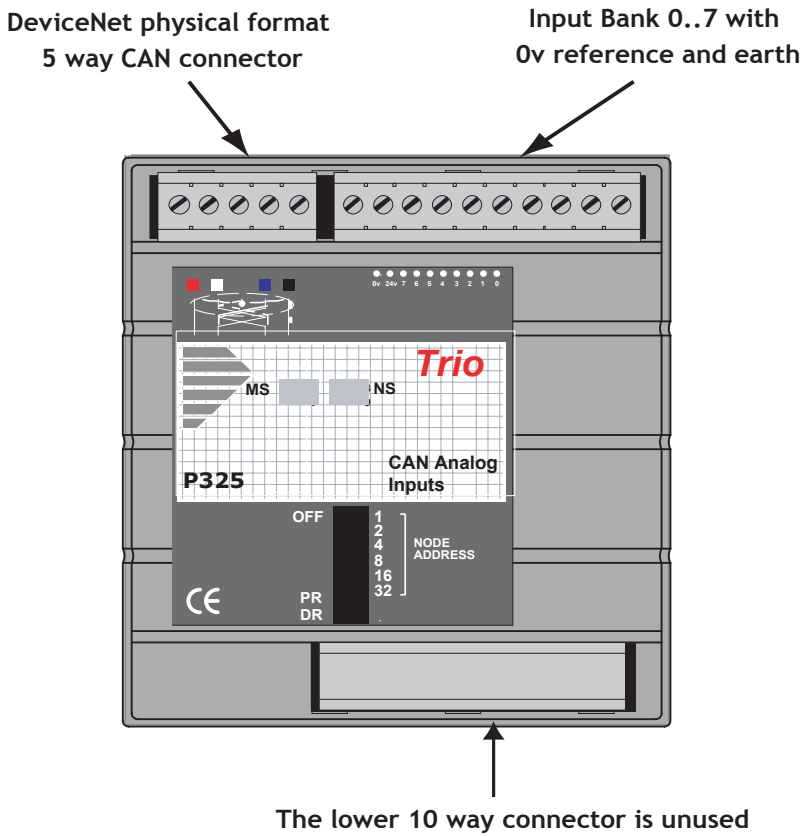
Inputs:	16 24volt inputs channels with 2500v isolation
Outputs:	16 24volt output channels with 2500v isolation
Configuration:	16 bi-directional channels
Output Capacity:	Outputs are rated at 250mA/channel. (1 Amp total/ bank of 8 I/O's)
Protection:	Outputs are overcurrent and over temperature protected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

CAN Analog Inputs Module (P325)

The CAN Analog Input Module allows the *Motion Coordinator* 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.

I/O Connections

The CAN Analog Input Module has 3 disconnect terminal connectors:



Bus Wiring

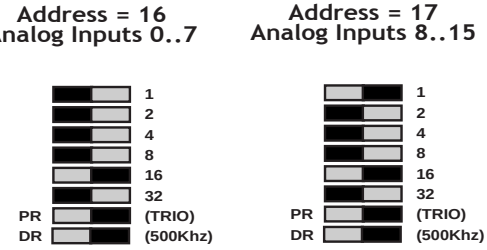
See Can 16 I/O for details

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

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.

LED Indicators

MS	“Module Status”	ON when module powered on OK
NS	“Network Status”	ON when module powered on OK and initialised.

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 and will return values -2047 to 2048 to the function AIN(). The first 4 channels are also available as system parameters AIN0, AIN1, AIN2, and AIN3. This allows these values to be seen using the SCOPE function.

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.

Troubleshooting

If the network configuration is incorrect 2 indications will be seen: The P325 module will indicate that it is uninitialised and the *Motion Coordinator* will report the wrong number when questioned:

```
>>? NAIO
```

If this is not as expected check:

- Terminating 120 Ohm Network Resistors fitted?
- 24Volt Power to Network?
- DIP switches in sequence starting 16,17,18...?
- System Software Version 1.42 (or higher)?
- *Motion Coordinator* CANIO_ADDRESS=32?

Specification

Analog Inputs:	8+/-10 volt inputs with 500v isolation from CAN bus
Resolution:	12 bit
Protection:	Inputs are protected against 24v over voltage.
Address Setting:	Via DIP switches
Power Supply:	24V dc, Class 2 transformer or power source. 18 ... 29V dc / 1.5W
Mounting:	DIN rail mount
Size:	95mm wide x 45mm deep x 105mm high
Weight:	200g
CAN:	500kHz, Up to 32 analog input channels
EMC:	BSEN50082-2 (1995) Industrial Noise Immunity / BS EN55022 (1995) Class A Industrial Noise Emissions

Operator Interfaces

There are two main options when considering Operator Interface products. You can utilise products connected to either the Trio Fibre-Optic Network, or third party products connected to one of the *Motion Coordinator*'s serial ports.

Using the Trio Fibre-Optic Network

Trio supply a range of operator interfaces which are designed to connect via the *Motion Coordinator*'s fibre-optic network connection. these are:

- P502 - Membrane Keypad
- P503 - Mini-membrane keypad
- Fibre-optic interface module (Allows users to design their own keypad on Trio fibre-optic network)

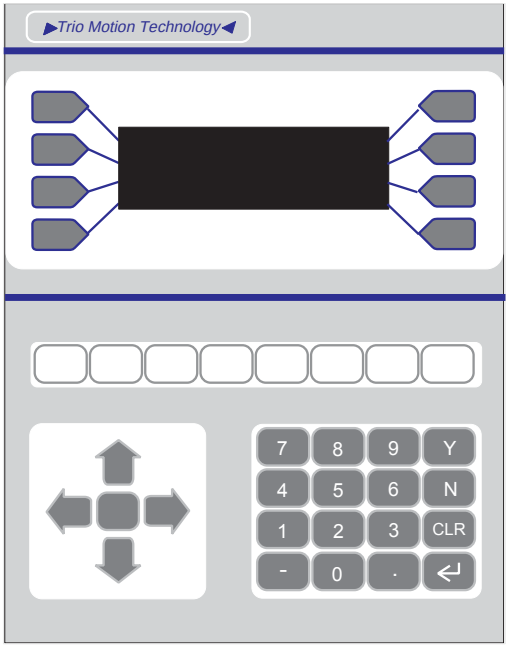
Third Party Modules

It is possible to connect to a wide variety of third-party operator interface panels via one of the *Motion Coordinator*'s serial ports (RS232 or RS485).

A growing number of programmable keypads and HMIs provide the user with a choice of serial interface protocols to enable communication with various PLCs and Industrial Computers. One such protocol is Modbus RTU. The *Motion Coordinator* system software provides built-in support for the Modbus protocol.

The Modbus protocol provides single point to point communication between the *Motion Coordinator* and a programmable keypad/display. Implementation of the protocol is provided on serial port 1 for RS232 and port 2 for RS485. Port 0 is the main programming port and does not have the Modbus option.

Membrane Keypad (P503)

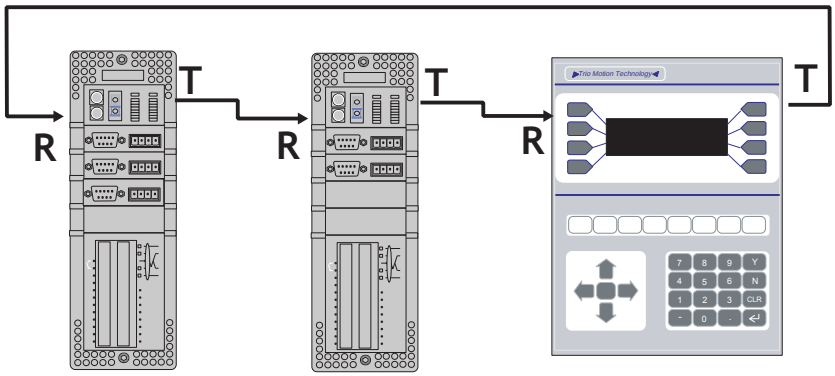


The Membrane Keypad brings together all the elements required for an effective man-machine interface in one package thus minimising the time taken to mount and connect to the rest of the system. The keypad has 37 tactile keys, eight of which can be defined by the user by inserting key legends from the 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* master module or network of master modules and membrane keypads. 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.

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*.

The physical connection of the network takes the form of a ring. The interconnections between nodes being made with fibre optic cable

.



Example of Network Connection

Connection of the Membrane Keypad

The fibre optic link to a master module or network of masters is the Hewlett-Packard “Versatile Link” format.



The fibre optic connectors are colour coded:

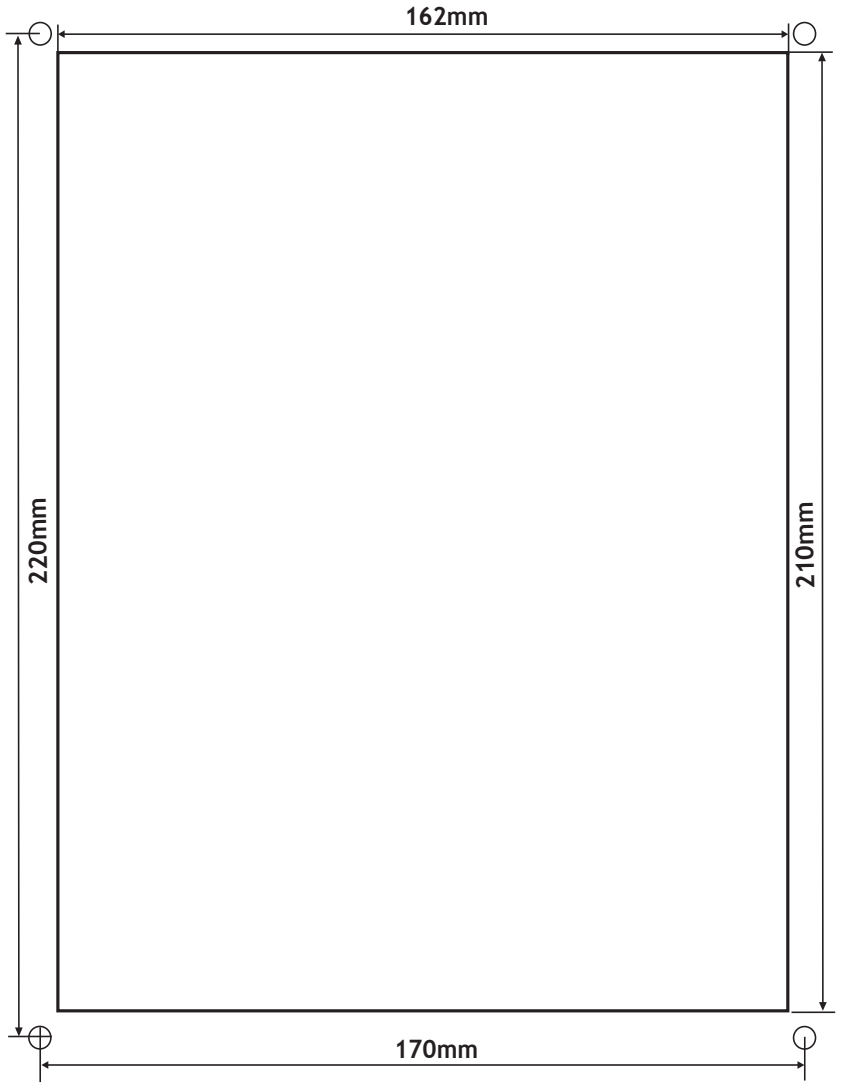
- Grey or Black- Transmitter
- Blue - Receiver

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.

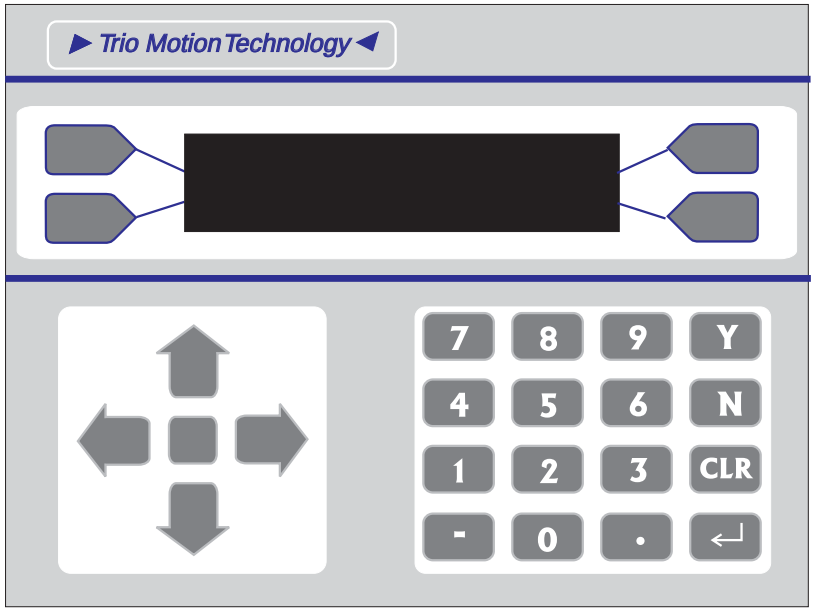
Power is applied to the two pin disconnect terminals on the side of the Keypad.

Mounting the Membrane Keypad

To mount the Membrane Keypad a rectangular cutout and four holes are required, as shown below. 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.



Mini-Membrane Keypad (P502)

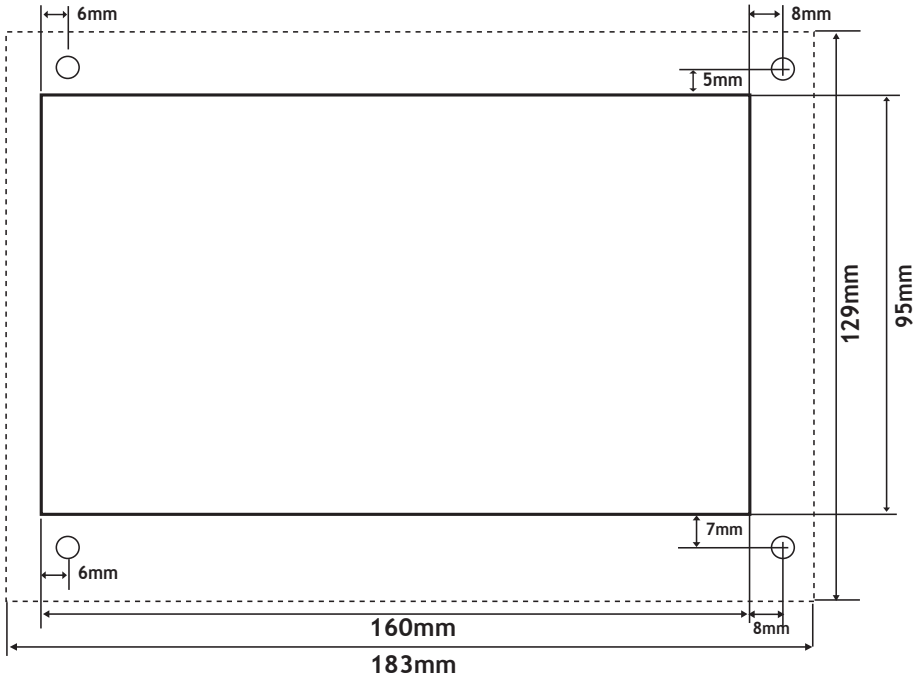


The Mini-Membrane Keypad is a lower cost alternative to the full membrane keypad. The keypad has 25 tactile keys. Incorporated into the keypad is a two line by twenty character vacuum fluorescent display. Connection to the control system is via a fibre optic cable to the *Motion Coordinator* master module 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.

Mounting the Mini-Membrane Keypad

The Mini-Membrane Keypad can be either mounted into a rectangular cutout or may the 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.



Connection of Mini-Membrane Keypad

The connections are identical to the membrane keypad.

Programming the 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.

Writing to the Membrane and Mini-Membrane Display

The Trio BASIC command **PRINT** 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:

CHR(..)	Function	Description
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	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.

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.

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 #	Get #4 Value	Get #3 Value
Up Arrow	1	33	20
Left Arrow	2	34	22
Centre 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

Key	Key #	Get #4 Value	Get #3 Value
<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

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.

Example: 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: In this mode the key presses should 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.

This character sequence is formatted as a Trio network message (type 4). It is designed to work only when there is one keypad and one *Motion Coordinator* OR where the keypad is the next node in the network. See chapter 11 for details of how to construct network messages for keypads at other nodes.

Summary of Features		
	P503 Membrane Keypad	**P502 Mini-Membrane**
Size	230mm x 180mm x 50mm	192mm z 183mm x 50mm
Weight	1.450Kg	0.600Kg
Operating Temperature	0-45 degrees C	0-45 degrees C
Power supply	24V dc, Class 2 transformer or power source. 18 ... 29V dc 500mA.	24V dc, Class 2 transformer or power source. 18 ... 29V dc 500mA.
Number of Keys	37	25
Type of Keys	Metal dome tactile, debounced.	Metal dome tactile, debounced.
Display	4x20 Vacuum Fluorescent, with anti-glare filter.	2x20 Vacuum Fluorescent, with anti-glare filter.
Environmental	Sealed to IP65, provided mating face to panel is sealed.	Sealed to IP65, provided mating face to panel is sealed.

Summary of Features		
	P503 Membrane Keypad	**P502 Mini-Membrane**
Material	Polyester top layer resistant to most solvents	Polyester top layer resistant to most solvents
Data Output	8 bit serial, no parity	8 bit serial, no parity

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.

FO-VFKB Display Interface

Display Interface DIL connector:

FUNCTION	PIN NO.		FUNCTION
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

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 pro-

gram 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.

Function	IDC plug pin no.		Function
Row 4	1	2	Column 0
Row 1	34		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

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.

Data Connection

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

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

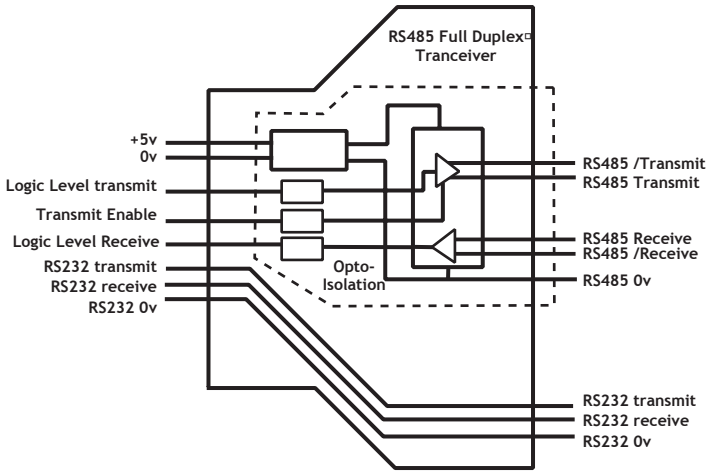
Communications Adaptors

Serial to RS485 Adapter: (P348)

The Serial to RS485 adapter is designed to buffer the logic level serial port connections on Trio Euro205, MC202 and MC216 *Motion Coordinator*s. It contains an opto-isolated RS485 full-duplex buffer. This gives 4 wire RS485 output. A direction control input is available on the logic level side of the converter. This allows the *Motion Coordinator* to enable the output transmitter only when required. If the transmit enable input is left unconnected the transmitter will be permanently enabled.

Note ***Two different adaptors are available:***
P348 for the Euro 205 and MC216, and P349 for the MC202.

RS485 Adapter Simplified Diagram



3 pins on the adapter can optionally be used to “Pass-through” the RS232 serial channel which is present on the same connector on the *Motion Coordinator* as the logic level serial port.

Motion Coordinator Side Pins:

9 Way Male 'D' Connector:	
1	+5v input to power adapter
2	RS232 Receive pass-through
3	RS232 Transmit pass-through
4	0v input to power adapter
5	RS232 Pass-through 0v
6	no connection
7	Transmit enable input
8	Logic level receive
9	Logic level transmit

RS485 Side Pins:

25 Way Female 'D' Connector:			
1..4	No connection		
5	RS485 Gnd		
6	RS485	A	A REC
7	RS485	B	B /REC
8	RS485	Z	Z /TRAN
9	RS485	Y	Y TRAN
10			
11	RS232 Receive pass-through		
12	RS232 Transmit pass-through		
13	RS232 Pass-through 0v		
14..25	No Connection		

P349 - Serial to RS485 adapter to an MC202:

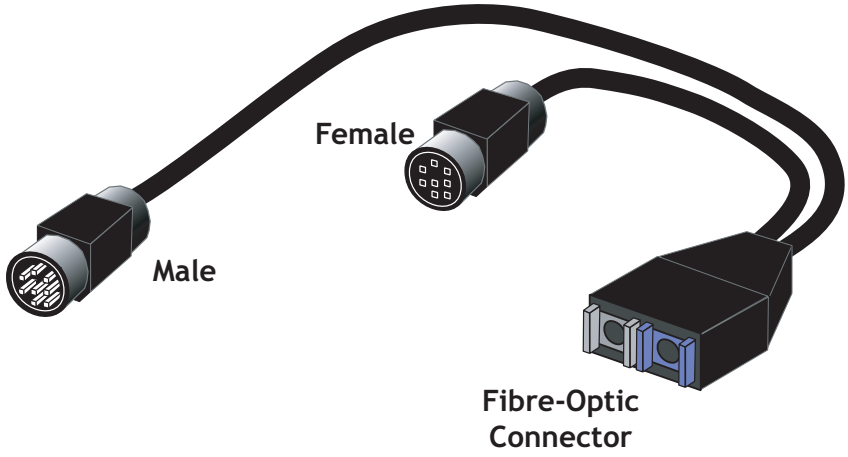
The following cable connections should be used for connecting an MC202 to the RS485 adapter:

MC Pin	9 Way Pin:	Function
1	1	5 volts
2	4	0 volts
3	3	RS232 transmit
4	5	RS232 0v
5	2	RS232 receive
6	7	logic level transmit enable (see note)
7	9	logic level transmit
8	8	logic level receive

Note: ***MC202 Only:***

MC202's with serial numbers 00081 and above have a transmit enable output on pin 6. MC202's with serial numbers less than 00081 do not have an RS485 transmit enable output available and can only be used with a point to point RS485 connection, not multi-drop.
To use the MC202 in a multi-drop configuration the MC202 system software must support this mode. If the RS485 is only point-to-point the transmit enable pin can be left floating.

P435 - Serial to Fibre-Optic Adapter



The P435 Fibre Optic Adapter may be connected to the MC202, Euro205 or MC206 controllers to enable the connection of the Trio Mini-membrane, Membrane Key-pad and the FO-VFKB.

The adapter provides a pass-through connector for the RS-232 serial port.

Controller Type	Serial Connector Used
MC202	A
Euro205	B
MC206	A
MC224	A

Note ***On the MC202, the fibre-optic adapter may be used to connect to the Membrane keypads and FO-VFKB only. The Trio Fibre-Optic Network is not supported by the MC202.***

CHAPTER

6

System Setup and Diagnostics

Preliminary Concepts:

Host Computer	A Windows PC running *Motion* Perfect 2.
Motor	A tuned servo drive / motor configuration for a servo axis or a stepper motor and drive combination
Prompt	When the controller is ready to receive a new command, the prompt >> will appear on the left hand side of the current line in the "terminal" under the "tools" menu
Axis Parameters	Can be written to or read from. For example the proportional gain of a servo axis has the name **P_GAIN**. It can be written to: **P_GAIN=0.5** or read from: **PRINT P_GAIN**. For further information see chapter 9.

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 *Motion Coordinator* modules.

Note:

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

Preliminary checks

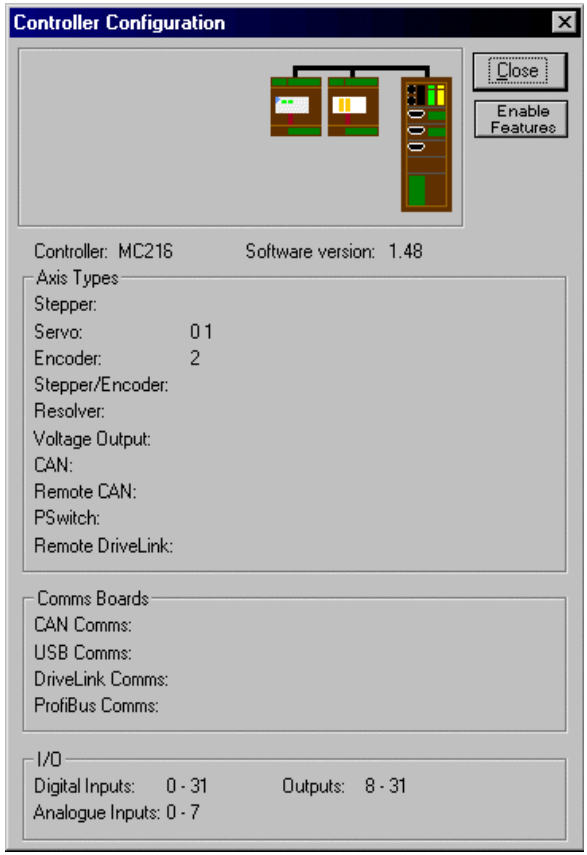
All wiring should be checked for possible misconnection and integrity before any power is applied.

- Disconnect all external connectors from the system, apart from the CANBus and ribbon cable bus (only required if using an Axis Expander module)
- Check bank selector switches on any Axis Expander modules, and the module address DIP switches on the CAN I/O
- Apply power to system and check the 24v power input on master module.
- Connect a serial lead between serial port 0 (or connector A) on the controllerand an unused serial port on your PC.

Checking Communications and System Configuration

- Ensure that the serial lead is connected between the *Motion Coordinator* master (serial connector A) and a free serial port on the computer.
- Apply 24v to the *Motion Coordinator*.
- 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 graphical representation of your system, as shown below.

Example:



This message would be produced by a *Motion Coordinator* MC216 with the following configuration:

- System Software version 1.48
- Two servo daughter boards in axis slots 0 and 1.
- A encoder daughter board in axis slot 2.
- A CAN16 I/O module, which adds a further 16 I/O channels, making a total of 32 channels (All 32 can be used as inputs, channels 8-31 are bidirectional and may be used as input or output as required)
- A CAN Analog Inputs module providing input channels 0-7.

Check that the system description corresponds with the modules that are actually present. If this is not so, check the CANBus and ribbon cable connections and the settings of the address switches on any CAN or axis expander modules.

Input/Output Connections:

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

Connecting a Servo Motor to a Servo Daughter Board

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

Each servo axis should be connected in turn.

- With the servo drive off or inhibited connect the motor encoder only (or the encoder emulation output from the servo drive).
- Check the encoder counts both up and down by looking at the measured axis position **MPOS** in the Axis parameter window of *Motion* Perfect (“Axis parameters” under the “Tools” menu) whilst turning the axis by hand.
- Ensure the **SERVO** axis parameter is set OFF (0) in the Axis 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 +/-10v output for the axis. Now connect the servo drive to the V+/V- connections.
- Enable the servo drive by clicking the “Drives disabled” button on the control panel. If the axis runs away the motor/drive 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:

  - Swap A and /A connections on the encoder input, or

- Swap BOTH motor terminals and tacho terminals (DC motors only!) On many digital brushless motors the direction can be reversed by a drive setting, or
- If the drive has differential inputs, reverse the voltage as it enters the drive. (This can cause problems with some servo-drives. The V- pins of the daughter board are internally connected inside the *Motion Coordinator* so the axis voltage outputs cannot float relative to each other), or
- set a negative **PP_STEP** axis parameter. (This is not possible using SSI encoders)
- If using an MC206 set a negative **DAC_SCALE** axis parameter.

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

Setting Servo Gains

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

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

The controller checks the following error typically 1000 times per second and updates the 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:

Gain	Parameter Name	Value
Proportional Gain	**P_GAIN**	1.0
Integral Gain	**I_GAIN**	0.0
Derivative Gain	**D_GAIN**	0.0
Output Velocity Gain	**OV_GAIN**	0.0
Velocity Feedforward Gain	**VFF_GAIN**	0.0

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

Example:

```
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 the *Motion* Perfect section for more details on how to use the editor. Once the program is entered, it can be run by clicking on the red button next to its name or the RUN button in the Controller Status panel.

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 feed forward to compensate for following errors together with higher proportional gains.

Exact achievement of end points of moves

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

Typically a combination of the above is required.

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

Proportional Gain

Description The proportional gain creates an output voltage, Op that is proportional to the following error E.

$$Op = Kp \; x \; E$$

Axis parameter is called **P_GAIN**

Syntax: **P_GAIN=0.8**

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

Integral gain

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

$$Oi=Ki \; x \; SE$$

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

Axis parameter is called **I_GAIN**

Syntax: **I_GAIN=0.0125**

Derivative gain

Description This produces an output Od 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.

$$Od = Kd \; x \; DE$$

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

Axis parameter is called **D_GAIN**

Syntax: **D_GAIN=5**

Output Velocity Gain

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

$$Oov = Kov \; x \; DPm.$$

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

Axis parameter is called **OV_GAIN**

Syntax: `OV_GAIN=-5`

Velocity Feed Forward Gain

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

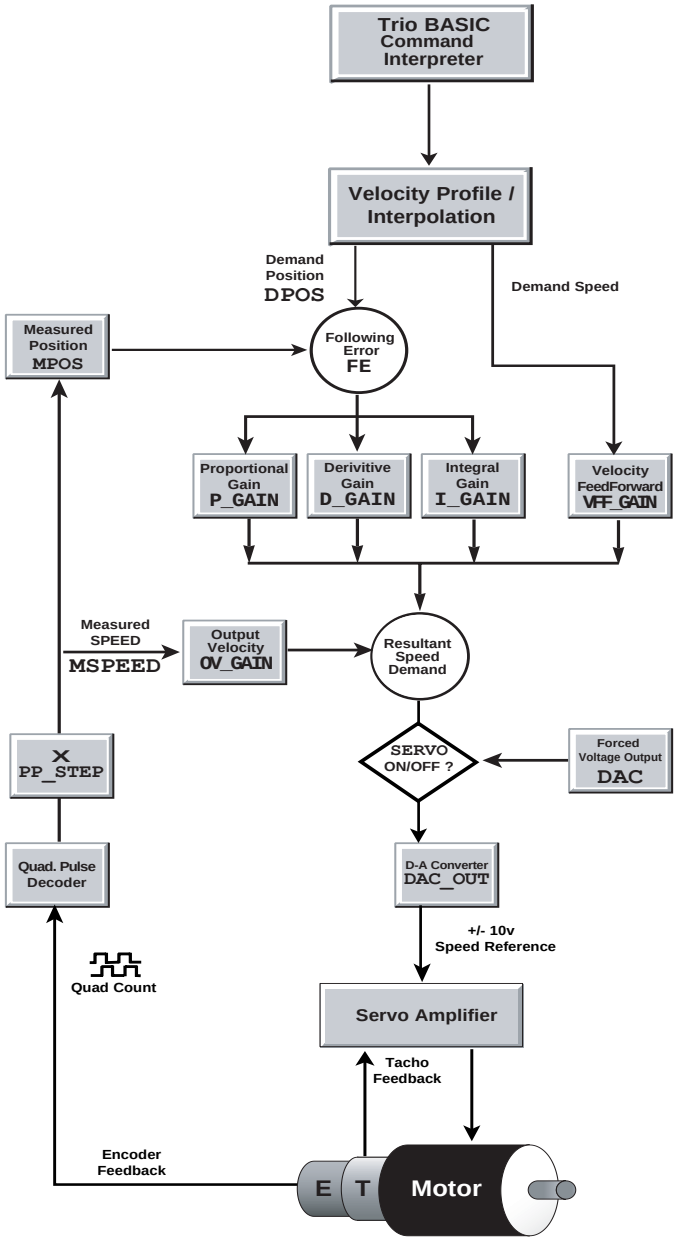
$$Ov = Kvff \; x \; DPd$$

Axis parameter is called **VFF_GAIN**

Syntax: `VFF_GAIN=10`

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

Servo Loop Diagram



Diagnostic Checklists

Problem	Potential reasons
No Status LEDs on any module	• Power Supply
LEDs lit on Master but not on other modules	• Ribbon cable / Bank select switch
OK LED ON & Status LED flashing	• Following error on at least one axis. The axis demand position and measured position exceed the programmed limit
Motor runs away without issuing a move command	• tacho/drive polarity • encoder/controller polarity • gains (drive and/or controller)
Motor runs away upon issuing a move command	• tacho feedback • encoder feedback • gains (drive and/or controller)
Motor does not move upon issuing a move command	• wiring (enables/inhibits/limits on drive and controller) • check status on all axes • drive power • feedhold applied • speed, acceleration and/or deceleration set to zero • servo set off/watchdog set off • gains (drive and/or controller) • axis is already running a move which has not completed - Check MTYPE and NTYPE
Axis goes out on following error after a time	• speed being requested requires more than 10v - check drive tacho gain and motor/ drive speed characteristics. • drive shutting down on current limit after a time

Problem	Potential reasons
Axis losing position	• encoder coupling • encoder signal (wire length, differential/single ended encoder) • mechanics
Motion Perfect cannot "connect" with the controller	• Controller running a program which transmits on serial port 0. 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, data bits, parity or stop bits have been changed. - Adjust to default of 9600 baud, 7 data bits, 2 stop bits, even parity under the "Options" menu. • Check *Motion* Perfect version. The latest version can be downloaded from **www.triomotion.com**

CHAPTER

7

PROGRAMMING

What is a program?

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

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

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

- Instructions
  - Program Flow
  - Controller Specific
- Identifiers
  - Labels
  - Data Storage

Controlling the Sequence of Events

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

Sequence

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

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

It is useful to split your program up into a series of routines, each of which handles a particular funtion of the machine. The GOSUB command will jump to a label and continue from its new location. When the program encounters a RETURN command, the program will jump back to the GOSUB from where it originally came. Take the following example:

```
PRINT "Hello"
GOSUB a_subroutine
STOP

a_subroutine:
  PRINT "World"
RETURN
```

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

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

i.e. `GOTO shut_down`

Trio BASIC instructions:

Labels, GOTO, GOSUB, RETURN, STOP

Selection

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

Example: `IF` we have made a complete batch `THEN` stop the machine

Trio BASIC Instructions:

```
IF … THEN … ELSE … ENDIF
ON ... GOTO
ON ... GOSUB
```

Iteration

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

Example

```
REPEAT
  GOSUB index_conveyor
UNTIL IN(product_sensor)=ON
```

Trio BASIC instructions:

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

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

Example:

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

The output to the screen would read:

```
1.0000 2.0000 3.0000 4.0000 5.0000
```

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

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

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

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

Such loops are especially useful for working on array data by using the variables that increment as position indexes for the arrays. As an example, we could list all our lottery numbers using the following example:

```
FOR y=1 TO 12
  FOR x=1 to 12
    MOVEABS(x,y)
  NEXT x
NEXT y
```

Controller Functions

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

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

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

examples:

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

Identifiers

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

Labels

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

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

e.g.
```
start:
enter_password:
error_handler:
```

Variables

Variables are storage locations for numeric values. they are called variables as they can be changed at any time. Just like labels, variables can often be given a user-defined name. Anywhere a number is required a variable can be used.

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

Trio BASIC *has three different variable types:*

named variables These are LOCAL variables - i.e. they are only valid within the task they are defined.

Each process can define up to 256 named variables (1024 on MC206).

Example:

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

VR() variables The controller has a global array of 251 **VR()** variables which are shared between tasks (1024 on MC206).

Example:

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

TABLE memory The **TABLE** memory is a large array of up to 16000 entries depending on the controller type. Normally used to store profiles for the **CAM**/**CAMBOX** commands.

If the controller features a battery backed memory, **VR()** variables and **TABLE** memory will be retained when the power is off. For controllers without a battery, the **FLASHVR()** command is provided to store the values in flash eprom memory.

Expressions

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

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

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

```
num_widgets = total_length / widget_length
```

has three operands, `num_widgets`, `total_length` and `widget_length` and two operators, = (assignment) & / (divide).

Reading the above as simple English would equate to:

Divide the variable `total_ length` by `widget_length` and assign the result to the variable `num_widgets`

Example 2: you could use an expression directly:

```
MOVE(widget_length+10) '(MOVE is a Trio BASIC instruction)
```

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

```
IF batch_count = batch_size THEN GOTO batch_done
```

Parameters

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

Axis Parameters

Each of the axes has its own set of axis parameters which are used to achieve many of the *Motion Coordinator* features. The axis parameters may be floating point or 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 Trio 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 AXIS(2)
```

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

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.

Process Parameters

Trio BASIC also holds a small number of parameters which are held separately for each PROCESS.

These are:

1) TICKS

2) PROCNUMBER

3) PMOVE

4) ERROR_LINE

5) INDEVICE / OUTDEVICE

6) BASE

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

Example:

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

Forcing priority of program execution

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

Allocation of Time

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

The default servo period is 1mS. This period is internally divided into 3 time slots of 1/3mS each, which are used internally for processing the servo functions, communications and general 'housekeeping' tasks respectively. The remaining time in each of these slots is used for running the users application programs.

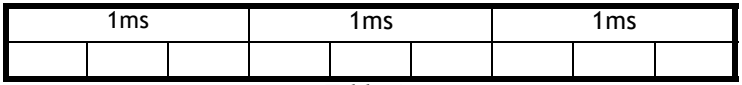


Table 1:

Process Numbers

The processes available for programs are identified by numbers, from 1 to the maximum available on the controller. For example, and MC216 can run 14 simultaneous programs. Process 0 is also allocated automatically to the *Motion Coordinator*'s command line interface / *Motion* Perfect connection.

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

Controller	Max # Programs	High Priority Tasks
MC202	3	3
Euro205	5	5,4
MC206	7	7,6
MC216	14	14,13

The two highest numbered processes (14 and 13 in our example MC216) are allocated a fixed time slot. These are referred to as the “fast” tasks. They should be used for processes which require:

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

Any other processes (including the command line) share the third time slot. Execution speed will therefore reduce as the number of programs running increases. In practice however, a useful execution speed is still obtained. Processes 1..12 are referred to as “slow” tasks.

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

>>RUN "progname",7	Run the named program immediately on specified task
>>RUNTYPE “progname”,ON,7	Assigns start-up mode for specified program

If equal time is required to be given to all programs, the high priority processes (14 and 13) should NOT be used. The time available will then be divided evenly between the remaining processes. The command line and processes 1, 2 & 3 share the remaining third. These programs and the command line use the available time slot with equal priority

Examples: No fast tasks, two normal / slow tasks

1ms			1ms			1ms		
1	2	C/L	1	2	C/L	1	2	C/L

Table 2:

No fast tasks, three normal / slow tasks

1ms			1ms			1ms		
1	2	3	C/L	1	2	3	C/L	1

Table 3:

Two fast tasks, two normal / slow tasks

1ms			1ms			1ms		
14	13	1	14	13	2	14	13	C/L

Table 4:

One fast task, two normal / slow tasks

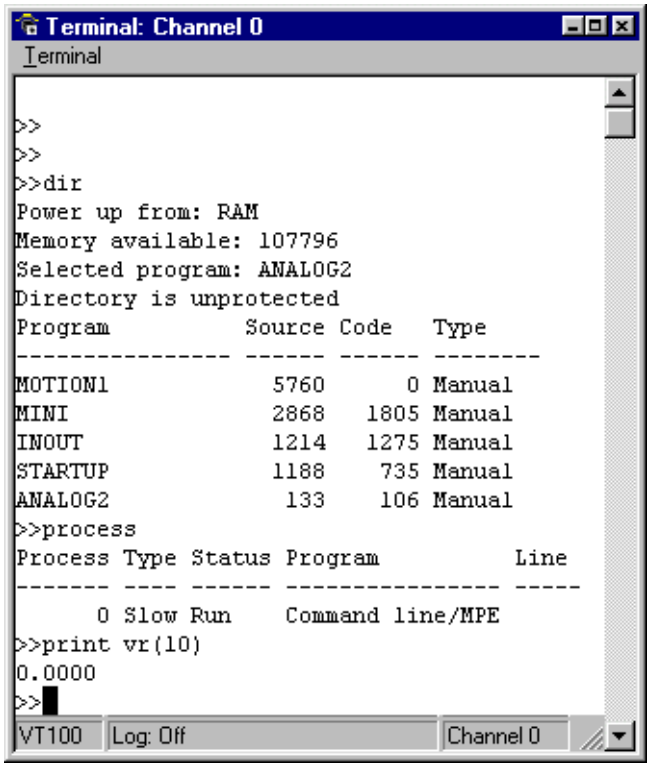
This example shows the case where there is one fast task only. This is the exception to the rule as it is allocated BOTH 'fast' time slots.

1ms		1ms		1ms	
14	1	14	2	14	C/L

Table 5:

Command Line Interface

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



Typing Commands for Immediate Execution

When the controller is waiting for a Trio 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>)***

Limitations of the command line

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

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

- Loop Instructions:
  FOR..NEXT, WHILE..WEND, REPEAT..UNTIL
- Wait Instructions:
  WA(time), WAIT UNTIL, WAIT IDLE
- Named variables:
  These are local to a program

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

Tip! ***The command line features a buffer of the last 10 commands used. This can save a lot of typing on the PC. Pressing the up arrow or down arrow cycles through the buffer.***
If you find a command you do not recognise it was probably put there by Motion Perfect!

