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 1023 (0 to 250 on MC202 & MC216)
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 1023 (0 to 250 on MC202 & MC216)
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 1023 (0 to 250 on MC202 & MC216)
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
Euro205x	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 Doard.

Once the SPC3 chip on the daughter board is initialised, the software is very efficient at transferring data in and out. In the MC302X, Euro205x and MC224 however, a fast process is recommended for optimum running. When using the MC206X or MC224, a low process number may be used with less impact on processing speed.

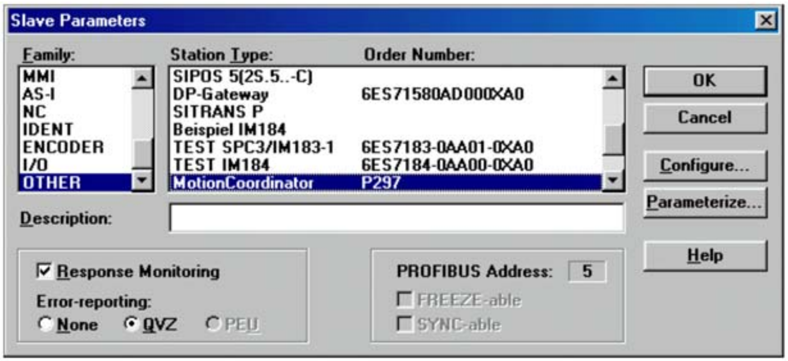
2. Profibus GSD File.

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.

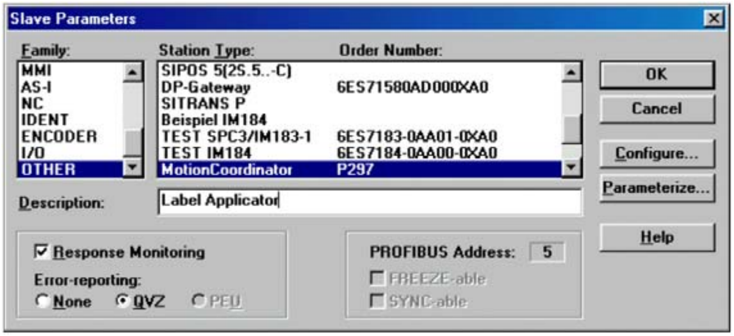
The following sequence shows how to include the *Motion Coordinator* in a fieldbus network using COM PROFIBUS.

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:

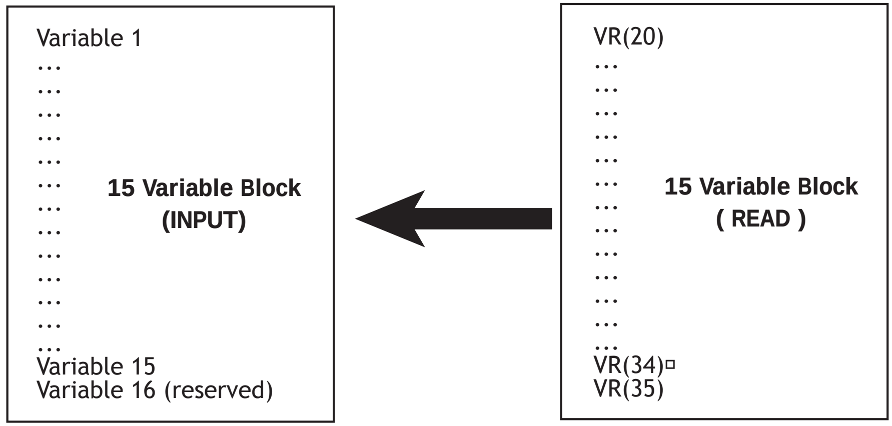
Now add any other nodes to the network that are required and close the window. Finally export the file in the required format, usually Binary, for use by the PLC or other Profibus Master. The master will now search for the *Motion Coordinator* on the Profibus network and when found will connect to it and start transferring the variable blocks.

Example: Trio BASIC Profibus Driver

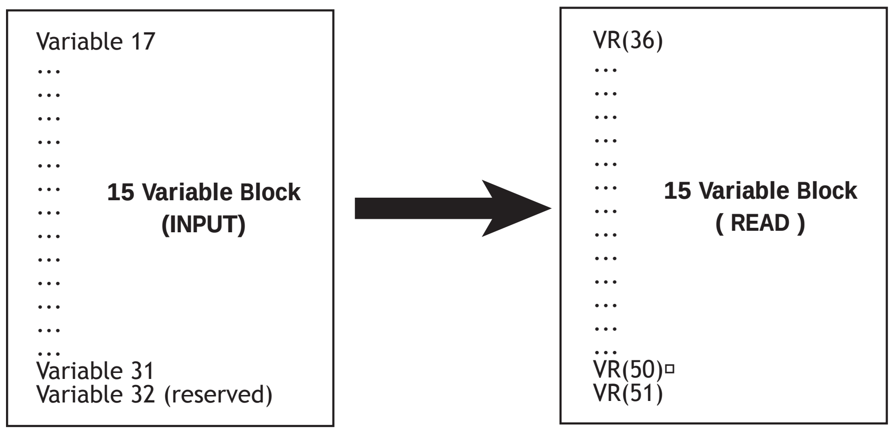
The latest driver can be obtained from www.triomotion.com. At the time of going to press, the most recent revision was 1.05. See below for header and revision information / dates.

```
'========================================================================
' 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* MC206X 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 MC206X 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 TABLE entries
0x38	Write Word - VR	Receives the specified number of values in 16 bit 2s complement format and writes them into the specified VR entries
0x39	Write IEEE - TABLE	Receives the specified number of values in 32 bit IEEE floating point format and writes them into the specified TABLE entries
0x3A	Write IEEE - VR	Receives the specified number of values in 32 bit IEEE floating point format and writes them into the specified VR entries

The following sections describe the message body area of the Explicit Message used to specify the different services. This ignores all of the fragmentation protocol.

Read word format

Request

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
byte 0	0	Service code = 0x33, or 0x34						
byte 1	Class ID = 0x8A							
byte 2	Instance ID = 0x01 (this is the only instance supported)							
byte 3	bits 15-8 of Source Address							
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 MC206X 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
```

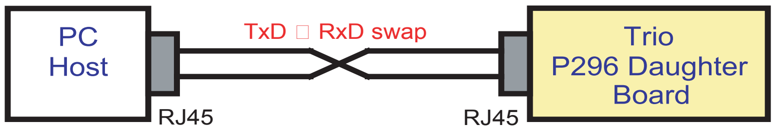
Or a string of all 1's for the first three quads (telling the router to look at these) and 0's for the host number (which the router doesn't need to look at). Subnet masking allows routers to move the packets on more quickly.

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.

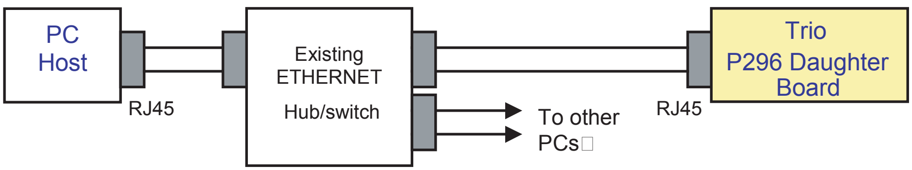
```
Host PC IP: 192.168.000.251
Subnet:     255.255.255.000

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.