Setting Programs to run on power up

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

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

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

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

Example Programs

Example 1

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

Example 2

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

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

Example 3

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

CHAPTER

8

TRIO BASIC COMMANDS

Motion and Axis Commands 8-10
ACC 8-10
ADD_DAC 8-10
ADDAX 8-11
AXIS 8-13
BASE 8-14
CAM 8-15
CAMBOX 8-18
CANCEL 8-25
CONNECT 8-26
DATUM 8-27
DEC 8-28
DEFPOS 8-29
FORWARD 8-30
MATCH 8-30
MHELICAL 8-31
MOVE 8-32
MOVEABS 8-33
MOVECIRC 8-35
MOVELINK 8-36
MOVEMODIFY 8-40
REGIST 8-41
RAPIDSTOP 8-43
REVERSE 8-43

Input / Output Commands 8-44
AIN 8-44
AIN0..7 / AINBI0..7 8-45
CURSOR 8-45
CHR 8-46
DEFKEY 8-46
FLAG 8-47
FLAGS 8-48
GET 8-48
GET# 8-49
HEX 8-50
IN()/IN 8-50
INPUT 8-51
INPUTS0 / INPUTS1 8-52
INVERT_IN 8-52
KEY 8-52
LINPUT 8-53

OP 8-54
PRINT 8-55
PRINT# 8-56
PSWITCH 8-57
READPACKET 8-59
RECORD 8-59
SEND 8-60
SETCOM 8-61

Program Loops and Structures 8-63
BASICERROR 8-63
ELSE 8-63
ELSEIF 8-63
ENDIF 8-65
FOR..TO.. STEP..NEXT 8-65
GOSUB 8-66
GOTO 8-67
IDLE 8-68
IF..THEN..ELSE.. ENDIF 8-68
NEXT 8-69
ON.. GOSUB 8-69
ON.. GOTO 8-70
REPEAT.. UNTIL 8-70
RETURN 8-71
STEP 8-71
STOP 8-71
THEN 8-72
TO 8-72
UNTIL 8-72
WA 8-73
WAIT IDLE 8-73
WAIT LOADED 8-74
WAIT UNTIL 8-74
WHILE 8-75
WEND 8-75

System Parameters and Commands 8-76
ADDRESS 8-76
APPENDPROG 8-76
AUTORUN 8-76
AXISVALUES 8-77
CAN 8-77

CANIO_ADDRESS 8-79
CANIO_ENABLE 8-80
CANIO_STATUS 8-80
CHECKSUM 8-80
CLEAR 8-81
CLEAR_PARAMS 8-81
COMMSERROR 8-82
COMMSTYPE 8-83
COMPILE 8-83
CONTROL 8-83
COPY 8-84
DATE 8-84
DATE$ 8-84
DAY 8-85
DAY$ 8-85
DEL 8-85
DEVICENET 8-86
DIR 8-87
DISPLAY 8-87
DLINK 8-88
EDPROG 8-92
EDIT 8-93
EPROM 8-93
ERROR_AXIS 8-94
ETHERNET 8-94
EX 8-95
EXECUTE 8-96
FEATURE_ENABLE 8-96
FLASHVR 8-98
FRAME 8-99
FREE 8-99
HALT 8-101
INITIALISE 8-101
LAST_AXIS 8-101
LIST 8-102
LOADSYSTEM 8-102
LOCK 8-103
MOTION_ERROR 8-104
MPE 8-104
NAIO 8-105
NETSTAT 8-106
NEW 8-106
NIO 8-106

PEEK 8-107
POKE 8-107
POWER_UP 8-107
PROCESS 8-108
PROFIBUS 8-108
REMOTE 8-109
RENAME 8-109
RUN 8-110
RUNTYPE 8-110
SCOPE 8-111
SCOPE_POS 8-112
SELECT 8-112
SLOT 8-113
SERCOS 8-113
SERVO_PERIOD 8-114
STEPLINE 8-115
STICK_READ 8-115
STICK_WRITE 8-116
STORE 8-116
TABLE 8-117
TABLEVALUES 8-118
TIME 8-119
TIME$ 8-119
TRIGGER 8-119
TROFF 8-119
TRON 8-120
TSIZE 8-120
UNLOCK 8-120
USB 8-121
USB_STALL 8-122
VERSION 8-122
VIEW 8-122
VR 8-123
WDOG 8-124
: 8-124
’ 8-125
. 8-125
$ 8-126

Process Parameters and Commands 8-127
ERROR_LINE 8-127
INDEVICE 8-127

LOOKUP 8-128
OUTDEVICE 8-128
PMOVE 8-128
PROC 8-129
PROC_LINE 8-129
PROCNUMBER 8-129
PROC_STATUS 8-129
RESET 8-130
RUN_ERROR 8-130
TICKS 8-130

Mathematical Operations and Commands 8-131
+ Add 8-131
- Subtract 8-131
* Multiply 8-132
/ Divide 8-132
= Equals 8-133
<> Not Equal 8-133
> Greater Than 8-134
>= Greater Than or Equal 8-134
< Less Than 8-135
<= Less Than or Equal 8-135
ABS 8-136
ACOS 8-136
AND 8-136
ASIN 8-137
ATAN 8-138
ATAN2 8-138
CLEAR_BIT 8-139
CONSTANT 8-139
COS 8-140
EXP 8-140
FRAC 8-140
GLOBAL 8-141
IEEE_IN 8-141
IEEE_OUT 8-142
INT 8-142
LN 8-143
MOD 8-143
NOT 8-143
OR 8-144
READ_BIT 8-145

SET_BIT 8-145
SGN 8-145
SIN 8-146
SQR 8-146
TAN 8-147
XOR 8-147

Constants 8-148
FALSE 8-148
OFF 8-148
ON 8-148
TRUE 8-149
PI 8-149

Axis Parameters 8-150
ACCEL 8-150
ADDAX_AXIS 8-150
ATYPE 8-150
AXISSTATUS 8-151
BOOST 8-152
CAN_ADDRESS 8-152
CAN_ENABLE 8-152
CLOSE_WIN 8-153
CLUTCH_RATE 8-153
CREEP 8-153
DAC 8-154
DAC_OUT 8-155
DAC_SCALE 8-155
DATUM_IN 8-156
DECEL 8-156
DEMAND_EDGES 8-156
DPOS 8-157
DRIVE_STATUS 8-157
D_GAIN 8-157
ENCODER 8-158
ENDMOVE 8-158
ERRORMASK 8-158
FAST_JOG 8-159
FASTDEC 8-159
FE 8-159
FE_LIMIT 8-160
FERANGE 8-160

FEGRAD 8-160
FEMIN 8-161
FHOLD_IN 8-161
FHSPEED 8-161
FS_LIMIT 8-162
FWD_IN 8-162
FWD_JOG 8-162
INVERT_STEP 8-163
I_GAIN 8-163
JOGSPEED 8-163
LINKAX 8-164
MARK 8-164
MARKB 8-164
MERGE 8-165
MICROSTEP 8-165
MPOS 8-166
MSPEED 8-166
MTYPE 8-167
NTYPE 8-167
OFFPOS 8-168
OPEN_WIN 8-169
OUTLIMIT 8-169
OV_GAIN 8-169
PP_STEP 8-170
P_GAIN 8-171
REG_MATCH 8-171
REG_POS 8-171
REG_POSB 8-172
REMAIN 8-173
REMOTE_ERROR 8-173
REPDIST 8-173
REP_OPTION 8-174
REV_IN 8-174
REV_JOG 8-174
RS_LIMIT 8-175
SERVO 8-175
SP 8-175
SPEED 8-176
SRAMP 8-176
SSI_BITS 8-176
TRANS_DPOS 8-177
TRANSITIONS 8-177
UNITS 8-177

VERIFY 8-178
VFF_GAIN 8-178
VP_SPEED 8-179

Motion and Axis Commands

ACC

Type: Axis Command

Syntax: **ACC(rate of acc)**

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:

rate of acc:	The units of the parameter are dependant on the **UNITS** axis parameter. The acceleration factor is entered in UNITS/SEC/SEC.

Example: **ACC(100)**

ADD_DAC

Type: Axis Command

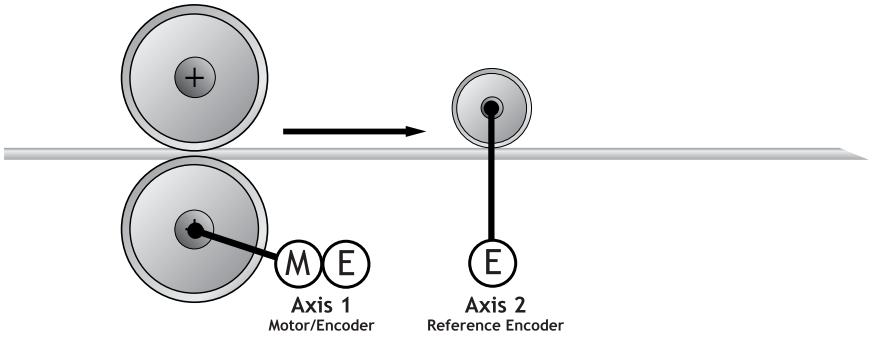
Syntax: **ADD_DAC(axis)**

Description: The **ADD_DAC** command is provided to allow a secondary encoder to be used on a servo axis to implement dual feedback control. This would typically be used in applications such as a roll-feed where you need a secondary encoder to compensate for slippage. The **ADD_DAC** function allows the output of 2 servo loops to be summed on to the speed demand output to a drive.

To use **ADD_DAC** it is necessary for the two axes with physical feedback to link to a common axis on which the required moves are executed. Typically this would be achieved by running the moves on a virtual axis and using **ADDAX** or **CONNECT** to produce a matching **DPOS** on BOTH axes. The servo algorithm gains are then set up on BOTH axes, and the output summed on to one physical output using **ADD_DAC**. Care should be taken if the required demand positions on both axes are not identical due to a difference in resolution between the 2 feedback devices.

Note that in the example below it would be necessary to set the **ATYPE** of the reference encoder axis as if it were a full servo axis. This is so that the software will perform the servo algorithm on that axis.

The **ADD_DAC** link can be terminated by setting **ADD_DAC(-1)**



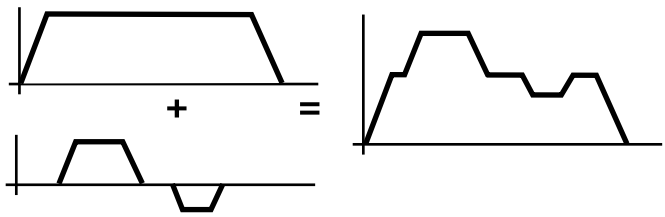
Example 1:
```
ADD_DAC(2) AXIS(1)  ' Combine axis(2) DAC with axis(1)
ADDAX(1) AXIS(2)  ' Superimpose axis 2 profile on axis 1
```

ADDAX

Type: Command

Syntax: **ADDAX(axis)**

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



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

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

Example 1:

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

Example 3: A **CAMBOX** movement is linked to an encoder input. In order to achieve registration a 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 linked to axis 2
' Axis 1 has encoder daughter board
' Axis 2 is "virtual" axis
SERVO AXIS(1)=OFF
ADDAX(1) AXIS(2)
...
CAMBOX(1000,1100,4,600,2) AXIS(0)
```

AXIS

Type: Modifier

Syntax: **AXIS(expression)**

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**

See Also: **BASE()**

BASE

Type: Motion Command

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

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

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

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
(0,2,3,1,4,5,6,7)
>>
```

This example is from an MC206 with 8 axes.

CAM

Type: Axis Command

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

Description: The **CAM** command is used to generate movement of an axis according to a table of POSITIONS which define a movement profile. The table of values is specified with the **TABLE** command. The movement may be defined with any number of points from 2 to 16000 (8000 on MC202). 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 (assuming the default **SERVO_PERIOD** of 1ms).

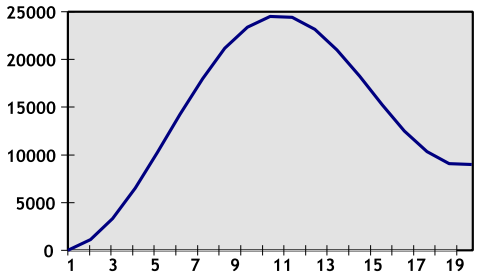
Example: Motion is required to follow the POSITION equation:

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, as shown in the graph below.

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



Note 2: When the **CAM** command is executing the **ENDMOVE** parameter is set to the end of the PREVIOUS move

CAMBOX

Type: Axis Command

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

Description: The **CAMBOX** command is used to generate movement of an axis according to a table of POSITIONS which define the movement profile. The motion is linked to the measured motion of another axis to form a continuously variable software gearbox. The table of values is specified with the **TABLE** command. The movement may be defined with any number of points from 2 to 16000 (8000 on MC202). 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..7 (MC206), 0..15 (MC216), 0..7 (Euro205), 0..2 (MC202)

link options: Bit Values:

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)

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

Note:

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

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

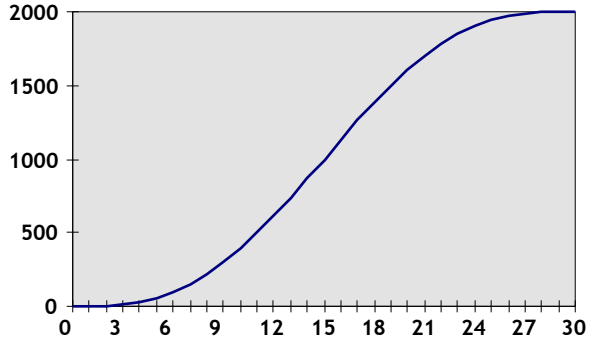
Note: When the **CAMBOX** command is executing the **ENDMOVE** parameter is set to the end of the PREVIOUS move. The **REMAIN** axis parameter holds the remainder of the distance on the link axis.

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

Example 1:

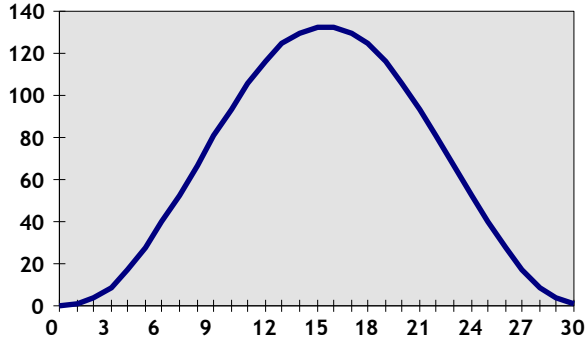
```
num_p=30
scale=2000
'
' Subroutine to generate a SIN shape speed profile
'
' Uses: p is loop counter
' num_p is number of points stored in tables pos 0..num_p
' scale is distance travelled scale factor

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:

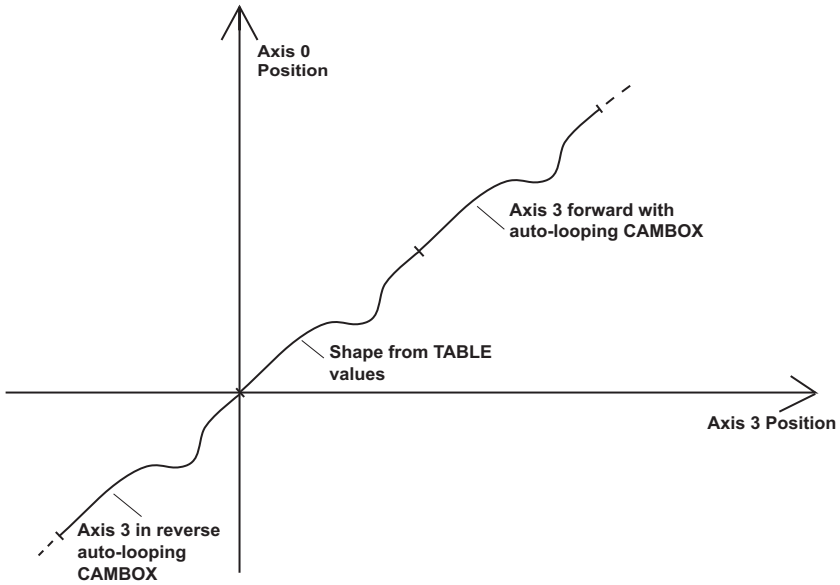
0	The start of the profile shape in the **TABLE**
30	The end of the profile shape in the **TABLE**

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

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.

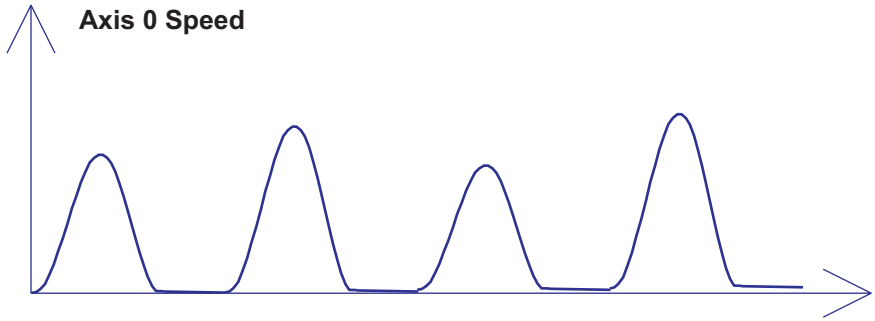
```
CAMBOX(1000,1100,45.68,360,3,4) AXIS(0)
REP_OPTION=2'    cancel repeating mode.
```

Note: The system software resets bit 1 of **REP_OPTION** to 0 when the mode has been cancelled.



Example 4: CAMBOX Pattern Mode

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



The parameters for this mode are treated differently to the standard **CAMBOX** function

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

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

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

SHAPE BLOCK This is directly pointed to by the **CAMBOX** command as in any **CAMBOX**.

CONTROL BLOCK This is pointed to by the third **CAMBOX** parameter in this options mode only. It is of fixed length (7 table values). It is important to note that the control block is modified during the **CAMBOX** operation. It must therefore be re-initialised prior to each use.

PATTERN BLOCK The start and end of this are pointed to by 2 of the CONTROL BLOCK values. The pattern sequence is a sequence of scale factors for the SHAPE.

Control Block Parameters

		R/W	Description
0	CURRENT POSITION	R	The current position within the TABLE of the pattern sequence. This value should be initialised to the START PATTERN number.
1	FORCE POSITION	R/W	Normally this value is -1. If at the end of a SHAPE the user program has written a value into this TABLE position the pattern will continue at this position. The system software will then write -1 into this position. The value written should be inside the pattern such that the value: CB(2)<=CB(1)<=CB(3)
2	START PATTERN	R	The position in the **TABLE** of the first pattern value.
3	END PATTERN	R	The position in the **TABLE** of the final pattern value
4	REPEAT POSITION	R/W	The current pattern repeat number. Initialise this number to 0. The number will increment when the pattern repeats if the link axis motion is in a positive direction. The number will decrement when the pattern repeats if the link axis motion is in a negative direction. Note that the counter runs starting at zero: 0,1,2,3…
5	REPEAT COUNT	R/W	Required number of pattern repeats. If -1 the pattern repeats endlessly. The number should be positive. When the ABSOLUTE value of CB(4) reaches CB(5) the **CAMBOX** finishes if CB(6)=-1. The value can be set to 0 to terminate the **CAMBOX** at the end of the current pattern. See note below on REPEAT COUNT in the case of negative motion on the link axis.
6	NEXT CONTROL BLOCK	R/W	If set to -1 the pattern will finish when the required number of repeats are done. Alternatively a new control block pointer can be used to point to a further control block.

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

Example: A machine cycles an initialisation shape cycle 1000 times prior to running a pattern continuously until requested to stop at the end of the pattern.

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

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

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

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

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

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

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

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

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

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

Note: Negative motion on link axis:

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

CANCEL

Type: Motion Command

Alternate Format: **CA**

Syntax: **CANCEL / CANCEL(1)**

Description: Cancels a move on an axis or an interpolating axis group. 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 terminated immediately.

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.

See also: **RAPIDSTOP.**

Example:
```
FORWARD
WA(10000)
CANCEL' stop movement 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 !
```

CONNECT

Type: **Axis Command**

Syntax: **CONNECT(ratio , driving axis)**

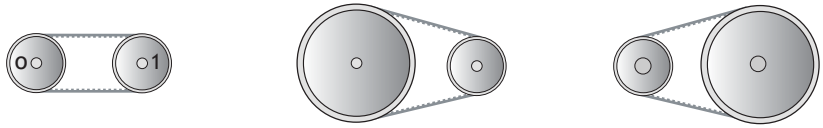
Alternate Format: **CO(ratio, driving axis)**

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

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.



CONNECT(1,1) **CONNECT(2,1)** **CONNECT(0.5,1)**

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)
```

Note 2: To achieve an exact connection of fractional ratio's of values such as 1024/3072. The MOVELINK command can be used with the continuous repeat link option set on.

DATUM

Type: Command

Syntax: **DATUM(sequence no)**

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:

Seq.	Description
0	The current measured position is set as demand position (this is especially useful on stepper axes with position verification). DATUM(0) also clears the following error exceeded FE_LIMIT condition in the **AXISSTATUS** register for ALL axes. It sets these bits in **AXISSTATUS** to zero: BIT 1 Following Error Warning BIT 2 Rmote Drive Comms Error BIT 3 Remote Drive Error BIT 8 Following Error Limit Exceeded BIT 11 Cancelling Move
1	The axis moves at creep speed forward till the Z marker is encountered. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
2	The axis moves at creep speed in reverse till the Z marker is encountered. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
3	The axis moves at the programmed speed forward until the datum switch is reached. The axis then moves backwards at creep speed until the datum switch is reset. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
4	The axis moves at the programmed speed reverse until the datum switch is reached. The axis then moves at creep speed forward until the datum switch is reset. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.

Seq.	Description
5	The axis moves at programmed speed forward until the datum switch is reached. The axis then reverses at creep speed until the datum switch is reset. It then continues in reverse looking for the Z marker on the motor. The demand position where the Z input was seen is then set to zero and the measured position corrected so as to maintain the following error.
6	The axis moves at programmed speed reverse until the datum switch is reached. The axis then moves forward at creep speed until the datum switch is reset. It then continues forward looking for the Z marker on the motor. The demand position where the Z input was seen is then set to zero and the measured position corrected so as to maintain the following error.

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 **FHOLD** inputs which are designed to be "fail-safe".

Example:
```
DATUM_IN=10
DATUM(5)
```

DEC

Type: Axis Command

Syntax: **DEC(rate)**

Description: Sets the deceleration rate for an axis. The **DEC()** command is provided to maintain compatibility with older controllers. The Axis Parameter **DECEL** provides the same functionality and is preferred.

Parameters: **rate:** The units of the parameter are dependant on the unit conversion factor. The deceleration factor is entered in UNITS/SEC/SEC.

Note: **ACC()** sets both the acceleration and deceleration rates. **DEC()** sets only the deceleration rate.

DEFPOS

Type: Motion Command.

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

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

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:

pos1: Absolute position to set on current base axis in user units.

pos2: Abs. position to set on the next axis in **BASE** array in user units.

pos3: Abs. position to set on the next axis in **BASE** array in user units.

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 user program 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 provide an easy way to avoid the problem described:

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

FORWARD

Type: Axis 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
```

MATCH

Type: Axis Command

Syntax: `MATCH(count, table address)`

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.

Parameters:

`count`	Number of Transitions to include in the pattern match.
`table address`	Address where pattern to use for comparison has been recorded.

See also: `REGIST` and `RECORD`

Example:
```
dec_dist=SPEED*SPEED*0.5/DECEL
length=10
REP_DIST=100*length
DEFPOS(0)
REGIST(5,length)
MOVE(2*length)
WAIT UNTIL MARK
IF TRANSITIONS>4 AND TRANSITIONS<12 THEN
  MATCH(8,10)
  IF REG_MATCH>0.8 THEN
```

```
    IF REG_POS<(MPOS+dec_dist-length) THEN
      MOVEMODIFY(2*length+REG_POS)
    ELSE
      MOVEMODIFY(length+REG_POS)
    ENDIF
  ELSE
    PRINT "marks give too poor fit"
  ENDIF
ELSE
  PRINT "marks not seen"
ENDIF
WAIT IDLE
PRINT REG_POS,ENDMOVE,TRANSITIONS,REG_MATCH
```

MHELICAL

Type: Motion Command.

Syntax: **MHELICAL(end1, end2, centre1,centre2, direction, distance 3)**

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:

end1:	position on **BASE** axis to finish at.
end2:	position on next axis in **BASE** array to finish at.
centre1:	position on **BASE** axis about which to move.
centre2:	position on next axis in **BASE** array about which to move.
direction:	The “direction” is a software switch which determines whether the arc is interpolated in a clockwise or anti- clockwise direction. The parameter is set to 0 or 1. See **MOVECIRC**.
distance3:	The distance to move on the third axis in the **BASE** array axis in user units

MOVE

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

Type: Motion 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 3 axes on MC202, 8 axes on MC206 or Euro205, and 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.]

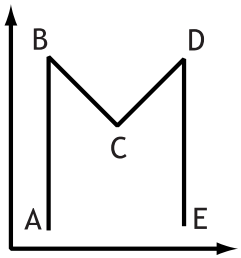
The number of parameters can increase to the number of axes on the controller

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

Type: Motion Command.

Syntax: **MOVEABS(1 pos [, 2 pos , 3 pos[, 4 pos...]]])**

Alternate Format: **MA()**

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

Parameters:

1 pos: position to move to on base axis.

2 pos: position to move to on next axis in BASE array.

3 pos: position to move to on next axis in BASE array.

n pos: position to move to on next axis in BASE array

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

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:

```
FOR x=0 TO 5
  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

Syntax: `MOVECIRC(finish1, finish2, centre1, centre2, direction)`

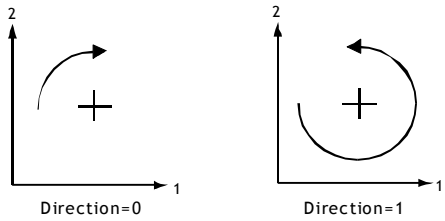
Type: Motion Command.

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.

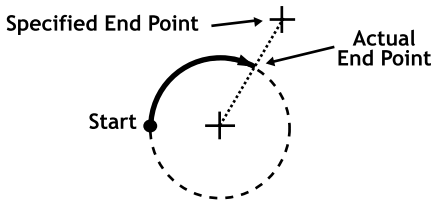
Parameters:

`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.



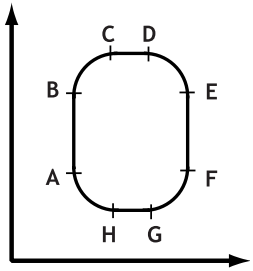
Note: In order for the **MOVECIRC()** command to be correctly executed, the two axes generating the circular arc must have the same number of encoder pulses/linear axis distance. If this is not the case it is possible to adjust the encoder scales in many cases by adjusting with **PP_STEP**.

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(O,-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

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

Type: Motion 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 (**MPOS**) of another axis.

Parameters:

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

link dec: positive incremental distance in user units on the input axis over which the base axis decelerates.

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.

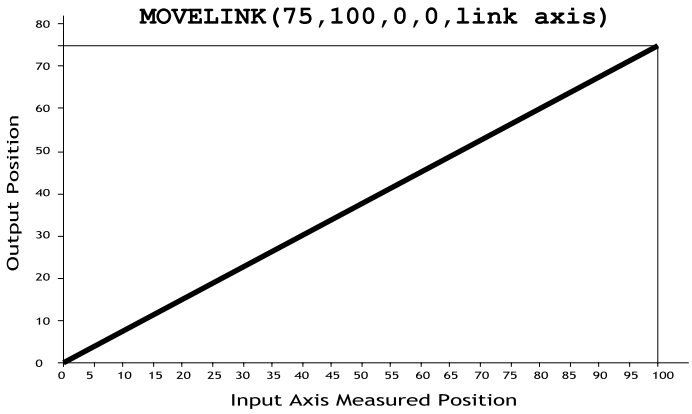
link axis: Specifies the axis to "link" to. It should be set to 0...number of available axes

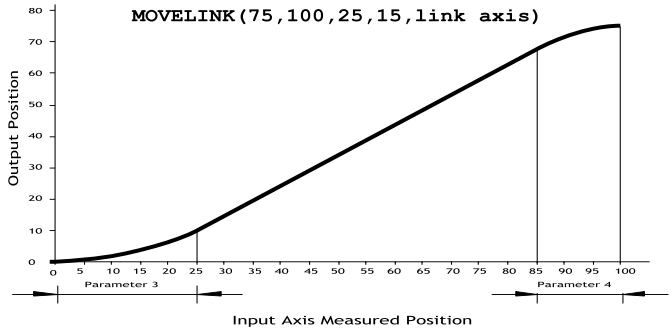
link options:
- **1** link commences exactly when registration event occurs on link axis
- **2** link commences at an absolute position on link axis (see **link pos** parameter)
- **4** **MOVELINK** repeats automatically and bi-directional when this bit is set. (This mode can be cleared by setting bit 1 of the **REP_OPTION** axis parameter)

link pos: This parameter is the absolute position where the **MOVELINK** 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 2: Parameters 6 and 7 are optional.





Example 2: 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 decel
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 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)' 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)' all constant speed
```

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

```
MOVELINK(0.4,0.8,0,0.8,1)' all decel
```

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

```
MOVELINK(1,1.8,0.8,0.8,1)' Same as 3 moves above
```

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

```
MOVELINK(0.4,0.8,0.8,0,1)
MOVELINK(0.6,1.0,0,0.8,1)
```

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

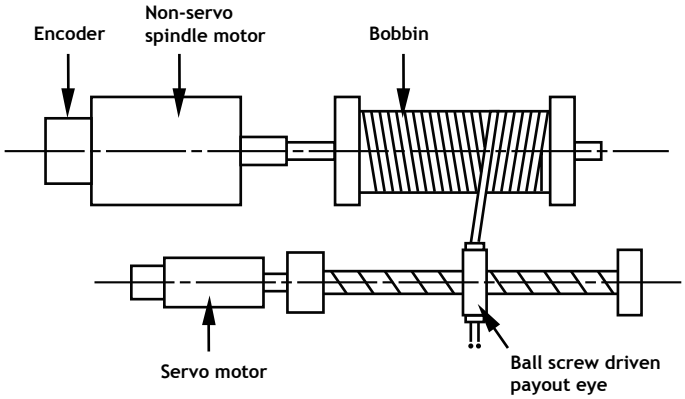
Example 3: Exact Ratio Gearbox

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

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

Example 4: Coil Winding

In this example the unit conversion factors **UNITS** are set so that the payout movements are in mm and the spindle position is measured in revolutions. The payout eye therefore moves 50mm over 25 revolutions of the spindle with the command **MOVELINK(50,25)**. If it were desired to accelerate up over the first spindle revolution and decelerate over the final 3 the command would be **MOVELINK(50,25,1,3)**. **MOVELINK** and **CAMBOX** can be programmed to commence automatically relative to an absolute position on the link axis.




```
' Trio BASIC Coil Winding Example Program:
OP(motor,ON)' - Switch spindle motor on
FOR turn=1 TO 10
  MOVELINK(50,25,0,0,1)
  MOVELINK(-50,25,0,0,1)
NEXT turn
WAIT IDLE
OP(motor,OFF)
```

MOVEMODIFY

Type: Axis Command.

Syntax: **MOVEMODIFY(absolute position)**

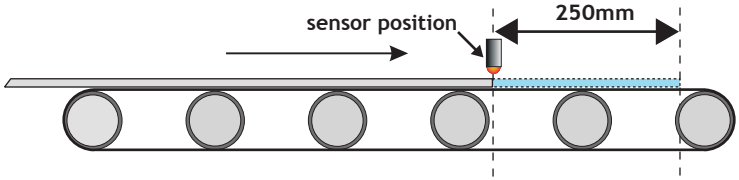
Alternate Format: **MM()**

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**.

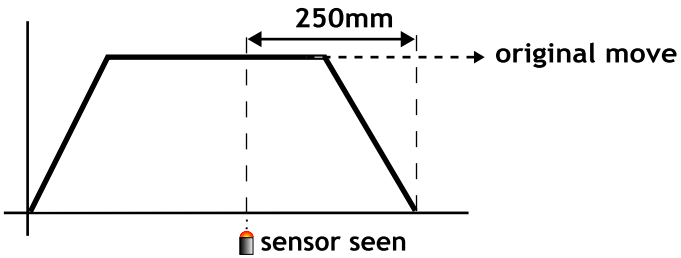
See also: **ENDMOVE**

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)' Start 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
```



REGIST

Type: Axis Command

Syntax: **REGIST(mode,{distance})**

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 within the specified window the **MARK** parameter is set **TRUE** and the position is stored in **REG_POS**. On the MC206 built-in axes 2 registration registers are provided for each axis. This allows 2 registration sources to be captured simultane-

ously and their difference in position determined. To use this dual registration mode the REGIST commands "mode" parameter is set in the range 6..9. Two additional axis parameters REG_POSB and MARKB hold the results of the Z mark registration in this mode.

Parameters: **mode:** Determines the position to capture:

1 - Absolute position when Z Mark Rising

2 - Absolute position when Z Mark Falling

3 - Absolute position when Registration Input Rising

4 - Absolute position when Registration Input Falling

5 - Sets pattern recognition mode

6 - R Input Rising into REG_POS and Z Mark Rising into REG_POSB.

7 - R Input Rising into REG_POS and Z Mark Falling into REG_POSB.

8 - R Input Falling into REG_POS and Z Mark Rising into REG_POSB.

9 - R Input Falling into REG_POS and Z Mark Falling into REG_POSB

(mode = 6 to 9 are only available on MC206 built-in axes)

distance: The distance parameter is used for the pattern recognition mode ONLY, and specifies the distance over which to record transitions

Note: **Windowing Functions**

Add **256** to the above mode 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:
```
REGIST(3+256)
WAIT UNTIL MARK
PRINT "Registration Input Seen at:";REG_POS
```

RAPIDSTOP

Type: Motion 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.

REVERSE

Type: Axis 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
```

Input / Output Commands

AIN

Type: Function

Syntax: **AIN(analog chan)**

Description Up to 4 analog input modules (P325) may be connected on the *Motion Coordinator*'s built in CAN bus port. Each P325 has 8 channels of +/-10v analog inputs which return 2047..-2048. The values from each P325 channel are updated every 10msec. On controllers with a built-in analog channel such as the MC206 channel 0 from the CAN bus is replaced by the built-in channel 0.

Parameters: **analog chan**: analog input channel number 0.31

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: Note that the analog input value is checked to ensure it is above zero even though it always should be positive. This is to allow for any noise on the incoming signal which could make the value negative and cause an error because a negative speed is not valid for any move type except **FORWARD** or **REVERSE**.

AIN0..7 / AINBI0..7

Type: System Parameter

Description: These system parameters duplicate the AIN() command.

They provide the value of the analog input channels in system parameter format to allow the **SCOPE** function (Which can only store parameters) to read the analog inputs.

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.

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.

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	45	47	48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79

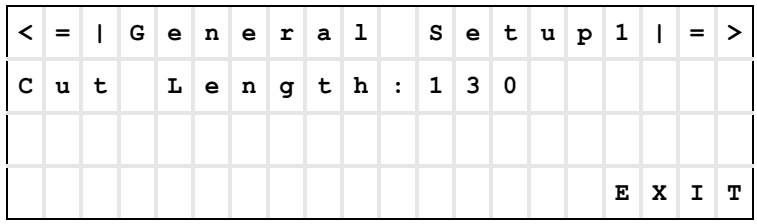
4 Line Display as featured on the Membrane Keypad

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

Example2: The following code

```
PRINT #kpd,CHR(12);CHR(14);CHR(20);
PRINT #kpd,CURSOR(00);"<=|General Setup1|=>";
PRINT #kpd,CURSOR(20);"Cut Length : ";VR(50+cut_length)[0];
```

would produce the following display:



CHR

Type: Command

Description: The **CHR(x)** command is used to send individual ASCII characters which are referred to by number. **PRINT CHR(x);** is equivalent to PUT(x) in some other version of BASIC.

Example:
```
>>PRINT CHR(65);
A
```

DEFKEY

Type: Command

Description: Under most circumstances this command is not required and it is recommended that the values of keys are input 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.

FLAG

Type: Command/Function

Syntax: **FLAG(flag no [,value])**

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

Type: Command/Function

Syntax: **FLAGS([value])**

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

Example: Set Flags 1,4 and 7 ON, all others OFF

Bit #	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1

```
FLAGS(146)' 2 + 16 + 128
```

Example 2: Test if FLAG 3 is set.

```
IF (FLAGS and 8) <>0 then GOSUB somewhere
```

GET

Type: 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 user program will wait until a character is available.

Example:
```
GET  k
```

GET#

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:
- 0 Serial port 0
- 1 Serial port 1
- 2 Serial port 2
- 3 Fibre optic port (value returned defined by **DEFKEY**)
- 4 Fibre optic port (returns raw keycode of key pressed)
- 5 *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.

Example 2: Get a key in a user menu routine

```
g_edit:
    PRINT #kpd,CHR(12);CHR(14);CHR(20);
    PRINT #kpd,CURSOR(00);"<=|General Setup1|=>";
    PRINT #kpd,CURSOR(20);"Cut Length : ";VR(clength)
    GET #kpd,option
    IF option=lastmenu OR option=f1 THEN RETURN
    IF option=menu_l2 THEN GOSUB set_cut_length
GOTO g_edit
```

HEX

Type: Command

Description: The **HEX** command is used in a print statement to output a number in hexadecimal format. The HEX command is available on MC206/MC224 only.

Example:
```
PRINT#5,HEX(IN(8,16))
```

IN()/IN

Type: Function.

Syntax: **IN(input no<,final input>)/IN**

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
  GOSUB place
```

Example 2: Move to the distance set on a thumb wheel multiplied by a factor. The thumb wheel is connected to inputs 4,5,6,7 and gives output in 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: absolute MOVE by this distance

Note: IN is equivalent to IN(0,23)

Example: Test if either input 2 or 3 is ON.

```
If (IN and 12) <> 0 THEN GOTO start
' (Bit 2 = 4 + Bit 3 = 8) so mask = 12
```

INPUT

Type: 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)
```

This will display 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 Parameter

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. Reading the inputs using these system parameters is not normally required. The IN(x,y) command should be used instead. They are made available in this format to make the input channels available accessible to the **SCOPE** command which can only store parameters.

INVERT_IN

Type: Command.

Syntax: **INVERT_IN(input,on/off)**

Description: The INVERT_IN command allows the input channels 0..31 to be individually inverted in software. This is important as these input channels can be assigned to activate functions such as feedhold. The INVERT_IN function sets the inversion for one channel ON or OFF. It can only be applied to inputs 0..31.

Example1:
```
>>? IN(3)
0.0000
>>INVERT_IN(3,ON)
>>? IN(3)
1.0000
>>
```

KEY

Type: Function.

Description: Returns **TRUE** or **FALSE** depending on whether a character has been received on an input device or not. This command does not read the character but allows the program to test if any character has arrived. A true result will be reset when the character is read with **GET**.

The **KEY** command checks the channel specified by **INDEVICE** or by a # channel number.

Input device:

Chan	Input device:-
0	Serial port 0
1	Serial port 1
2	Serial Port 2
3	Fibre optic port (value returned defined by DEFKEY)
4	Fibre optic port (returns raw keycode of key pressed)
5	*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 1:
```
main:
  IF KEY#1 THEN GOSUB read
...
read:
  GET#1  k
RETURN
```

Example 2: To test for a character received from the fibre optic network:

```
IF KEY#4 THEN GET#4,ch
```

LINPUT

Type: Command

Syntax: **LINPUT variable**

Description: Waits for an input string and stores the ASCII values of the string in an array of variables starting at a specified numbered variable. The string must be terminated with a carriage return <CR> which is also stored. The string is not echoed by the controller.

Parameters: None.

Example: **LINPUT VR(0)**

Now entering: **START<CR>** will give:

VR(0)	83	ASCII'S'
VR(1)	84	ASCII 'T'
VR(2)	65	ASCII 'A'
VR(3)	82	ASCII 'R'
VR(4)	84	ASCII 'T'
VR(5)	13	ASCII carriage return

OP

Type: Command/Function.

Syntax: **OP[([output no,] value)]**

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 *Motion Coordinator* 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 0 to7 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)
```

PRINT

Type: Command.

Description: The **PRINT** command allows the Trio 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"
```
Suppose VR(1)=6 and variab=1.5: print output will be:

Example 2:
```
>>PRINT 123.45,VR(1)
123.4500 1.5000
>>
```
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.

Example 3:
```
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 4:
```
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 5:

```
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 line feed 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 versions of BASIC.

Note: The **PRINT** statements are normally transmitted to serial port 0. They can be redirected to other output ports by using **PRINT#**.

PRINT#

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	Serial port 0
1	Serial port 1
2	Serial port 2
3	Fibre optic port
4	Fibre optic port duplicate
5	RS-232 port A - channel 5
6	RS-232 port A - channel 6

n:	Output device:-
7	RS-232 port A - channel 7
8	RS-232 port A - channel 8 - reserved for use by *Motion* Perfect
9	RS-232 port A - channel 9 - reserved for use by *Motion* Perfect
10..24	send text string to fibre optic network node 1..15

Example: `PRINT#10,"SPEED=";SPEED[6,1];`

PSWITCH

Type: Command

Syntax: `PSWITCH(sw,en,[,axis,opno,opst,setpos,rspos])`

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 16 (8 on MC202) position switches each of which can be assigned to any axis, and can be assigned ON/OFF positions and OUTPUT numbers.

Multiple PSWITCH's can be assigned to a single output. The result on the output will be the OR of the position switches and the standard BASIC OP setting.

The command must be used with all 7 parameters to enable a switch, just the first 2 parameters are required to disable a switch.

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 `PSWITCH`
3 to enable hardware `PSWITCH`
(hardware PSWITCH can only be used with a P242 daughter board

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 Trio 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)
```

READPACKET

Type: Command

Syntax: **READPACKET(port#,vr#,vr count, format)**

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. A detailed description of the READPACKET format can be downloaded from WWW.TRIOMOTION.COM

Parameters:

Port Number	This value should be 0 or 1
VR Number	This value tells the *Motion Coordinator* where to start setting the variables in the **VR()** global memory array.
VR count.	The number of variables to download
Format	The number format for the numbers being downloaded

RECORD

Type: Command

Syntax: **RECORD(count, table address)**

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:

1 - Reduce the number of transitions to a number defined by the programmer
2 - Store the transition pattern for subsequent comparison with **MATCH**

Parameters:

count	Number of transitions to record. The actual transitions seen may be greater than this number but the shortest ones are removed so that only the programmed transition count is stored.
table address	This value tells the *Motion Coordinator* where to store the pattern information in the global **TABLE** memory. Table used is address to address+24.

Note: See the **MATCH** command for an example of a complete recognition sequence.

SEND

Type: Command

Syntax: **SEND(n,type,data1[,data2])**

Description: Outputs a fibre-optic network message of a specified type to a given node.

Parameters:

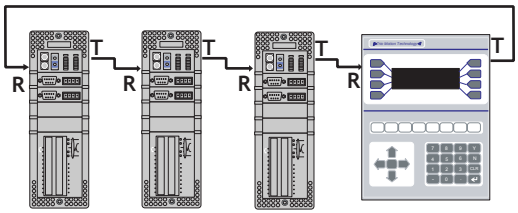
n:	Number from 10 to 24 defining the destination node.
type:	Message type: 1 - Direct variable transfer 2 - Keypad offset
data1:	If message type 1, data1 is the numbered variable number, 0..250, on the destination *Motion Coordinator* to modify. If message type 2; data1 is the number of nodes from the 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 Coordinator*s 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 Coordinator*s and one membrane keypad connected as follows:

MCa ---> MCb---> MCc ---> Keypad ---> (back to MCa)



	MCa	MCb	MCc	Keypad
Offset from MCa	0	10	11	12
Offset from Keypad	10	11	12	0

If MCa is to initialise the keypad (offset of 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)
```

SETCOM

Type: Command

Syntax: **SETCOM(baudrate,databits,stopbits,parity,port,mode)**

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:	0,1 or 2
mode:	This switch is available on serial ports #1 and #2 ONLY. 0=XON/XOFF inactive,1=XON/XOFF active,4=MODBUS protocol
Note:	On power up the controllers always set the serial ports to 9600,7,2,even, XON/XOFF active.

Example 1:

```
' Set port 1 to 19200 baud, 7 data bits, 2 stop bits
' even parity and XON/XOFF enabled

SETCOM(9600,7,2,2,1,1)
```

Example 2: The Modbus protocol is initialised by setting the mode parameter of the **SETCOM** instruction to 4. The **ADDRESS** parameter must also be set *before* the Modbus protocol is activated.

```
' set up RS485 port at 19200 baud, 8 data, 1 stop, even parity
' and enable the MODBUS comms protocol

ADDRESS=1
SETCOM(19200,8,1,2,2,4)
```

Program Loops and Structures

BASICERROR

Type: Program Structure

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 Trio BASIC command.

Example:
```
ON BASICERROR GOTO error_routine
....(rest of program)

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

ELSE

Type: Program Structure

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

See Also `IF, THEN, ENDIF`

ELSEIF

Type: Program Structure

Syntax:
```
IF  <condition1> THEN
  commands
ELSEIF <condition2> THEN
  commands
ELSE
  commands
ENDIF
```

Description: The command is used within an **IF .. THEN .. ENDIF**. It evaluates a second (or subsequent) condition and if TRUE it executes the commands specified, otherwise the commands are skipped. MC206 and MC224 only.

Parameters: **condition(s):** Any logical expression.

commands: Any valid Trio BASIC commands including further **IF..THEN ..{ELSEIF}..{ELSE} ENDIF** sequences

Example 1:

```
IF IN(stop)=ON THEN
  OP(8,ON)
  VR(cycle_flag)=0
ELSEIF IN(start_cycle)=ON THEN
  VR(cycle_flag)=1
ELSEIF IN(step1)=ON THEN
  VR(cycle_flag)=99
ENDIF
```

Example 2:

```
IF key_char=$31 THEN
  GOSUB char_1
ELSEIF key_char=$32 THEN
  GOSUB char_2
ELSEIF key_char=$33 THEN
  GOSUB char_3
ELSE
  PRINT "Character unknown"
ENDIF
```

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

See Also **IF, THEN, ELSE, ENDIF**

ENDIF

Type: Program Structure

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

Example:
```
IF count >= batchsize THEN
  PRINT #3,CURSOR(20);"   BATCH COMPLETE   ";
  GOSUB index ' Index conveyor to clear batch
  count=0
ENDIF
```

See Also **IF, THEN, ELSE**

FOR..TO.. STEP..NEXT

Type: Program Structure

Syntax:
```
FOR variable=start TO end [STEP increment]
  ...
  block of commands
  ...
NEXT variable
```

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**. The **STEP** parameter is optional. If not 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 program drops out of the **FOR..NEXT** loop.

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. (Optional)

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

Note: **FOR..NEXT** loops can be nested up to 8 deep in each program.

GOSUB

Type: Program Structure

Syntax: **GOSUB label**

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 routine1
  GOSUB routine2
GOTO main

routine1:
  PRINT "Measured Position=";MPOS;CHR(13);
RETURN

routine2:
  PRINT "Demand Position=";DPOS;CHR(13);
RETURN
```

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

GOTO

Type: Program Structure

Syntax: `GOTO label`

Description: Identifies the next line of the program to be executed.

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:
```
loop:
  PRINT "Measured Position=";MPOS;CHR(13);
  WA(1000)
  GOTO loop
```

Note: Labels may be character strings of any length. Only the first 15 characters are significant. Alternatively line numbers may be used as labels.

IDLE

Type: Command Modifier

Description: Only used in conjunction with the **WAIT** command, **WAIT IDLE** 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 Complete"
```

IF..THEN..ELSE.. ENDIF

Type: Program Structure

Syntax:
```
IF  <condition> THEN
  commands
ELSE
  commands
ENDIF
```

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 ex_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 controller 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.

NEXT

Type: Program Structure

Description: Used to mark the end of a **FOR..NEXT** loop. See **FOR**.

ON.. GOSUB

Type: Program Structure

Syntax: **ON expression GOSUB label[,label[,...]]**

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.. GOTO

Type: Program Structure

Syntax: **ON expression GOSUB label[,label[,...]]**

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
```

REPEAT.. UNTIL

Type: Program Structure

Syntax: **REPEAT commands UNTIL condition**

Description: The **REPEAT..UNTIL** construct allows a block of commands to be continuously repeated until a condition becomes **TRUE**. **REPEAT..UNTIL** loops can be nested without limit.

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
```

RETURN

Type: Program Structure

Description: Instructs the program 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
```

STEP

Type: Program Structure

Description: This optional parameter specifies a step size in a **FOR**..**NEXT** sequence. See **FOR**.

Example:
```
FOR x=10 TO 100 STEP 10
   MOVEABS(x) AXIS(9)
NEXT x
```

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
```

THEN

Type: Program Structure

Description: Forms part of an **IF** expression. See IF for further information.

Example:
```
IF MARK THEN
    offset=REG_POS
ELSE
    offset=0
ENDIF
```

Note: Comments should not be placed after a **THEN** statement

TO

Type: Program Structure

Description: Precedes the end value of a **FOR**..**NEXT** loop.

Example:
```
FOR x=10 TO 0 STEP -1
```

UNTIL

Type: Program Structure

Description: Defines the end of a **REPEAT**..**UNTIL** multi-line loop, or part of a **WAIT UNTIL** structure. After the **UNTIL** statement is a condition which decides if program flow continues on the next line or at the **REPEAT** statement. **REPEAT**..**UNTIL** loops can be nested without limit.

Example:
```
' This loop loads a CAMBOX move each time Input 0 comes on.
' It continues until Input 6 is switched OFF.
```

```
REPEAT
  WAIT UNTIL IN(0)=OFF
  WAIT UNTIL IN(0)=ON
  CAMBOX(0,150,1,10000,1)
UNTIL IN(6)=OFF
```

WA

Type: Command

Syntax: `WA(delay time)`

Description: Holds up program execution for the number of milliseconds specified in the parameter.

Parameters: `time:` The number of milliseconds to wait for.

Example:
```
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

Type: Command

Syntax: **WAIT UNTIL** condition

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.

WHILE

Type: Program Structure

Syntax: **WHILE condition**

Description: The commands contained in the **WHILE..WEND** loop are continuously executed until the condition becomes **FALSE**.

Execution then continues after the **WEND**.

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
```

WEND

Type: Program Structure

Description: Marks the end of a **WHILE..WEND** loop.

See also: **WHILE**

Note: **WHILE..WEND** loop can be nested without limit other than program size.

System Parameters and Commands

ADDRESS

Type: System Parameter

Syntax: **ADDRESS=value**

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

Example: **ADDRESS=5**

APPENDPROG

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

Syntax: **APPENDPROG <string>**

Alternate Format: **@ <string>**

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

AUTORUN

Type: System Command

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

See Also: **RUNTYPE.**

AXISVALUES

Type: System Command

Syntax: **AXISVALUES(axis,bank)**

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

Parameters **The data is given in the format:**

<Parameter><type>=<value>

<Parameter> is the name of the parameter

<type> is the type of the value.

- **i** integer
- **f** float
- **c** 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

CAN

Type: System Function

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

Description: This function allows the CAN communication channels to be controlled from the Trio BASIC programming system. All *Motion Coordinator*'s have a single built-in CAN channel which is normally used for digital and analog I/O using Trio's I/O modules. With up to 4 CAN daughter boards plus the built-in CAN channel the units can control a maximum of 5 CAN channels:

Channel:	Channel Number:	Maximum Baudrate:
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 built-in CAN channel will automatically scan for Trio I/O modules if the system parameter CANIO_ADDRESS is set to its default value of 32.

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 10 separate functions:

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

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

Function #:
There are 10 CAN functions 0..9:

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 msg 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..)**

8	Read CanOpen Object	**CAN(channel#,8,transbuf,recbuf,object, subindex,variable#)**
9	Write CanOpen Object	**CAN(channel#,9,transbuf,recbuf,format, object,subindex,value,{valuems})**

Notes: **register#** is the 81C91 register number.

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 hold the data bytes from the can packet.

Functions 8 and 9 are only available on the MC206 and MC224.

CANIO_ADDRESS

Type: System Parameter (Stored in FLASH Eprom)

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 eprom in the controller and for most systems does not need to be set from the default value of 32. The value may be changed to a different value in the range 33..47 but in this case the *Motion Coordinator* will not connect to CAN-I/O modules following reset. The value of CANIO_ADDRESS should be changed from 32 if it is required to use the built-in CAN channel for functions other than controlling Trio I/O modules.

CANIO_ENABLE

Type: System Parameter

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 Trio BASIC using the **CAN** command. The system software will set CANIO_ENABLE to **ON** on power up if the **CANIO_ADDRESS** is set to 32 and any P315 CAN I/O or P325 CAN analog modules have been detected, otherwise it will be set to **OFF**.

CANIO_STATUS

Type: System Parameter

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 programs in battery backed RAM. On power up the checksum is recalculated and compared with the previously held value. If the checksum is incorrect the programs will not run.

CLEAR

Type: System Command

Description Sets all global (numbered) variables to 0 and sets local variables on the process on which command is run to 0.

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 12.345 62.0000
0.0000 0.0000 0.0000
```

CLEAR_PARAMS

Type: System Command

Description Clears all variables and parameters stored in flash eprom to their default values. On the MC202 CLEAR_PARAMS will erase all the VR's stored using FLASHVR. CLEAR_PARAMS cannot be performed if the controller is locked.

COMMSERROR

Type: System Parameter

Description: This parameter returns all the communications errors that have occurred since the last time that it was initialised. It is a bitwise value defined as follows:

Bit	Value
0	RX Buffer overrun on Network channel
1	Re-transmit buffer overrun on Network channel
2	RX structure error on Network channel
3	TX structure error on Network channel
4	Port 0 Rx data ready
5	Port 0 Rx Overrun
6	Port 0 Parity Error
7	Port 0 Rx Frame Error
8	Port 1 Rx data ready (MC202 port 1 Parity Error)
9	Port 1 Rx Overrun (MC202 port 1 Rx Frame Error)
10	Port 1 Parity Error (MC202 port 1 Rx Overrun)
11	Port 1 Rx Frame Error (MC202 - no function)
12	Port 2 Rx data ready
13	Port 2 Rx Overrun
14	Port 2 Parity Error
15	Port 2 Rx Frame Error
16	Error FO Network port
17	Error FO Network port
18	Error FO Network port
19	Error FO Network port

COMMSTYPE

Type: Slot Parameter

Syntax: **COMMSTYPE SLOT(slot#)**

Description: This parameter returns the type of communications daughter board in a controller slot. On the MC206, a communications daughter board will respond with its type if the COMMSTYPE is requested from slot(0).

See Appendix 1 for complete list of values returned by communications boards.

COMPILE

Type: System Command

Description: Forces compilation (to intermediate code) of the currently selected program. Program compilation is performed automatically by the system software prior to program **RUN** or when another program is **SELECT**ed. This command is not therefore normally required.

CONTROL

Type: System Parameter (Read Only)

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

Controller	**CONTROL**
MC202:	202
Euro205	205
Euro205X	255
MC206	206
PCI208	208
MC216:	216
MC224:	224

Note: **When the Motion Coordinator is LOCKED, 1000 is added to the above numbers. eg a locked MC206 will return 1206.**

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.

DATE

Type: System Parameter (MC216 Only)

Description: Returns/Sets the current date held by the MC216's real time clock. The number may be entered in DD:MM:YY or DD:MM:YYYY format.

Example 1:
```
>>DATE=20:10:98
or
>>DATE=20:10:2001
```

Example2:
```
>>PRINT DATE
36956
```

This prints the number representing the current day. This number is the number of days since 1st January 1900, with 1 Jan. 1900 as 1. Trio has issued a year 2000 compliance statement which describes the year 2000 issue in relation to all Trio products.

DATE$

Type: Command (MC216 Only)

Description: Prints the current date DD/MM/YY as a string to the port. A 2 digit year description is given.

Example:
```
PRINT #3,DATE$
```

This will print the date in format for example: 20/10/01

DAY

Type: System Parameter (MC216 only)

Description: Returns the current day as a number 0..6, Sunday is 0. The **DAY** can be set by assignment.

Example:
```
>>DAY=3
>>? DAY
3.0000
>>
```

DAY$

Type: System Command (MC216 only)

Description: Prints the current day as a string.

Example:
```
>>? DAY$
3.0000
>>
```

DEL

Type: System Command

Alternate Format: **RM**

Syntax: **DEL progname**

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"
```

DEVICENET

Type: System Command

Syntax: **DEVICENET(slot,func,baud,mac id,poll base,poll inlen,poll outlen)**

Description: The command DEVICENET is used to start and stop the DeviceNet slave function which is built into the *Motion Coordinator*. Once the DEVICENET command is running the process can be used to execute other BASIC commands in the usual way. This command is available on MC206 and MC224 only.

Parameters:

slot:	Specifies the communications slot where the CAN daughter baord is placed. Set -1 for built-in CAN port and 0 for a CAN daughter board in the MC206.
func:	0 = Start the DeviceNet slave protocol on the given slot. 1 = Stop the DeviceNet protocol.
baud:	Set to 125, 250 or 500 to specify the baudrate in kHz.
mac id:	The ID which the *Motion Coordinator* will use to identify itself on the DeviceNet network. Range 0..63.
poll base:	The first TABLE location to be transfered as poll data
poll in len:	Number of words to be received during poll. Range 0..4
poll out len:	Number of words to be sent during poll. Range 0..4

Polled IO data is transferred periodically:
From PLC to [TABLE(poll_base) -> TABLE(poll_base + poll_in_len)]
To PLC from [TABLE(poll_base + poll_in_len + 1) -> TABLE(poll_base + poll_in_len + poll_out_len)]

Example 1:
```
' Start the DeviceNet protocol on the built-in CAN port
DEVICENET(-1,0,500,30,0,4,2)
```

Example 2:
```
' Stop the DeviceNet protocol on the CAN board in slot 2
DEVICENET(2,1)
```

DIR

Type: System Command

Alternate Format: **LS**

Description: Prints a list of all programs in memory, their size and their **RUNTYPE**. The alternative format **DIR F** may be used to list the programs stored in the FlashStick if present.

Note: This command should only be used on the *Motion Coordinator* Command Line

DISPLAY

Type: System Parameter

Description: Determines the I/O channels to be displayed on the front panel LEDS.

Certain controllers, such as the Euro205 and MC206 do not have LEDs for every I/O channel. The **DISPLAY** parameter may be used to select which bank of I/O should be displayed.

The parameter default value is 0.

Parameters:

0	Inputs 0-7
1	Inputs 8-15
2	Inputs 16-23
3	Inputs 24-31
4	Outputs 0-7 (unused on existing controllers)
5	Outputs 8-15
6	Outputs 16-23
7	Outputs 24-31
8	DeviceNet Status (MC206 only)

Example:

```
DISPLAY=5
' Show outputs 8-15
```

DLINK

Type: System Command

Syntax: **DLINK(function,…)**

Description: This is a specialised command, to allow access to the SLM™ digital drive interface. During the power sequence, when a SLM™ interface card is found, all the ASICs are initialised, starting the communications protocol.

The axis parameters have to be initialised by the **DLINK** function 2 command before the interface can be used for controlling an external drive.

Parameters:

Function: Specifies the required function.
0 = Read a register on the SLM™ ASIC
1 = Write a register on the SLM™ ASIC
2 = Check for presence SLM module
3 = Check for presence of SLM servo drive
4 = Assign a *Motion Coordinator* axis to a SLM channel
5 = Read an SLM parameter
6 = Write an SLM parameter
7 = Write an SLM command
8 = Read a drive parameter
9 = Returns slot and asic number associated with an axis
10 = Read an EEPROM parameter

Read a register on the SLM™ ASIC.

Parameters:

Function	0
Slot	The communications slot in which the interface daughter board is inserted.
ASIC	The number of the ASIC to be used. Each SLM™ daughter board has 3 ASICs. The master ASIC is 0, the first slave is 1 and the second slave is 2.
Register	The number of the register to be read.

Example:

```
>>PRINT DLINK(0,0,0,3)
117.0000
>>
```

Write a register on the SLM™ ASIC.

Parameters:

Function	1
Slot	The communications slot in which the interface daughter board is inserted.
ASIC	The number of the ASIC to be used.
Register	The number of the register to be written to.
Value	The value to be written.

Example:

```
>>DLINK(1,0,0,1,244)
>>
```

Check for presence SLM module on rear of motor. Returns 1 if the SLM is answering, otherwise it returns 0.

Parameters:

Function	2
Slot	The communications slot in which the interface daughter board is inserted.
ASIC	The number of the ASIC to be used.

```
>>? DLINK(2,0,0)
1.0000
>>
```

Check for presence of SLM servo drive, such as MultiAx. Returns 1 if the drive is answering, otherwise it returns 0. **The current SLM software dictates that the drive MUST be powered up after power is applied to the** ***Motion Coordinator*** **/ SLM.**

Parameters:

Function	3
Slot	The communications slot in which the interface daughter board is inserted.
ASIC	The number of the ASIC to be used.

Example:

```
>>? DLINK(3,0,0)
0.0000
>>
```

Assign a *Motion Coordinator* axis to a SLM channel.

Parameters:

Function	4
Slot	The communications slot in which the interface daughter board is inserted.
ASIC	The number of the ASIC to be used.
Axis	The axis to be associated with this drive. If this axis is already assigned then it will fail. The ATYPE of this axis will be set to 11.

Example:

```
>>DLINK(4,0,0,0)
```

Read an SLM parameter

Parameters:

Function	5
Axis	The axis to be associated with this drive. If this axis is out of range, or is not of the correct type (see function 2) then the function will fail.
Parameter	The number of the SLM parameter to be read. This is normally in the range 0…127. See the drive documentation for further information.

Example:

```
>>PRINT DLINK(5,0,1)
463.0000
>>
```

Write an SLM parameter

Parameters:

Function	6
Axis	The axis to be associated with this drive.
Parameter	The number of the SLM parameter to be read. See Function 4
Value	The value to be set.

Example:

```
>>DLINK(6,0,0,0)
>>
```

Write an SLM command. If command is successful this function returns a TRUE, otherwise it returns FALSE

Parameters:

Function	7
Axis	The axis to be associated with this drive.
Command	The command number. (see drive documentation)

Example:
```
>>PRINT DLINK(7,0,250)
1.0000
>>
```

Read a drive parameter

Parameters:

Function	8
Axis	The axis to be associated with this drive.
Parameter	The number of the drive parameter to be read. This must be in the range 0…127. See the servo drive documentation for further information.

Example:
```
>>PRINT DLINK(8,0,53248)
20504.0000
>>
```

Return slot and asic number associated with an axis

Parameters:

Function	9
Axis	Axis number.
Returns	10 x slot number + ASIC number.

Example:
```
>>PRINT DLINK(9,2)
>>11.0000
```
This example is for slot 1, asic 1

Read an EEPROM parameter

Parameters:

Function	10

Axis	The axis to be associated with this drive/SLM.
Parameter	EEPROM parameter number. (see drive documentation)

Example:
```
>>PRINT DLINK(10,0,29)
>>62128.0000
```
Returns EEPROM parameter 29, the Flux Angle

EDPROG

Type: System Command

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:

&C	Print the name of the currently selected program
&<line>D	Delete line <line> from the currently selected program
&<line>I,<string>	Insert the text <string> in the currently selected program at the line <line>. Note - you should NOT enclose the string in quotes unless they need to be inserted into the program.
&K	Print the checksum of the system software
&<start>,<end>L	Print the lines of the currently selected program between <start> and <end>
&N	Print the number of lines in the currently selected program
&<line>R,<string>	Replace the line <line> in the currently selected program with the text <string>. Note - you should NOT enclose the string in quotes unless they need to be inserted into the program.
&Z,<progname>	Print the CRC checksum of the specified program. This uses the standard CCITT 16 bit generator polynomial

EDIT

Type: System Command

Syntax: **EDIT [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.

Quit Editor	-Control K then D
Delete line	-Control Y
Cursor Control	-Cursor Keys

EPROM

Type: System Command

Description: Stores the Trio BASIC programs in the controller in the FLASH EPROM. This information is be retrieved on power up if the **POWER_UP** parameter has been set to 1. The **EPROM(n)** functions are only usable on an MC206 and MC224..

EPROM or **EPROM(0)**	Stores application programs in ram into on board flash.
EPROM(1)	Stores application programs in ram into FlashStick.
EPROM(2)	Stores application programs in ram into the FlashStick and marks the EPROM request flag so that the programs are copied from the FlashStick into on board flash when the stick is inserted into a controller which is unlocked.
EPROM(3)	Deletes all programs in the FlashStick, leaves data sectors intact.

Note: This command should only be used on the command line. *Motion* Perfect performs the **EPROM** command automatically when the *Motion Coordinator* is set to “Fixed”

See Also: **STICK_WRITE, STICK_READ, DIR**

When using the Memory Stick users should refer to the overview in the MC206 Hardware Overview for a complete description of the Memory Stick functionality.

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
```

ETHERNET

Type: System Command

Syntax:
```
ETHERNET(read/write, slot number, function [,data])
```

Description: The command ETHERNET is used to read and set certain functions of the Ethernet daughter board. The ETHERNET command should be entered on the command line with *Motion* Perfect in "disconnnected" mode via the serial port 0. This command is available on MC206 and MC224 only.

Parameters:

read / write: Specifies the required action.
0 = Read
1 = Write to Flash EPROM
2 = Write to RAM

slot number: The daughter board slot where the Ethernet port has been installed. On the MC206 this is always slot 0.

function: Function number must be one of the following values.

0 = IP Address
1 = Static(1) or dynamic(0) addressing. (Only static addressing is supported.)
2 = Subnet Mask
3 = MAC address
4 = Default Port Number (initialised to 23)
5 = Token Port Number (initialised to 3240)
6 = Ethernet daughter board firmware version (read only)
7 = Modbus TCP mode. Integer (0) or Floating point (1). (R)
8 = Default Gateway
9 = Data configuration. VR() variables (0) or TABLE (1). (R)

data: The optional data is used when changing a parameter value.

When writing to the EPROM on the Ethernet daughter board, the new value will only be used after power has been cycled to the controller. Any data written to RAM (R) is used straight away.

Example 1: Set the IP address for the Ethernet daughter board in slot 0.

```
ETHERNET(1,0,0,192,200,185,2)
```

Example 2: Read the firmware version number in the Ethernet daughter board in slot 2.

```
ETHERNET(0,2,6)
```

EX

Type: System 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.

EXECUTE

Type: System Command

Description: Used to implement the remote command execution via the USB interface. For more details see the section on using the OCX control.

FEATURE_ENABLE

Type: System Function

Syntax: **FEATURE_ENABLE(feature number)**

Description: The EURO205 and MC206 *Motion Coordinator*s, have the ability to unlock additional axes by entering a “Feature Enable Code”. This function is used to enable such protected features of a controller. It is recommended to use *Motion* Perfect 2 to enter and store the feature enable codes.

Note: To add servo axes to a EURO205 stepper base card the DAC chip must be added to the Euro205 in addition to enabling the feature.

Feature:	Function:
0	Stepper generation hardware on axis 0
1	Stepper generation hardware on axis 1
2	Stepper generation hardware on axis 2
3	Stepper generation hardware on axis 3
4	Servo generation hardware on axis 0
5	Servo generation hardware on axis 1
6	Servo generation hardware on axis 2
7	Servo generation hardware on axis 3

Controllers with features which can be enabled are fitted with a unique security code number when manufactured. This security code number can be found by typing **FEATURE_ENABLE** with no parameters:

Example 1:
```
>>feature_enable
Security code=17980000000028
Enabled features: 0 1
```
If you require additional features for a controller. These can be enabled by the entry of a password which is unique for each feature and controller security code. To obtain a feature enable code, the feature must be ordered and the security code FAXed to Trio or a distributor.

Example 2: In example one axes 0 and 1 are enabled for stepper operation. If axis 2 was required to operate as a stepper axis it would be necessary to obtain the password. For this card and this feature only the password is 5P0APT.

```
>>feature_enable(2)
Feature 2 Password=5P0APT
>>
>>feature_enable
Security code=17980000000028
Enabled features: 0 1 2
```

Note: When entering the passwords always enter the characters in upper case. Take care to check that 0 (zero) is not confused with O and 1 (one) is not confused with I.

FLASHVR

Type: System Function

Syntax: **FLASHVR([variable number])**

Description: Stores a single **VR()** global variable into permanent flash memory. VR variables stored in this way will have their value restored to the current value when the unit is powered up again. This feature is provided on controllers which do not feature battery backed ram **VR()** storage. Each **FLASHVR** command generates a write to a block of flash eprom. After 8000 block writes the flash sector will be erased. The controller writes into a second sector during the erase. Each sector can be erased over 1,000,000 times. It is therefore possible to use the FLASHVR() command many hundreds of millions of times. It does however have a finite life and cannot easily be replaced. Programmers MUST allow for this fact.

Note: The FLASHVR function is provided on controllers without batteries such as the MC202. However the FLASHVR(-1) and FLASHVR(-2) functions can be used with all *Motion Coordinator*'s. These functions write a whole block of data to flash memory and the programmer must ensure that they are only used occasionally.

The **FLASHVR** command can also be used to store the TABLE memory. By using the command **FLASHVR(-1)**, the entire contents of the TABLE memory will be written to the flash memory.

To revert to the standard power-up mode, i.e. not reading the values from the eprom, you should use the command **FLASHVR(-2)**

Parameters: **variable number:** The variable to be stored into flash

Example 1:
```
VR(25)=k
FLASHVR(25)
```

Example 2:
```
FOR v=1 to 10
   FLASHVR(v)
NEXT v
```

Example 3:
```
FLASHVR(-1)' Store TABLE memory to flash eprom
```

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:

Byte	Length	Value
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

HALT

Type: System Command

Description: Halts execution of all running programs. The **STOP** command will stop a specific program.

Example:
```
HALT '  Stop ALL programs
```
or
```
STOP "main"
' Stop only the program called 'MAIN'
```

Note: **HALT** does not stop any motion. Currently executing, or buffered moves will continue unless they are terminated with a **CANCEL** or **RAPIDSTOP** command.

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

LAST_AXIS

Type: System Parameter (Read Only)

Description: In order to maximise the processor time available to BASIC, the *Motion Coordinator* keeps a record of the highest axis number that is in use. This axis number is held in the system parameter **LAST_AXIS**. Axes higher than **LAST_AXIS** are not processed.

LAST_AXIS is set automatically by the system software when an axis command is used.

LIST

Type: System Command

Alternate Format: **TYPE**

Description: Prints the current **SELECT**ed 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. Use of LIST in Motion Perfect is not recommended.***

LOADSYSTEM

Type: System Command

Description: Loads new version of system software:

On the *Motion Coordinator* 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 RAM 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 the system software on disk supplied by TRIO in COFF format. (Files have a.OUT suffix, for example C140.OUT)

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 the user to confirm that the file should 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 1: All *Motion Coordinator* models have different system software files. The file name indicates the controller type.

Controller Type	Filename
MC202	Fnnn.OUT
Euro205	Ennn.OUT
Euro205X	Knnn.OUT
MC206	Hnnn.OUT
PCI208	Jnnn.OUT
MC216	Cnnn.OUT
MC224	Innn.OUT

Updates can be obtained from Trio's website at WWW.TRIOMOTION.COM

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

LOCK

Type: System Command

Syntax: LOCK(code)

Description: LOCK is designed to prevent programs from being viewed or modified by personnel unaware of the security code. The lock code number is stored in the flash eprom.

When a *Motion Coordinator* is locked, it is not possible to view, edit or save any programs and command line instructions are limited to those required to execute the program.

To unlock the *Motion Coordinator*, the UNLOCK command should be entered using the same lock code number which was used originally to LOCK it.

The lock code number may be any integer and is held in encoded form. Once LOCKed, the only way to gain full access to the *Motion Coordinator* is to UNLOCK it with the correct code. For best security the lock number should be 7 digits.

Parameters: code Any integer number

Example: >>LOCK(5619234)
The program cannot now be modified or viewed.

>>UNLOCK(5619234)
The system is now unlocked.

Note 1: LOCK and UNLOCK are available from the *Motion Coordinator* menu in *Motion* Perfect.

Note 2 ***If you forget the security code number, the Motion Coordinator may have to be returned to your supplier to be unlocked!***

Note 3 ***It is possible to compromise the security of the lock system. Users must consider if the level of security is sufficient to protect their programs.***

MOTION_ERROR

Type: System Parameter

Description: This system parameter returns a non-zero value when a motion error has occured on at least one axis, (normally a following error, but see **ERRORMASK**), and the value 0 when none of the axes 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 watchdog 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.

MPE

Type: System Command

Description: Sets the type of channel handshaking to be performed on the serial port 0. This is normally only used by the *Motion* Perfect program, but can be used for user applications. There are 4 valid settings

Parameters **channel type:** Any valid Trio BASIC expression

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.

3 Channel handshaking on, XON/XOFF controller by the channel. In **MPE(3)** mode the system transmits and receives using a protected packet protocol using a 16 bit CRC.

Example1:
```
>> PRINT #5,"Hello"
Hello
```
Example2:
```
MPE(1)
>> PRINT #5,"Hello"
<ESC>5Hello
<ESC>0
>>
```

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:

Bit Set	Error Type	Value
0	TX Timeout	1
1	TX Buffer Error	2
2	RX CRC Error	4
3	RX Frame Error	8

NEW

Type: System 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” must be in quotes)

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.

Note: Depending on the particular controller type, there may be a number of channels which are input only. For example, on the MC216 the first 8 channels are inputs, the next 8 bi-directional. If an MC216 has 2 P315 CAN-16 I/O modules connected the NIO parameter will return 48.

All channels on the CAN-16 I/O modules are bi-directional.

PEEK

Type: System Command

Syntax: **PEEK(address<,mask>)**

Description: The PEEK command returns value of a memory location of the controller ANDed with an optional mask value.

POKE

Type: System Command

Syntax: **POKE(address,value)**

Description: The POKE command allows a value to be entered into a memory location of the controller. The POKE command can prevent normal operation of the controller and should only be used if instructed by Trio Motion Technology.

POWER_UP

Type: Flash EPROM stored System Parameter

Description: This parameter is used to determine whether or not programs should be read from Flash Eprom on power up or software reset (**EX**).

Two values are possible:

0 Use the programs in battery backed RAM

1 Copy programs from the controllers Flash Eprom or Memory Stick (if present) into RAM.

The EPROM request flag held on the Memory Stick itself controls if the programs loaded are to be written into the internal Flash Eprom after loading (See **EPROM**)

Programs are individually selected to be run at power up with the **RUNTYPE** command

Note: **POWER_UP** is always an immediate command and therefore cannot be included in programs.

This value is normally set by *Motion* Perfect.

When using the Memory Stick users should refer to the overview in the MC206 Hardware Overview for a complete description of the Memory Stick functionality.

PROCESS

Type: System Command

Description: Displays the running status and process number for each current process.

PROFIBUS

Type: System Command

Syntax: **PROFIBUS(slot,function<,register><,value>)**

Description: The command PROFIBUS provides access to the registers of the SPC3 ASIC used on the Profibus daughter board. Trio can supply sample programs using this command to setup and control a Profibus daughter board.

Parameters:		
	slot:	Specifies the slot on the controller to be used. Set 0 for the daughter board slot of an MC206/Euro205 or the slot number of an MC216.
	function:	Specifies the function to be performed. 0: read register 1: write register
	register:	The SPC3 register number to read or write
	value:	The value to write into an SPC3 register

REMOTE

Type: System Command

Syntax: **REMOTE(slot)**

Description: Transfers control of a process to the remote computer via a USB interface and the Trio OCX control. The REMOTE command is normally inserted automatically on to a process by the system software. When a process is performing the REMOTE function execution of BASIC statements is suspended.

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.

RUN

Type: System 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 the named program on a particular process)

RUNTYPE

Type: System Command

Syntax: **>>RUNTYPE progname,autorun[,process#]**

Description: Sets whether program is run automatically at power up, and which process it is to run on. The current status of each program's **RUNTYPE** is displayed when a **DIR** command is performed. For any program to run automatically on power-up ALL the programs on the controller must compile without errors.

Parameters:

program name	Can be in inverted commas or without autorun
autorun	1 to run automatically, 0 for manual running
<process number>	optional to force process number

Example: **>>RUNTYPE progname,1,10**
- Sets program "progeny" 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.

See Also: **POWER_UP**

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 “Save 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. The sampling does not start until the **TRIGGER** command is executed.

Example 2: `SCOPE(OFF)`

Note: The `SCOPE` facility is a “one-shot” and needs to be re-started by the `TRIGGER` command each time an update of the samples is required.

Note2: Data saved to the TABLE memory by the `SCOPE` command is not placed in battery backed memory so will be lost when power is removed.

SCOPE_POS

Type: System Parameter (Read Only)

Description: Returns the current index position at which the `SCOPE` function is currently storing its parameters.

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 `SELECT`ed the commands `EDIT`, `RUN`, `LIST`, `NEW` etc. assume that the `SELECT`ed 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 `SELECT`ed program cannot be changed when programs are running.

Note 2: *Motion* Perfect automatically `SELECT`s programs when you click on their entry in the list in the control panel.

SLOT

Type: Slot Modifier

Description: Modifier specifies the slot number for a slot parameter such as COMMSTYPE.

SERCOS

Type: System Function

Syntax: **SERCOS(function#,slot,{parameters})**

Description: This function allows the SERCOS ring to be controlled from the Trio BASIC programming system. A SERCOS ring consists of a single master and 1 or more slaves daisy-chained together using fibre-optic cable. During initialisation the ring passes through several 'communication phases' before entering the final cyclic deterministic phase in which motion control is possible. In the final phase, the master transmits control information and the slaves transmit status feedback information every cycle time.

The *Motion Coordinator* SERCOS hardware uses the Sercon 816 SERCOS interface chip which allows connection speeds up to 16Mhz. This chip can be programmed at a register level using the **SERCOS** command if necessary. To program in this way it is necessary to obtain a copy of the chip data sheet.

The **SERCOS** command provides access to 11 separate functions:

Slot:
The slot number is in the range 0 to 3 and specifies the hardware channel

Function:

0	Read SERCOS Asic:	**val=SERCOS(0,slot,RAM/register,address)**
1	Write SERCOS Asic:	**SERCOS(1,slot,RAM/register,value)**
2	Initialise command:	**SERCOS(2,slot[,intensity[,baudrate[,period]]])**
3	Link SERCOS drive to Axis	**SERCOS(3,slot,slave_addr,axis[,slave_drive_type])**

4	Read parameter	`val=SERCOS(4,slot,slave_address, parameter_IDN[,parameter_size[,element_type[,list_length_offset]]])`
5	Write parameter	`SERCOS(5,slot,slave_address, parameter_IDN,parameter_size,parameter_value[,parameter_value …])`
6	Run SERCOS procedure command	`SERCOS(6,slot,slave_address, parameter_IDN[,timeout,[command_type]])`
7	Check for drive present	`SERCOS(7,slot,slave_address)`
8	Print network parameter	`SERCOS(8,slot,required_parameter)`
9	Reserved	
10	SERCOS ring status	`SERCOS(10,<slot>)`

Notes: MotionPerfect2 contains support for commissioning SERCOS rings. This tool simplifies the creation of a TrioBASIC startup program which consists of SERCOS statements to initialise the ring following power-on, and configure the ring in the deterministic cyclic phase.

SERVO_PERIOD

Type: System Parameter

Description: This parameter allows the controller servo period to be specified. On the MC202, MC216 and Euro205 controllers this period is specified in seconds and on the MC206 controller it is specified in microseconds. It is recommended not to adjust the default 1msec servo period on the MC202 and Euro205 controllers or where the stepper daughter board is being used. Where the MC216 is being used for servo axes the servo period may be adjusted down to around 0.0005 seconds.

Note: The *Motion Coordinator* must be reset using the **EX** command before the new servo period will be applied.

On the MC206 the servo period may be set to 1000,500 or 250 usec.

STEPLINE

Type: System Command

Syntax: **STEPLINE {Program name}{,Process number}**

Description: Steps one line in a program. This command is used by *Motion* Perfect to control program stepping. It can also be entered directly from the command line or as a line in a program with the following parameters.

Parameters:

Program name: This specifies the program to be stepped. All copies of this named program will step unless the process number is also specified. If the program is not running it will step to the first executable line on either the specified process or the next available process if the next parameter is omitted. If the program name is not supplied, either the SELECTed program will step (if command line entry) or the program with the STEPLINE in it will stop running and begin stepping.

Process number: This optional parameter determines which process number the program will use for stepping, or, if multiple copies of the same program exist, it is used to select the required copy for stepping.

Example 1:
```
>>STEPLINE "conveyor"
```

Example 2:
```
>>STEPLINE "maths",2
```

STICK_READ

Type: System Command

Syntax: **STICK_READ(sector, table start)**

Description: Copy one block of 128 values from a sector on the FlashStick to TABLE memory. The function returns TRUE (-1) if the **STICK_READ** was successful and FALSE (0) if the command failed, if for example the FlashStick is not present.

Parameters:

sector:	A number between 0 and 2047 that is used as a pointer to the sector to be read from the FlashStick.
table start:	The start point in the TABLE where the 128 values will be transferred to.

Example:
```
IF STICK_READ(25, 1000) THEN PRINT "Stick read OK"
```

STICK_WRITE

Type: System Command

Syntax: **STICK_WRITE(sector, table start)**

Description: Copy one block of 128 values from TABLE memory to a sector on the FlashStick. The function returns TRUE (-1) if the **STICK_WRITE** was successful and FALSE (0) if the command failed, if for example the FlashStick is not present.

Parameters:

sector:	A number between 0 and 2047 that is used as a pointer to the sector to be written to the FlashStick.
table start:	The start point in the TABLE where the 128 values will be transferred from.

Example:
```
check = STICK_WRITE(25, 1000)
IF check=TRUE THEN PRINT "Stick write OK"
```

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: ***Removing the controller power during a STORE sequence can lead to the controller having to be returned to Trio 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

Type: System Command

Syntax: **TABLE(address [, data1..data20])**

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 on all *Motion Coordinator*s EXCEPT the MC202 which has an 8000 point table length. 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.

Except in the MC202 the table entries are automatically battery backed. If FLASH Eprom storage is required it is recommended to set the values inside a program or use the FLASHVR(-1) function. 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)
```

This loads the internal table:

Table Entry:	Value:
100	0
101	120
102	250
103	370
104	470
105	530

Example 2:
```
>>PRINT TABLE(1000)
0.0000
>>
```

Note: The Oscilloscope 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 be the Oscilloscope function.

TABLEVALUES

Type: System Command

Syntax: **TABLEVALUES(first table number, last required table number, format)**

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.

TIME

Type: System Parameter (MC216 only)

Description: Returns the time from the real time clock. The time returned is the number of seconds since midnight 24:00 hours.

TIME$

Type: System Command (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
>>
```

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)
```

TSIZE

Type: System Parameter

Description: Returns one more than the highest currently defined table value.

Example:
```
>>TABLE(1000,3400)
>>PRINT TSIZE
1001.0000
```

Note: **TSIZE** can be reset using **>>DEL "TABLE"**

UNLOCK

Type: System Command

Syntax: **UNLOCK(code)**

Description: Enables full access to a *Motion Coordinator* which has a security lock code applied via the **LOCK()** command.

When a *Motion Coordinator* is locked, it is not possible to view, edit or save any programs and command line instructions may be limited to those required to execute the program only.

To unlock the *Motion Coordinator*, the **UNLOCK** command should be entered using the same security code number which was used originally to **LOCK** it.

The security code number may be any integer and is held in encoded form. Once **LOCK**ed, the only way to gain full access to the *Motion Coordinator* is to **UNLOCK** it with the correct code.

Parameters:

code	Any integer number

Example:

```
>>LOCK(561234)
```
The program cannot now be modified or seen.

```
>>UNLOCK(561234)
```
The system is now unlocked.

Note 1: It is not normally necessary to use the **LOCK**/**UNLOCK** commands from the command line as the they are available directly from the Controller menu in *Motion* Perfect 2.

USB

Type: System Command

Syntax: **USB(slot,function<,register><,value>)**

Description: The command USB provides access to the registers of the USBN9602 USB controller chip used in the MC206 and USB daughter board. It is not required to use this command as the functions are included in the *Motion Coordinator* system software.

Parameters:

slot:	Specifies the slot on the controller to be used. Set 1 for the built-in USB of the MC206 or the slot number of an MC216/ Euro205.
function:	Specifies the function to be performed. 0: read register 1: write register
register:	The register number to read or write
value:	The value to write into a register

USB_STALL

Type: System Parameter

Description: This parameter returns TRUE if the USB controller chip has its "stalled" (unable to communicate) bit set.

VERSION

Type: System Parameter

Description: Returns the version number of the system software installed on the *Motion Coordinator*.

Example:
```
>>? VERSION
  1.5000
```

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:
```

VR

Type: System Command

Syntax: **VR(expression)**

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, except on the MC206 which has 1024 VR's numbered from 0..1023.

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 (except on MC202) 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, or from a master via a MODBUS or other supported 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 'van' 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 ten1's or 0's to signal that the positions are full (1) or EMPTY (0). The gantry puts the load down in the first free position. Part of the program to achieve this would be:

```
movep:
  MOVEABS(115) ' MOVE TO FIRST PUT DOWN POSITION:
  FOR VR(0)=101 TO 110
    IF VR(VR(0))=0) THEN GOSUB load
    MOVE(200)' 200 IS SPACING BETWEEN POSITIONS
  NEXT VR(0)
  PRINT "All Positions Are Full"
  WAIT UNTIL IN(3)=ON
```

```
GOTO movep

load:
   'PUT LOAD IN POSITION AND MARK ARRAY
   OP(15,OFF)
   VR(VR(0))=1
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:
```
'Assign VR(65) to VR(0) multiplied by Axis 1 measured position
VR(65)=VR(0)*MPOS AXIS(1)
PRINT VR(65)
```

WDOG

Type: System Parameter

Description: Controls the **WDOG** relay contact used for enabling external drives. 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

Note 2: Analog outputs and step/direction outputs are also disabled when WDOG=OFF

:

Type: Special Character

Description: The colon character is used to terminate labels used as destinations for **GOTO** and **GOSUB** commands.

Labels may be character strings of any length. (The first 15 characters are significant) Alternatively line numbers can be used. Labels must be the first item on a line and should have no leading spaces.

Example:
```
start:
```

The colon is also used to separate Trio 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 remaining statements will not be executed:

```
PRINT "Hello":GOTO Routine:PRINT "Goodbye"
```
Goodbye will not be printed.

Similarly with GOSUB because subroutine calls return to the following line.

’

Type: Special Character

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 after the executable statement. 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 required
MOVE(turns)’ the movement occurs here
```

#

Type: Special Character

Description: The # symbol is used to specify a communications channel to be used for serial input/output commands.

Note: Communications Channels greater than 3 will only be used when the controller is running in *Motion* Perfect mode (See **MPE** command).

Example 1:
```
PRINT #3,"Membrane Keypad"
PRINT #2,"Port 2"
```

Example 2:
```
' Check membrane keypad on fibre-optic channel
IF KEY #3 THEN GET #3,k
```

$

Type: Special Character

Description: The $ symbol is used to specify that the number that follows is in hexadecimal format.

Note: Only the MC206 and MC224 controllers support hexadecimal format.

Example 1:
```
VR(10)=$8F3B
OP($CC00)
```

Process Parameters and Commands

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)
```

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)

Chan	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
```

LOOKUP

Type: Process Command

Syntax: **LOOKUP(format,entry) <PROC(process#)>**

Description: The LOOKUP command allows *Motion* Perfect to access the local variables on an executing process. It is not normally required for BASIC programs.

Parameters:

format:	0: Prints (in binary) floating point value from an expression 1: Prints (in binary) integer value from an expression 2: Prints (in binary) local variable from a process 3: Returns to BASIC local variable from a process
entry:	Either an expression string (format=0 or 1) or the offset number of the local variable into the processes local variable list.

OUTDEVICE

Type: Process Parameter

Description: The value in this parameter determines the serial output device for the **PRINT** command for the process. The channel numbers are the same as described in **INDEVICE**.

PMOVE

Type: Process Parameter

Modifier: **PROC**

Description: Returns 1 if the process move buffer is occupied, and 0 it is empty. When one of the *Motion Coordinator* processes encounters a movement command the process loads the movement requirements into its “process move buffer”. This can hold one movement instruction for any group of axes. When the load into the process move buffer is complete the **PMOVE** parameter is set to 1. When the next servo interrupt occurs the motion generation program will load the movement into the “next move buffer” of the required axes if these are available. When this second transfer is complete the **PMOVE** parameter is cleared to 0. Each process has its own **PMOVE** parameter.

PROC

Type: Process Modifier

Description: Allows a process parameter from a particular process to be read or set.

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

PROC_LINE

Type: Process Parameter (Read Only)

Description: Allows the current line number of another program to be obtained with the **PROC(x)** modifier.

Example:
```
PRINT PROC_LINE PROC(2)
```

PROCNUMBER

Type: Process Parameter

Description: Returns the process on which a Trio BASIC program is running. This is normally required when multiple copies of a program are running on different processes.

Example:
```
MOVE(length) AXIS(PROCNUMBER)
```

PROC_STATUS

Type: Process Parameter (Read Only)

Description: Returns the status of another process, referenced with the **PROC(x)** modifier.

Example:
```
WAIT UNTIL PROC_STATUS PROC(12)=0
```

Returns

0	Process Stopped
1	Process Running
2	Process Stepping
3	Process Paused

RESET

Type: Process Command

Description: Sets the value of all the local named variables of a Trio BASIC process to 0.

RUN_ERROR

Type: Process Parameter

Modifier: PROC

Description: Contains the number of the last program error that occurred on the specified process.

Example:
```
>>? RUN_ERROR PROC(5)
  9.0000
>>
```

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 cycle 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.

Mathematical Operations and Commands

+ Add

Type: Arithmetic operation

Syntax **<expression1> + <expression2>**

Description: Adds two expressions

Parameters: **Expression1:** Any valid Trio BASIC expression

Expression2: Any valid Trio BASIC expression

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

Trio BASIC evaluates the parentheses first giving the value 18.9 and then adds the two expressions. Therefore result holds the value 28.9

- Subtract

Type: Arithmetic operation

Syntax **<expression1> - <expression2>**

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)
```

Trio BASIC evaluates the parentheses first giving the value 18.9 and then subtracts this from 10. Therefore VR(0) holds the value -8.9

* Multiply

Type: Arithmetic operation

Syntax **<expression1> * <expression2>**

Description: Multiplies expression1 by expression2

Parameters: **Expression1:** Any valid Trio BASIC expression

Expression2: Any valid Trio BASIC expression

Example:

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

Trio BASIC evaluates the brackets first giving the value 11.1 and then multiplies this by 10. Therefore factor holds the value 111

/ Divide

Type: Arithmetic operation

Syntax **<expression1> / <expression2>**

Description: Divides expression1 by expression2

Parameters: **Expression1:** Any valid Trio BASIC expression

Expression2: Any valid Trio BASIC expression

Example:

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

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

Therefore a holds the value 0.9009

= Equals

Type: Arithmetic Comparison Operation

Syntax **<expression1> = <expression2>**

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 program execution will continue at line starting "**label:**"

<> Not Equal

Type: Arithmetic Comparison Operation

Syntax **<expression1> <> <expression2>**

Description: Returns **TRUE** if expression1 is not 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 MTYPE<>0 THEN GOTO scoop
If axis is not idle (MTYPE=0 indicates axis idle) then goto label "scoop"

> Greater Than

Type: Arithmetic Comparison Operation

Syntax **<expression1> > <expression2>**

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:
```
WAIT UNTIL MPOS>200
```
The program will wait until the measured position is greater than 200

Example 2:
```
VR(0)=1>0
```
1 is greater than 0 and therefore VR(0) holds the value -1

>= Greater Than or Equal

Type: Arithmetic Comparison Operation

Syntax **<expression1> >= <expression2>**

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: **Expression1:** Any valid Trio BASIC expression

Expression2: Any valid Trio BASIC expression

Example:
```
IF target>=120 THEN MOVEABS(0)
```
If variable target holds a value greater than or equal to 120 then move to the absolute position of 0.

< Less Than

Type: Arithmetic Comparison Operation

Syntax **<expression1> < <expression2>**

Description: Returns **TRUE** if expression1 is less 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:
```
IF AIN(1)<10 THEN GOSUB rollup
```
If the value returned from analog input 1 is less than 10 then execute subroutine "**rollup**"

<= Less Than or Equal

Type: Arithmetic Comparison Operation

Syntax **<expression1> <= <expression2>**

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:

Expression1: Any valid Trio BASIC expression

Expression2: Any valid Trio BASIC expression

Example:
```
maybe=1<=0
```
1 is not less than or equal to 0 and therefore variable **maybe** holds the value 0

ABS

Type: Function

Syntax: **ABS(expression)**

Description: The **ABS** function converts a negative number into its positive equal. Positive numbers are unaltered.

Parameters: **Expression:** Any valid Trio BASIC expression

Example:
```
IF ABS(AIN(0))>100 THEN
  PRINT "Analog Input Outside +/-100"
ENDIF
```

ACOS

Type: Function

Syntax: **ACOS(expression)**

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
```

AND

Type: Logical and bitwise operator

Syntax **<expression1> AND <expression2>**

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:

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)
```
Trio 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
```
AND is a bitwise operator and so the binary action taking place is:

	0	1
0	0	0
1	0	1

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

Therefore **VR(0)** holds the value 2

Example 3:
```
IF MPOS AXIS(0)>0 AND MPOS AXIS(1)>0 THEN GOTO cyc1
```

ASIN

Type: Mathematical Function

Syntax: **ASIN(expression)**

Alternate Format: **ASN(expression)**

Description: The **ASIN** function returns the arc-sine of a number which should be in the range +/-1. The result in radians is in the range -PI/2.. +PI/2 (Numbers outside the +/-1 input range will return zero)

Parameters: **Expression:** Any valid Trio BASIC expression.

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

ATAN

Type: Mathematical Function

Syntax: **ATAN(expression)**

Alternate Format: **ATN(expression)**

Description: The **ATAN** function returns the arc-tangent of a number. The result in radians is in the range -PI/2.. +PI/2

Parameters: **Expression:** Any valid Trio BASIC expression.

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

ATAN2

Type: Mathematical Function

Syntax: **ATAN2(expression1,expression 2)**

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
```

CLEAR_BIT

Type: Command

Syntax: **CLEAR_BIT(bit#,vr#)**

Description: **CLEAR_BIT** can be used to clear the value of a single bit within a **VR()** variable.

Example: **CLEAR_BIT(6,23)**
Bit 6 of VR(23) will be cleared (set to 0).

Parameters:

bit #	Bit number within the VR. Valid range is 0 to 23
vr#	VR() number to use

See also **READ_BIT, SET_BIT**

CONSTANT

Type: System Command

Syntax: **CONSTANT "name", value**

Description: Declares the *name* as a constant for use both within the program containing the **CONSTANT** definition and all other programs in the Motion Coordinator project. MC206 and MC224 only.

Parameters:

name:	Any user-defined name containing lower case alpha, numerical or underscore (_) characters.
value	The value assigned to *name*.

Example:

```
CONSTANT "nak",$15
CONSTANT "start_button",5

IF IN(start_button)=ON THEN OP(led1,ON)
IF key_char=nak THEN GOSUB no_ack_received
```

Note: The program containing the **CONSTANT** definition must be run before the name is used in other programs. For fast startup the program should also be the ONLY process running at power-up.

A maximum of 128 **CONSTANTs** can be declared.

COS

Type: Mathematical Function

Syntax: **COS(expression)**

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
>>
```

EXP

Type: Mathematical Function

Syntax: **EXP(expression)**

Description: Returns the exponential value of the expression.

FRAC

Type: Mathematical Function

Syntax: **FRAC(expression)**

Description: Returns the fractional part of the expression.

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

GLOBAL

Type: System Command

Syntax: **GLOBAL "name", vr_number**

Description: Declares the *name* as a reference to one of the global VR variables. The name can then be used both within the program containing the **GLOBAL** definition and all other programs in the Motion Coordinator project. MC206 and MC224 only.

Parameters:

name:	Any user-defined name containing lower case alpha, numerical or underscore (_) characters.
vr_number	The number of the VR to be associated with *name*.

Example:

```
GLOBAL "srew_pitch",12
GLOBAL "ratio1",534

ratio1 = 3.56
screw_pitch = 23.0
PRINT screw_pitch, ratio1
```

Note: The program containing the **GLOBAL** definition must be run before the name is used in other programs. For fast startup the program should also be the ONLY process running at power-up.

In programs that use the defined **GLOBAL**, **name** has the same meaning as **VR(vr_number)**. Do not use the syntax: VR(name).

A maximum of 128 **GLOBALs** can be declared.

IEEE_IN

Type: Mathematical Function

Syntax: **IEEE_IN(byte0,byte1,byte2,byte3)**

Description: The **IEEE_IN** function returns the floating point number represented by 4 bytes which typically have been received over a communications link such as Modbus. MC206 and MC224 only.

Parameters:

byte0 - 3:	Any combination of 8 bit values that represents a valid IEEE floating point number.

Example: `VR(20) = IEEE_IN(b0,b1,b2,b3)`

Note: Byte 0 is the high byte of the 32 bit floating point format.

IEEE_OUT

Type: Mathematical Function

Syntax: `byte_n = IEEE_OUT(value, n)`

Description: The `IEEE_OUT` function returns a single byte in IEEE format extracted from the floating point value for transmission over a bus sytem. The function will typically be called 4 times to extract each byte in turn. MC206 and MC224 only.

Parameters:

`value:`	Any TrioBASIC floating point variable or parameter.
`n:`	The byte number (0 - 3) to be extracted.

Example:

```
byte0 = IEEE_OUT(MPOS AXIS(2), 0)
byte1 = IEEE_OUT(MPOS AXIS(2), 1)
byte2 = IEEE_OUT(MPOS AXIS(2), 2)
byte3 = IEEE_OUT(MPOS AXIS(2), 3)
```

Note: Byte 0 is the high byte of the 32 bit IEEE floating point format.

INT

Type: Mathematical Function

Syntax: `INT(expression)`

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 integer value take the `INT` function of the (number + 0.5)

LN

Type: Mathematical Function

Syntax: **LN(expression)**

Description: Returns the natural logarithm of the expression.

Parameter: Any valid Trio BASIC expression.

MOD

Type: Mathematical Function

Syntax: **MOD(expression)**

Description: Returns the integer modulus of an expression.

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

NOT

Type: Mathematical Function

Description: The **NOT** function truncates the number and inverts all the bits of the integer remaining.

Parameter: **expression:** Any valid Trio BASIC expression.

Example:
```
PRINT 7 AND NOT(1.5)
  6.0000
>>
```

OR

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)
```

Trio 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

READ_BIT

Type: Command

Syntax: **READ_BIT(bit#,vr#)**

Description: **READ_BIT** can be used to test the value of a single bit within a **VR()** variable.

Example: **res=READ_BIT(4,13)**

Parameters:

bit #	Bit number within the VR. Valid range is 0 to 23
vr#	VR() number to use

See also **SET_BIT, CLEAR_BIT**

SET_BIT

Type: Command

Syntax: **SET_BIT(bit#,vr#)**

Description: **SET_BIT** can be used to set the value of a single bit within a **VR()** variable. All other bits are unchanged.

Parameters:

bit #	Bit number within the VR. Valid range is 0 to 23
vr#	**VR()** number to use

Example:
```
SET_BIT(3,7)
Will set bit 3 of VR(7) to 1.
```

See also **READ_BIT, CLEAR_BIT**

SGN

Type: Mathematical Function

Syntax: **SGN(expression)**

Description: The **SGN** function returns the SIGN of a number.

1	Positive non-zero
0	Zero
-1	Negative

Parameters: **expression:** Any valid Trio BASIC expression.

Example:
```
>>PRINT SGN(-1.2)
-1.0000
>>
```

SIN

Type: Mathematical Function

Syntax: **SIN(expression)**

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
```

SQR

Type: Mathematical Function

Syntax: **SQR(number)**

Description: Returns the square root of a number.

Parameters: **number:** Any valid Trio BASIC number or variable.

Example:
```
>>PRINT SQR(4)
2.0000
```

```
>>
```

TAN

Type: Mathematical Function

Syntax: **TAN(expression)**

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
```

XOR

Type: Logical and bitwise operator

Description: This performs and 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:

Parameters: **Expression1:** Any valid Trio BASIC expression

Expression2: Any valid Trio BASIC expression

Example:
```
a = 10 XOR (2.1*9)
```

Trio 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: **a=10 XOR 18**. The **XOR** is a bitwise operator and so the binary action taking place is:

```
    01010
XOR 10010
----------
    11000
```

The result is therefore 24.

Constants

FALSE

Type: Constant

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

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

OFF

Type: Constant

Description: **OFF** returns the value 0

Example:
```
IF IN(56)=OFF THEN GOTO label
'Branch if input 56 is off.
```

ON

Type: Constant

Description: **ON** returns the value 1.

Example:
```
OP(lever,ON)'  This sets the output named lever to ON.
```

TRUE

Type: Constant

Description: The constant **TRUE** takes the numerical value of -1.

Example:
```
t=IN(0)=ON AND IN(2)=ON
IF t=TRUE THEN
  PRINT "Inputs are on"
ENDIF
```

PI

Type: Constant

Description: **PI** is the circumference/diameter constant of approximately 3.14159

Example:
```
circum=100
PRINT "Radius=";circum/(2*PI)
```

Axis Parameters

ACCEL

Type: Axis Parameter

Syntax: **ACCEL=value**

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 " Accel rate:";ACCEL;" mm/sec/sec"
```

ADDAX_AXIS

Type: Axis Parameter (Read Only)

Syntax: **ADDAX_AXIS**

Description: Returns the axis currently linked to with the **ADDAX** command, if none the parameter returns -1.

ATYPE

Type: Axis Parameter

Description: The **ATYPE** axis parameter indicates the type of axis fitted. On daughter board based axes, the **ATYPE** axis parameter is set by the system software at power up.

Controllers that use Feature Enable Codes to activate axes, such as the Euro205 and MC206, have the **ATYPE** of each axis set by the system software depending on the Enabled Features on that Motion Coordinator. The **ATYPE** of Remote Axes must be set during initialisation in a suitable Trio BASIC program. e.g. STARTUP.BAS.

On the MC202 the **ATYPE** parameter must be set to a value between 1 and 4 to select the axis function.

See Appendix 1 for a complete list of ATYPE values.

Example:
```
>>PRINT ATYPE AXIS(2)
1.0000
```
This would show that an stepper daughter board is fitted in this axis slot.

```
ATYPE AXIS(20)=16
```
Sets axis 20 to be a remote SERCOS speed axis. (This feature must be enabled with the correct Feature Enable Code first)

```
ATYPE AXIS(0)=4
```
Sets axis 0 to be a stepper with encoder verification axis on the MC202.

AXISSTATUS

Type: Axis Parameter (Read Only)

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

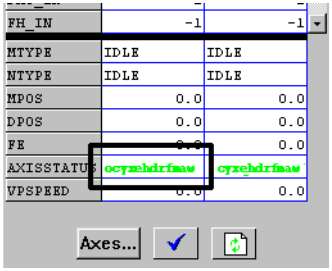
Bit	Description	Value	char
0	Unused	1	
1	Following error warning range	2	**w**
2	Communications error to remote drive	4	**a**
3	Remote drive error	8	**m**
4	In forward limit	16	**f**
5	In reverse limit	32	**r**
6	Datuming	64	**d**
7	Feedhold	128	**h**
8	Following error exceeds limit	256	**e**
9	In forward software limit	512	**x**
10	In reverse software limit	1024	**y**
11	Cancelling move	2048	**c**
12	Encoder power supply overload (MC206)	4096	**o**
13	Set on SSI axis after initialisation	8192	

The **AXISSTATUS** axis parameter is set by the system software is read-only..

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

Note: In the *Motion* Perfect parameter screen the **AXISSTATUS** parameter is displayed as a series of characters, **ocyxehdrfmaw**, as listed in the table above.

These characters are displayed in green lowercase letters normally, or red uppercase when set.



See Also: **ERRORMASK**

BOOST

Type: Axis Parameter

Syntax: **BOOST=ON / BOOST=OFF**

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**

CAN_ADDRESS

Type: Axis Parameter

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

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.

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
```

CLUTCH_RATE

Type: Axis Parameter

Description: This affects operation of **CONNECT** by changing the connection ratio at the specified rate/second.

Default **CLUTCH_RATE** is set very high to ensure compatibility with earlier versions.

Example:
```
CLUTCH_RATE=5
```

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)
```

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 a 12 bit DAC can take is:

DAC=-2048 corresponds to a voltage of 10V

to

DAC=2047 corresponds to a voltage of -10v

The range of values that a 16 bit DAC (MC206 built-in axes only) can take is:

DAC=32767 corresponds to a voltage of 10V

to

DAC=-32768 corresponds to a voltage of -10v

Note: The SERVO DAUGHTER BOARD/MC202 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.

```
WDOG=ON
SERVO AXIS(0)=OFF
square:
  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 drive 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
>>
```

DAC_SCALE

Type: Axis Parameter

Description: The **DAC_SCALE** axis parameter is only available on the MC206. The parameter has 2 purposes: It is set to value 16 on power up on the built-in axes of the MC206. This scales the values applied to the higher resolution DAC so that the gains required on the axis are similar to those required on the other controllers. **DAC_SCALE** may be set negative (-16) to reverse the polarity of the DAC output signal. When the servo is off the magnitude of **DAC_SCALE** is not important as the voltage applied is controlled by the DAC parameter. The polarity is still reversed however by **DAC_SCALE**. When a Servo Daughter Board is used in an MC206 the default **DAC_SCALE** parameter for that axis will be 1.

Example:

```
>>DAC_SCALE AXIS(3)=-16
>>
```

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 (default) 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.

DECEL

Type: Axis Parameter

Syntax: **DECEL=value**

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

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

DEMAND_EDGES

Type: Axis Parameter (Read Only)

Description: Allows the user to read back the current **DPOS** in encoder edges.

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

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.

DRIVE_STATUS

Type: Axis Parameter

Description: Returns the status register of a drive with digital communications capability connected to the *Motion Coordinator*.

In the case of an SLM axis it returns the SLM and drive status:
Bits 0..7 return bits 0..7 of register 0x8000 on the drive. Bits 8..23 return register 0xD000 on the SLM.

Example:
```
>>PRINT DRIVE_STATUS AXIS(8)
0.0000
>>
```

D_GAIN

Type: Axis Parameter

Syntax: **D_GAIN=value**

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 *Kd* and a change in following error *Đe* the contribution to the output signal is:

$$Od = Kd \times Đe$$

Example: `D_GAIN=0.25`

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

ENCODER

Type: Axis Parameter (Read Only)

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. On MC202 and the built-in axes of a Euro205 or MC206 the **ENCODER** register is a 14 bit register.

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 although may be written to if required provided that **SERVO=ON** and no move is in progress. This will produce a step change in **DPOS**. Making step changes in **DPOS** can easily lead to "Following error exceeds limit" errors unless the steps are small or the **FE_LIMIT** is high.

ERRORMASK

Type: Axis Parameter

Description: The value held in this parameter is bitwise **AND**ed 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.

On the MC202, Euro 205 and MC216 the default setting is 256. This will trip the enable relay only if a following error condition occurs.

For the MC206, the default value is 268 which is set to also trap critical errors with digital drive communications.

See Also: **AXISSTATUS**

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 (default) 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 a reserved word, but 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**

Syntax: **FE_LIMIT=value**

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

Syntax: **FERANGE=value**

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

Syntax: **FEGRAD=value**

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

Syntax: **FEMIN=value**

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**

Syntax: **FHOLD_IN=value**

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 (default) 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.

FHSPEED

Type: Axis Parameter

Syntax: **FHSPEED=value**

Description: When the feedhold input is set motion is usually ramped down to zero speed as the feedhold speed is set to its default zero value. In some cases it may be desirable for the axis to ramp to a known constant speed when the feedhold input is set. To do this the **FHSPEED** parameter is set to a non zero value. The value is in user units/sec.

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**.

FWD_IN

Type: Axis Parameter

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 (default) 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

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.

INVERT_STEP

Type: Axis Parameter

Description: **INVERT_STEP** is used to switch a hardware inverter into the stepper pulse output circuit. This can be necessary in for connecting to some stepper drives. The electronic logic inside the *Motion Coordinator* stepper pulse generation assumes that the FALLING edge of the step output is the active edge which results in motor movement. This is suitable for the majority of stepper drives. 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**. Default=OFF.

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

I_GAIN

Type: Axis Parameter

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 0 (Default). 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 *Ki* and a sum of position errors $\oplus e$, the contribution to the output signal is:

$$Oi = Ki \times \oplus e$$

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**.

LINKAX

Type: Axis Parameter (Read Only)

Description: Returns the axis number that the axis is linked to during any linked moves. Linked moves are where the demand position is a function of another axis. E.G. **CONNECT**, **CAMBOX**, **MOVELINK**

MARK

Type: Axis Parameter (Read Only)

Description: Returns **TRUE** when a registration event has occurred. This is set to **FALSE** by the **REGIST** command and set to true when the registration event occurs. When **TRUE** the **REG_POS** is valid.

Example:
```
loop:
  WAIT UNTIL IN(punch_clr)=ON
  MOVE(index_length)
  REGIST(3)' rising edge of R
  WAIT UNTIL MARK  MOVEMODIFY(REG_POS + offset)
  WAIT IDLE
GOTO loop
```

MARKB

Type: Axis Parameter (Read Only)

Description: Only usable on MC206 built-in axes. MARKB returns **TRUE** when the Z registration position has been latched. This is set to **FALSE** by the **REGIST** command and set to true when the registration event occurs. When **TRUE** the **REG_POSB** is valid. See REGIST() and REG_POSB.

MERGE

Type: Axis Parameter

Syntax: **MERGE=ON / MERGE=OFF**

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 does 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
```

MICROSTEP

Type: Axis Parameter

Description: Sets microstepping mode when using a stepper daughter board, MC202 or Euro205. On these controllers the stepper pulse circuit contains a circuit which places the step pulses more evenly in time by dividing the pulse rate by 2 or 16:

MICROSTEP=OFF (DEFAULT)	62.5 kHz Maximum
MICROSTEP=ON	500 kHz Maximum

(On the MC206 a different pulse generation circuit is used which always divides the pulse rate by 16 and is NOT affected by the MICROSTEP parameter. This circuit can generate pulses up to 2Mhz) 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 pulse rate 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
```

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 1mSec. 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.

MTYPE	Move Type
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 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.

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)**

OFFPOS

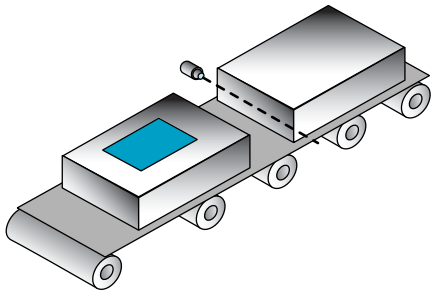
Type: Axis Parameter

Description: The **OFFPOS** parameter allows the axis position to be offset by any value without affecting motion. **OFFPOS** can therefore be used to effectively datum a system at full speed. Values loaded into the **OFFPOS** axis parameter are reset to 0 by the system software as the axis position is changed.

Example: Define the current demand position as zero:

```
OFFPOS=-DPOS
WAIT UNTIL OFFPOS=0'   wait until applied
```
This is equivalent to **DEFPOS(0)**

Example 2: A conveyor is used to transport boxes onto which labels must be applied.



Using the **REGIST()** function, we can capture the position at which the leading edge of the box is seen, then by using **OFFPOS** we can adjust the measured position of the axis to be zero at that point. Therefore, after the registration event has occurred, the measured position (seen in **MPOS**) will actually reflect the absolute distance from the start of the box, the mechanism which applies the label can take advantage of the absolute position start mode of the **MOVELINK** or **CAMBOX** commands to apply the label.

```
BASE(conv)
REGIST(3)
WAIT UNTIL MARK
OFFPOS = -REG_POS ' Leading edge of box is now zero
```

Note: The **OFFPOS** adjustment is executed on the next servo period. Several Trio 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.

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.

OUTLIMIT

Type: Axis Parameter

Description: The output limit restricts the voltage output from a servo axis to a lower value than the maximum. The value required varies depending on whether the axis has a 12 bit or 16 bit DAC. If the voltage output is generated by a 12 bit DAC values an OUTLIMIT of 2047 will produce the full +/-10v range. If the voltage output is generated by a 16 bit DAC values an OUTLIMIT of 32767 will produce the full +/-10v range. Only the MC206 internal axes have 16 bit DAC's

Example:
```
OUTLIMIT AXIS(0)=1023
```

The above will limit the voltage output to a ±5V output range on a servo daughter board axis. 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 gain constant which is multiplied by the change in measured position. The result is summed with all the other gain terms and applied to the servo DAC. Default value is 0. Adding NEGATIVE output velocity gain to a system is mechanically equivalent to adding damping. It is likely to produce a smoother response and allow the use of a higher proportional gain than could otherwise be used, but at the expense of higher following errors. High values may lead to oscillation and produce high following errors. For an output velocity term Kov and change in position ΔPm, the contribution to the output signal is:

$$Oov = Kov \times \Delta Pm$$

Note: Negative values are normally required. Servo gains have no effect on stepper motor axes.

PP_STEP

Type: Axis parameter

Description: This parameter allows the incoming raw encoder counts to be multiplied by an integer value in the range -1024 to 1023. 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. 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 is 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
```

Note: If used in a Servo axis, increasing PP_STEP will require a proportionate decrease of loop gain.

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 *Kp* and position error *e*, its contribution to the output signal is:

$$Op = Kp \times e$$

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
```

REG_MATCH

Type: Axis Parameter (Read Only)

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 (Read Only)

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)

REG_POSB

Type: Axis Parameter (Read Only)

Description: Useable only on MC206 built-in axes. REG_POSB returns the position at which a registration Z mark was seen on an axis. See **REGIST()** for more details.

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.

```
start:
  SPEED=10
  MOVE(45)
  WAIT UNTIL REMAIN<5
  SPEED=1
  WAIT IDLE
```

REMOTE_ERROR

Type: Axis Parameter

Description: Returns the number of errors on a drive's digital communication link.

Example:
```
>>PRINT REMOTE_ERROR
1.0000
>>
```

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 **REP_OPTION** parameter controls the way the REPDIST is applied. In the default setting (**REP_OPTION bit 0=0**) REPDIST operation is selected in the range –**REPEAT DISTANCE** to **+REPEAT DISTANCE**. In some circumstances it more convenient for the axis positions to be specified from 0 to **+REPEAT DISTANCE**. (**REP_OPTION bit 0=1**)

REP_OPTION bit 1 is set **ON** to switch OFF the automatic repeat option of the **CAMBOX** or MOVELINK function. When the system software has set the option OFF it automatically clears bit 1 of **REP_OPTION**.

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 (default) 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 (default) then no input is used as a reverse jog. When the 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.

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**

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**.

SP

Type: Axis 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
```

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. **SRAMP** can be used with **MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL**, **FORWARD**, **REVERSE** and **MOVEMODIFY** move types.

Note: The **SRAMP** 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.

If the number of SSI_BITS is to be changed, the parameter must first be set to zero before entering the new value.

Example:
```
SSI_BITS AXIS(3)=12
SSI_BITS AXIS(7)=21
```

Example2:
```
're-initialise MPOS using absolute value from encoder
SERVO=OFF
SSI_BITS =0
SSI_BITS =databits
```

TRANS_DPOS

Type: Axis Parameter (Read Only)

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 machines 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 if you want to install frame transformations.

Note: See also **FRAME**

TRANSITIONS

Type: Axis Parameter

Description: Records the number of register input transitions in a **REGIST** sequence

See also **REGIST**, **RECORD**, **MATCH**

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, acceleration 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 x 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**.

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 signal

Note: On the MC202 and the Euro205 when **VERIFY=OFF**, the encoder counting circuit is configured to accept **STEP** and **DIRECTION** signals hard wired to the encoder A and B inputs. If **VERIFY=ON**, the encoder circuit is configured for the usual quadrature input.

Take care that the encoder inputs do not exceed 5 volts.

Example:
```
VERIFY AXIS(3)=ON
```

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 ΔPd, the contribution to the output signal is:

$$Ovff = Kvff \times \Delta Pd$$

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

VP_SPEED

Type: Axis Parameter (Read Only)

Alternate Format: **VPSPEED**

Description: The velocity profile speed is an internal speed which is ramped up and down as the movement is velocity profiled. It is reported in user units/sec.

Example: Wait until command speed is achieved:

```
MOVE(100)
WAIT UNTIL SPEED=VP_SPEED
```

CHAPTER

9

PROGRAMMING EXAMPLES

Example Programs

1 - Fetching an Integer Value from the Membrane Keypad

The subroutine “getnum” 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 “CLR” key and the ENTER key are used. Other keys are ignored.

```
' Demonstrate integer number entry via Membrane Keypad:
getnum: pos=70
  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
```

Example 2 - Fetching a Real Value from the Membrane Keypad

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.

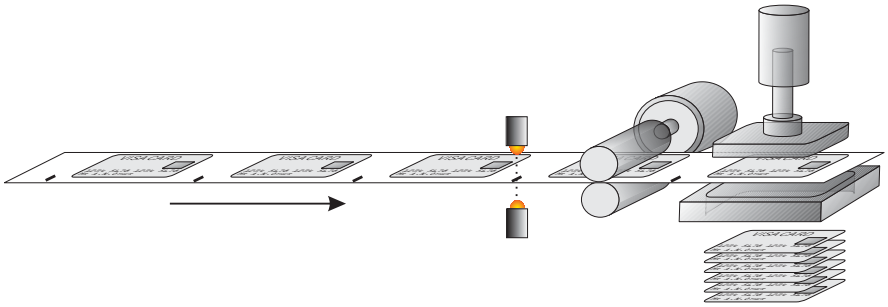
```
getnum:
  pos=40
  dpoint=0
  num=0
  negative=1
  PRINT#5,CHR(20);
  REPEAT
    PRINT#5,CURSOR(pos);num*negative[8,2];
```

```
      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
```

Example 3 - ATM Card Production

Key Features Used: `REGIST, MOVEMODIFY`

An automated die-cutting machine, is designed to punch out pre-printed plastic cards for use in ATM machines etc.



There is one servo axis which is connected to the draw rollers which feed the card into the machine. A printed registration mark appears once per card and is sensed by an optical sensor connected to the Registration input of the MC2xx's Servo Daughter Board.

The operation of the machine is quite simple, the cards are printed at a known fixed-pitch. Each cycle, the draw rolls must feed the card into position, an output is then fired to operate the punch. An input signals that the punch is clear of the cards and the cycle can repeat.



In an ideal situation we would simply datum the first card and then move a fixed pitch every cycle,

```
loop:
  MOVE(card_pitch)
  WAIT IDLE
  OP(punch,ON)
  WAIT UNTIL IN(punch_clear)=OFF
  WAIT UNTIL IN(punch_clear)=ON
  OP(punch,OFF)
GOTO loop
```

In the real world we must allow for mechanical slippage and any inconsistencies which may occur in the printing. Therefore we will use the registration mark to synchronise the position of the draw each cycle

```
loop:
  DEFPOS(0)
  REGIST(3)
  MOVE(card_pitch)
  WAIT UNTIL MARK
  MOVEMODIFY(REG_POS+20)
  WAIT IDLE
  OP(punch,ON)
  WAIT UNTIL IN(punch_clear)=OFF
  WAIT UNTIL IN(punch_clear)=ON
  OP(punch,OFF)
GOTO loop
```

The above example shows only the simplest form of the main loop. It allows for a fixed offset value of 20, but there is no provision for error handling etc. An example where the code might be expanded to check for registration errors would be:

```
loop:
  DEFPOS(0)
  REGIST(3)
  MOVE(card_pitch)
  WAIT UNTIL MARK OR MTYPE=0
  IF MTYPE=0 THEN
    ' Indicate error to user
    PRINT #3,"Registration Error!"
    errors=errors+1
    if errors>max_errors then GOTO reg_failed
    OP(error_lamp,ON)
  ELSE
    OP(error_lamp,OFF)
    MOVEMODIFY(reg_pos+20)
    WAIT IDLE
  ENDIF
  ' Rest of loop as before
GOTO loop

reg_failed:
  PRINT #3,CURSOR(00);"Too many reg errors!"
  PRINT #3,CURSOR(20);"Press any key...    "
  GET #3,k
GOTO start
```

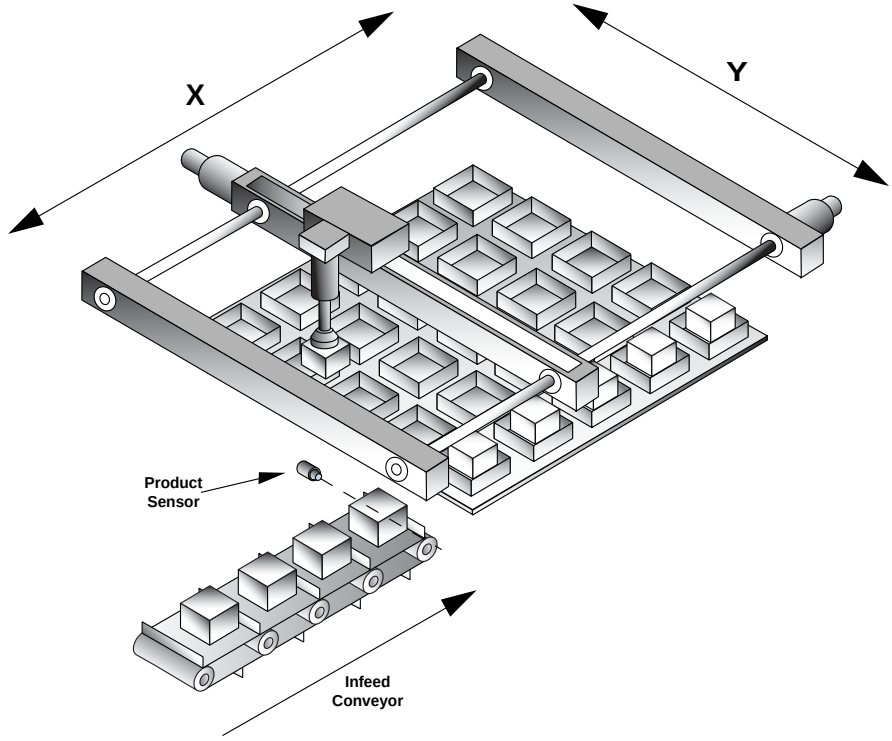
Example 4 - 2 Axis Pick & Place System

Overview

A square palette has sides 1200mm long.

It must be divided into a grid, each of these positions on the palette contains a box into which a widget must be placed:

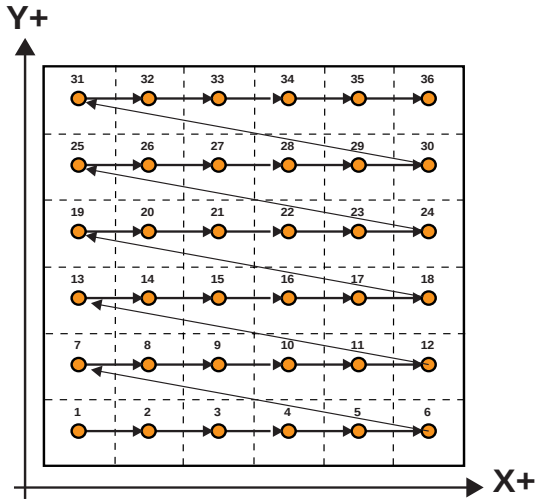
A vacuum operated pick-up mechanism collects objects from a conveyor and fills each of the boxes in turn.



Additional Information:

Whist the palette size is fixed at the maximum size of 1.2m square, the program should be flexible enough to allow for a user-defined number of boxes on the pallette. The grid could contain up to 10 divisions in each direction and they may be combined in any ratio, i.e. 2x4, 6x3, 1x10, 9x4 etc.

The illustration below shows a sample palette with a 6x6 grid of boxes. The numbers/arrows show the order in which the boxes are filled. Note that we step through the rows (Y axis) in turn, filling each box (move along X axis) before moving onto the next row.



Structuring the program

This example can be solved with a very simple structure using two nested **FOR..NEXT** loops.

Firstly we create a loop to step through each row (Y) in turn:

```
FOR y=0 to ydiv-1

NEXT y
```

'ydiv' is the number of rows, and the -1 is because we start counting at 0 rather than 1.

Now, within this loop we create another for the 'X' direction:

```
FOR y=0 to ydiv-1
  FOR x=0 to xdiv-1
    '
  NEXT x
NEXT y
```

Calculating the box positions

So now we have a sequence which steps sequentially through each row, and then through each position on that row in turn. We can use the absolute move (**MOVEABS**) command to position the axes at an absolute position in our X/Y coordinate system in the form

```
MOVEABS(x,y)
```

The x and y variables with the **FOR..NEXT** loop are simply logical box coordinates and therefore need to be scaled to the correct positions.

If we know the palette size (1200) and the number of divisions in each direction (xdiv/ydiv) then we can simply calculate an appropriate scaling, thus for the x axis:

```
xscale = 1200/xdiv
```

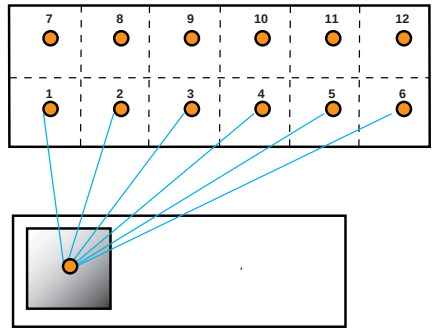
The actual position (of the box's corner) would therefore be (x*xscale), we could adjust this for the centre of the box by adding half the box size (xscale/2). So the position would be:

```
(x*xscale)+(xscale/2)
```

Our final consideration is that for each box we first have to move to the preset pick-up position, fetch the product and then move to the appropriate empty box to place the product in.

The pick up point is at a known absolute position and so we can simply use a pair of constants (pick_x and pick_y) to reference this point.

```
MOVEABS(pick_x, pick_y)
```



```
constants:
  nozzle=8 '    output - nozzle raise/lower
  vacuum=9 '    output - vacuum on / off

  xdiv=6
  ydiv=6

start:
  xscale=1200/xdiv
  xmid=xscale/2
  yscale=1200/ydiv
  ymid=yscale/2

  FOR y=0 TO xdiv-1
    FOR y=0 TO ydiv-1
    GOSUB pick
       MOVEABS((x*xscale)+xmid,(y*yscale)+ymid)
       WAIT IDLE
       GOSUB place
    NEXT x
  NEXT y
GOTO start

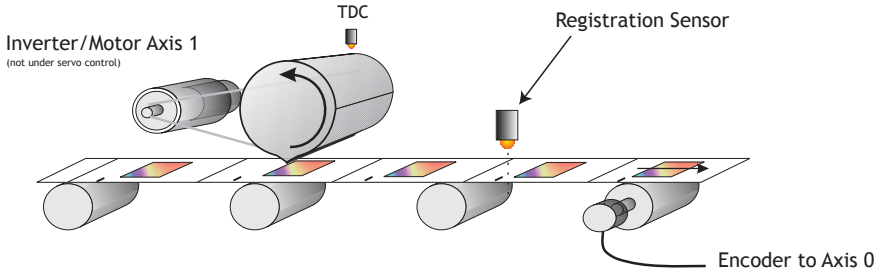
pick:
  MOVEABS(pick_x, pick_y)
  WAIT IDLE

  OP(nozzle,ON)
  OP(vacuum,ON)
  wa(500)
  OP(nozzle,OFF)
  wa(500)
RETURN

place:
  OP(nozzle,ON)
  OP(vacuum,OFF)
  wa(250)
  OP(nozzle,OFF)
  wa(500)
RETURN
```

Example 5 - Rotating Print Head with Registration

Description A rotating print head 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 print head 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 print head. 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.

Program Listing

```
'  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)
    ‘ 75 is the diff. between the mark spacing and
    ‘ the print head circumference
    ' Move linked to conveyor
    WAIT IDLE
    BASE(0)
    REGIST(3)
  GOTO loop

  initial:
    BASE(0)'   Setup axis 0
    UNITS=20'  Edges/mm
    P_GAIN=0.5
    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 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
```

Example 6 - *Motion Coordinator* programs sharing data

Description These two programs run multi-tasking on a *Motion Coordinator*. 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

Operator Interface Program

```
' Motion Coordinator 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 getnum
  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 "
  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
```

Motion Cycle Program

```
'
' Motion Cycle 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
```

Example 7 - Handling Axis Errors

The *Motion Coordinator* controllers are designed to trap error conditions in hardware, and if required to automatically open the drive enable relay (watchdog) and to disable the output to the drives.

As this mechanism happens automatically, it may not be immediately apparent that an error has occurred and therefore we need a mechanism in the software to recognise it, and to set up the type of errors which will cause the controller to disable the drive / output.

The relevant parameters are:

- **AXISSTATUS**
- **ERRORMASK**
- **MOTION_ERROR**
- **ERROR_AXIS**

```
start:
   ' Monitor constantly until axis error occurs
   ' Set ERRORMASK so that Following Errors and Fwd/Rev limit
   ' switches will automatically trip the watchdog relay
  ERRORMASK = 256+16+32

REPEAT
  IF MOTION_ERROR<>0 THEN
    ax = ERROR_AXIS
    BASE(ax)
    PRINT #3,CURSOR(0);"Error on Axis ";ax[0]
    IF (AXISSTATUS AND 256)>0 THEN PRINT #3,"Fol. Error";
    IF (AXISSTATUS AND (16+32))>0 THEN PRINT #3,"H/W Limit";
    IF (AXISSTATUS AND 512+1024)>0 THEN PRINT #3,"S/W Limit";
  ENDIF

  WAIT UNTIL KEY#3
  GET #3,k
GOTO start
```

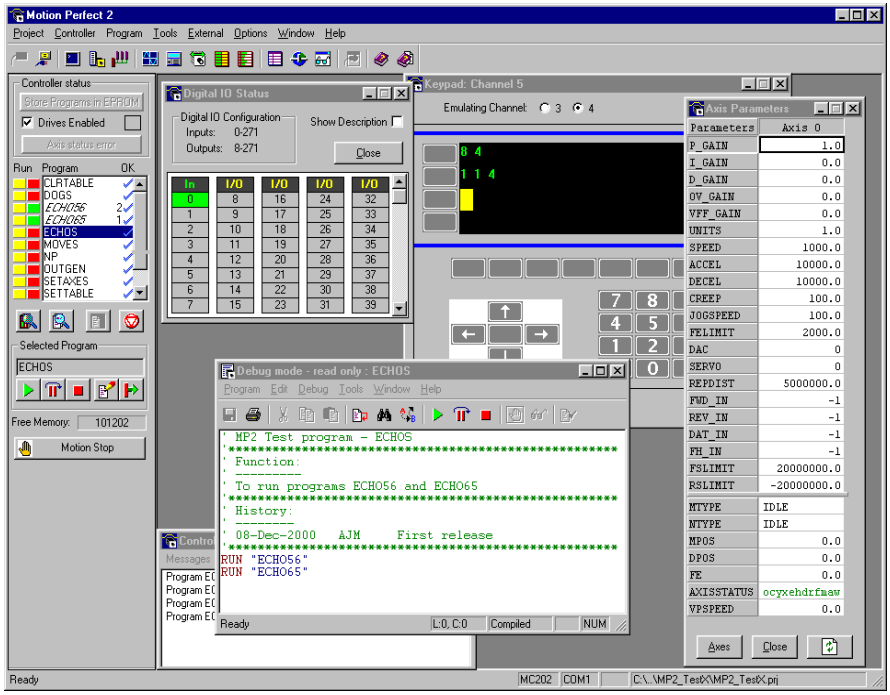
CHAPTER

10

SUPPORT SOFTWARE

Motion Perfect 2

Motion Perfect 2 is an application for the PC, designed to be used in conjunction with the *Motion Coordinator* range of multi tasking motion controllers.



Motion Perfect provides the user with an easy to use Windows based interface for controller configuration, rapid application development, and run-time diagnostics of processes running on the *Motion Coordinator*.

System Requirements:

The following equipment is required to use *Motion* Perfect 2.

PC

	Minimum Specification	Recommended
CPU	Pentium class processor, operating at 75MHz	Pentium class processor, operating at 200MHz
RAM	16 Mb	32Mb RAM (64Mb for Windows 2000)
Hard disk space	10 Mb	10 Mb
	For Serial Operation: Windows 95 or Windows NT v4 **For USB Communications:** Windows 98, Windows ME, or Windows 2000	Windows 98 SE, Windows ME, Windows 2000 or Windows XP.
Display	800 x 600 256 colours	1024 x 768 16-bit colour
Communications	Single RS232 Serial Port	RS232 serial port, USB port
Other	Mouse or similar pointing device	

Motion Coordinator

- *Motion Coordinator* controller or compatible controllers.

  Compatible controllers include:
  MC2, MC202, MC204, MC402e, Euro 205, MC206 and MC216

In order to use the Packet Communications mode, system software version 1.49 or higher is required.

Note: *You should always try to use the most recent version of* ***Motion Perfect****. Updates are available from your local distributor or you can download the latest version from the Trio Web site:* **WWW.TRIOMOTION.COM**

New Features for *Motion* Perfect 1 Users

If you have previously used the first version of *Motion* Perfect you will find version 2 very familiar. We have maintained a core functionality and working environment as close to the original as possible whilst adding a number of new and enhanced features.

New in Version 2

- Full Windows 32 bit Application
- Designed for 32 Bit Windows (Windows 95,98, ME, and Windows 2000)
- Full Support for Latest Trio Products
- Supports controller locking for project security
- Supports Feature Enable Codes for the Euro 205 and MC206
- New Editor Features:
  Syntax highlighting
  Error highlighting
  Offline program editing
  Windows restored after connection
- Links to External Applications
- Integrated support for CAD2Motion, DocMaker
- Enhanced Communications Features
  Protected RS-232 communications mode
  USB support for high speed communications
  Motion Coordinator simulation
- New/Enhanced Tools
- TABLE Editor
- Variable Editor
- Load / Save TABLE files
- Resizable Oscilloscope

Connecting *Motion* Perfect to a controller

Motion Perfect can be connected to the *Motion Coordinator* using a serial connection, or USB if fitted to the controller. A suitable serial cable can be supplied by Trio Motion Technology. *Motion* Perfect may use either of the standard serial ports on a PC, COM1 or COM2, but it will connect quicker to your system if COM1 is used as this is the first port it tries.

If you wish to edit a project but do not have a controller connected to your PC, it is possible to edit off-line by connecting to a 'virtual' controller running on your PC. See the information on the MCSimulation at the end of this chapter.

Running *Motion* Perfect 2 for the First time

Turn your PC on and enter Windows. Make sure the *Motion Coordinator* is turned on and then launch *Motion* Perfect. During initialisation you will see a splash screen such as the one below.

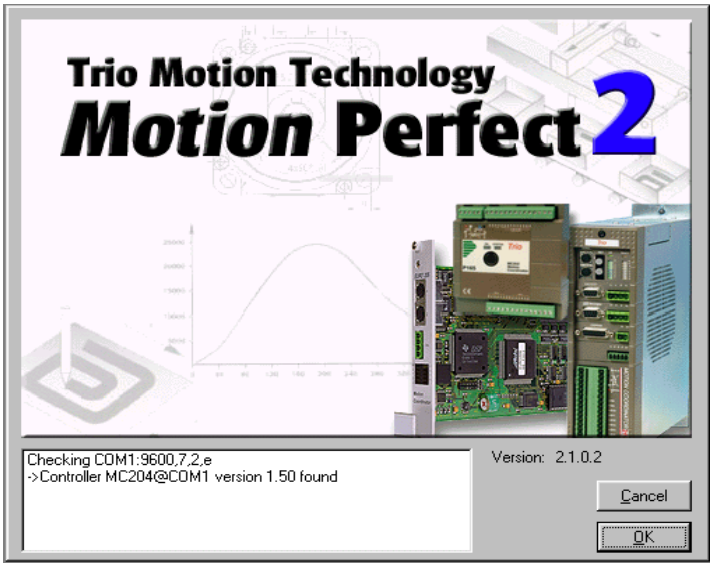


Figure 1: Splash screen

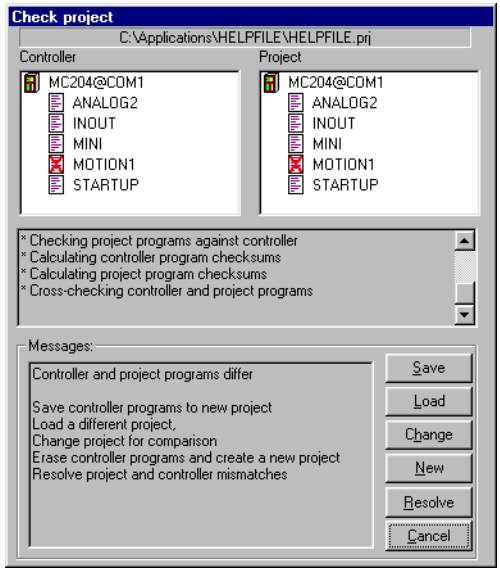
The splash screen features a small messages window (bottom left) which is used to display the status of the connection process. In this example *Motion* Perfect is connected to an MC204 controller via serial port COM1.

Motion Perfect 2 Projects

One of the keys to using *Motion* Perfect is to understand its concept of a "Project". The project facilitates the application design and development process, by providing a disk based copy of the multiple controller programs, parameters and data which may be used for a single motion application. Once the user has defined a project, *Motion* Perfect works behind the scenes automatically maintaining consistency between the programs on the controller and the files on the PC. When creating or editing programs on the controller they are automatically duplicated on the PC which means you do not have to worry about loading or saving programs and you can be confident that next time you connect to the controller you will have the correct information on your PC .

The Project Check Window

Whenever you connect to the controller, *Motion* Perfect will perform a **project check** to compare the programs on the controller with those defined in the current project on the pc. During the project check a window similar to the one below will be displayed. If the projects match then you will see a "project checked ok" message and an OK button to continue. If however there is any inconsistency between the controller and the PC, the display will feature a number of addition options, shown below.



You can force *Motion* Perfect to perform a project check at any time with the "Check Project" option from the project menu. (Ctrl+Alt+P)

Project Check Options

Save **Save the controller contents to disk.**

If you have never connected with this controller before, and therefore do not have the project on your PC, or if there in an inconsistency in the project check and you are sure that the project on the controller is the correct version, then select SAVE to copy the programs on the controller to disk.

Note: This will of course overwrite any programs already in the PC copy of the project. If you are unsure which is the correct versiom, you should save the project with a new name to avoid overwriting any existing project programs on the PC.

Load **Load the PC files onto the controller**

If you are uploading a complete project from the PC to the controller, or the project check fails and you are sure the version on your PC is correct, then you should use this option to upload the entire project from the PC to the controller.

Note: The entire contents of the programs on the controller will be erased. If you are unsure, SAVE the controller contents first!

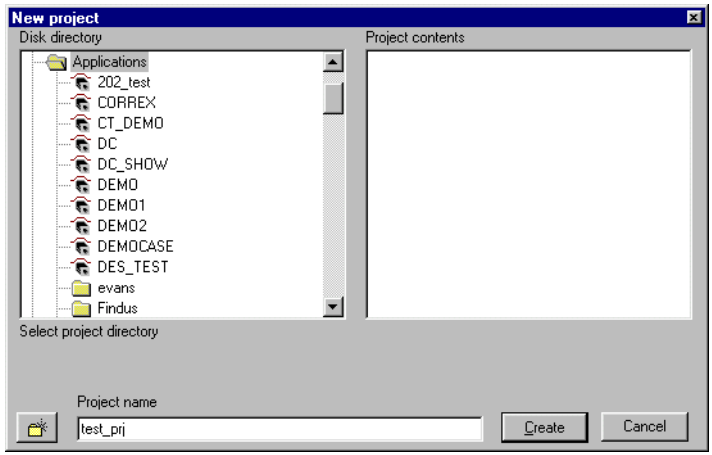
Change **Change the project on the PC to compare with..**

If you have been working on more than one project, the project on the controller may not match the 'last project' remembered by *Motion* Perfect. If this is the case you can use this option to select another project on the PC. Once you select an alternative, *Motion* Perfect will perform a fresh project check and the above process will be repeated.

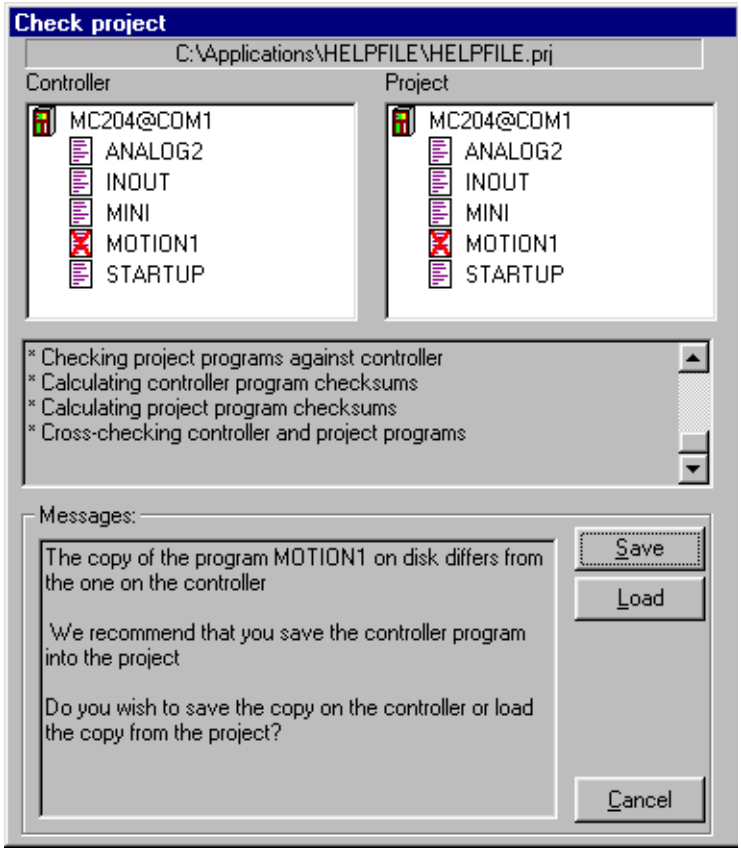
New **Create a new project**

The controller contents will be erased and a new project created on the PC. You will be prompted to select a directory and project name.

When you create a new project, *Motion* Perfect will make a new directory with the project name, and within that directory a project file with the same name (the .PRJ extension is added to the filename).



Resolve This option should be used when you have the correct project selected, but one or more of the files differ between the controller and PC version, or do not exist in one of the copies.

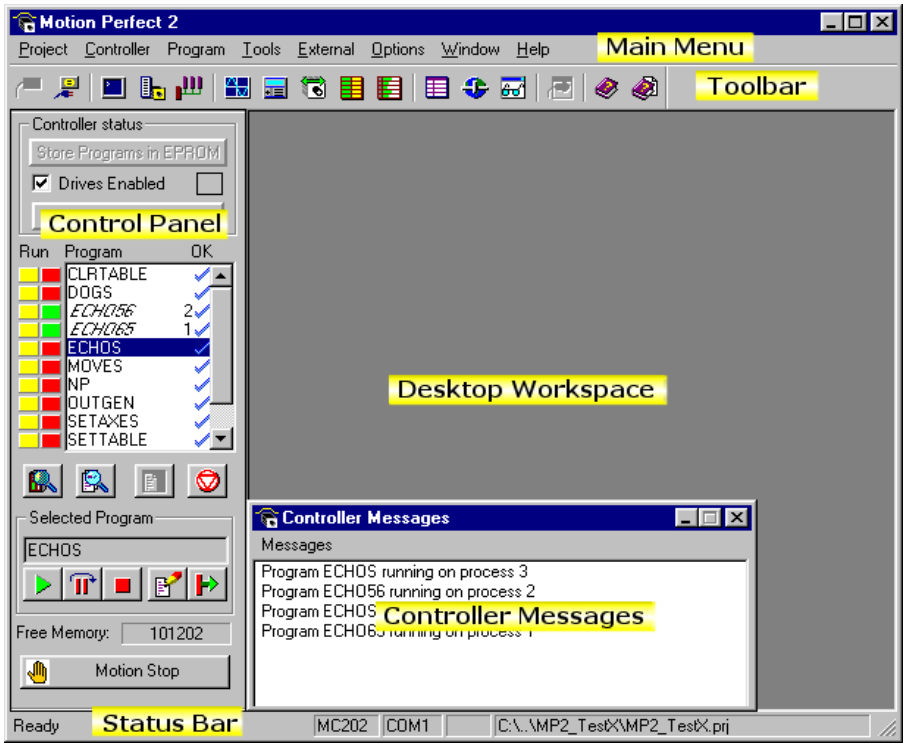


You will need to use your judgment to decide whether the disk or controller version is correct. Typically, if you are recovering the project after a comms failure or PC crash then the version on the controller should be saved. If you have modified the disk based copy of the program then you will need to load this version onto the controller.

Cancel Cancels the connection process and starts *Motion* Perfect in disconnected mode.

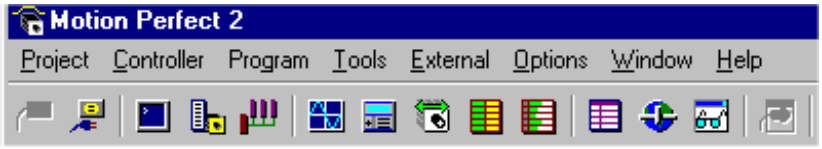
Once the project has been checked and is consistent then a backup copy of the PC project will be created.

The *Motion* Perfect Desktop



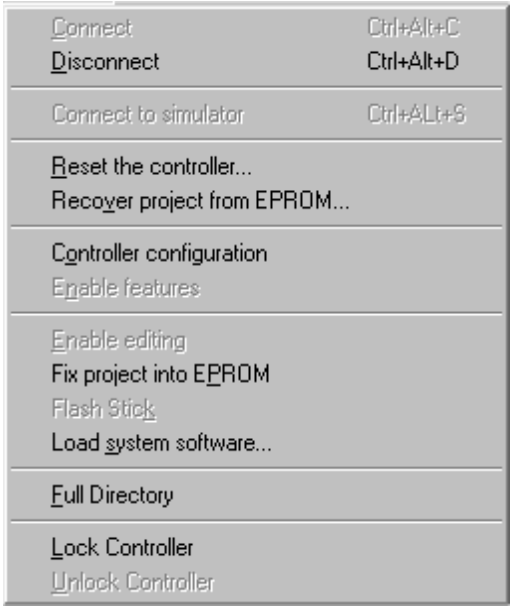
Main Menu	Standard Windows menu to access all features of the *Motion* Perfect application.
Toolbar	Shortcut buttons to access the *Motion* Perfect tools
Control Panel	Displays the current controller contents and provides controls for interrogating the controller status, running / editing programs
Desktop Workspace	This area is used to display the user windows and tools
Controller Messages	Status and error messages reported by the controller
Status Bar	Information about the current project and controller connection.

Main Menu



Project	Options for Creating, Loading & Saving *Motion* Perfect Projects, Loading/Saving program files and Table data
Controller	Options relating to the controller hardware, including connecting/disconnecting and checking configuration information.
Program	Program specific options, including creating, editing and running controller tasks.
Tools	Access to the main *Motion* Perfect tools. These options are also available from the Toolbar
Options	Configure the *Motion* Perfect Environment. Includes options to setup the communications ports and to customise the editor display.
Window	Control the appearance of the *Motion* Perfect desktop.
Help	Access the help files and version information.

Controller Menu

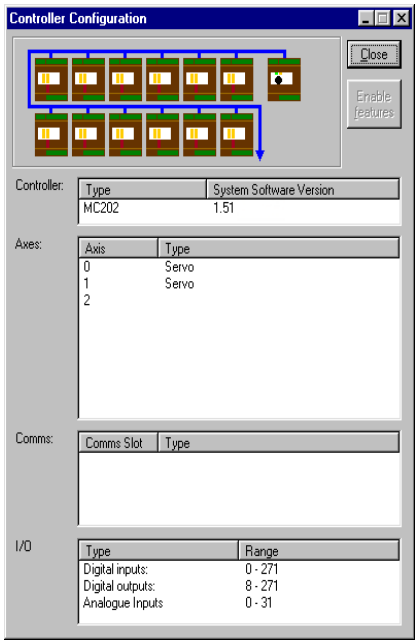


The controller menu contains the following items:

Connect	Connect using serial communications to the controller and start the project manager. This is only available if *Motion* Perfect is currently disconnected from the controller.
Disconnect	Disconnect the serial communications, and stop using the project tools. Only available if *Motion* Perfect 2 is currently connected to the controller.
Reset Controller	Perform a software-reset (EX) on the controller. This will cause *Motion* Perfect 2 to disconnect from the controller
Recover Project from EPROM	Reset the controller and restore the programs which were previously stored in the EPROM

Controller Configuration

This screen interrogates the hardware and displays the configuration information reported back by the controller.



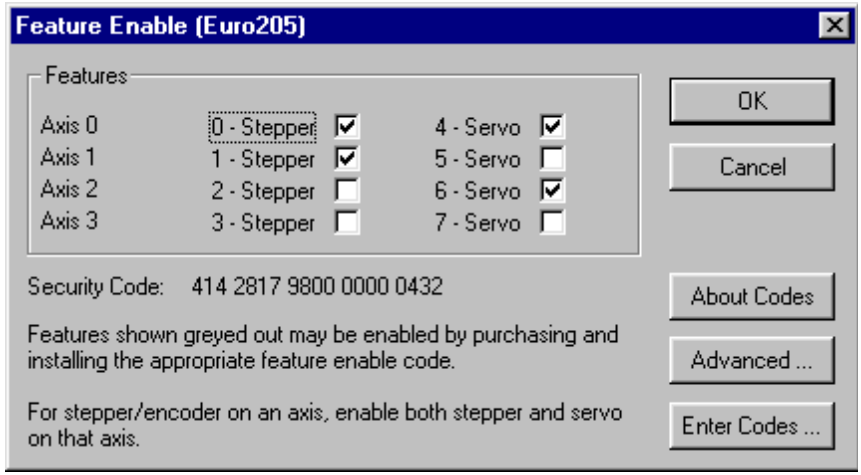
Looking at the example screen shown here from top to bottom:

Controller:	We are connected to a *Motion Coordinator* MC202
Software Version:	The controller is running version 1.51 of the system software.
Axis Types	A list of the types of all available axes.
Comms Boards	If the controller is fitted with any of the extended / communications daughter boards, that capability will be indicated here.
I/O	The channel range available for each type of I/O. Remember that on many *Motion Coordinator*s the channels are shared, i.e. if Output 15 is available, then it implies that Input 15 is also available and shares the same connector.

Feature Enable

Certain *Motion Coordinator*s, such as the EURO205 and MC206, have the ability to unlock additional axes by entering a "Feature Enable Code".

When you access the Feature Enable dialog, you will be presented with a display similar to the following:



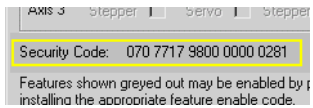
This display illustrates the feature which are currently available. If the codes for additional features have been purchased, the relevant boxes will be available for checking.

Enabling Additional Features

To enable a feature you must enter a Feature Enable Code, which is unique to each controller and feature. To obtain a Feature Enable Code, you will need to specify the feature required and the security code for the specific controller to be updated. The order for the required codes should be FAXed to Trio or an authorised Trio distributor.

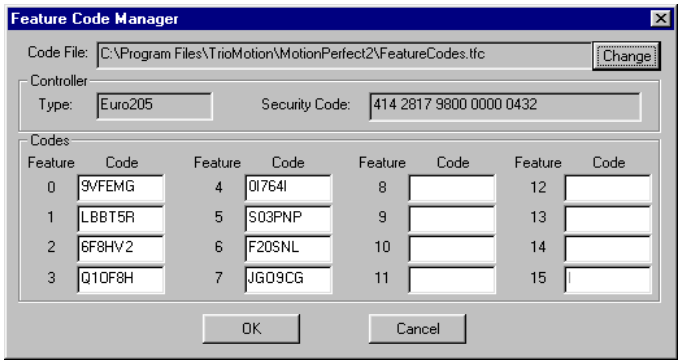
Security Code

Controllers with features which can be enabled each have a unique security code number which is implanted when the unit is manufactured. This security code number is displayed on the above screen (as highlighted right).



Once you have the required codes, select the  button.

A dialog similar to the following example will appear.



Each feature requested has a feature number. Enter the relevant code for each feature number, being careful to enter the characters in upper case. Take care to check that 0 (zero) is not confused with O and 1 (one) is not confused with the letter I.

Feature Code File

Motion Perfect stores all of the Feature Enable Codes of which it is aware in a file called "**FeatureCodes.TFC**". By default this file is located in the same directory as the *Motion* Perfect 2 executable file.

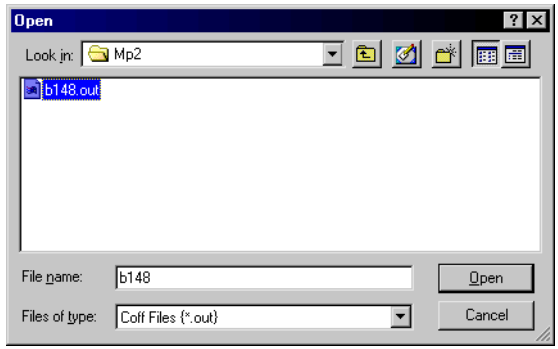
Enable Editing	Restore the power-up state of a controller currently starting from EPROM to run from RAM and allow editing.
Fix Project into EPROM	Store the programs in RAM into the controllers flash-eprom memory. The startup state for each program will not be changed.
Full Directory	Display a complete listing of all files on the controller, details of memory used and the run status of each program.

Loading New System Software

*Motion Coordinator*s feature a flash eprom for storage of both user programs and the system software. From *Motion* Perfect 2 it is possible to upgrade the software to a newer version using a system file supplied by Trio.

NOTE: ***We do not advise that you load a new version of the system software unless you are specifically advised to do so by your distributor or by Trio.***

When you select the 'Load System Software' option from the controller menu, you will first be presented with a warning dialog to ensure you have saved your project and are sure you wish to continue. When you press OK you will be presented with the standard Windows file selector to choose the file you wish to load.



Each *Motion Coordinator* controller has its own system file, identified by the first letter of the file name.

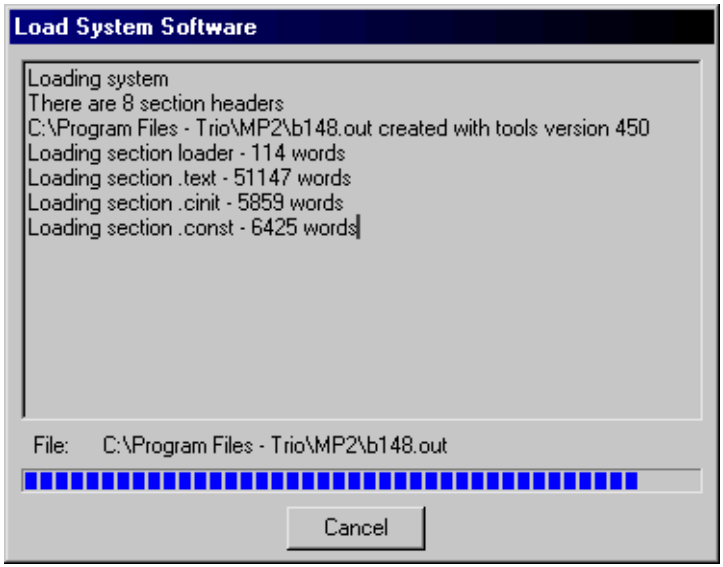
System Software File Prefix Codes:

Filename	Controller Type
`Annn.OUT`	MC2
`Bnnn.OUT`	MC204
`Cnnn.OUT`	MC216
`Dnnn.OUT`	MC402
`Ennn.OUT`	Euro205
`Fnnn.OUT`	MC202 / Motion Module
`Gnnn.OUT`	MCW202
`Hnnn.OUT`	MC402e
`Jnnn.OUT`	MC206

You must ensure that you load only software designed for your specific controller, other versions will not work

When you have chosen the appropriate file you will be prompted once again to check that you wish to continue. Press OK to start the download process.

Downloading will take several minutes, depending on the speed of your PC. During the download, you should see the progress of each section updated as follows:-

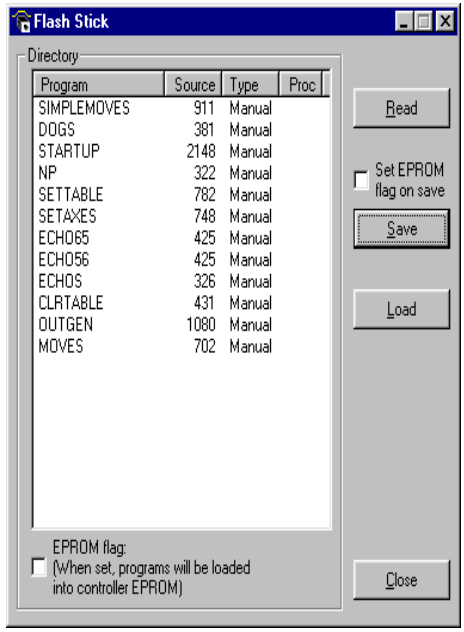


When the download is complete, a checksum is performed to ensure that the download process was successful. If it saw you will be presented with a confirmation screen and asked if you wish to store the software into Eprom.

When you press Yes, the controller will take a few moments to fix the project into the eprom and you can then continue as normal.

At this point you can check the controller configuration to confirm the new software version.

Flashstick support



This only applies to controllers which are fitted with a flashstick socket (eg MC206).

When a controller with flashstick support is powered on with a flashstick inserted, then the controller will automatically load the programs from the flashstick into the controller RAM.

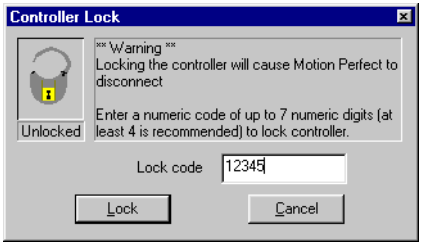
The **Read** button will read the directory of the flashstick and display it. As the directory is automatically read when the tool window is created, this button only needs to be used when the flashstick is changed.

The **Save** button will save the programs on the controller to the flashstick eassing any programs already stored on the flashstick. If the **set EPROM fag on save** box is checked then a flag is set on the flashstick which makes the controller store the programs on the flashstick in controller EPROM as well as in controller RAM at power-up.

The **Load** button will load the programs from the flashstick onto the controller. This is done by resetting the controller.

Lock / Unlock

Lock Controller

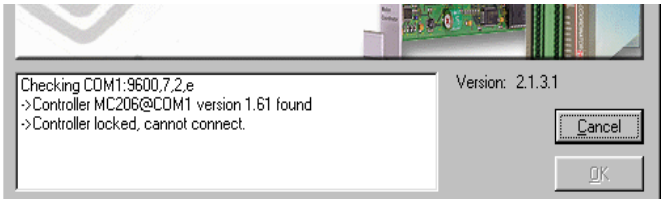


Locking the controller will prevent any unauthorised user from viewing or modifying the programs in memory.

You simply need to enter a numeric code (up to 7 digits). This value will be encoded by the system and used to lock the directory structure. The lock code is held in encrypted form in the flash memory of the *Motion Coordinator*.

This can also be achieved by issuing the LOCK(lock_code) command from the controller's command line.

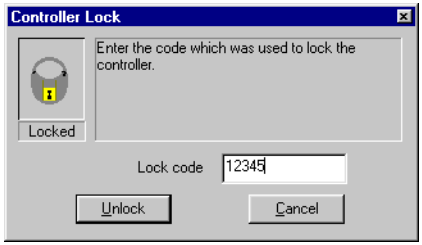
Once the *Motion Coordinator* is locked it is not possible to list, edit or save any of the controller programs. You cannot connect to the controller with *Motion* Perfect 2, although the terminal screen and unlock dialog will still be available.



WARNING: ***If you forget the lock code there is no way to unlock the controller. You will need to return it to Trio or a distributor to have the lock removed.***

Unlock Controller

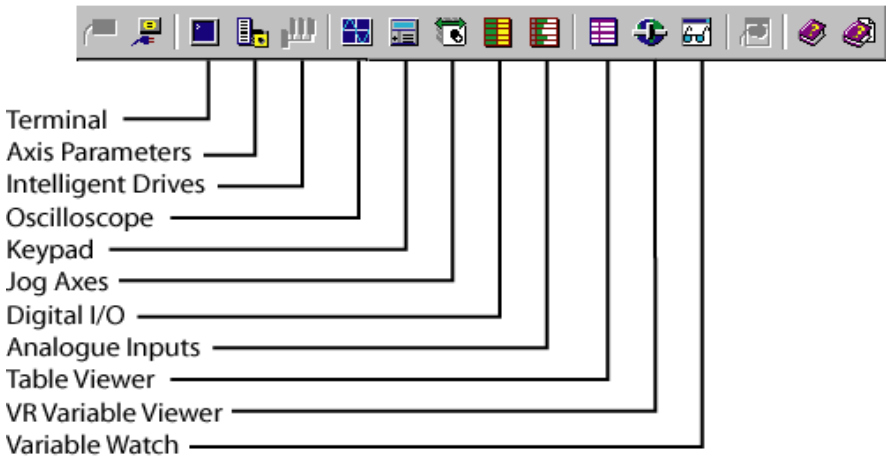
In order to unlock the controller you need to enter the same numeric code which was used to lock it. Once the unlock code is entered it will be possible to gain full access to the programs in memory.



Motion Perfect Tools

The *Motion* Perfect tools can be accessed from either the Tools Menu or the Tool-bar buttons

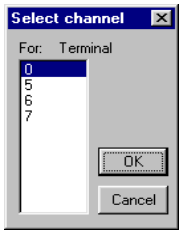




Terminal

The terminal window provides a direct connection to the *Motion Coordinator*. Most of the functions that must be performed during the installation, programming and commissioning of a system with a *Motion Coordinator* have been automated by the options available in the *Motion* Perfect menu options. However, if direct intervention is required the terminal window may be used.

Selecting a communications channel

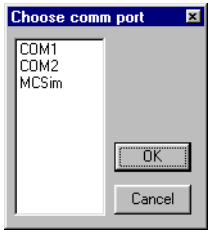


If *Motion* Perfect is connected to the controller, you will be prompted to select from any of the active user communications channels:

For this example we will select channel 0 which is used for the *Motion Coordinator* Command Line interface.

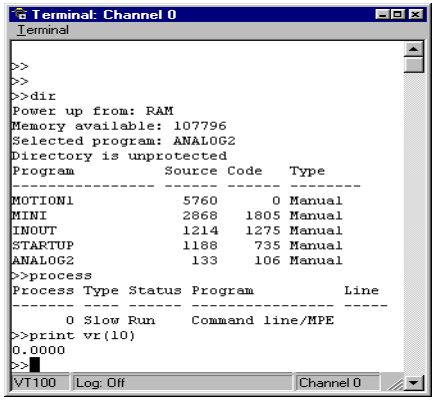
Using the terminal without a *Motion* Perfect Connection

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”. In this mode the terminal window has complete control over the serial link.

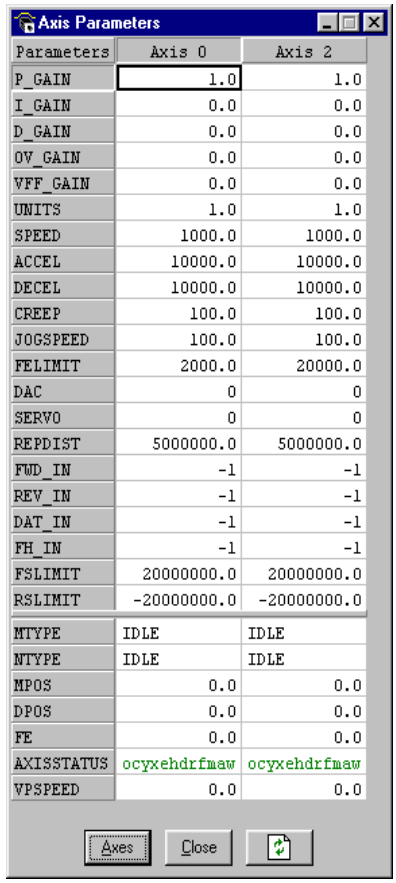


Once connected to the controller, the terminal window opens a communications channel to the controller. The user must select the required channel.

If this channel is already in use, either by another terminal window or a keypad emulation then access will be denied.



Axis Parameters



The Axis Parameters window enables you to monitor and change the motion parameters for any axis on the controller.

The window is made up of a number of cells, separated into two banks, bank 1 at the top and bank 2 at the bottom:

Bank 1. contains the values of parameters that may be changed by the user.

Bank 2. contains the values of parameters that cannot be changed by the user, as these values are set by the system software of the *Motion Coordinator* as it processes the Trio BASIC motion commands and monitors the status of the external inputs.

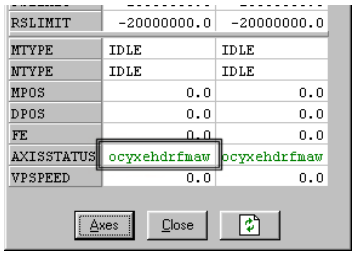
The black dividing bar that separates the two banks may be repositioned using the mouse to redistribute the space occupied by the different banks, for example to allow the user to shrink the window and view other windows whilst still watching the bank 2 information.

When there are more parameters in a bank that can be shown in the window a scroll bar will appear beside that bank so that the user can scroll up and down the parameter list to see the required values.

The user can select different parameters using the cursor keys or using the mouse. Multiple items may be selected by pressing the shift key and then using the cursor keys or the clicking the mouse to select a different cell, or by pressing the left mouse button in the start cell and the moving the mouse to select the last cell in the selection. Functions may be implemented in the future that work on a selection of multiples cells.

When the user changes the **UNITS** parameter for any axis, all the data for this axis is re-read as many of the parameters, such as the **SPEED**, **ACCEL**, **MPOS**, etc., are adjusted by this factor to be shown in user units.

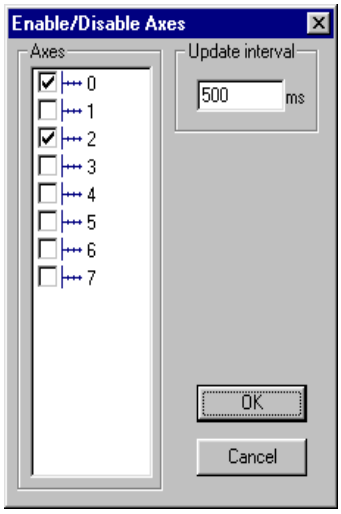
In the *Motion* Perfect parameter screen the **AXISSTATUS** parameter is displayed as a series of characters, **ocyxehdrfmaw.**



These characters represent **AXISSTATUS** bits in order, as follows:-

char	status bit
w	Warning FE Range
a	Drive Comms Error
m	Remote Drive Error
f	Forward Limit
r	Reverse Limit
d	Datum Input
h	Feed Hold Input
e	Following Error
x	Forward Soft Limit
y	Reverse Soft Limit
c	Cancelling Move
o	Encoder Overcurrent

Parameter Screen Options





Select Axes

This shows a dialog that allows the user to select the axes for which the data will be displayed.

The axes set by the last Create Startup, Jog Axes window or Axes Parameters window will be displayed by default.

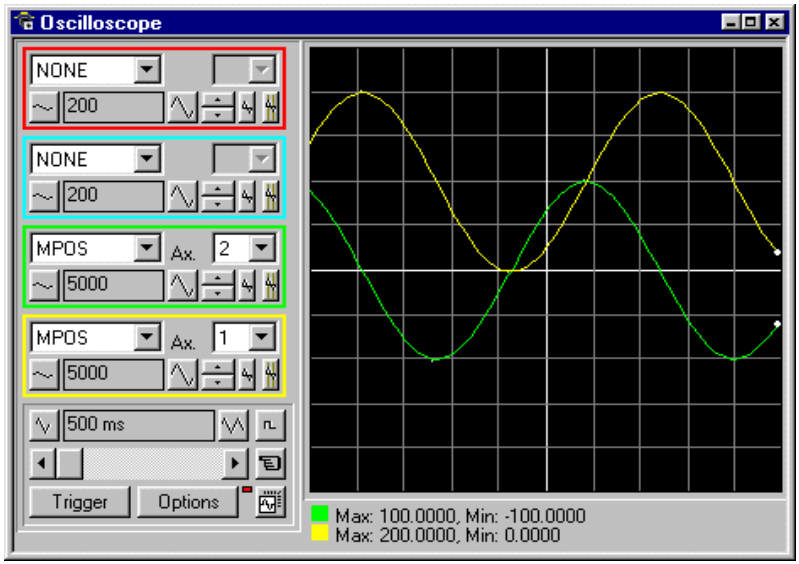


Refresh Display

In order to minimise the load placed upon the controller communications, the parameters in the bank 1 section are only read when the screen is first displayed or the parameter is edited by the user. It is possible that if a parameter is changed in a user program then value displayed may be incorrect. The refresh button will force *Motion* Perfect to read the whole selection again.

Note ***If there is any possibility that a program has changed any of the parameters then you should ensure that your refresh the display before making changes.***

Oscilloscope



The software oscilloscope can be used to trace axis and motion parameters, aiding program development and machine commissioning.

There are four channels, each capable of recording at up to 1000 samples/sec, with manual cycling or program linked triggering.

The controller records the data at the selected frequency, and then uploads the information to the oscilloscope to be displayed. If a larger time base value is used, the data is retrieved in sections, and the trace is seen to be plotted in sections across the display. Exactly when the controller starts to record the required data depends upon whether it is in manual or program trigger mode. In program mode, it starts to record data when it encounters a `TRIGGER` instruction in a program running on the controller. However, in manual mode it starts recording data immediately.

Controls

The oscilloscope controls are organised into four channel specific control blocks, followed by the oscilloscope general controls.

Oscilloscope Channel Controls



Each oscilloscope 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 oscilloscope can record and display are selected using the pull-down list box in the upper left hand corner of each channel control block. Depending upon the parameter chosen, the next label switches between 'axis' or 'ch' (channel). This leads to the second pull-down list box which enables the user to select the required axis for a motion parameter, or channel for a digital input/output or 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 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 oscilloscope channel is not in use, 'axis' if an axis parameter has been selected, or 'ch' if a channel parameter has been selected.

Vertical Scaling



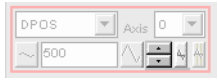
The vertical scale (units per grid division on the display) are selected per channel, and these can be configured in either automatic or manual mode.

In automatic mode the oscilloscope calculates the most appropriate scale when it has finished running, prior to displaying the trace. Hence if the oscilloscope 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.

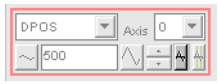
Channel Trace Vertical Offset



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 scroll bars can be cleared by pressing this vertical offset reset button.

The button latches on/off. When the button is latched ON then the vertical offset will automatically rescale each time the oscilloscope display is redrawn.

Cursor Bars



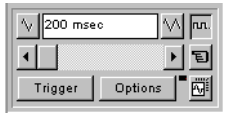
After the oscilloscope has finished running, and has displayed a trace, cursor bars can be enabled. These are displayed as two vertical bars, of the same colour as the channel trace, and initially located at the maximum and minimum trace location points. The values these represent are shown below the oscilloscope display, and again the text is of the same colour as the channel values represented.

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, 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.

Oscilloscope General Controls

The oscilloscope general controls appear at the bottom left of the oscilloscope window. From here you can control such aspect as the time base, triggering modes and memory used for the captured data.



Time Base



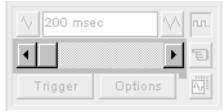
The required time base is selected using the up/down scale buttons either side of the current time base scale text box (left side button decreases the scale, and the right side button increases the scale value.) The value selected is the time per grid division on the display.

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 compete trace of data at one time.) These sections of data are plotted on the display as they are received, and the last point plotted is seen as a white spot.

After the oscilloscope has finished running and a trace has been displayed, the time base scale may be changed to view the trace with respect to different horizontal time scales. If the time base scale is reduced, a section of the trace can be viewed in greater detail, with access provided to the complete trace by moving the horizontal scrollbar.

Horizontal scrollbar

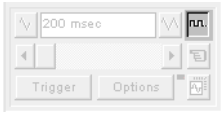
Once the oscilloscope has finished running and displayed the trace of the recorded data, if the time base is changed to a faster value, only part of the trace is displayed. The remainder can be viewed by moving the thumb box on the horizontal scrollbar.



Additionally, if the oscilloscope is configured to record both motion parameters and plot table data, then the number of points plotted across the display can be determined by the motion parameter. If there are additional table points not visible, these can be brought into view by scrolling the table trace using the horizontal scrollbar. The motion parameter trace does not move.

One Shot / Continuous Trigger Mode

Button Raised = One Shot Trigger Mode:

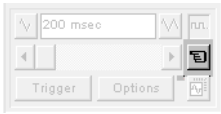


In one-shot mode, the oscilloscope runs until it has been triggered and one set of data recorded by the controller, retrieved and displayed.

Button Pressed = Continuous Trigger Mode:

In continuous mode the oscilloscope continues running and retrieving data from the controller each time it is re-triggered and new data is recorded. The oscilloscope continues to run until the trigger button is pressed for a second time.

Manual/Program Trigger Mode



The manual/program trigger mode button toggles between these two modes. When pressed, the oscilloscope is set to trigger in the program mode, and two program listings can be seen on the button. When raised, the oscilloscope is set to the manual trigger mode, and a pointing hand can be seen on the button.

Button Raised = Manual Trigger Mode:

In manual mode, the controller is triggered, and starts to record data immediately the oscilloscope trigger button is pressed.

Button Depressed = Program Trigger Mode:

In program mode the oscilloscope starts running when the trigger button is pressed, but the controller does not start to record data until a TRIGGER instruction is executed by a program running on the controller. After the trigger instruction is executed by the program, and the controller has recorded the required data. The required data is retrieved by the oscilloscope and displayed.

The oscilloscope stops running if in one-shot mode, or it waits for the next trigger on the controller if in continuous mode

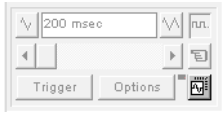
Trigger Button



When the trigger button is pressed the oscilloscope is enabled. If it is manual mode the controller immediately commences recording data. If it is in program mode then it waits until it encounters a trigger command in a running program.

After the trigger button has been pressed, the text on the button changes to 'Halt' whilst the oscilloscope is running. If the oscilloscope is in the one-shot mode, then after the data has been recorded and plotted on the display, the trigger button text returns to 'Trigger', indicating that the operation has been completed.The oscilloscope can be halted at any time when it is running, and the trigger button is displaying the 'Halt' text, by pressing this button.

Reset Oscilloscope 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.

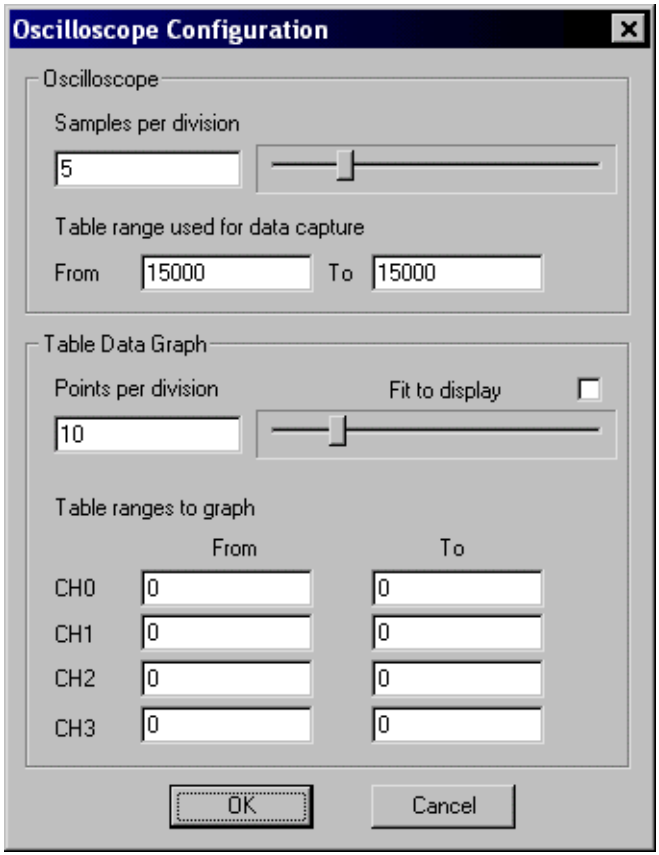
Status Indicator

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	oscilloscope 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 oscilloscope defaults to recording five points per horizontal (time base) grid division. This value can be adjusted using the adjacent scrollbar.

To achieve the fastest possible sample rate it is necessary to reduce the number of samples per grid division to 1, and increase the time base scale to its fastest value (1 servo period per grid division).

It should be noted that the trace might not be plotted completely to the right hand side of the display, depending upon the time base scale and number of samples per grid division.

Oscilloscope Table Values

The controller records the required parameter data values in the controller as table data prior to uploading these values to the scope. By default, the lowest oscilloscope table value used is zero. However, if this conflicts with programs running on the controller which might also require this section of the table, then the lower table value can be reset.

The lower table value is adjusted by setting focus to this text box and typing in the new value. The upper oscilloscope table value is subsequently automatically updated (this value cannot be changed by the user), based on the number of channels in use and the number of samples per grid division. If an attempt is made to enter a lower table value which causes the upper table value to exceed the maximum permitted value on the controller, then the original value is used by the oscilloscope.

Table Data Graph

It is possible to plot controller table values directly, in which case the table limit text boxes enable the user to enter up to four sets of first/last table indices.

Parameter Checks

If analog inputs are being recorded, then the fastest oscilloscope 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.

It is not possible to enter table channel vales in excess of the controllers maximum TABLE size, nor to enter a lower oscilloscope table value. Increasing the samples per grid division to a value which causes the upper oscilloscope table value to exceed the controller maximum table value is also not permitted.

If the number of samples per grid division is increased, and subsequently the time base scale is set to a faster value which causes an unobtainable resolution, the oscilloscope automatically resets the number of samples per grid division.

General Oscilloscope Information

Displaying Controller Table Points -

If the oscilloscope is configured for both table and motion parameters, then the number of points plotted across the display is determined by the time base (and samples per division). If the number of points to be plotted for the table parameter is greater than the number of points for the motion parameter, the additional table points are not displayed, but can be viewed by scrolling the table trace using the horizontal scrollbar. The motion parameter trace does not move.

Data Upload from the controller to the oscilloscope -

If the overall time base is greater than a predefined value, then the data is retrieved from the controller in blocks, hence the display can be seen to be updated in sections. The last point plotted in the current section is seen as a white spot.

If the oscilloscope is configured to record both motion parameters, and also to plot table data, then the table data is read back in one complete block, and then the motion parameters are read either continuously or in blocks (depending upon the time base).

Even if the oscilloscope is in continuous mode, the table data is not re-read, only the motion parameters are continuously read back from the controller.

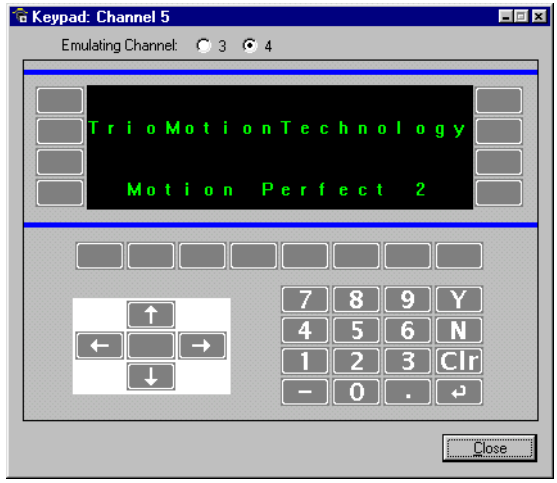
Enabling/Disabling of oscilloscope controls -

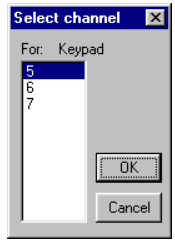
Whilst the oscilloscope is running all the oscilloscope controls except the trigger button are disabled. Hence, if it is necessary to change the time base or vertical scale, the oscilloscope must be halted and re-started.

Display accuracy -

The controller records the parameter values at the required sample rate in the table, and then passes the information to the oscilloscope. Hence the trace displayed is accurate with respect to the selected time base. However, there is a delay between when the data is recorded by the controller and when it is displayed on the oscilloscope due to the time taken to upload the data via the serial link.

Keypad Emulation





The keypad requires one of the user communications channels, and so you will be prompted for the channel to use.

If the specified channel is already in use, either by another keypad or a terminal window, the window will not open. Once a channel has been reserved then the keypad will be shown.

In the Trio BASIC program the channel definition for the commands that are associated with the Keypad must be changed from 3 (or 4) to the channel that corresponds with the channel selected for the emulation. We recommend that the channel assignment be made through a variable, so when time comes to run the program on the real machine, only one program change will be required.

example:

```
kpd=5
PRINT #kpd, "Press any key.."
```

Emulating Channel

The normal operation of the keypad emulation returns the characters as if they were read from channel #3 with the **DEFKEY** translation. Alternatively, the *Motion Coordinator* can read the characters returned directly from the Keypad using channel 4. If the emulate #4 codes is selected then the keypad emulation will return the raw characters.

Note: It is only possible to emulate the default **DEFKEY** table.

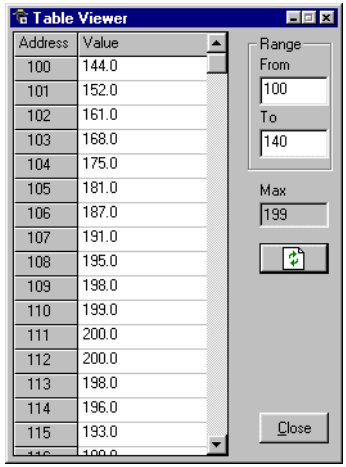
Key Functions

menu keys	This is a keypad menu key. Normally it is associated with a message on the display. This button can only be pressed by clicking the mouse over it.
function keys 1-8	This is the keypad function key 1. Normally it has an associated user label. This button can be pressed by clicking the mouse over it or using the '1' - '8' keys in the QWERTY area of the PC keyboard.
number keys	This is a keypad number key. It can be pressed by clinking the mouse over it or using the corresponding number in the numerical keypad of your PC keyboard.
Y/N keys	This is the keypad 'Y' and 'N' keys. This is usually used to respond YES or NO to some question on the display. It can be pressed by clicking the mouse over it or using the 'Y'/'N' keys in the QWERTY area of the PC keyboard.
CLR key	This is the keypad 'CLR' key. This is usually used to perform some form of CANCEL operation. It can be pressed by clicking the mouse over it or using the 'ESC' in the QWERTY area of the PC keyboard.
Return key	This is the keypad Return key. This is usually used to perform some form of ACCEPT operation. It can be pressed by clicking the mouse over it or using the 'Enter' in the QWERTY area or numerical keypad of the PC keyboard.
- key	This is the keypad '-' key. This is usually used for entering negative numbers. It can be pressed by clicking the mouse over it or using the '-' in the QWERTY area or numerical keypad of the PC keyboard.
. key	This is the keypad '.' key. This is usually used for entering fractional numbers. It can be pressed by clicking the mouse over it or using the '.' in the QWERTY area or numerical keypad of the PC keyboard.
arrow keys	This is the keypad up arrow key. This is usually used to select between options on the display. It can be pressed by clicking the mouse over it or using the appropriate arrow key of the PC keyboard.
centre button	This is the keypad centre key. It can only be pressed by clicking the mouse over it.

Table / VR Editor

The Table and VR Editor tools are very similar. You are presented with a spreadsheet style interface to view and modify a range of values in memory.

To modify a value, click on the existing value with the mouse and type in the new value and press return. The change will be immediate and can be made whilst programs are running.



Options

Range

In both tools you have the option to set the start and end of the range to view. In the Table view tool the max value displays the highest value you can read (this is the system parameter TSIZE).

If the range of values is larger than the dialog box can display, then the list will have a scrollbar to enable all the values to be seen.



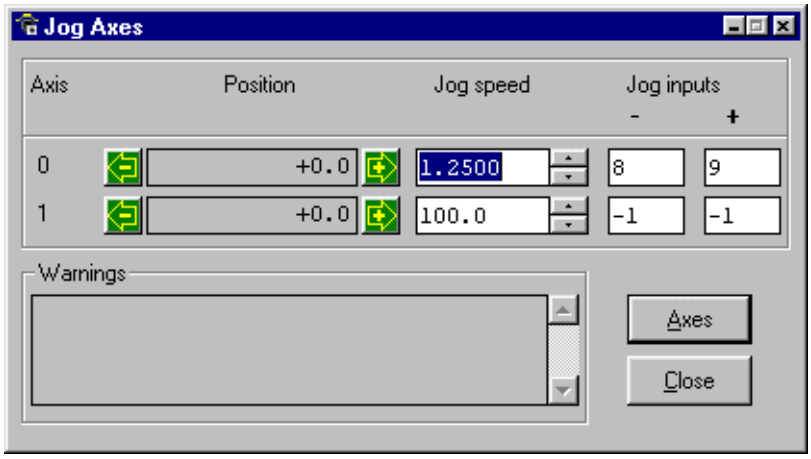
Refresh Button



This screen does not update automatically, so if a Table or VR is changed by the program you will not see the new value until you refresh the display.

Jog Axes

This window allows the user to move the axes on the *Motion Coordinator*.



This window takes advantage of the bi-directional I/O channels on the *Motion Coordinator* to set the jog inputs. The forward, reverse and fast jog inputs are identified by writing to the corresponding axis parameters and are expected to be connected to NC switches. This means that when the input is on (+24V applied) then the corresponding jog function is DISABLED and when the input is off (0V) then the jog function is ENABLED.

The jog functions implemented here disable the fast jog function, which means that the speed at which the jog will be performed is set by the JOGSPEED axis parameter. What is more this window limits the jog speed to the range 0..demand speed, where the demand speed is given by the SPEED axis parameter.

Before allowing a jog to be initiated, the jog window checks that all the data set in the jog window and on the *Motion Coordinator* is valid for a jog to be performed.



Jog Reverse

This button will initiate a reverse jog. In order to do this, the following check sequence is performed:

- If this is a SERVO or RESOLVER axis and the servo is off then set the warning message
- If this axis has a daughter board and the WatchDog is off then set the warning message
- If the jog speed is 0 the set the warning message
- If the acceleration rate on this axis is 0 then set the warning message
- If the deceleration rate on this axis is 0 then set the warning message
- If the reverse jog input is out of range then set the warning message
- If there is already a move being performed on this axis that is not a jog move then set the warning message

If there were no warnings set, then the message “Reverse jog set on axis?” is set in the warnings window, the **FAST_JOG** input is invalidated for this axis, the **CREEP** is set to the value given in the jog speed control and finally the **JOG_REV** output is turned off, thus enabling the reverse jog function.



Jog Forward

This button will initiate a forward jog. In order to do this, 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

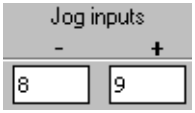
If there were no warnings set, then the message “Forward jog set on axis?” is set in the warnings window, the **FAST_JOG** input is invalidated for this axis, the **CREEP** is set to the value given in the jog speed control, and finally the **JOG_FWD** output is turned off, thus enabling the forward jog function.

Jog Speed



This is the speed at which the jog will be performed. This window limits this value to the range from zero to the demand speed for this axis, where the demand speed is given by the **SPEED** axis parameter. This value can be changed by writing directly to this control or using the jog speed control. The scroll bar changes the jog speed up or down in increments of 1 unit per second

Jog Inputs



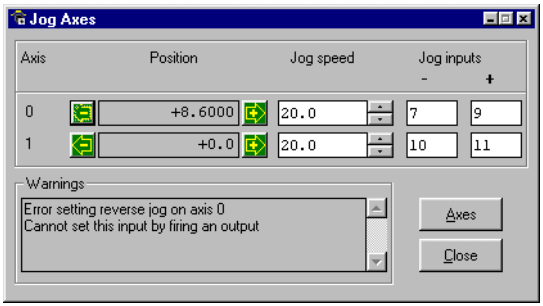
These are the inputs which will be associated with the forward / reverse jog functions.

They must be in the range 8 to the total number of inputs in the system as the input channels 0 to 7 are not bi-directional and so the state of the input cannot be set by the corresponding output.

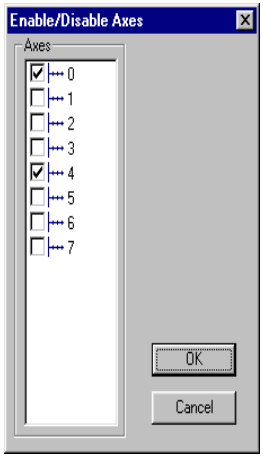
The input is expected to be ON for the jog function to be disabled and OFF for the reverse jog to be enabled. In order to respect this, when this is set to a valid input number, the corresponding output is set ON and then the corresponding **REV_JOG** axis parameter is set.

Warnings

This shows the status of the last jog request. For example, the screen below shows axis 0 with IO channel 7 selected. This is an Input-only channel and therefore cannot be used in the jog screen.



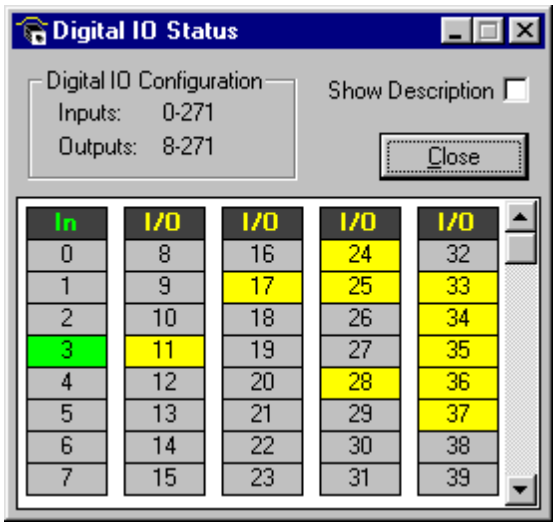
Axes



This displays an axis selector box which enables the user to select the axis to include in the jog axes display. By default, the physical axes fitted to the controller will be displayed.

Digital IO Status

This window allows the user to view the status of all the IO channels and toggle the status of the output channels. It also optionally allows the user to enter a description for each I/O line.



Digital Inputs

This shows the total number of input channels on the *Motion Coordinator*.

Digital Outputs

This shows the total number of output channels on the *Motion Coordinator*.

IO Mimic

Input Bank 0-7

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

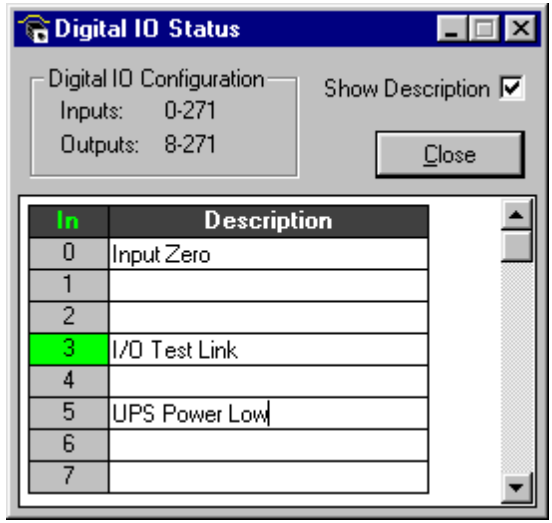
These banks of LEDs show the status the bi-directional IO channels. If an LED is yellow then the corresponding input is ON. If an LED is white then the corresponding input is OFF. Under normal conditions the input status mimics the output status, except:

5) If this input is connected to an external 24V then it may be on without the corresponding output being on.

6) 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.

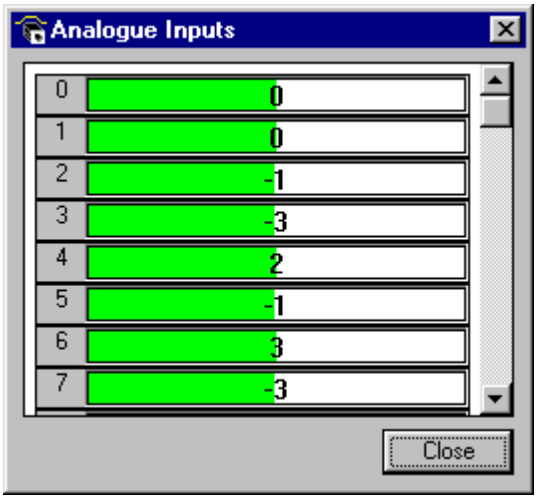
If the LED is clicked with the mouse the status corresponding output channel is toggled, i.e. if the LED is white then the output will be turned on, if the LED is yellow then the output channel will be turned off.

Checking the **Show Description** check box will toggle between dsecriptions on, and descriptions off. Descriptions are stored in the project file

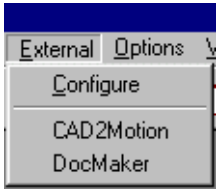


Analogue Input Viewer

The analogue input viewer is only available if the system has analogue inputs. It displays the input values of all analogue inputs in the system using a bar-graph with numeric display. All inputs have the range -2048 to 2047.



Linking to External Tools

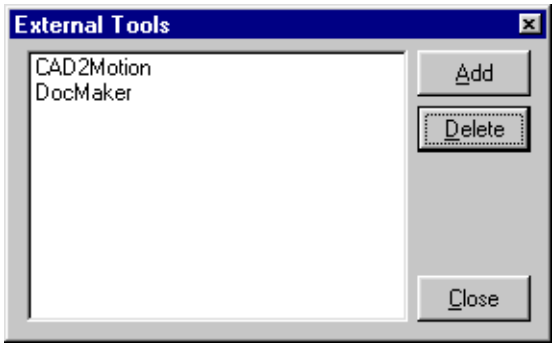


The EXTERNAL menu in *Motion* Perfect allows you to run other programs directly from the main *Motion* Perfect menu. In our example shown here, the menu has been configured to launch two other Trio applications, CAD2Motion and DocMaker. Further information on these applications is given at the end of this chapter.

Note: ***Cad2Motion and DocMaker are available to download from the Trio Website at*** `www.triomotion.com`.

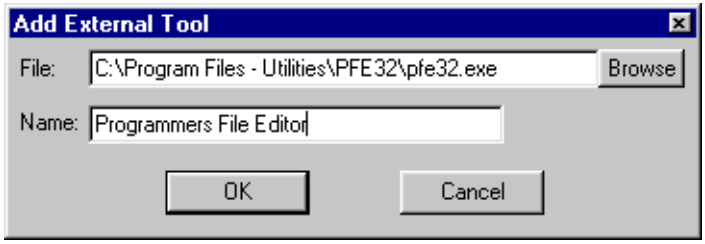
Configuring Items on the External menu

Clicking on the Configure item will bring up a list of all installed applications and from here we can add or delete items from this list.



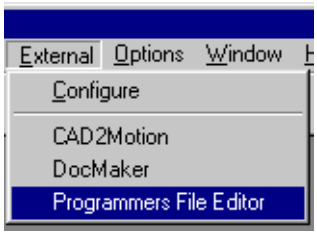
Adding a new programs to the menu

Clicking on the Add button will open the following dialog:



You can either directly enter the path and program file name in the "File" box, or use the "Browse" option to open up a standard windows file selector box which you can use to locate the file on your computer.

Once you have selected the file, it will automatically appear in the External menu every time you run *Motion* Perfect 2.

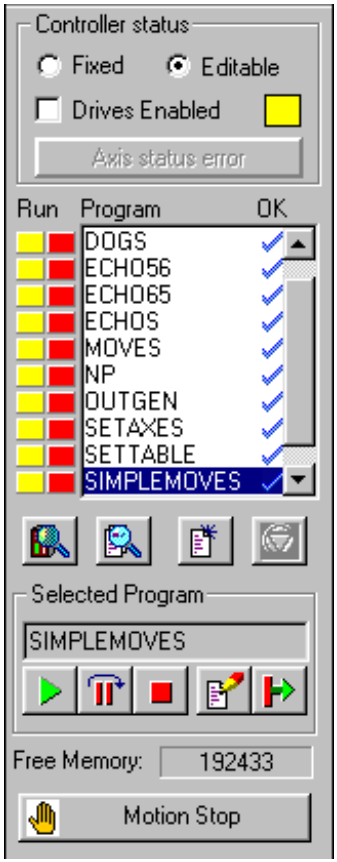


Removing program items from the menu

To delete a program from the External menu, you simply need to click on the program name in the list and press the Delete button.

Note: ***This simply removes the program from the menu. it does NOT affect the original program on disk!***

Control Panel



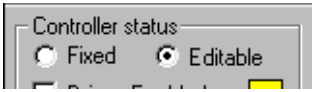
The control panel appears on the left hand side of the main *Motion* Perfect window.

It provides direct links to many of the frequently used operations within *Motion* Perfect, in particular the file and directory functions.

Please Note: Certain Control Panel Features behave differently on controller without a battery backup. The differences are described later in this section.

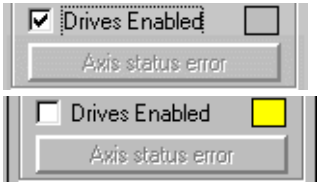
Control Panel Features

Fixed/Editable radio buttons



When the project is “fixed”, the programs are copied to the Flash EPROM on the *Motion Coordinator*, the *Motion Coordinator* is set to run from EPROM and the programs cannot be modified by *Motion* Perfect. Usually this is done when the machine programs are completed. The Flash EPROM provides a reliable permanent storage for the programs.

Drives radio button



Drives Enabled

Drives Disabled

This radio button toggles the state of the enable (watchdog) relay on the controller, going between drives disabled (watchdog off) and drives enabled (watchdog on).

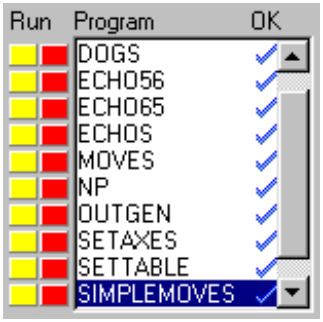
The LED mimic next to this control shows the status of the error LED on the *Motion Coordinator*. If it is yellow then the drives are disabled, if it is grey the drives are enabled and if it is flashing then there has been a motion error on at least one axis of the controller.

Axis Status Error



This will normally be greyed out unless a motion error occurs on the controller.
When an error does occur you can use this button to clear the error condition.

Program directory



This is a scrollable list of the programs on the controller. The list shows the program name followed by two optional indicators. The first is a number which specifies which process that program is running on. If it is not running then this space is blank. The second indicator shows the status of the program. If it has a tick then the program has been compiled successfully and is ready to be run. If it has a cross then there was an error during the compilation of the program and it cannot be run.

If it is blank then it has not been compiled.

If a program name is clicked then it will become the selected program if there are no programs running. If there are programs running the select will be ignored.

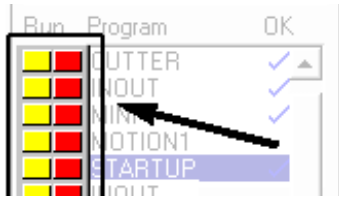
If a program name is double clicked then it will be opened for editing assuming that there are no programs running. If there are programs running then the edit will be ignored.

The names of programs which are currently running are displayed in italics.



Right clicking on an entry in the program list causes a pop-up menu to appear allowing easy access to operations commonly performed on programs. The right click operation highlights the entry in the program list under the cursor whilst the pop-up menu is visible. It reverts to the program which is currently selected on the controller when the menu is closed.

Run buttons



The run buttons provide short cut keys for running, stopping and single stepping programs. They can be in one of three states, red, green or yellow.

RED	Click on the red button to start the corresponding program running. The button will turn Green. *
GREEN	Click on the green button to stop the corresponding program. The button will turn red. *
YELLOW	Click on the yellow button to single step through the program.

* If the program goes into trace mode, through the use of a trace button, the selected program step button, the debug option of the program menu, the debug button on the tool bar or a TRON/STEPLINE command in a program or the terminal window, then the red/green run button will turn yellow. If the button is clicked when it is yellow then the program will be stepped one line.

General Options




Show Controller Configuration


Show a full directory of all programs in memory


Create a New Program. Same as the Program menu item.


HALT - Stop all programs which are running

Selected Program



The text box displays the currently selected program and the buttons below, the operations which can be performed on that program.

From left to right they are:
Run, Step, Stop, Edit and Power Up Mode

Free Memory



Shows the total free memory available on the controller

Motion Stop



Cancel moves on all axes and disable the watchdog relay.

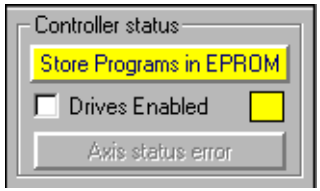
Note ***MOTION STOP is a software function. It is not a substitute for a hardware E_Stop circuit and should not be used as an emergency stop.***

Control Panel Variations for Motion Coordinators without battery backup

On those controllers without a battery backup such as the MC202, the fixed / editable radio buttons are replace by a single button labeled "Store Programs into Eprom".

As the controller does not feature a battery, it is essential that you store your programs into the eprom to avoid loss of data.

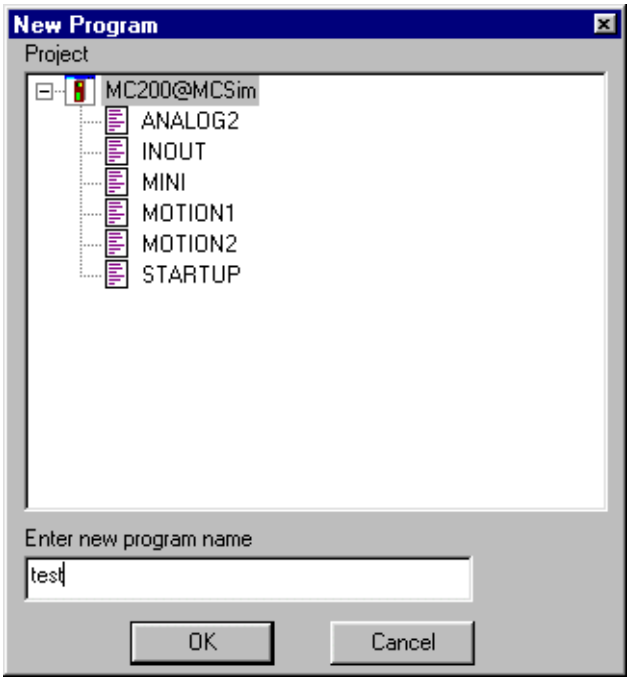
If the programs in memory have been edited, the button will be highlighted to remind you to fix into eprom before exiting from the program



Creating and Running a program

In order to create a new program on the controller, you must first have an active project. If you have already connected to the controller then you can use the default project which was created at this time.

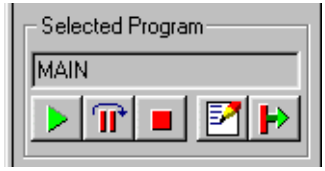
You will be presented with a program selector dialog and prompted to enter a name for your program file. It is a good idea to make this name representative of the task performed by the program, for example “mmi”, “motion”, “logic” or something similar. In the following example, we will add a program called “test” to the current project.



Once you have created a new program it will be added to both the controller and the *Motion* Perfect project file. You can now edit the file in *Motion* Perfect Editor.

The *Motion* Perfect Editor

You can start the Editor from the main Program Menu, the Edit button in the program section of the control panel or by right clicking in an entry in the control panel program list and selecting **Edit** from the pop-up menu.



Program Section of the Control Panel

If you launch the editor from the control panel it will start immediately. From the program menu you will first be prompted with a program selector dialog to confirm the file you wish to edit.

The *Motion* Perfect Editor is designed to operate in a similar manner to any simple text editor found on a PC. Standard operations such a block editing functions, text search and replace and printing are all supported and conform to the standard Windows shortcut keys. In addition it provides TrioBASIC syntax highlighting, program formatting and program debugging facilities.

Editor Options

Options for the editor are controlled by the **Editor Options** dialog. This can be opened by selecting **Options/Editor** from *Motion* Perfect's main menu. The dialog allows to user to change the fonts used for screen display and printing, the colours used for syntax and line highlighting and the spacings used when automatically formatting a program.

Screen Font

This is the font used by *Motion* Perfect to display text in the editor window on the screen. The font is restricted to fixed pitch fonts only.

Printer Font

This is the font used by *Motion* Perfect to print program listings. The font is restricted to fixed pitch fonts only.

Colours

Colours can be specified for the following:

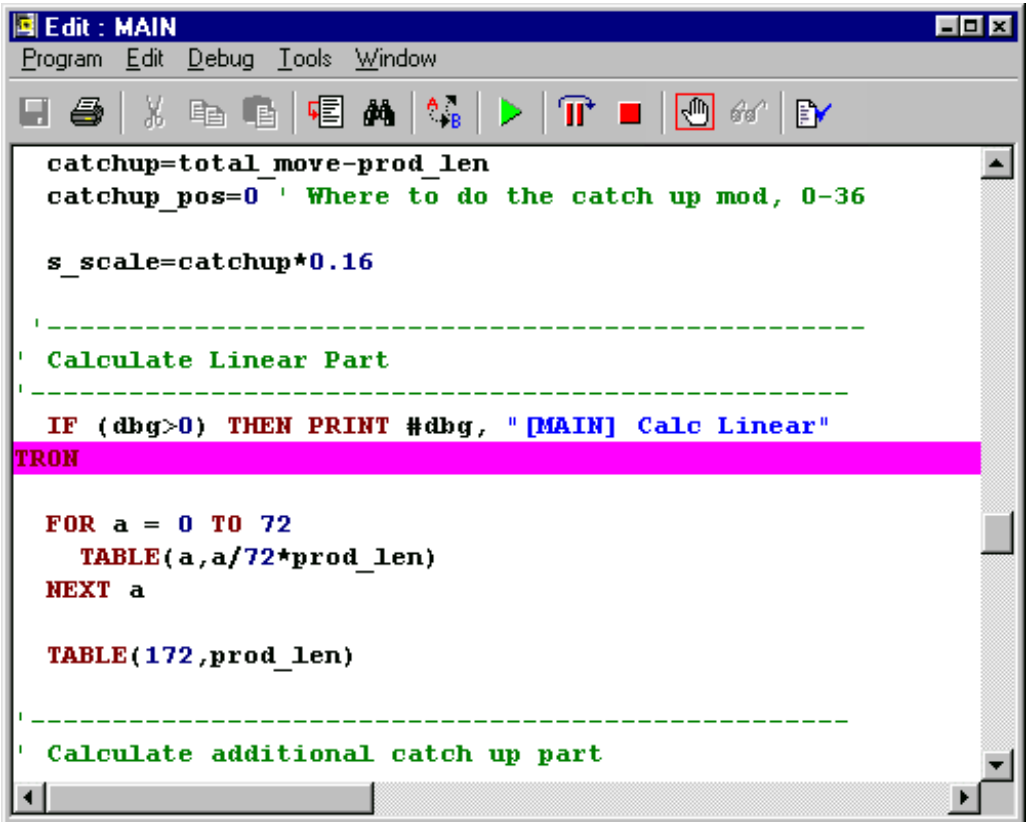
- Normal Text - Text which is not highlighted using systax highlighting.
- Screen Background.
- Current Line (background) - the current line during debugging.
- Break Line (background) - a line containing a break (TRON) command.
- Current Line Break (background) - a line containing a break command which is also the current line.
- Error Line (background) - The first line containing a compilation error.
- BASIC key word - A key word in the TrioBASIC language, usually a command or some type of system variable.

- Comment Text
- Constant Text - text making up a constant value (number).
- Strings
- Label Defenition - where a program label is defined.
- Label Reference - where a jump or branch (GOTO, GOSUB etc.) in program execution is required. The jump or branch changes the execution point to the place where the label is defined.

Format

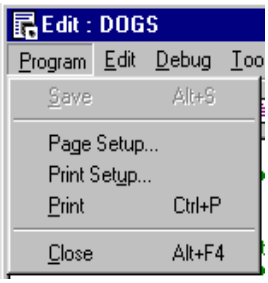
The format options affect text entry and the automatic reformatting preformed by *Motion* Perfect.
For automatic reformatting the code start column and the tab width are specifiable. As label definitions always start in column 0, the code start column can be used to indent all lines containing code thus making label definitions clearer.

The when **Auto tab on enter** check box is checked pressing the “enter” key will automatically indent the next (new) line to the same position as the current one.


Edit : MAIN
Program Edit Debug Tools Window
catchup=total_move-prod_len
catchup_pos=0 ' Where to do the catch up mod, 0-36
s_scale=catchup*0.16
' Calculate Linear Part
IF (dbg>0) THEN PRINT #dbg, "[MAIN] Calc Linear"
TRON
FOR a = 0 TO 72
TABLE(a,a/72*prod_len)
NEXT a
TABLE(172,prod_len)
' Calculate additional catch up part

Editor Menus

Program



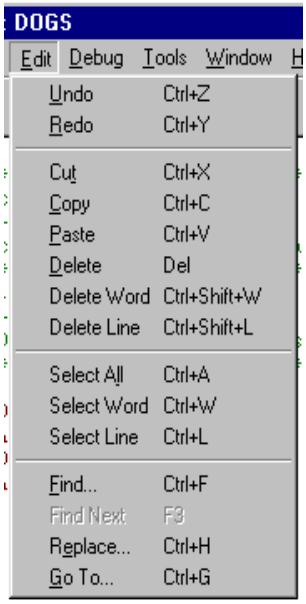
Save

Normally the program is only saved to disk when the editor is closed or a program is run, however if you have modified the program the Save Button will be available and will force *Motion* Perfect to save the file immediately.

Printing

Use **Page Setup** to set the page mrrgins, **Print Setup** to configure you printer settings and the **Print** option to send the program to the printer.

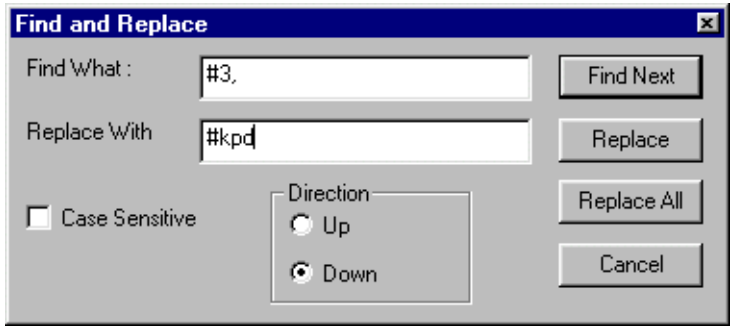
Edit



The edit menu functions are similar to many other text editors and provide the standard block cut/copy/paste operations as well as a simple text find/replace, and various select and delete functions.

Find/Replace

The options for the find & replace dialogs are very similar and feature many of the same options



You should enter the text to search for in the “Find What” box, and if using find and replace, the text to replace it with in the “Replace With” box.

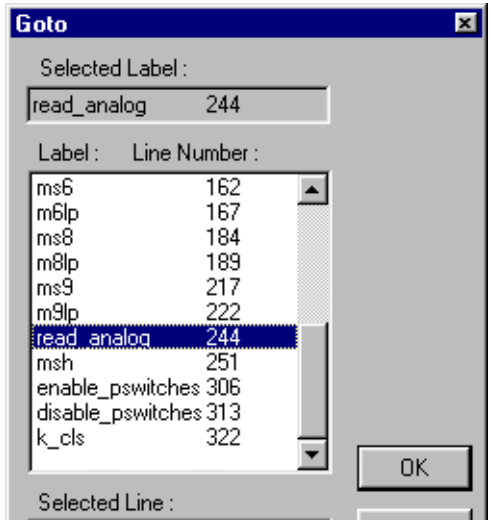
Normally the “Case Sensitive” search option is not selected, You should only use this option if you have an exact pattern to match, generally the default option is best.

GOTO

The Goto option will bring up the following dialog:

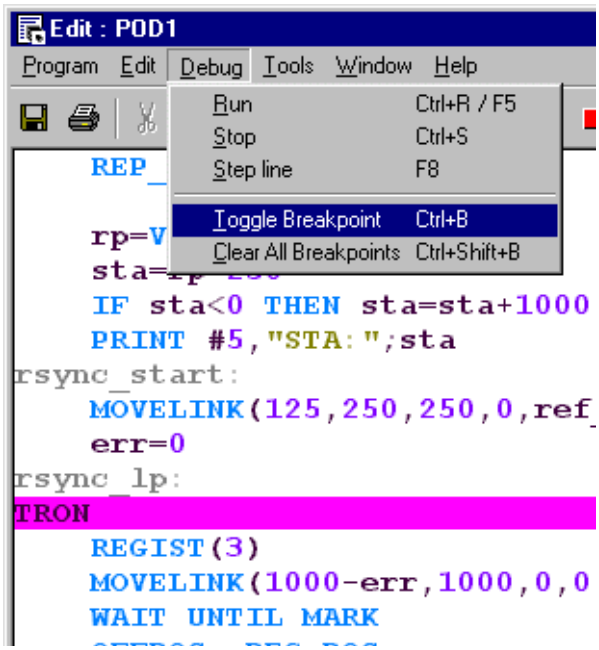
A list of labels defined in the program is displayed. You can either select a label from here, or enter a line number directly in the “Selected Line” field.

Press OK and the cursor will jump directly to the beginning of the selected line.



Program Debugger

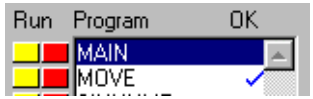
The *Motion* Perfect debugger allows you to run a program directly from the editor window in a special 'trace mode, executing one line at a time (known as stepping) whilst viewing the line in the window. It is also possible to set breakpoints in the program, and run it at normal speed until it reaches the breakpoint where it will stop, and this line of code will be highlighted in the debug window.



When programs are running on the controller, any open editor windows will automatically switch to Debug Mode and will become read-only. Hence, breakpoints are set in the edit window, and the code viewed in the same window in debug mode when the program is running.

Stepping Through a program

To commence stepping a program:



Use the mouse to press the yellow button alongside the required program name in the list box on the control panel

if the required program is currently selected, press the 'Step' button on the control panel( ) or use the menu item 'Debug-Step line'

The currently executing line of code is indicated in the debug window by highlighting it with a green background, and a breakpoint is highlighted with a red background.

To continue stepping the program, repeatedly press the yellow button alongside the program name in the list box on the control panel, or press the 'Step' button or the 'F8' functional key if the program required is currently selected on the *Motion Coordinator*.

Breakpoints

Breakpoints are special place markers in the code which allow a particular section (or sections) of the program to be identified when debugging the code. If a breakpoint is inserted, the program will pause at that point and return control to *Motion* Perfect where the controller may be interrogated or the program run in step mode as described above.

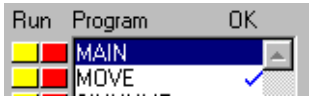
To insert a breakpoint, first position the text cursor on the line at which you want the break to occur, then use either Ctrl-B or the menu item to insert the breakpoint.

The Trio BASIC instruction TRON is used to mark a breakpoint and TROFF to terminate a 'traced' block.

Note: ***It is not possible to add or remove breakpoints whilst any programs are running.***

Running to a breakpoint

A program can be run to the next break point by:



- using the mouse to press the red button alongside the program name in the list box on the control panel.
- if it is the currently selected program on the *Motion Coordinator*, you can pressing the 'Run' button ( ) on the control panel/editor tool bar, or by using the keyboard <F5> function key.
- by selecting the 'Debug->Run' menu option

Stopping a Program

If it is necessary to stop the program running before it reaches the breakpoint then:



- press the green button alongside the program name (running on the required process) in the list box on the control panel.
- press the stop button ( )on the control panel if the program is currently selected (this will stop all running copies of the program)
- use the 'Debug-Stop' menu option.

Alternatively all programs can be stopped by pressing the 'Halt' button on the control panel, or selecting the 'Program' 'Halt all programs' menu option, or using the <Ctrl><F> key combination.

Switching a running program into trace mode

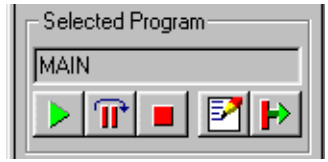
A running program can enter trace (stepping) mode by pressing the yellow button alongside the required program name in the list box on the control panel, or the 'Step' button if the required program is currently selected on the *Motion Coordinator*.

Running Programs

You can start/stop programs running in one of four ways:

From the control panel

If the program is currently selected (highlighted in the control panel), you can press the green start arrow in the "selected program" box.



From the program list

Pressing the red button to the left of the program name in the list will start it running, the button will change to green and it will then function as a stop button for the same task.



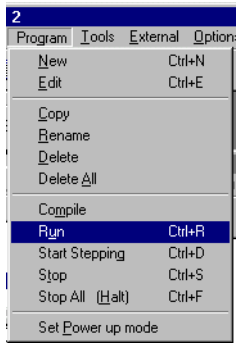
From the editor toolbar



If you have an editor or debug window open for the program you can use either the Debug menu or toolbar buttons to start the program running

From the Program Menu

The program menu provides us with a slightly different option when running the program as we are presented with a program selector box which includes an option to choose which task we want to run the program on

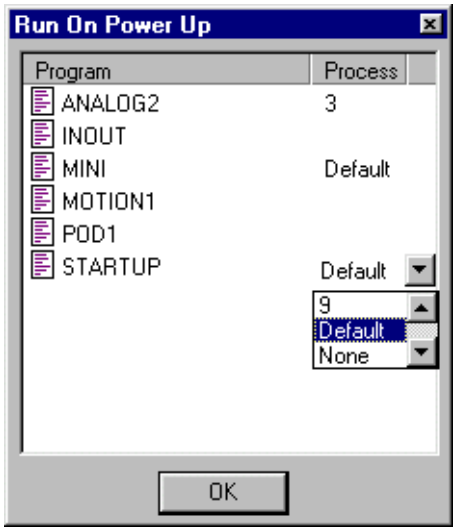


Making programs run automatically

Set Powerup Mode

It is possible to make the programs on the controller run automatically when the system first starts up. From the **Program** Menu, select **"Set Powerup Mode"** to open the following dialog.

Click on the program you want to auto run and a small drop-down list will appear to the right of the window. If you are happy to let the controller allocate which task to run on then you should choose "default" as the process number, otherwise you can specify the task explicitly in the box.



Storing Programs in the Flash EPROM



This is accomplished by selecting the “Fixed” option in the controller status section of the control panel, or the “Fix Program Into EPROM” option from the controller menu.

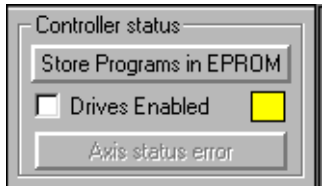
When the controller is fixed into eprom, the programs actually still run from RAM. The information stored is copied into RAM when the controller is first started, therefore if the controller has been switched off for an extended period, or there is any corruption of the RAM, it will be refreshed with a correct copy of the programs.

When the controller is set to fixed you will not be able to edit any programs. In order to make changes you must select “Editable” from the control panel or “Enable Editing” from the controller menu.

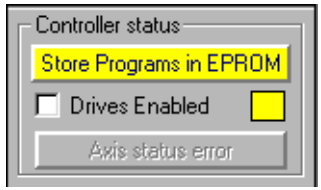
Variations for controllers without battery backup

On those controllers without battery backup, such as the MC202, it is essential that you store your programs into the eprom to avoid loss of data.

The control panel the fixed / editable radio buttons are replaced by a single button labelled "Store Programs into Eprom".

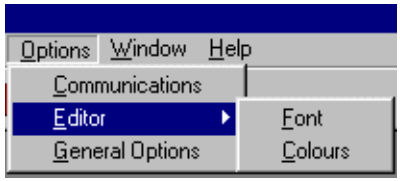


If the programs in memory have been edited, the button will be highlighted to remind you to fix into eprom before exiting the program.



Configuring The *Motion* Perfect 2 Desktop

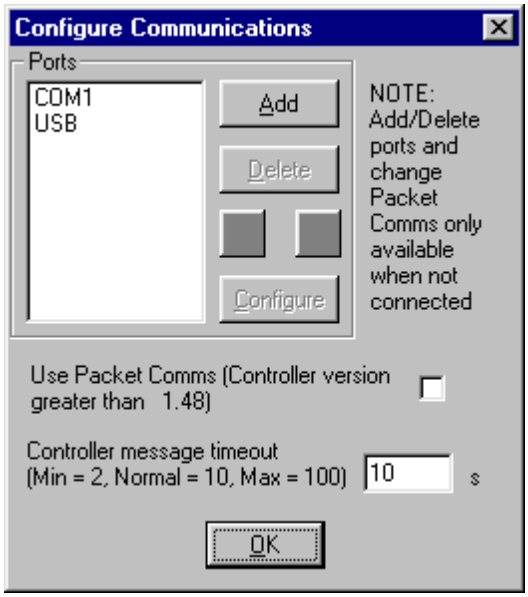
There are a number of ways in which you can configure *Motion* Perfect 2 to suit your requirements. The Options menu provides a number of choices:-



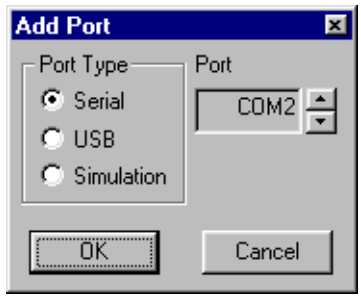
Communications

Set up the default communications device for *Motion* Perfect 2 to use.

Motion Perfect 2 needs a single serial connection to the controller in order to operate. This can be an RS-232 serial connection, or a USB connection if your controller and PC have USB ports. It will normally recognise the ports installed in your PC and will display these in the Configure Communications window.



If the port you wish to use is not shown, you need to select the Add Port option which will select the following dialog.

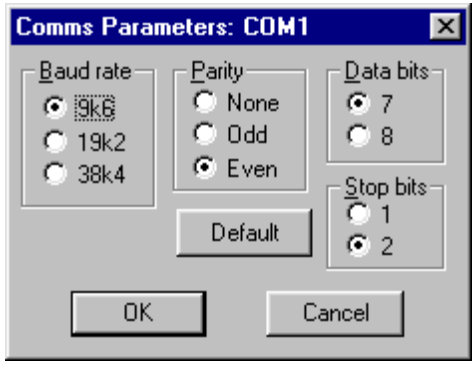


Most *Motion* Perfect 2 users will connect via a standard RS-232 serial (COM) port. Other options include a connection via a high speed Universal Serial Bus (USB) port or a controller simulation for offline editing.

Choose the port options you require and click on the "OK" button to install the new port.

Changing Comms Port Parameters

The default settings for serial communications is:- 9600 baud, 7 data bits, 2 stop bits, even parity. If you wish to change these values you can do so with the configure button in the Configure Communications dialog.

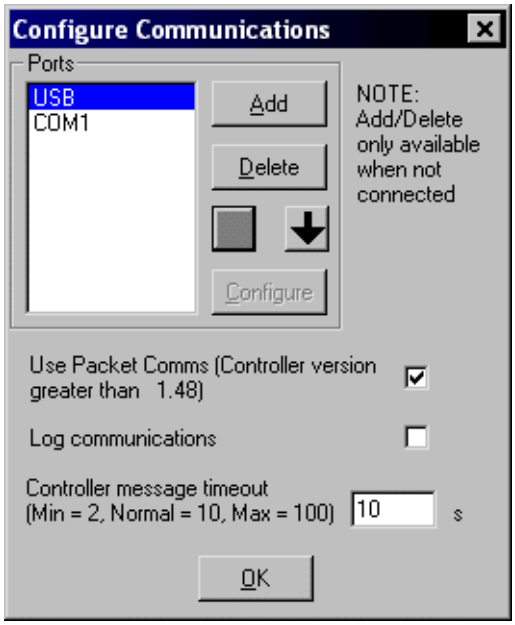


Note: ***If you change the port setting to anything other than the default, you may encounter problems when the controller is reset because it will revert to the default values.***

In order to avoid this your controller will need to set the comms parameters within an auto-running program. See the SETCOM instruction in the Trio BASIC reference for further information.

Setting the Default Port

You can change the order in which the ports are scanned by the program by selecting the port in the list and using the up/down arrows to move the items as required.



Packet Based Communications

Motion Perfect 2 features an enhanced communications protocol which uses a packet based structure with error checking to significantly improve the reliability of the serial port communications. Packet communications can be used over any type of communications link but as some of these (USB, ethernet) use packetised communications as part of their own protocol, adding an extra level of packetisation may make communications less efficient without giving any reliability improvement.

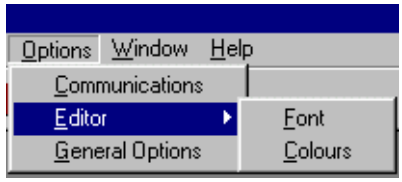
In order to use the Packet Comms mode you must have system software 1.49 or higher.

Controller message timeout

Certain controller operations may cause the controller to stop communicating for a few seconds. If you find that your PC seems to disconnect often, you can change *Motion* Perfect 2's default timeout value to allow the program to wait for a longer time before disconnecting.

Editor Options

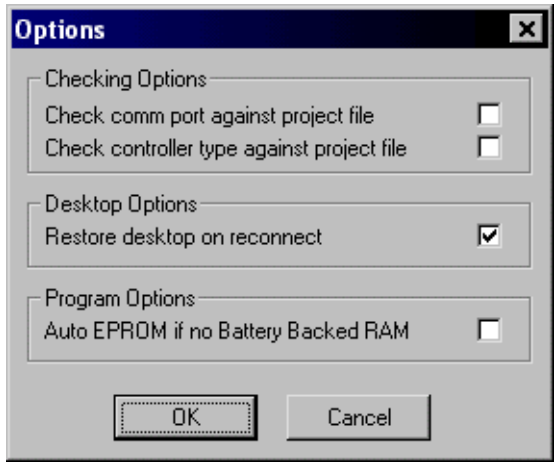
The Editor sub-menu allow you to modify the appearance of the *Motion* Perfect 2 editor to suit your own personal taste. You can change both the default font used and the colours used by the syntax highlighting feature.



General Options

This dialog allows the user to change a number of options relating to how *Motion* Perfect 2 starts up and handles projects.

When you select General Options you will be presented with the following screen.



The check-boxes enable the following features:

Check comm port against project file

Checks the comm port being used against the one in the project file when a Check Project operation is performed.

Check controller type against project file

Checks the type of the connected controllerg used against the one in the project file when a Check Project operation is performed.

Restore desktop on reconnect

If this option is selected, the program will attempt to automatically save the desktop layout when disconnecting from the controller.

When you reconnect, *Motion* Perfect 2 will automatically restore the last desktop layout saved.

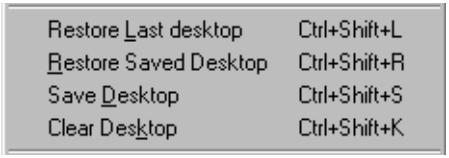
Auto EPROM if no Battery Backed RAM

If the controller does not feature battery-backed RAM memory and this option is selected, the program will attempt to automatically save the controller memory to EPROM before disconnecting from the controller.

Saving the Desktop Layout

When you have a number of windows open, you can save the layout so that it can be quickly restored later. Alternatively the desktop can be set to restore automatically on each re-connection by ticking the checkbox under the menu: Options/General options.

From the Window menu...



Restore Last Desktop

Restores the last desktop which was automatically saved by *Motion* Perfect 2 when it disconnected from the controller

Restore Saved Desktop

Restore the last desktop saved using the Save Desktop option.

Save Desktop

Saves the current desktop layout to a file on disk

Clear Desktop

Closes all open tool windows.

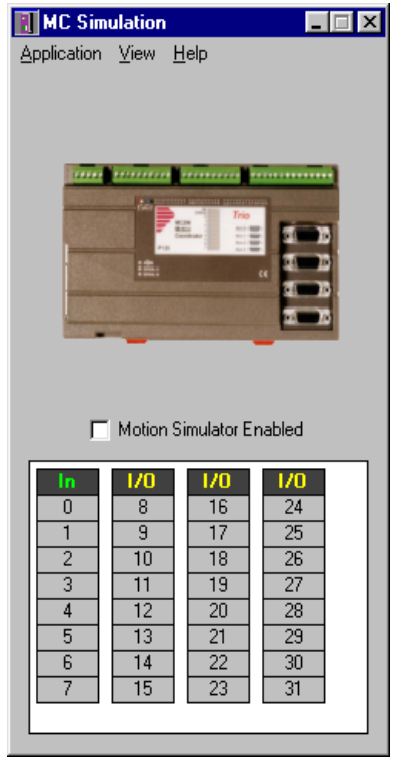
Running *Motion* Perfect 2 Without a Controller

Normally you will run *Motion* Perfect 2 on-line, that is connected to a controller. In fact *Motion* Perfect 2 is designed to operate in this manner and has little functionality without the connection.

In order that you can view or edit your project programs without a controller connected there is a special application to simulate the controller operation and to allow *Motion* Perfect 2 to operate in many ways as if a real controller were connected.

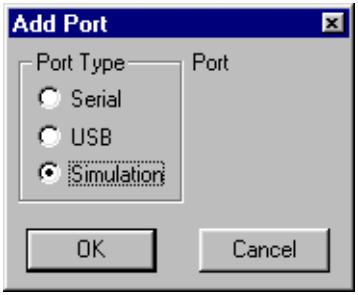
MC Simulation

MC Simulation (MCSim) is a very simple program designed to run alongside *Motion* Perfect 2 in the background. There are no options or configurations to worry about, you just have to run the program and connect as usual.

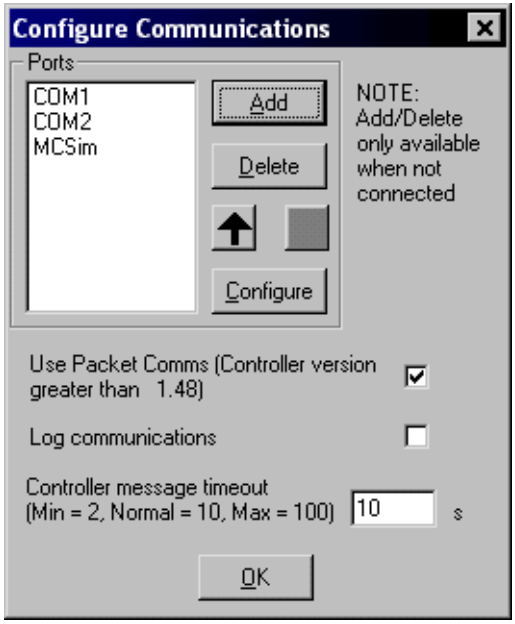


Starting MC Simulatiion from *Motion* Perfect 2

MC Simulatoion is automatically started (if it is not already running) when *Motion* Perfect tries to connect to it. To connect to MCSimuolation either use the **Connect to Simulator** tool button  or set up an **MCSim** port in the connection list.



Use the Add Port option to select a new port and choose "Simulation" as the Port Type.



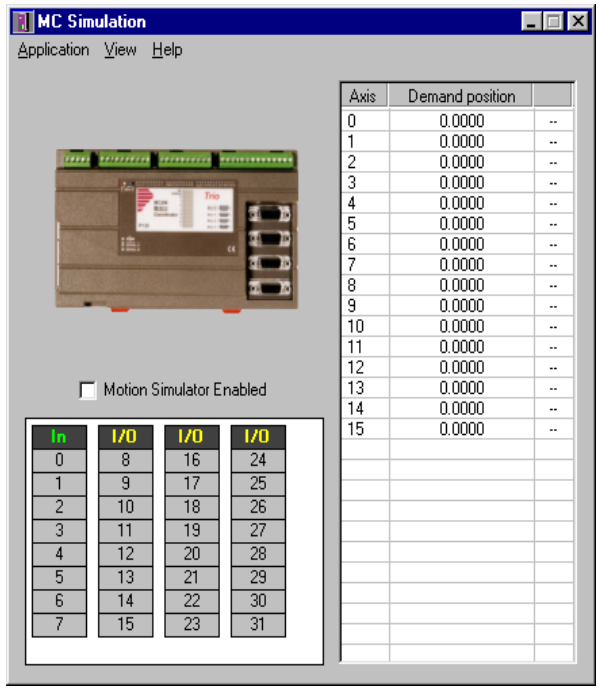
The new device will normally appear at the end of the list. Use the "move up" button to make it the default option.



Limitations of MC Simulation

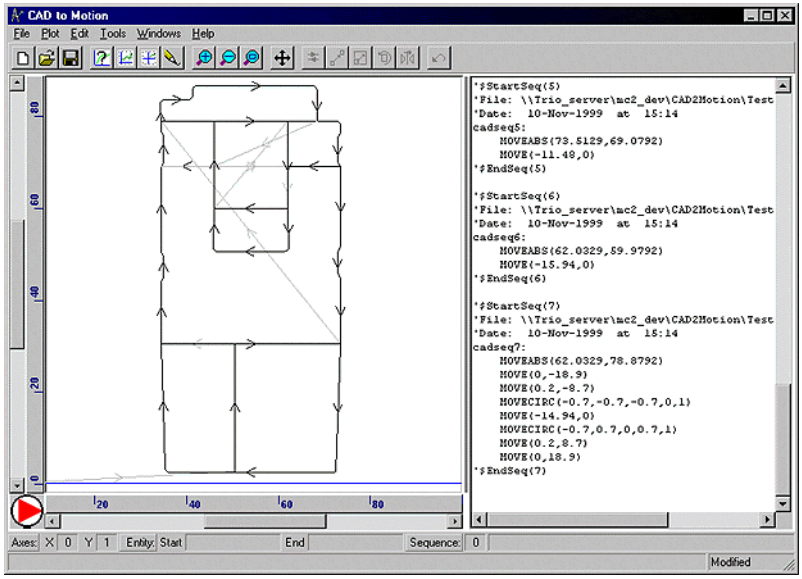
The MCSimulation program does not yet cover all the functionality present in a real controller. It does allow connection to *Motion* Perfect for program editing and the running of programs in the simulated environment. There are some unsupported TrioBASIC commands (mainly those related to communications busses such as CAN).

The motion engine built into the simulation is still under development although it will handle all move types except linked moves. There is an axis demand position display which can be used to monitor the axes when moves are taking place. This can be toggled on and off by selecting **View/Axes** from the MCSimulation main menu. The motion engine can be enabled/disabled by checking/unchecking the the **Motion Simulator Enabled** check box.



CAD2Motion

CAD2Motion is a program designed to allow users to translate CAD generated two dimensional motion paths into Trio BASIC programs.



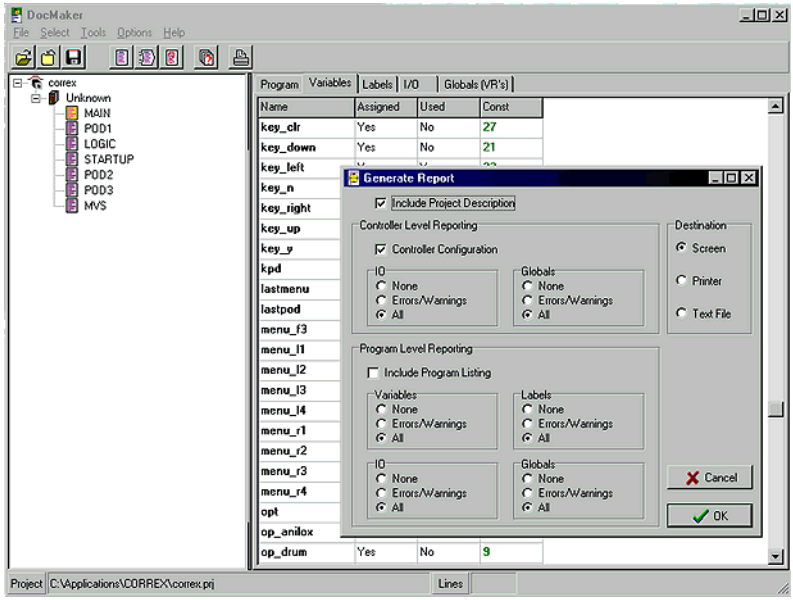
The program allows the user to create motion paths in a CAD package such as AutoCAD and convert them into code executable by a *Motion Coordinator*. Typically the path information will be drawn on a single layer in the CAD package and exported as a DXF file. The DXF file (layer with motion path only) is read into CAD2Motion to create a program to follow the motion path.

The motion path can be manipulated and edited before being saved as a Trio BASIC program file which can be loaded on to a *Motion Coordinator*.

- Can read Industry standard DXF files and TrioBASIC files.
- Outputs files as TrioBASIC programs.
- Graphical display of motion path with full pan and zoom facilities.
- Built in program text editor.
- Program list and graphical display linked together to show correlation between program code and moves within the motion path.
- Built in tools to mirror, scale, shift, reverse and rotate motion paths.
- Full undo on all tool and editor operations.
- Will handle multiple motion paths (sequences) in the same program.

DocMaker

DocMaker is a Windows application designed to assist in documenting a Trio BASIC project created with *Motion* Perfect.



DocMaker analyses the content of the program files in the project. It can be used to print program listings and to report on the programs (variables, labels, I/O and VR's) and on overall I/O and VR usage. There is also a checking routine which does a quick check on the whole project and flags up possible errors

DocMaker Benefits

- Automatic Analysis of MotionPerfect Project Files
- Highlight potential errors due to labels or variables
- Generates fully cross-referenced reports
- Reformat programs with auto-indenting

Docmaker Hardware Requirements

- IBM PC or Compatible running Microsoft Windows 95 or higher
- Works with all current Motion Coordinators: MC202, MC204, MC216, MC2 and Euro205

Project Autoloader

Trio Project Autoloader is a stand alone program to load projects created using Motion Perfect 2 onto a Trio Motion Coordinator.

The program is small enough to fit onto a 1.44MByte floppy disk and is intended for easy loading of projects onto controllers without the need to run Motion Perfect and so allows OEM manufacturers to update customers equipment easily.

Operation of the program is controlled using a script file which gives a series of commands to be processed, in order, by the program.

Using the Autoloader

General

The autoloader is primerally intended to be used from a floppy disk to update controllers already installed in equipment to allow OEM manufacturers to update customers equipment easily. It can also be used from a hard disk or CD-ROM.

Script File

The commands to be executed are held in a script file AutoLoader.tas which must be in the LoaderFiles directory.

Project

The project to be loaded using LOADPROJECT is in the form of a normal Motion Perfect 2 project. This consists of a directory containing a project definition file and TrioBASIC program files. The directory must have the same name as the project definition file less the extension.

i.e. project definition file TestProj.prj, directory TestProj

The project directory must be in the LoaderFiles directory.

Tables

Any tables to be loaded must be in the form of *.lst files produced by Motion Perfect 2.

Normally these table files will be in the LoaderFiles directory.

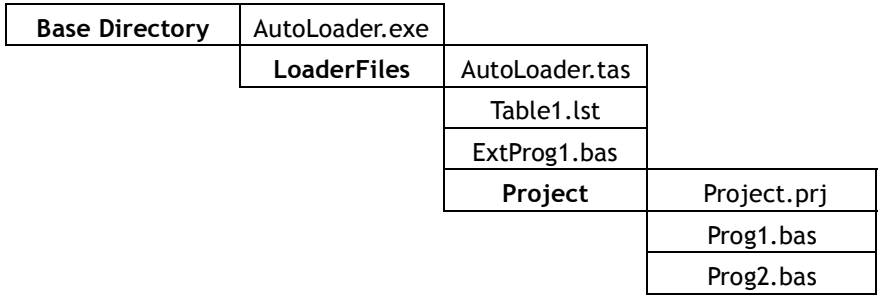
Extra Programs

Programs which need to be loaded using LOADPROGRAM because they are not in the project being loaded (or if no project is being loaded)

Normally these program files will be in the LoaderFiles directory.

Files

The autoloader is designed to work with the following file structure (fixed names are shown in bold type).



Where:

Base Directory is normally the root directory on a floppy disk (A:\), but can be any directory.

Project is the Motion Perfect 2 project directory for the project to be loaded using the LOADPROJECT command, Project.prj being the project file and Proj?.bas are the program files in the project.

Table?.lst are the table files to be loaded using the LOADTABLE command.

ExtProg?.bas are the extra programs to be loaded using the LOADPROGRAM command.

Running the program

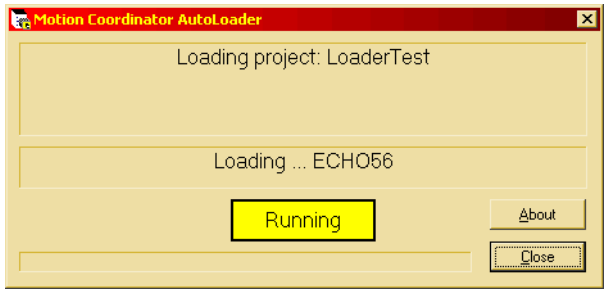
The program can be started in the same way as any other Windows program.

Start Dialog



The start dialog displays a message specified in the script and has continue and cancel buttons so that the user can exit from the program without running the script.

Main Window



The program main window consists of two message windows; one to display the current command and the other to display the name of the program or file currently being loaded. There is a button to show the current status (Starting, running, pass or fail) and a progress bar to show the progress during file and table loading.

The close button closes the dialog. If it is pressed while a script is being processed then script processing will be terminated at the end of the current operation.

Script Commands

The following commands are available for use in script files

COMMLINK
CHECKTYPE
CHECKVERSION
CHECKUNLOCKED
LOADPROGRAM
LOADPROJECT
EPROM
SETRUNFROMEPROM
LOADTABLE
Comment

All commands return a result of OK or Fail. An OK result allows script execution to continue, a Fail result will make script execution terminate at that point.

COMMLINK

Purpose: To set the communications port and parameters.

Syntax: COMMLINK <PortSpec>

Where <PortSpec> is a string specifying a communications port and the connection parameters.

For a serial port this string is similar to COM1:9600,7,e,2 to specify the port, speed, number of data bits, parity and number of stop bits. 9600,7,e,2 are the default parameters for a controller.

Examples: COMMLINK COM2:9600,7,e,2

CHECKTYPE

Purpose: To check the controller type.

Syntax: **CHECKTYPE <Controller List>**

Where <Controller List> is a comma separated list of one or more valid controller ID numbers.

i.e. 206,216

Examples:
```
CHECKTYPE 206
CHECKTYPE 202,216,206
```

Controller ID Numbers

Each type of controller returns a different ID number in response to the TrioBASIC command ?CONTROL[0] . The table below gives the ID number for current controllers.

Controller	ID Number
MC202	202
MC204	204
Euro205	205
MC206	206
MC216	216

CHECKVERSION

Purpose: To check the version of the controller system code.

Syntax: `CHECKVERSION <Operator><Version>`
`CHECKVERSION <LowVersion>-<HighVersion>`

Examples: `CHECVERSION >1.49`
`CHECKVERSION >= 1.51`
`CHECKVERSION 1.42-1.50`

CHECKUNLOCKED

Purpose: To check that the controller is not locked.

Syntax: `CHECKUNLOCKED`

LOADPROJECT

Purpose: To load a project from disk onto the controller.

Syntax: **LOADPROJECT <ProjectName>**
Where <ProjectName> is the path of the project directory. If the project directory is in the LoaderFiles directory then it is just the name of the of the project directory.

Examples: **LOADPROJECT TestProj**

LOADPROGRAM

Purpose: To load a program which is not part of a project from disk onto the controller.

Syntax: **LOADPROGRAM <ProgramFile>**

Where <ProgramFile> is the path of the program file. If the program file is in the LoaderFiles directory then it is just the name of the of the program file.

Note: ***This command should always be used after the LOADPROJECT command.***

Examples: **LOADPROGRAM TestProg.bas**

LOADTABLE

Purpose: **To load a table onto the controller.**

Syntax: **LOADTABLE <TableFile>**
Where <TableFile> is the path of the table file. If the table file is in the LoaderFiles directory then this is just the file name of the table file.

This command should always be used after the LOADPROJECT command.

Examples: **LOADTABLE Tbl.lst**

EPROM

Purpose: To store the project currently in controller RAM into EPROM

Syntax: **EPROM**

SETRUNFROMEPROM

Purpose: To set the controller to use the programs stored in its EPROM. (It actually copies the programs from EPROM into RAM at startup).

Syntax: **SETRUNFROMEPROM <State>**

Where <State> is 1 for copy from EPROM and 0 is use programs currently in RAM.

A single @ character can be used to specify state in the project file.

Examples:
```
SETRUNFROMEPROM 1
SETRUNFROMEPROM @
```

Note: This command only applies to controllers which have battery backed RAM (controllers with no battery backed RAM will always copy programs from EPROM).

Script File

The autoloader program uses a script file AutoLoader.tas as a source of commands. These commands are executed in order until all commands have been processed or an error has occurred.

If any command fails the exececution terminates without completing the scripted command sequence.

Sample Script

```
' Test Script
' **************
' Startup Message
# ***
# This autoloader was set up by TRIO to load a test project
# onto a controller of fixed type.
# ***
COMMLINK COM1:9600,7,e,2
CHECKTYPE 206
CHECKVERSION > 1.45
CHECKUNLOCKED
LOADPROJECT LoaderTest
LOADTABLE tbl_1.lst
CHECKPROJECT LoaderTest
LOADPROGRAM flashop.bas
LOADPROGRAM clrtable.bas
LOADPROGRAM settable.bas
EPROM
SETRUNFROMEPROM @
```

For this script to work correctly the LoaderFiles directory must contain a project directory LoaderTest, a table file tbl_1.lst and three program files: flashop.bas, clrtable.bas and settable.bas.

Creating an AutoLoader floppy disk

To create an AutoLoader floppy disk follow the steps below:

- Take a blank floppy disk and copy the AutoLoader.exe program file into the root directory.
- Create a sub-directory called LoaderFiles in the root directory.
- If you are going to load a project then copy the project directory of the project to be loaded into the LoaderFiles directory created above, so that it is a sub-directory of LoaderFiles.
- If you are going to load any tables then copy all the table files into the LoaderFiles directory.
- If you are going to load any extra programs then copy the program files into the LoaderFiles directory.
- Using an ASCII text editor (i.e. Windows Notepad) create and edit the script file (AutoLoader.tas) in the LoaderFiles directory, using the appropriate commands for the operations to be carried out.

CHAPTER

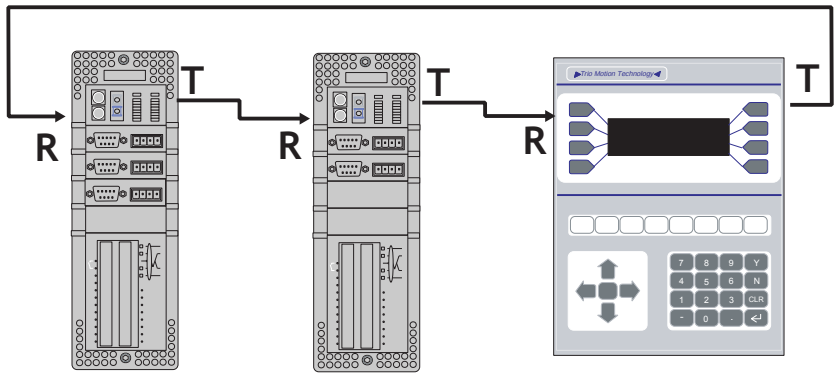
11

FIBRE-OPTIC NETWORK

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*.

The physical connection of the network takes the form of a ring. The interconnections between nodes being made with fibre optic cable.



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 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.

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.

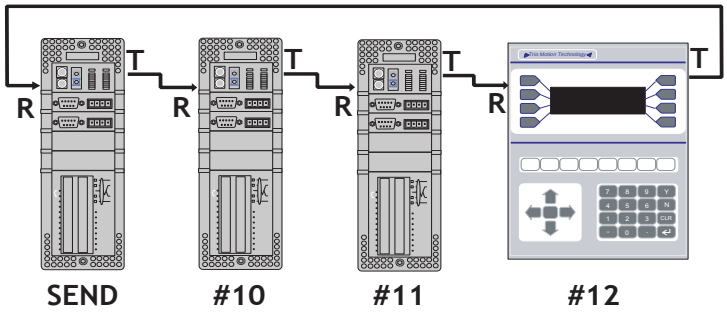
The order in which nodes are connected is not critical but the performance of the network can be affected by rearranging the order of the nodes. This is especially true if there are a large number of nodes in the network and a large amount of data being passed between two nodes. If these two nodes are not adjacent in the ring any nodes between them will be tied up re-transmitting messages and will not be able to transmit as priority is given to re-transmission of messages. This can be a particular problem if one of the nodes re-transmitting is a membrane keypad as this can cause poor response to key presses.

Fibre-optic cables

Trio can supply a fibre-optic connection kit (P570 Simplex Cable Kit) comprising a length of fibre-optic cable, connectors and assembly components.

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.



Node Addressing Example

The example above 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.

Network Programming

Trio BASIC Commands

The transmission and reception of messages on the network is performed by four Trio BASIC commands.

GET #n

Type: Command.

Syntax: **GET#n,VR(x)**

Description: Waits for the arrival of a single character on 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.

3 value returned is defined by DEFKEY

4 value returned is character received

x Variable number in the range 0 to 250.

KEY#n

Type: Function.

Syntax: **IF KEY#n THEN GET #n,VR(x)**

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

PRINT#n,

Type: Command.

Syntax: `PRINT #11, CURSOR(20);"Printed on Keypad 2"`

Description: The **PRINT#** command allows the 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.

SEND

Type: Command.

Syntax: `SEND(n,type,data1[,data2])`

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.

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.

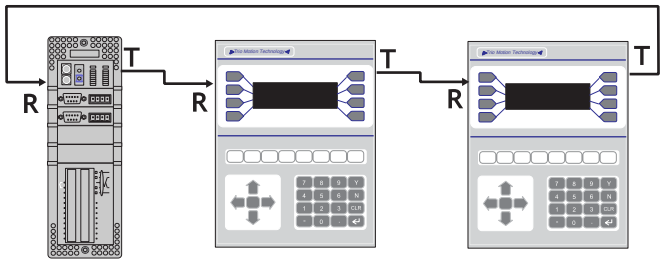
data2: If type=1: data2 is the value to change the specified variable to.

If type=2: data2 is not used.

Examples of network programming

Example 1 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 below.



Network Example 1

The program could look something like this:

```
length=2
' Set offset on first keypad to 2, i.e. send key press to MC
SEND(10,2,11)
' Set offset on second keypad to 1, i.e. send key press to MC
SEND(11,2,10)

' Clear first display & make cursor invisible
PRINT #10,CHR(12);CHR(14);CHR(20)
' Clear second display & make cursor invisible
PRINT #11,CHR(12);CHR(14);CHR(20)
' setup display
PRINT #10,"SPEED:":PRINT #10,"LENGTH:"
PRINT #11,"SPEED:":PRINT #11,"LENGTH:"

REPEAT
  IF KEY#3 THEN GOSUB read_key ELSE GOSUB do_motion
UNTIL FALSE
```

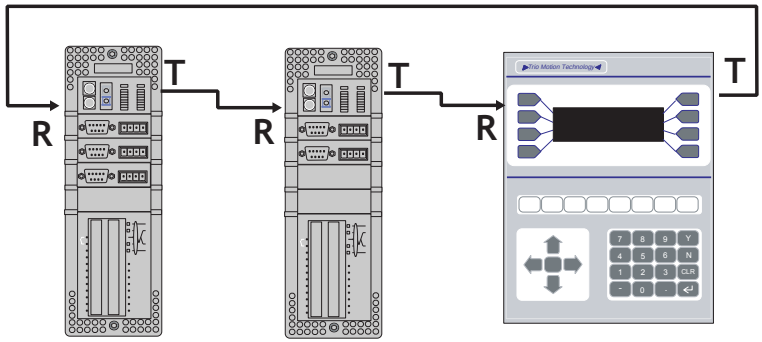
```
read_key:
  GET #3,k
  IF k>9 THEN GOTO not_num
  a=a*10+k
  IF a>9999 THEN a=0
  PRINT #10,CHR(27);CHR(72);CHR(7);a[6,0]
  PRINT #11,CHR(27);CHR(72);CHR(7);a[6,0]
not_num:
  IF (k=10)&(VR(length)<10000) THEN VR(length)=VR(length)+0.1
  IF (k=11)&(VR(length)>0.1) THEN VR(length)=VR(length)-0.1
  PRINT #10,CURSOR(28);VR(length)[8,1]
  PRINT #11,CURSOR(28);VR(length)[8,1]
RETURN

do_motion:
' Main motion routine
```

Example 2 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.

This could also be achieved with an MC206 or an MC216 + Axis expander, but in this example because each block can be built as a stand-alone unit, each requires it's own *Motion Coordinator*. In this example our system uses two MC216 controllers.

Our network will consist of the two *Motion Coordinator*s and the membrane keypad. It should be connected as shown in below.



Network Example 2

As there are now two *Motion Coordinator*s in the network it is 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 previous example highlights a couple of useful points:

1) 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. 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 Coordinator*s in the network to have a delay at the start of their program to allow time for the membrane keypads to be initialised.

2) 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.

Network Specification

Message Format

Each message is constructed of a header character, the message, a cwt. check (on direct variable transfers only) and an end of message character.

Bit	7	6	5	4	3	2	1	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:
- 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 Trio BASIC and is for internal use only.
- The number of message bytes (n) is determined by the message type as described below
- CRC bytes are used for direct variable transfers (message type 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 re-transmits 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 re-transmission is invisible to the user.

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 user program. 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.

4 Keypad Mode
Message is 1 byte long and controls the response of the keypad to key presses.

0 = Character sent on Key ON only
127 = Key ON/Key OFF each send seperate characters

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 Trio BASIC program.

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:

Bit	Value	Error Type	Description
0	1	TX Timeout	Shows that a problem has occurred while trying to put information into the transmit buffer.
1	2	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.
2	4	RX CRC Error	the received variable has become corrupt. The error flag is set but the variable isn't.
3	8	RX Frame Error	an incomplete or corrupt message has been received.

CHAPTER

12

USING THE TRIO ACTIVEX CONTROL

Introduction

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

Requirements

- PC with one or more of USB interface, Ethernet network interface, or PCI based *Motion Coordinator*.
- Windows 98, ME, 2000 or XP (Windows 2000 or XP only for PCI connection)
- TrioUSB driver - for USB connection
- Trio PCI driver - for PCI connection (Windows 2000 and XP systems only)
- TrioPC OCX
- Knowledge of the Trio *Motion Coordinator* to which the TrioPC ActiveX controls will connect.
- Knowledge of the Trio BASIC programming language.

Installation of the ActiveX Component

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

Using the Component

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

From the Menu select Project then Components... (or use shortcut ctrl+T).

When the Components dialogue box has opened, scroll down until you find “TrioPC ActiveX Control Module” then click in the block next to TrioPC. (A tick will appear)

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

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

Connection Commands

Open

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

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

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

Syntax: `Open(PortType, PortMode)`

Parameters

`short PortType:`	0: USB, 1:N/A, 2:Ethernet, 3:PCI
`short PortMode:`	0: Synchronous Mode, 1:Asynchronous Mode, 3240:Synchronous Mode (for Ethernet connections only).

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

Example

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

Close

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

Syntax: `Close(PortMode)`

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

Return Value: `None.`

Example

```
Rem Close the connection when form unloads
Private Sub Form_Unload(Cancel As Integer)
    TrioPC1.Close(0)
    frmMain.Refresh
End Sub
```

IsOpen

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

Syntax: `IsOpen(PortMode)`

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

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

Example

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

SetHost

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

Syntax: `SetHost(host)`

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

Return Value: `None`

Example

```
Rem Set up the Ethernet IP Address of the target Motion
Coordinator
TrioPC1.SetHost("192.168.000.001")
Rem Open a Synchronous connection
TrioPC_Status = TrioPC1.Open(2, 0)
frmMain.Refresh
```

GetConnectionType

Description Gets the connection type of the current connection.

Syntax: `GetConnectionType()`

Parameters `None`

Return Value: -1: No Connection, 0: USB, 1:N/A, 2: Ethernet, 3: PCI

Example

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

Properties

Board

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

Type Long

Access Read / Write

Default Value 0

Example

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

HostAddress

Description Used for reading or changing the ethernet host IP address, and must be set prior to opening an ethernet connection. The **SetHost** command can also be used for setting the host adddress.

Type String

Access Read / Write

Default Value "192.168.0.250"

Example

```
Rem Open a Ethernet connection and refresh the TrioPC indicator
if TrioPC.HostAddress <> "192.168.0.111" Then
    TrioPC.HostAddress = "192.168.0.111"
End If
TrioPC_Status = TrioPC1.Open(2, 0)
frmMain.Refresh
```

CmdProtocol

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

0: for ethernet daughterboard firmware version 1.0.4.0 or earlier.
1: for ethernet daughterboard firmware version 1.0.4.1 or later.

Type Long

Access Read / Write

Default Value `1`

Example

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

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

Motion Commands

MoveRel

Description Performs the corresponding `MOVE(…)` command on the *Motion Coordinator*

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

Parameters:

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

Return Value: `TrioPC STATUS.`

Base

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

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

Parameters:

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

Return Value: `TrioPC STATUS.`

MoveAbs

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

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

Parameters:

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

Return Value: **TrioPC STATUS.**

MoveCirc

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

Syntax: **MoveCirc(EndBase, EndNext, CentreBase, CentreNext, Dir, [Axis])**

Parameters:

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

Return Value: **TrioPC STATUS.**

AddAxis

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

Syntax: **AddAxis(LinkAxis, [Axis])**

Parameters:

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

Return Value: **TrioPC STATUS.**

CamBox

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

Syntax: **CamBox(TableStart, TableStop, Multiplier, LinkDist, LinkAxis, LinkOpt, LinkPos, [Axis])**

Parameters:

short TableStart:	The position in the table data on the *Motion Coordinator* where the cam pattern starts
short TableStop:	The position in the table data on the *Motion Coordinator* where the cam pattern stops
double Multiplier:	The scaling factor to be applied to the cam pattern
double LinkDist:	The distance the input axis must move for the cam to complete
short LinkAxis:	Definition of the Input Axis
short LinkOpt:	**1** link commences exactly when registration event occurs on link axis **2** link commences at an absolute position on link axis (see param 7) **4** CAMBOX repeats automatically and bi-directionally when this bit is set.
double LinkPos:	The absolute position on the link axis where the cam will start.
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **TrioPC STATUS.**

Cam

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

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

Parameters:

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

Return Value: `TrioPC STATUS.`

Cancel

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

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

Parameters:

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

Return Value: `TrioPC STATUS.`

Connect

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

Syntax: **Connect(Ratio, LinkAxis, [Axis])**

Parameters:

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

Return Value: **TrioPC STATUS.**

Datum

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

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

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

0. The current measured position is set as demand position (this is especially useful on stepper axes with position verification). **DATUM(0)** will also reset a following error condition in the **AXISSTATUS** register for all axes.
1. The axis moves at creep speed forward till the Z marker is encountered. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
2. The axis moves at creep speed in reverse till the Z marker is encountered. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
3. The axis moves at the programmed speed forward until the datum switch is reached. The axis then moves backwards at creep speed until the datum switch is reset. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.

4. The axis moves at the programmed speed reverse until the datum switch is reached. The axis then moves at creep speed forward until the datum switch is reset. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
5. The axis moves at programmed speed forward until the datum switch is reached. The axis then moves at creep speed until the datum switch is reset. The axis is then reset as in mode 2.
6. The axis moves at programmed speed reverse until the datum switch is reached. The axis then moves at creep speed forward until the datum switch is reset. The axis is then reset as in mode 1.

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

Return Value: `TrioPC STATUS.`

Forward

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

Syntax: `Forward([Axis])`

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

Return Value: `TrioPC STATUS.`

Reverse

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

Syntax: `Reverse([Axis])`

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

Return Value: `TrioPC STATUS.`

MoveHelical

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

Syntax: **MoveHelical(FinishBase, FinishNext, CentreBase, CentreNext, Direction, LinearDistance, [Axis])**

Parameters

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

Return Value: **TrioPC STATUS.**

MoveLink

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

Syntax: **MoveLink(Distance, LinkDistance, LinkAcc, LinkDec, LinkAxis, LinkOptions, LinkPosn, [Axis])**

Parameters:

double Distance:	Total distance to move on the base axis
double LinkDistance:	Distance to be moved on the driving axis

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

Return Value: **TrioPC STATUS.**

MoveModify

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

Syntax: **MoveModify(Position, [Axis]**

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

Return Value: **TrioPC STATUS.**

RapidStop

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

Parameters: None

Return Value: `TrioPC STATUS`.

Process Control Commands

Run

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

Syntax: `Run(Program, Process)`

Parameters:

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

Return Value: `TrioPC STATUS.`

Stop

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

Syntax: `Stop(Program, Process)`

Parameters:

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

Return Value: `TrioPC STATUS.`

Variable Commands

GetTable

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

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

Parameters

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

Return Value: `TrioPC STATUS`.

GetVariable

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

Syntax: `GetVariable(Variable, Value)`

Parameters

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

Return Value: `TrioPC STATUS`.

GetVr

Description: Returns the current value of the specified Global variable.

Syntax: `GetVr(Variable, Value)`

Parameters:

`short Variable:`	Number of the VR variable to read.
`double *Value:`	Variable in which to store the value read.

Return Value: `TrioPC STATUS.`

SetTable

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

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

Parameters

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

Return Value: `TrioPC STATUS.`

SetVariable

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

Syntax: `SetVariable(Variable, Value`

Parameters

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

Return Value: `TrioPC STATUS.`

SetVr

Description: Sets the value of the specified Global variable.

Syntax: `SetVr(Variable, Value)`

Parameters:

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

Return Value: `TrioPC STATUS.`

Input / Output Commands

Ain

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

Syntax: **Ain(Channel, Value)**

Parameters

short Channel:	AIN channel to be read.
double *Value:	Variable in which to store the value read.

Return Value: **TrioPC STATUS.**

Get

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

Syntax: **Get(Channel, Value)**

Parameters

short Channel:	Comms channel to be read
double *Value:	Variable in which to store the value read.

Return Value: **TrioPC STATUS.**

In

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

Syntax: **In(StartChannel, StopChannel, Value)**

Parameters:

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

Return Value: `TrioPC STATUS.`

Input

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

Syntax: `Input(Channel, Value)`

Parameters:

`short Channel:`	Comms channel to be read
`double *Value:`	Variable in which to store the value read.

Return Value: `TrioPC STATUS.`

Key

Description Performs the corresponding `KEY #…` command on the *Motion Coordinator*.

Syntax: `Key(Channel, Value)`

Parameters

`short Channel:`	Comms channel to be read
`double *Value:`	Variable in which to store the value read.

Return Value: `TrioPC STATUS.`

Linput

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

Syntax: `Linput(Channel, Startvr)`

Parameters:

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

Return Value: `TrioPC STATUS.`

Op

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

Syntax: `Op(Output, State)`

Parameters:

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

Return Value: `TrioPC STATUS.`

Pswitch

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

Syntax: **Pswitch(Switch, Enable, Axis, OutputNumber, OutputStatus, SetPosition, ResetPosition)**

Parameters:

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

Return Value: **TrioPC STATUS.**

ReadPacket

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

Syntax: **ReadPacket(PortNumber, StartVr, NumberVr, Format)**

Parameters:

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

Return Value: **TrioPC STATUS.**

Record

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

Syntax: `Record(Transitions, TablePosition)`

Parameters:

`short Transitions:`	Number of transitions to record.
`long TablePosition:`	Start position in the table to store the transitions.

Return Value: `TrioPC STATUS.`

Regist

Description Performs the corresponding `REGIST(…)` command on the *Motion Coordinator*

Syntax: `Regist(Mode, Dist)`

Parameters: `short Mode:` Registration mode

1. Axis absolute position when Z Mark Rising
2. Axis absolute position when Z Mark Falling
3. Axis absolute position when Registration Input Rising
4. Axis absolute position when Registration Input Falling
5. Sets pattern recognition mode

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

Return Value: `TrioPC STATUS.`

Send

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

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

Parameters:

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

Return Value: `TrioPC STATUS.`

Setcom

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

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

Parameters:

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

Return Value: `TrioPC STATUS.`

General commands

Execute

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

Syntax: **Execute(Command)**

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

Return Value: **TrioPC STATUS: TRUE** if the command was sent successfully to the *Motion Coordinator* and the **EXECUTE** command on the *Motion Coordinator* was completed successfully and the command specified by the **EXECUTE** command was tokenised, parsed and completed successfully.

GetData

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

Syntax: **GetData(channel, data)**

Parameters **short channel:** Channel over which the required data was received (5,6,7, or 9).

VARIANT data: data received by the control from the *Motion Coordinator*

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

SendData

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

Syntax: `SendData(channel, data)`

Parameters

`short channel:` channel over which to send the data (5,6,7, or 9).

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

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

Events

OnBufferOverrunChannel5/6/7/9

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

Syntax: `OnBufferOverrunChannelx()`

Parameters: `None.`

Return Value: `None.`

OnReceiveChannel5/6/7/9

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

Syntax: `OnReceiveChannelx()`

Parameters: `None.`

Return Value: `None.`

TrioPC status

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

CHAPTER

13

COMMUNICATIONS PROTOCOLS

MODBUS RTU

Introduction

A growing number of programmable keypads and HMIs provide the user with a choice of serial interface protocols to enable communication with various PLCs and Industrial Computers. One such protocol is Modbus RTU. The *Motion Coordinator* system software provides built-in support for the Modbus protocol.

Scope of Operation

This document applies to *Motion Coordinator*s with system software version 1.48 and above.

The Modbus RTU protocol provides single point to point communication between the *Motion Coordinator* and a programmable keypad/display. Implementation of the protocol is provided on serial port 1 for RS232 and port 2 for RS485. Port 0 is the main programming port and does not have the Modbus option. Baud rate and slave address can be set in the Trio BASIC program during serial port initialisation.

Initialisation and Set-up

The Modbus protocol is initialised by setting the mode parameter of the SETCOM instruction to 4. The **ADDRESS** parameter must also be set *before* the Modbus protocol is activated.

example:

```
ADDRESS=1
SETCOM(9600,8,1,2,1,4) 'Port 1 as MODBUS port at 9600 baud
ADDRESS=1
SETCOM(19200,8,1,2,2,4) 'set up the RS485 port at 19200 baud
```

The protocol can be de-selected by setting the option to 0 in the SETCOM command.

```
SETCOM(19200,8,1,2,2,0) 'set the RS485 port to normal mode
```

Example The following shows a typical set-up for a HMI panel running a Modbus Link. All references below are to the programming software supplied by the HMI manufacturer and are not specific to any individual programming environment. See your HMI programming instructions for the actual set-up sequence.

In the Controller Driver section choose “Modicon Modbus”, choose any Modicon PLC type from the PLC setup section.

Program the panel to display a variable and open up a dialog box to **Define**

Choose	Example
Input bits, Output bits, Holding Register.	Holding Register
Data size/type	WORD (Binary)
Address Offset.	
Display format and field width to be displayed.	Numeric 4 digits

The *Motion Coordinator* is the slave so it will always wait for the HMI to request the data required. With the set-up shown above, the display should poll the *Motion Coordinator* for the value of VR(12) and display the data as a 4 digit number.

Modbus Technical Reference

This section lists the *Motion Coordinator*’s response to each supported Modbus Function.

Modbus Code Table

The following Modbus Function Codes are implemented:

Code	Function Name	Action
1	Read Coil Status	Returns input/output bit pattern
2	Read Input Status	Returns input/output bit pattern
3	Read Holding Registers	Returns data from VR() variables
5	Force Single Coil	Sets single output ON/OFF
6	Preset Single Register	Sets the value of a single VR() variable
16	Preset Multiple Registers	Sets the values of a group of VR() variables

(1 and 2) Read Coil Status / Read Input Status

Modbus Function Code	1 & 2
Mapped Trio Function	Read input word: IN(nn,mm)
Starting Address Range	0 to NIO-1 (NIO = Number of Input/Output Bits on Controller)
Number of Points Range	1 to (NIO-1) – Starting Address
Returned Data	Bytes containing “Number of Points” bits of data

(3) Read Holding Registers

Modbus Function Code	3
Mapped Trio Function	Read VR() Global Variable
Starting Address Range	0 to 250
Number of Points Range	1 to 127 (Number of VR() variables to be read)
Returned Data	2 to 254 bytes containing up to 127 16-bit Signed Integers.

(5) Force Single Coil

Modbus Function Code	5
Mapped Trio Function	Set Single Output: OP(n,ON/OFF)
Starting Address Range	8 to NIO-1
Data	00 = Output OFF, ffH = Output ON
Returned Data	None

(6) Preset Single Register

Modbus Function Code	6
Mapped Trio Function	Set VR() Global Variable: VR(addr)=data
Register Address Range	0 to 250
Data	-32768 to 32767 (16 bit signed)
Returned Data	None

(16) Preset Multiple Registers

Modbus Function Code	6
Mapped Trio Function	Set VR() Global Variables: VR(addr)=$data_1$ …… VR(addr+n)=$data_n$
Starting Address Range	0 to 250
Number of Points Range	1 to 127
$Data_1$ to $Data_n$	-32768 to 32767 (16 bit signed)
Returned Data	None

Notes The following baud rate limitations should be observed when attaching a HMI panel to the *Motion Coordinator* using Modbus.

Motion Coordinator	Maximum Baud Rate
MC202	9600
Euro205	38400
MC206	38400
MC216	38400
MC224	38400

Some HMI's use the standard MODICON addressing for registers and I/O. If this is the case, use the following mappings:

- Holding Registers 40001 + are mapped to VR(0) +
- Inputs 10001 to 10272 are mapped to IN(0) to IN(271) when the appropriate I/O expansion is fitted.
- Output Coils 9 to 272 are mapped to OP(8,s) to OP(271,s) where s is the state (ON or OFF)

Glossary

HMI	Human – Machine Interface
MODBUS	A communications protocol developed by Modicon, part of Groupe Schneider.
RTU	One of two serial transmission modes used by Modbus, the other being ASCII.
Holding Register	A read/write variable as defined for Modicon PLC.
Coil	A programmable output as defined for Modicon PLC.

Profibus

This section applies to the BASIC program developed for Motion Coordinator types that can take the P297 Profibus DP Daughter Board. The program is provided for evaluation and example purposes and no guarantee is made as to its suitability for a particular Profibus application.

In order to include the Motion Coordinator in a Profibus network the following components are required:

1. Trio BASIC program P297DRxxx.bas (where xxx is the version number of the program)

2. Profibus GSD file; TRIO0595.GSD. (Electronic Data Sheet for COM PROFIBUS)

3. Motion Perfect and serial programming cable.

The program example and Profibus GSD file are available to download from the Trio Website **www.triomotion.com**.

Installation and Set-up

1. Trio BASIC program.

The program must be loaded into the Motion Coordinator and set to run from power-up. Set the "node" variable in the program to the required Profibus Address for the Motion Coordinator. Make sure the "db" variable is set to the slot number of the Profibus Daughter Board.

Once the SPC3 chip on the daughter board is initialised, the software is very efficient at transferring data in and out. In the MC204, Euro205 and MC216 however, a fast process is recommended for optimum running. When using the MC206 or MC224, a low process number may be used with less impact on processing speed.

2. Profibus GSD File.

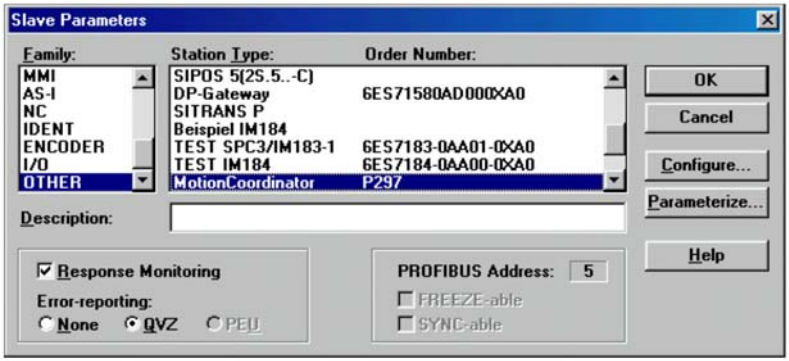
The GSD file supplied (TRIO0595.GSD) must be copied to the GSD folder used by COM PROFIBUS or the equivalent Profibus configuration tool supplied by the PLC vendor. The Motion Coordinator can then be added to the Profibus network by selecting OTHER and P297 Motion Coordinator from the list.

The following sequence shows how to include the Motion Coordinator in a fieldbus network using COM PROFIBUS.

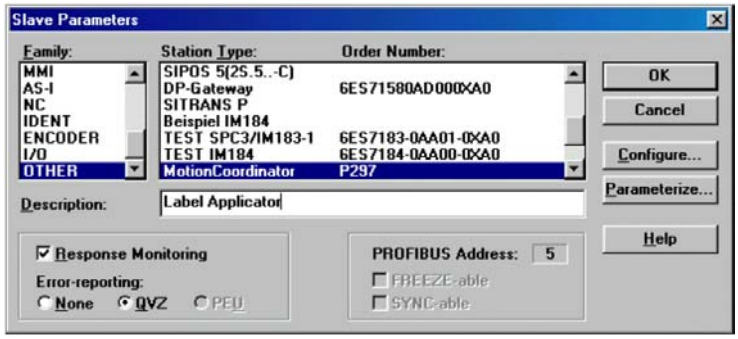
1. Launch the window shown below and click on “Others”.



2. Add the Box Icon to the network on the left and the Slave Parameters dialogue will open. Choose Motion Coordinator P297 from the list as shown here:



3. Add your own description in the text box like this:



4. Click OK and the Motion Coordinator will appear on the diagram like this:



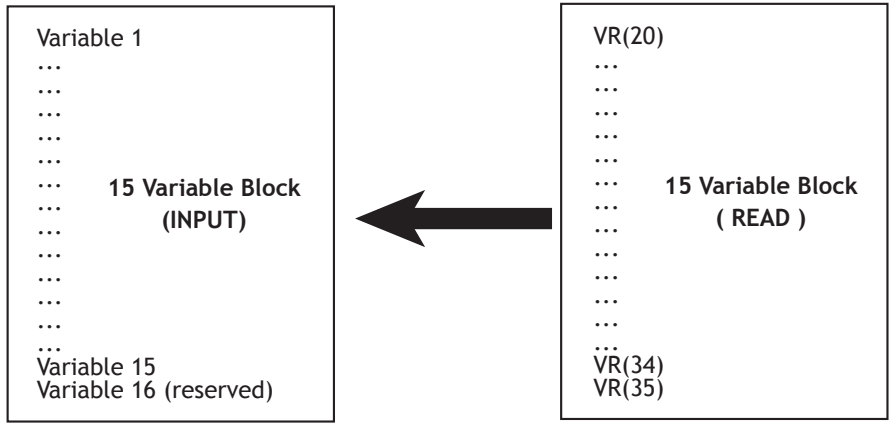
Now add any other nodes to the network that are required and close the window. Finally export the file in the required format, usually Binary, for use by the PLC or other Profibus Master. The master will now search for the Motion Coordinator on the Profibus network and when found will connect to it and start transferring the variable blocks.

Example: Trio BASIC Profibus Driver

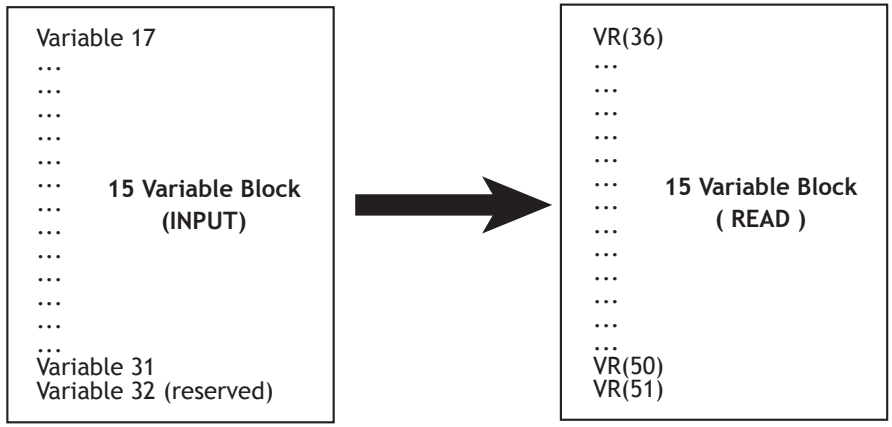
The latest driver can be obtained from www.triomotion.com. At the time of going to press, the most recent revision was 1.05. See below for header and revision information / dates.

```
'=========================================================================
' Title:        Profibus DP SPC3 Driver for P297 Daughter Board
' Module:       P297DR104.BAS
' Platform:     MC Series 2xx + P297 Daughter Board
'-------------------------------------------------------------------------
' Revision:     1.05
' Rev Date:     25 May 2003
'-------------------------------------------------------------------------
'             Copyright (c) 2002-2003 Trio Motion Technology Ltd
'
=========================================================================
' DESCRIPTION:
'  Profibus driver for Cyclic Data Transfer
'  This program sets up the SPC3 chip for transfer of 16 integers from
'  master and 16 intergers to master on a cylcic basis as determined by
'  master unit.
'  Variables:
'   VR(vbase) to VR(vbase+15)    : Data TO master (16bit int)
'   VR(vbase+16) to VR(vbase+31) : Data FROM master (16bit int)
'   VR(vbase+32) = PLC status. 0=running, 2=PLC in configure mode
'
' V1.00: 08/01/2001 Beta Release includes full 16+16 VR transfer
' V1.01: 15/03/2001 Reversed byte order for Word transfer
' V1.02: 24/04/2002 Added support for "Leave Data Ex". (S7 config sequence)
'                   Added local timout if no connection after BR detect.
' V1.03: 08/08/2002 Allows VR segment used to be changed by setting a variable
' V1.04: 23/10/2002 Made local timeout value settable.  See "localtimeout"
' V1.05: 25/03/2003 Fixed bug in sending negative numbers.

restart:
RESET
node = 5 '    Profibus node address
debug = TRUE 'Set TRUE to get debug messages printed to terminal
db=0 '        Daughter Board slot number
vbase = 20 '  VRs for data transfer
localtimeout = 5000 'time in msec
```



Profibus Cyclic Data Transfer
16-bit signed integer



DeviceNet

The *Motion Coordinators* MC206 and MC224 have a ***DeviceNet*** option that allows the *Motion Coordinator* to be attached as a slave node to a ***DeviceNet*** factory network. Either to built-in CANbus port or a P290 daughter board may be used as the physical connection to the ***DeviceNet*** network.

If the built-in CANbus port is used, it will not be available for CAN I/O expansion, so the digital I/O will be limited to the 8 in and 8 bi-directional on the *Motion Coordinator* itself.

Note that the CAN daughter board P290 is not tailored for use on ***DeviceNet*** and a ***DeviceNet*** buffer/isolator may be required.

Installation and Set-up

The DEVICENET TrioBASIC command must be in a program that runs at power-up. See the command reference in chapter 8 for infromation about the use of the DEVICENET command. In order to prevent the *Motion Coordinator* from acting as a CANIO master and generating non-DeviceNet CANbus messages on power-up, set the CANIO_ADDRESS to 33. This parameter is written directly into Flash EPROM adn so it is only necessary to set CANIO_ADDRESS once.

e.g. in an intialisation program:

```
IF CANIO_ADDRESS<>33 THEN CANIO_ADDRESS = 33
DEVICENET(slot, 0, baudrate, macid, pollbase, pollin, pollout)
```

DeviceNet Information

This Section contains DeviceNet information for the multi-axis Trio *Motion Coordinator* model MC206 and MC224.

The *Motion Coordinator* operates as a slave device on the DeviceNet network and supports Explicit Messages of the predefined master/slave connection set and Polled I/O. It does not support the Explicit Unconnected Message Manager (UCMM).

Polled I/O allows the master to send up to 4 integer variables to the Motion Coordinator and to read up to 4 integer variables from the Motion Coordinator. These values are mapped to the TABLE memory in the *Motion Coordinator.* The values are transferred periodically at a rate determined by the DeviceNet Master. The Global variables (VRs) and TABLE memory are also accessible over DeviceNet individually by way of the Explicit Messaging service.

Connection Types Implemented

There are 3 independent connection channels in this DeviceNet implementation:

1. Group 2 predefined master/slave connection

   This connection will only handle Master/Slave Allocate/Release messages. The maximum message length for this connection is 8 bytes.

2. Explicit message connection

   This connection will handle explicit messaging for the DeviceNet objects defined below. The maximum message length for this connection is 242 bytes.

3. I/O message connection

   This connection will handle the I/O poll messaging. The maximum message length for this connection is 32 bytes.

DeviceNet Objects Implemented

The *Motion Coordinator* supports the following DeviceNet object classes.

Class	Object	Description
0x01	Identity	Identification and general information about the device
0x02	Router	Provides a messaging connection point through which a Client may address a service to any object class or instance residing in the physical device
0x03	DeviceNet	Provides the configuration and status of a DeviceNet port
0x04	Assembly	Permits access to the I/O poll connection from the explicit message channel
0x05	Connection	Manages the characteristics of the communications connections
0x8a	MC	Permits access to the VR variables and TABLE data on the *Motion Coordinator*

Identity Object

Class Code: 0x01

Instance Services

Id	Service	Description
0x05	Reset	Reinitialises the DeviceNet protocol
0x0E	Get Attribute Single	Used to read the instance attributes

Instance Attributes

Attribute ID	Access Rule	Name	DeviceNet Data Type	Data Value
1	Get	Vendor	UINT	0x0115 (277)
2	Get	Product Type	UINT	Generic Device (0x0000)
3	Get	Product Code	UINT	The MC type as returned by the CONTROL system variable.
4	Get	Revision Major Revision Minor Revision	Structure of: USINT USINT	 3 2
5	Get	Status	WORD	Ony bit 0 (owned) is implemented
6	Get	Serial Number	UDINT	The MC Serial Number
7	Get	Product Name String Length ASCII String1	Structure of: USINT STRING(30)	11 "Trio MC_<product code>", where <product code> is the same as defined for attribute 3.

DeviceNet Object

Class Code: 0x03

Class Services

Id	Service	Description
0x0E	Get Attribute Single	Used to read the class attributes

Class Attributes

Attribute ID	Access Rule	Name	DeviceNet Data Type	Data Value
1	Get	Revision	UINT	2

Number of Instances: 1

Instance Services

Id	Service	Description
0x0E	Get Attribute Single	Used to read the instance attributes
0x10	Set Attribute Single	Used to write the instance attributes
0x4B	Allocate Master/ Slave Connection Set	Requests the use of the Predefined Master/Slave Connection set
0x4C	Release Group 2 Identifier Set	Indicates that the specified Connections within the Predefined Master/Slave Connection Set are no longer desired. These Connections are to be released (Deleted).

Instance Attributes

Attribute ID	Access Rule	Name	DeviceNet Data Type	Data Value
1	Get	MAC ID	USINT	DeviceNet node address. Software defines
5	Get	Allocation Information	Structure of: BYTE USINT	 0-63 = master address The current allocation choice

Allocation_byte

bit 0	explicit message	Supported, 1 to allocate
bit 1	Polled	Supported, 1 to allocate
bit 2	Bit_strobed	Not supported, always 0
bit 3	reserved	always 0

Assembly Object

Class Code: 0x04

Number of Instances: 2

There are 2 instances implemented. Instance 100 is a static input object, associated with the I/O poll producer. Instance 101 is a static output object, associated with the I/O poll consumer.

Instance Services

Id	Service	Description
0x0E	Get Attribute Single	Used to read the instance attributes
0x10	Set Attribute Single	Used to write the instance attributes

Instance Attributes

Attribute ID	Access Rule	Attibute	Description
3	Get / Set	Data	Get Instance 100 : The I/O poll producer is executed and the output buffer returned Set Instance 100: Error Get Instance 101: The last received I/O poll buffer is returned Set Instance 101: The buffer received is passed to the I/O poll consumer

Connection Object

Class Code: 0x05

Instance Services

Id	Service	Description
0x0E	Get Attribute Single	Used to read the instance attributes
0x10	Set Attribute Single	Used to write the instance attributes

Number of Instances: 2

The values for these attributes are defined in the "Predefined master/slave connection set" of the "ODVA DeviceNet specification".

Instance Attributes (Instance 1)

Instance Type : Explicit Message

Attribute ID	Access Rule	Name	DeviceNet Data Type	Data Value
1	Get	State	USINT	0 = nonexistent 1 = configuring 3 = established 4 = timed out
2	Get	Instance Type	USINT	0 = explicit message
3	Get	Transport Class Trigger	USINT	83 hex
4	Get	Produced Connection ID	UINT	10xxxxxx011 binary xxxxxx = node address
5	Get	Consumed Connection ID	UINT	10xxxxxx100 binary xxxxxx = node address
6	Get	Initial Comm Characteristics	USINT	21 hex
7	Get	Produced Connection Size	UINT	7
8	Get	Consumed Connection Size	UINT	7
9	Get / Set	Expected Packet Rate	UINT	2500 default (msec) with timer resolution of 1mS
12	Get	Watchdog Timeout Action	USINT	1 = autodelete
13	Get	Produced Connection Path Length	USINT	0
14	Get	Produced Connection Path		Null (no data)
15	Get	Consumed Connection Path Length	USINT	0
16	Get	Consumed Connection Path		Null (no data)

Instance Attributes (Instance 2)

Instance Type : Polled I/O

Attribute ID	Access Rule	Name	DeviceNet Data Type	Data Value
1	Get	State	USINT	0 = nonexistent 1 = configuring 3 = established 4 = timed out
2	Get	Instance Type	USINT	1 = Polled I/O
3	Get	Transport Class Trigger	USINT	0x83
4	Get	Produced Connection ID	UINT	01111xxxxxx binary xxxxxx = node address
5	Get	Consumed Connection ID	UINT	10xxxxxx101 binary xxxxxx = node address
6	Get	Initial Comm Characteristics	USINT	0x01
7	Get	Produced Connection Size	UINT	0x08
8	Get	Consumed Connection Size	UINT	0x08
9	Get / Set	Expected Packet Rate	UINT	2500 default (msec) with timer resolution of 1 msec
12	Get	Watchdog Timeout Action	USINT	0
13	Get	Produced Connection Path Length	USINT	0
14	Get	Produced Connection Path		Null (no data)
15	Get	Consumed Connection Path Length	USINT	0
16	Get	Consumed Connection Path		Null (no data)
17	Get	Production Inhibit Time	USINT	0

MC Object

Class Code: 0x8A

Instance Services

Id	Service	Description
0x05	Reset	Performs EX on the Motion Coordinator. This will reset the DeviceNet as well.
0x33	Read Word - TABLE	Reads the specified number of TABLE entries and sends their values in 16 bit 2s complement format
0x34	Read Word - VR	Reads the specified number of TABLE entries and sends their values in 16 bit 2s complement format
0x35	Read IEEE - TABLE	Reads the specified number of TABLE entries and sends their values in 32 bit IEEE floating point format
0x36	Read IEEE - VR	Reads the specified number of VR entries and sends their values in 32 bit IEEE floating point format
0x37	Write Word - TABLE	Receives the specified number of values in 16 bit 2s complement format and writes them into the specified VR entries
0x38	Write Word - VR	Receives the specified number of values in 16 bit 2s complement format and writes them into the specified TABLE entries
0x39	Write IEEE - TABLE	Receives the specified number of values in 32 bit IEEE floating point format and writes them into the specified TABLE entries
0x3A	Write IEEE - VR	Receives the specified number of values in 32 bit IEEE floating point format and writes them into the specified VR entries

The following sections describe the message body area of the Explicit Message used to specify the different services. This ignores all of the fragmentation protocol.

Read word format

Request

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	0	Service code = 0x33, or 0x34						
byte 1	Class ID = 0x8A							
byte 2	Instance ID = 0x01 (this is the only instance supported)							
byte 3	bits 15-8 of Source Address							
byte 4	bits 7-0 of Source Address							
byte 5	ignored							
byte 6	Number of word values to be read							

Response

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	1	Service code = 0x33, or 0x34						
byte 1	bits 15-8 of Value 0							
byte 2	bits 7-0 of Value 0							
	...							
byte n	bits 15-8 of Value m							
byte n + 1	bits 7-0 of Value m							

Write word format

Request

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	0	Service code = 0x37, or 0x38						
byte 1	Class ID = 0x8A							
byte 2	Instance ID = 0x01 (this is the only instance supported)							
byte 3	bits 15-8 of Source Address							
byte 4	bits 7-0 of Source Address							
byte 5	ignored							
byte 6	Number of word values to be written							
byte 7	bits 15-8 of Value 0							
byte 8	bits 7-0 of Value 0							
	...							
byte n	bits 15-8 of Value m							
byte n + 1	bits 7-0 of Value m							

Response

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	1	Service code = 0x37, or 0x38						

Read IEEE format

Request

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	0	Service code = 0x35, or 0x36						
byte 1	Class ID = 0x8A							
byte 2	Instance ID = 0x01 (this is the only instance supported)							
byte 3	bits 15-8 of Source Address							
byte 4	bits 7-0 of Source Address							
byte 5	ignored							
byte 6	Number of IEEE values to be read							

Response

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	1	Service code = 0x35, or 0x36						
byte 1	bits 7-0 of Value 0							
byte 2	bits 15-8 of Value 0							
byte 3	bits 23-16 of Value 0							
byte 4	bits 31-24 of Value 0							
	...							
byte n	bits 7-0 of Value m							
byte n + 1	bits 15-8 of Value m							
byte n + 2	bits 23-16 of Value m							
byte n + 3	bits 31-24 of Value m							

Write IEEE format

Request

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	0	Service code = 0x39, or 0x3A						
byte 1	Class ID = 0x8A							
byte 2	Instance ID = 0x01 (this is the only instance supported)							
byte 3	bits 15-8 of Source Address							
byte 4	bits 7-0 of Source Address							
byte 5	ignored							
byte 6	Number of IEEE values to be written							
byte 7	bits 7-0 of Value 0							
byte 8	bits 15-8 of Value 0							
byte 9	bits 23-16 of Value 0							
byte 10	bits 31-24 of Value 0							
	...							
byte n	bits 7-0 of Value m							
byte n + 1	bits 15-8 of Value m							
byte n + 2	bits 23-16 of Value m							
byte n + 3	bits 31-24 of Value m							

Response

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	1	Service code = 0x39, or 0x3A						

DeviceNet Status LEDs

To see the DeviceNet status on the Motion Coordinator's LEDs, you must set the DISPLAY system parameter to 8. (See the DISPLAY system variable in chapter 8)

The Motion Coordinator's I/O status LEDs are now read from 0 to 7 as follows:

LED	Function
0 - Amber	Module Status - DeviceNet GREEN LED
1 - Amber	Module Status - DeviceNet RED LED
2 - Amber	Network Status - DeviceNet GREEN LED
3 - Amber	Network Status - DeviceNet RED LED
4 - Amber	Bus Off Count - bit 0
5 - Amber	Bus Off Count - bit 1
6 - Amber	Bus Off Count - bit 2
7 - Amber	Bus Off Count - bit 3

LEDs 0 to 3 have the standard meanings as stated in the ODVA *DeviceNet* manual.

Module Status:

Off - No Power
Flashing Green/Red - Module self test
Flashing Green - Standby
Steady Green - Operational
Flashing Red - Minor Fault (software recoverable)
Steady Red - Major Fault

Network Status:

Off - Not On-line
Flashing Green - On-line, not connected
Steady Green - On-line, connected
Flashing Red - Connection timeout
Steady Red - Critical link failure

Ethernet

The Ethernet daughter board P296 can be fitted to both the MC206 and MC224 *Motion Coordinators*. This section describes how to set up a simple Ethernet connection to the P296.

Default IP Address

The IP address (Internet Protocol address) is a 32-bit address that has two parts: one part identifies the network, with the network number, and the other part identifies the specific machine or host within the network, with the host number. An organization can use some of the bits in the machine or host part of the address to identify a specific subnet. Effectively, the IP address then contains three parts: the network number, the subnet number, and the machine number.

The 32-bit IP Address is often depicted as a dot address (also called dotted quad notation) - that is, four groups of decimal digits separated by points.

For example, the Trio Ethernet daughter board has a default IP of:

```
192.168.000.250
```

Each of the decimal numbers represents a string of eight binary digits. Thus, the above IP address really is this string of 0s and 1s:

```
11000000.10101000.00000000.11111010
```

As you can see, points are inserted between each eight-digit sequence just as they are in the decimal version of the IP address. Obviously, the decimal version of the IP address is easier to read and that's the form most commonly used (192.168.000.250).

Part of the IP address represents the network number or address and another part represents the local machine address (also known as the host number or address). IP addresses can be one of several classes, each determining how many bits represent the network number and how many represent the host number. IP addresses are grouped by classes A,B,C, D and E. The Trio ethernet board is set up for a Class C address.

Using the above example, here's how the IP address is divided:

```
<-Network address->.<-Host address->
         192.168    .    000.250
```

The beginning Network Address portion of 192 begins with the first three bits as 110... and classifies it as a Class C address. This means you can have up to 256 host addresses on this particular network.

If you wanted to add sub-netting to this address, then some portion (in this example, eight bits) of the host address could be used for a subnet address. Thus:

```
<-Network address->.<-Subnet address->.<-Host address->
       192.168    .          000       .      250
```

To simplify this explanation, the subnet has been divided into a neat eight bits but an organization could choose some other scheme using only part of the third quad or even part of the fourth quad.

A subnet (short for "sub-network") is an identifiably separate part of an organization's network. Typically, a subnet may represent all the machines at one geographic location, in one building, or on the same local area network (LAN). Having an organization's network divided into subnets allows it to be connected to the Internet with a single shared network address. Without subnets, an organization could get multiple connections to the Internet, one for each of its physically separate sub-networks, but this would require an unnecessary use of the limited number of network numbers the Internet has to assign. It would also require that Internet routing tables on gateways outside the organization would need to know about and have to manage routing that could and should be handled within an organization.

The Subnet Mask

Once a packet has arrived at an organization's gateway or connection point with its unique network number, it can be routed within the organization's internal gateways using the subnet number as well. The router knows which bits to look at (and which not to look at) by looking at a subnet mask. A mask is simply a screen of numbers that tells you which numbers to look at underneath. In a binary mask, a "1" over a number says "Look at the number underneath"; a "0" says "Don't look." Using a mask saves the router having to handle the entire 32-bit address; it can simply look at the bits selected by the mask.

Using the Trio default IP address, the combined network number and subnet number occupy 24 bits or three of the quads. The default subnet mask carried along with the packet is:

```
255.255.255.000
```

Or a string of all 1's for the first three quads (telling the router to look at these) and 0's for the host number (which the router doesn't need to look at). Subnet masking allows routers to move the packets on more quickly.

Connecting to the Trio Ethernet Daughter Board

The following steps can be followed to establish an ethernet connection from a PC to the MC controller.

1. One-to-One Connection

For a PC to the MC controller direct connection, use a "null modem" data cable.



The IP address of the Host PC can be set to the match the default value of the Trio ethernet card.

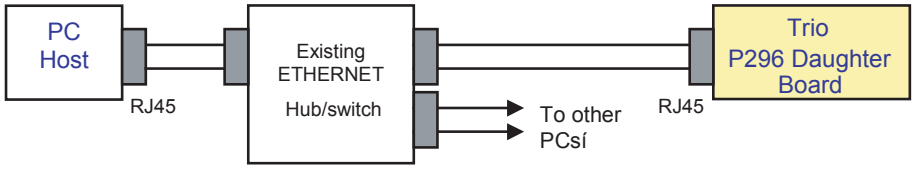
```
Host PC IP: 192.168.000.251
Subnet:     255.255.255.000

Trio IP:    192.168.000.250
Subnet:     255.255.255.000
```

If leaving the Trio's IP address as default, proceed to step 6 to test communications.

2. Connecting the Trio to Network through an ethernet hub/switch

When connecting the Trio MC controller to an existing ethernet network on a hub, no swap in the data cable is required since the hub or router will handle the data inversion.



The IP address of the Trio ethernet board can be set to the match the network address. The Trio's default subnet (255.255.255.000) is generic and allows any host PC to communicate with the controller regardless of a specific sub-network mask. Below is a typical example.

```
Host PC IP: 192.200.185.001
Subnet:     255.255.255.224

Trio IP:    192.200.185.a
Subnet:     255.255.255.000
```

Where:

a = Valid IP address for the Trio ethernet board on the given network

3. Select a valid IP address for the Trio

For this network example, the 224 in the subnet indicate the network can have up to (6) sub-networks (224 = 11100000). The (5) remaining bits within the 224 mask will allow up to 30 valid host addresses ranging from 1 to 30.

Valid IP Addresses (a) for above example:

```
       002 = 11100010 to 030 = 11111110

New Trio IP: 192.200.185.002
Trio Subnet: 255.255.255.000
```

4. Checking and Setting The Trio's IP Address

The IP address of the ethernet daughter board can be verified using the RS232 command line interface ">>" of the *Motion Coordinator*. The command line can be accessed via the terminal 0 in *Motion* Perfect 2.

At the command line, use the ETHERNET command and type:

```
>>ETHERNET(0,0,0)
```

When connected correctly the controller will respond with the line:

```
>>192.168.000.250
```

The sequence (192.168.000.250) is the IP address of the *Motion Coordinator*.

5. To change the IP address to a different

Set a new IP address to match the network:

At the command line, use the ETHERNET command and type:

```
>>ETHERNET(1,0,0,192,200,185,2)
```

Verify the new IP address:

```
>>ETHERNET(0,0,0)
```

The new IP address value prints out:

```
>>192.200.185.002
```

NOTE: Cycle power to the Motion Coordinator for the new IP address to take effect

6. Test the Communications

The easiest way to test the ethernet link is to "ping" the Trio MC controller. This can be done using the ping command at a DOS prompt.

From the START button in Windows, select Accessories and then Command Prompt utility.

At the DOS prompt type the Trio's appropriate IP address:

```
C:\>ping 192.168.0.250
```

Successful reply from controller

```
Pinging 192.168.0.250 with 32 bytes of data:

Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64
Reply from 192.168.0.250: bytes=32 time<10ms TTL=64

Ping statistics for 192.168.0.250:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum =  0ms, Average =  0ms
```

If the ping command is unsuccessful you will see

```
C:\>ping 192.168.0.250

  Pinging 192.168.0.250 with 32 bytes of data:

  Request timed out.
  Request timed out.
  Request timed out.
  Request timed out.

  Ping statistics for 192.168.0.250:
  Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
  Approximate round trip times in milli-seconds:
  Minimum = 0ms, Maximum =  0ms, Average =  0ms
```

7. Telnet/Command-line Prompt

If the controller was successfully 'pinged', then the telnet application can be used to open a remote command-line prompt connection to the controller. This tests the TCP socket connection.

Type:

`telnet 192.168.0.250`
using the DOS prompt on the PC. This should open a telnet session, and by typing <return> the characteristic Trio command-line prompt ('>>') should be seen.

It should be noted that it is possible to use other port numbers with the controller, hence if port number 1025 has been configured then a telnet session can still be started by typing:

`telnet 192.168.0.250 1025`
If the serial lead is also connected to the controller then the Ethernet connection will grab and release the port 0 communications as the socket connection (telnet session) is opened and closed.

8. Modbus TCP

The Modbus/TCP communication protocol is supported by the P296 Ethernet Daughter Board and allows the Motion Coordinator acts as a Modbus/TCP Slave device. Its functionality is similar to the existing Trio Modbus RTU Slave (over RS-232 or RS-485), except an Ethernet connection is used and there are 3 extensions to the basic serial Modbus functionality.

1. Floating point transfers are allowed as an alternative to 16 bit integer.

2. Table memory can be read and written to instead of the VR() variables.

3. Function number 23 (17 Hex) “Read / Write 4x Registers” is supported.

Modbus TCP connects via Ethernet Port 502.

The following ETHERNET command is used to set which data type, integer or floating point, is used for communications.

ETHERNET(2, slot number, 7, data type)

slot number = comms slot where Ethernet Daughter Board is installed

data type = 0, for 16-bit signed integer data communication (default)
data type = 1, for 32-bit signed floating point data communication

This parameter only needs to be initialized if passing floating point data, and must be set before Modbus/TCP communications are attempted. It is not maintained through power cycles, so it must be initialized once after each power-up.

It should be noted that floating point mode isn't covered by the Modbus specification, so each manufacturer's chosen implementation may differ.

To use the TABLE memory instead of VR() variables set ETHERNET function 9 to 1.

ETHERNET(2, slot number, 9, data target)

slot number = comms slot where Ethernet Daughter Board is installed

data target = 0, for read/write VR() variables (default)
data target = 1, for read/write TABLE memory (TABLE(0) to TABLE(16384) max)

This parameter only needs to be initialized if TABLE mamory is required, and must be set before Modbus/TCP communications are attempted. It is not maintained through power cycles, so it must be initialized once after each power-up.

APPENDIX

1

REFERENCE

ATYPE

#	Description
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
11	Remote DriveLink axis
12	Reserved
13	Embedded axis
14	Encoder Output
15	Reserved
16	Remote SERCOS speed axis
17	Remote SERCOS position axis
18	Remote CanOpen position axis
19	Remote CanOpen speed axis
20	Reserved
21	Remote User Specific CAN axis

COMMSTYPE

#	Description
20	CAN Communications card in axis slot
21	USB Communications card in axis slot
22	DriveLink Communications card in axis slot
23	Profibus Communications card in axis slot
24	SERCOS Communications card in axis slot
25	Ethernet Communications card in axis slot

AXISSTATUS / ERRORMASK

Bit	Description:	Value:
0	Unused	1
1	Following error warning range	2
2	Communications error to remote drive	4
3	Remote drive error	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
12	Encoder power supply overload (MC206)	4096
13	Set on SSI axis after initialisation	8192

CONTROL

Controller	CONTROL
Motion Coordinator MC202	202
Motion Coordinator Euro205	205
Motion Coordinator Euro205X	225
Motion Coordinator MC206	206
Motion Coordinator PCI208	208
Motion Coordinator MC216	216
Motion Coordinator MC224	224

Communications Ports

Chan	Device:-
0	Serial Port 0 - RS232 - Motion Perfect / Command Line
1	Serial Port 1
2	Serial Port 2
3	Fibre optic port (value returned defined by DEFKEY)
4	Fibre optic port (returns raw keycode of key pressed)
5	*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

Communications Errors

Bit	Value
0	RX Buffer overrun on Network channel
1	Re-transmit buffer overrun on Network channel
2	RX structure error on Network channel
3	TX structure error on Network channel
4	Port 0 Rx data ready
5	Port 0 Rx overrun
6	Port 0 parity error
7	Port 0 Rx frame error
8	Port 1 Rx data ready (MC202 port 1 parity error)
9	Port 1 Rx overrun (MC202 port 1 Rx frame error)
10	Port 1 parity error (MC202 port 1 Rx overrun)
11	Port 1 Rx frame error (MC202 - no function)
12	Port 2 Rx data ready
13	Port 2 Rx overrun
14	Port 2 parity error
15	Port 2 Rx frame error
16	Error FO Network port
17	Error FO Network port
18	Error FO Network port
19	Error FO Network port

MTYPE

MTYPE	Move Type
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

NETSTAT

Bit Set	Error Type	Value
0	TX Timeout	1
1	TX Buffer Error	2
2	RX CRC Error	4
3	RX Frame Error	8

Data Formats and Floating-Point Operations

The TMS320C3x processor used by the *Motion Coordinator* features several different data types. In the *Motion Coordinator* we use two main formats. The following descriptions are taken directly from the TI documentation.

Single-Precision Floating Point Format

In the single precision format, the floating-point number is represented by an 8-bit exponent field (e) and a twos complement 24-bit mantissa field (man) with and implied significant non-sign bit.

31 24	23	22 0
e	s	f

Operations are performed with an implied binary point between bits 23 and 22. When the implied most significant non-sign bit is made explicit, it is located to the immediate left of the binary point.

The floating point number *'x'* is given by:

$$x = \begin{cases} 01.f \times 2^{e} & \text{if } s=0 \\ 10.f \times 2^{e} & \text{if } s=1 \\ 0 & \text{if } e=-128 \end{cases}$$

The following examples illustrate the range and precision if the single-precision floating-point format:

Most Positive:	$x = (2 - 2^{-23}) \times 2^{127}$	$= 3.4028234 \times 10^{38}$
Least Positive:	$x = 1 \times 2^{-127}$	$= 5.8774717 \times 10^{-39}$
Least Negative:	$x = (-1 - 2^{-23}) \times 2^{-127}$	$= -5.8774724 \times 10^{-39}$
Most Negative:	$x = -2 \times 2^{127}$	$= -3.4028236 \times 10^{38}$

Single-Precision Integer Format

In the single precision integer format, the integer is represented in twos complement notation.

31	0
s	

The range of an integer *x*, represented in the single-precision integer format, is:

$$-2^{31} <= x <= 2^{31} - 1$$

Product Codes

Processors	
P135	MC206
P140	MC216
P150	Euro205 Stepper base card
P151	Euro205X Stepper base card
P155	Euro205 Servo base card
P156	Euro205X Servo base card
P165	MC202
P170	MC224

Options - MC216 & MC224 Only	
P300	Axis Expander, for 1 to 4 daughter baords
Options - Euro205 & Euro205X Only	
P445	Daughter Board Mounting Kit
Options - MC206 Only	
P399	Daughter Board Adapter

Daughter Boards - Euro205-MC206-MC216-MC224	
P200	Servo Encoder Daughter
P210	Servo Resolver Daughter
P220	Reference Encoder Daughter
P230	Stepper Daughter
P240	Stepper Encoder Daughter
P242	Hardware PSWITCH Daughter
P250	Reference Resolver Daughter
P260	Analog Output Daughter
P270	SSI Servo Encoder Daughter

P280	Differential Stepper Daughter
P290	CAN Daughter(2)
P291	SERCOS Daughter(1)
P292	SLM Daughter
P295	USB Daughter
P296	Ethernet Daughter(1)
P297	Profibus Daughter

Options - I/O	
P315	CAN 16-I/O
P325	CAN 8 analog Inputs
P330	MC Network Interface (MC216)

Keypads & Cables	
P350	RS232 Serial Cable
P435	Serial to Fibre Optic Adapter (Euro205/MC202)
P348	Serial to RS485 Adapter (Euro205 / MC216)
P349	Serial to RS485 Adapter (MC202)
P502	Mini-Membrane Keypad
P503	Membrane Keypad

(1) MC206, Euro205X and MC224 only.

(2) CanOpen function available on MC206, Euro205X and MC224 only.

INDEX

Symbols

- 8-133
8-127
$ 8-128
* 8-134
+ 8-133
/ 8-134
: 8-126
< 8-137
<= 8-137
<> 8-135
= 8-135
> 8-136
>= 8-136
’ 8-127

A

ABS 8-138
ACC 8-11
ACCEL 8-152
ACOS 8-138
ADD_DAC 8-11
ADDAX 8-12
ADDAX_AXIS 8-152
ADDRESS 8-78
AIN 8-45
AIN0-7 8-46
AINBI0-7 8-46
AND 8-138
APPENDPROG 8-78
Applications 1-15
ASIN 8-139
ATAN 8-140
ATAN2 8-140
ATYPE 8-152
AUTORUN 8-78
AXIS 8-14
Axis Parameters 7-11
AXISSTATUS 8-153
AXISVALUES 8-79

B

BASE 8-15
BASICERROR 8-65
Board 12-7
BOOST 8-154
Breakpoints 10-59

C

CAD2Motion 10-72
CAM 8-16
CAMBOX 8-19
CAN 8-79
CAN_ADDRESS 8-154
CAN_ENABLE 8-154
CANCEL 8-26
CANIO_ADDRESS 8-81
CANIO_ENABLE 8-82
CANIO_STATUS 8-82
CHECKSUM 8-82
CHR 8-47
CLEAR 8-83
CLEAR_BIT 8-141
CLEAR_PARAMS 8-83
CLOSE_WIN 8-155
CLUTCH_RATE 8-155
CmdProtocol 12-8
Command Line Interface 7-15
COMMSERROR 8-84
COMMSTYPE 8-85
Communications Adaptors 5-28
Communications Protocols 13-1
COMPILE 8-85

CONNECT 8-27
CONSTANT 8-141
CONTROL 8-85
COPY 8-86
COS 8-142
CREEP 8-155
CURSOR 8-46

D

D_GAIN 6-10, 8-159
DAC 8-156
DAC_OUT 8-157
DAC_SCALE 8-157
DATE 8-86
DATE$ 8-86
DATUM 8-28
DATUM_IN 8-158
Daughter Boards 1-10
    Daughter Boards 1-6
DAY 8-87
DAY$ 8-87
DEC 8-29
DECEL 8-158
DEFKEY 8-47
DEFPOS 8-30
DEL 8-87, 8-88
DEMAND_EDGES 8-158
Desktop 10-11
DEVICENET 8-88
Diagnostic Checklists 6-13
DIP Switch Settings 5-6, 5-12
DIR 8-89
DISPLAY 8-89
DLINK 8-90
DocMaker 10-73
DPOS 8-159
DRIVE_STATUS 8-159

E

EDIT 8-95
EDPROG 8-94
ELSE 8-65, 8-70
ELSEIF 8-65
EMC 3-9
ENCODER 8-160
ENDIF 8-67, 8-70
ENDMOVE 8-160
Environmental Considerations 3-7
EPROM 8-95
Error Codes 5-8
ERROR_AXIS 8-96
ERROR_LINE 8-129
ERRORMASK 8-160
ETHERNET 8-96
EX 8-97
Example Programs 9-3
EXECUTE 8-97, 8-98
EXP 8-142

F

FALSE 8-150
FAST_JOG 8-161
FASTDEC 8-161
FE 8-161
FE_LIMIT 8-162
FEATURE_ENABLE 8-98
FEGRAD 8-162
FEMIN 8-163
FERANGE 8-162
FHOLD_IN 8-163
FHSPEED 8-163
Fibre-Optic Network 11-1
FLAG 8-48
FLAGS() 8-49
Flash Eprom 10-63
FLASHVR 8-100
FLASHVR() 8-100
FOR 8-67
FORWARD 8-31
FO-VFKB 5-26
FRAC 8-142
FRAME 8-101

FREE 8-101
FS_LIMIT 8-164
FWD_IN 8-164
FWD_JOG 8-164

G

Gains 6-8
GET 8-49
GET# 8-50
GetConnectionType 12-6
GLOBAL 8-143
GOSUB 8-68
GOTO 8-69

H

HALT 8-103
HEX 8-51
HostAddress 12-7

I

I_GAIN 6-10, 8-165
IDLE 8-70, 8-75
IEEE_IN 8-141, 8-143
IEEE_OUT 8-144
IF 8-70
IN 8-51
INDEVICE 8-129
INITIALISE 8-103
INPUT 8-52
INPUTS0/INPUTS1 8-53
Installation 3-1, 12-3
INT 8-144
INVERT_IN 8-53
INVERT_STEP 8-165
IsOpen 12-5

J

JOGSPEED 8-165

K

KEY 8-53

L

LAST_AXIS 8-103
LENZE 5-7
LINKAX 8-166
LINPUT 8-54
LIST 8-104
LN 8-145
LOADED 8-76
LOADSYSTEM 8-104
LOCK 8-105
LOOKUP 8-130

M

MARK 8-166
MARKB 8-166
MATCH 8-31
MC Simuation 10-69
Membrane Keypad 5-16
MERGE 8-167
MHELICAL 8-32
MICROSTEP 8-167
Mini-Membrane Keypad 5-19
MOD 8-145
MODBUS RTU 13-3
Module Interconnection 3-7
MOTION 8-106
Motion Perfect 10-3
    Communications 10-64
    Flashstick Support 10-19
  Analogue Input Viewer 10-44
  Axis Parameters 10-23
  Connecting to a Controller 10-6
  Control Panel 10-47
  Controller Configuration 10-14
  Debugger 10-58
  Desktop 10-11

Digital IO Status 10-42
Editor 10-53
External Tools 10-45
Feature Enable 10-15
Jog Axes 10-38
Keypad
Emulation 10-35
Loading System Software 10-17
Lock/Unlock Controller 10-20
Main Menu 10-12
New Features 10-5
Oscilloscope 10-26
Projects 10-7
Setting Powerup Mode 10-62
System Requirements 10-4
Table/VR Editor 10-37
Terminal 10-22
Tools 10-21
MOTION_ERROR 8-106
MOVE 8-33
MOVEABS 8-34
MOVECIRC 8-36
MOVELINK 8-37
MOVEMODIFY 8-41
MPE 8-106
MPOS 8-168
MSPEED 8-168
MTYPE 8-169

N

NAIO 8-107
NETSTAT 8-108
Network 11-1
Connection of 11-4
Node Addressing 11-5
Programming 11-6
Specification 11-12
NEW 8-108
NEXT 8-67, 8-71
NIO 8-108
NOT 8-145
NTYPE 8-169

O

OCX 12-1
OFF 8-150
OFFPOS 8-170
ON 8-150
ON .. GOSUB 8-71
ON .. GOTO 8-72
OP 8-55
OPEN_WIN 8-171
Operator Interfaces 1-10, 5-15
OR 8-146
OUTDEVICE 8-130
OUTLIMIT 8-171
Output Velocity Gain 6-11
OV_GAIN 6-11, 8-171

P

P_GAIN 6-10, 8-173
Packaging
Mounting 3-7
PEEK 8-109
PI 8-151
PMOVE 8-130
POKE 8-109
POWER_UP 8-109
PP_STEP 8-172
PRINT 8-56
PRINT# 8-57
PROC 8-131
PROC_LINE 8-131
PROC_STATUS 8-131
PROCESS 8-110
Process Numbers 7-12
Process Parameters 7-11
PROCNUMBER 8-131
Products 1-5
I/O Expansion options 1-9
Master Controllers 1-5
PROFIBUS 8-110

Programming 7-1
Project Autoloader 10-74
PSWITCH 8-58

R

RAPIDSTOP 8-44
READ_BIT 8-147
READPACKET 8-60
RECORD 8-60
REG_MATCH 8-173
REG_POS 8-173
REG_POSB 8-174
REGIST 8-42
REMAIN 8-175
REMOTE 8-111
REMOTE_ERROR 8-175
RENAME 8-111
REP_OPTION 8-176
REPDIST 8-175
REPEAT 8-72
RESET 8-132
RETURN 8-73
REV 8-176
REV_IN 8-176
REV_JOG 8-176
REVERSE 8-44
RS_LIMIT 8-177
RUN 8-112
RUN_ERROR 8-132
RUNTYPE 8-112

S

SCOPE 8-113
SCOPE_POS 8-114
SELECT 8-114
SEND 8-61
SERCOS 8-115
SERVO 8-177
Servo Gains 6-8
Servo Loop 6-12
SERVO_PERIOD 8-116
SET_BIT 8-147
SETCOM 8-62
SGN 8-147
SIN 8-148
SLM 8-90
SLOT 8-115
SP() 8-177
SPEED 8-178
SQR 8-148
SRAMP 8-178
SSI_BITS 8-178
STEP 8-67, 8-73
STEPLINE 8-117
STICK_READ 8-117
STICK_WRITE 8-118
STOP 8-73, 8-74
STORE 8-118
System Building 1-11
System Parameters 7-11

T

TABLE 8-119
TABLEVALUES 8-120
TAN 8-149
Testing Standards 3-10
THEN 8-70, 8-73, 8-74
TICKS 8-132
TIME 8-121
TIME$ 8-121
TO 8-67, 8-74
TRANS_DPOS 8-179
TRANSITIONS 8-179
TRIGGER 8-121
TROFF 8-121
TRON 8-122
TRUE 8-151
TSIZE 8-122
Typical applications 1-15

U

UNITS 8-179

UNLOCK 8-122
UNTIL 8-72, 8-74, 8-76
USB 8-123
USB_STALL 8-124

V

VERIFY 8-180
VERSION 8-124
VFF_GAIN 6-11, 8-180
VIEW 8-124
VP_SPEED 8-181
VR 8-125, 8-181

W

WA 8-75
WAIT 8-75
WAIT IDLE 8-75
WAIT LOADED 8-76
WAIT UNTIL 8-76
WDOG 8-126
WEND 8-77
WHILE 8-77

X

XOR 8-149

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP