Generation D Real-Time Operating System DeviceNet Reference Guide

GE Fanuc Automation

Programmable Control Products

Generation D
Real-Time Operating System

DeviceNet Reference Guide

GFK-2208 *June 2002*

Warnings, Cautions, and Notes as Used in this Publication

Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where equipment might be damaged if care is not taken.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Fanuc Automation assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Fanuc Automation makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. No warranties of merchantability or fitness for purpose shall apply.

Related Publications

The following publications are available at http://www.gefanuc.com/support/plc/m-MotionSolutions.htm.

Generation D RTOS Programming Manual, GFK-2205

IMC Hardware Manual, GFK-2201

Target® ARS Field Service Manual, GFK-2200

IMCjr Hardware Manual, Pub 330

DeviceNet Reference Guide (for Early Firmware Revisions), Pub 305

For an in-depth DeviceNet resource, please consult the *DeviceNet Specification*, release 2.0, Errata 3, published by the Open DeviceNet Vendor Association (www.odva.org).

Chapter 1 DeviceNet Overview

The contents of this chapter describe:

- DeviceNet: What It Is; How It Works
- Network Size and Device Types
- Generation D Real-time Operating System (RTOS)
- Related Publications

DeviceNet: What It Is; How It Works

DeviceNet is a digital, serial, multidrop network that connects and serves as a communication path for DspMotion® controllers and other industrial controls and I/O devices. Using DeviceNet allows one network to connect both simple and high-level devices from many manufacturers. Those devices may each contain one or more nodes, or connection points, to the network. Each node gets a unique *node address* (MAC ID) that serves as its network address.

DeviceNet is a producer-consumer network that supports multiple communication hierarchies and message prioritization. Therefore, when used on DeviceNet, GE Fanuc Motion products can act as slaves to a DeviceNet master and/or communicate with other DeviceNet products in a peer-to-peer fashion. Those two network architectures can also coexist on a single DeviceNet, using both *master/slave* and *peer-to-peer* communication to create a *distributed control architecture*.

DeviceNet Cable and Installation

A DeviceNet network uses 5-wire, multi-conductor cable. Two wires form a twisted pair transmission line for network communications. A second pair transmits network power. The fifth conductor forms an electromagnetic shield. Cable is available in a variety of current-carrying capacities. On a DeviceNet field bus, every device must power its network transceiver from the network power source. Some devices draw all their power from the network supply. The Motion controller powers only its transceiver and requires 40ma maximum per node from the network power supply.

A network can include both high-capacity trunk cable and lower capacity cable for individual branch circuits. DeviceNet specifies two types of network cable, thick and thin cable. Thick cable provides for longer distances and more power and is typically used for the trunk line. Thin cable is used for shorter distances and is generally used for dropline cables or where cable flexibility is

necessary. A cable should be selected to provide sufficient current carrying capacity in the network power pair to provide power to the sum of all network power consumption. It is important to carefully select a quality network cable and install it with attention to cable routing and grounding.

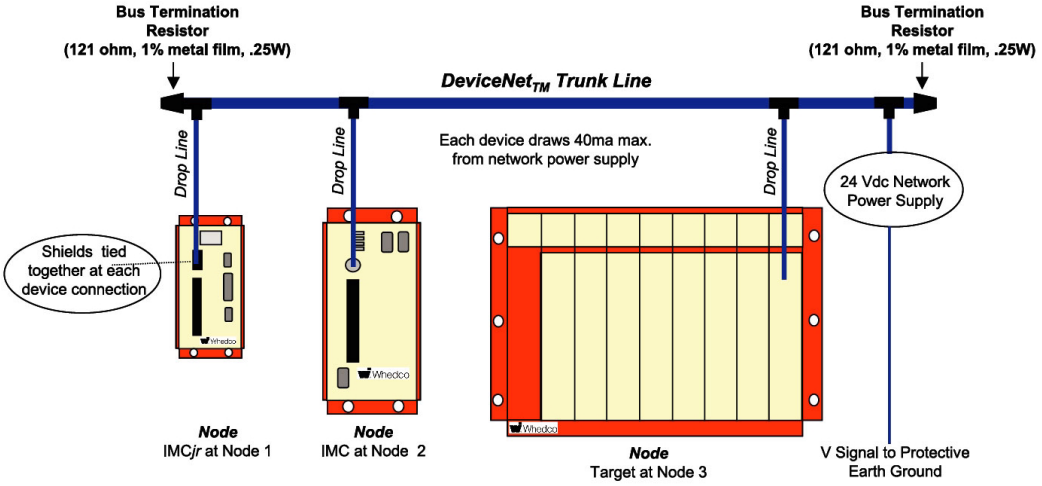


Figure 1.1. DeviceNet Trunk Line with Dropline Connections

Figure 1.2. DeviceNet Cable and General Network Specifications

Thick Cable General Specifications	Two shielded pairs – Common axis with drain wire in the center Overall braid shield – 65% coverage; 36 AWG or 0.12mm minimum individually tinned copper braid Drain wire – #18 Copper min.; 19 strands minimum (individually tinned) Outside diameter – 0.410 inches (min.) to 0.490 inches (max.) Roundness – radius delta to be within 15% of 0.5 O.D.
Thin Cable General Specifications	Two shielded pairs – Common axis with drain wire in the center Overall braid shield – 65% coverage; 36 AWG or 0.12mm minimum individually tinned copper braid Drain wire – #22 Copper min.; 19 strands minimum (individually tinned) Outside diameter – 0.240 inches (min.) to 0.280 inches (max.) Roundness – radius delta to be within 20% of 0.5 O.D.
Network Topology	Bus with limited branching (trunkline/dropline)
Redundancy	Not Supported
Network Power Supply	Nominal 24 volt DC +/- 4%
Allowed Nodes (Bridging excluded)	64 Nodes
Data Packet Size	0-8 bytes with allowance for message fragmentation
Duplicate Address Detection	Address verified at power-up
Error Detection / Correction	CRC – retransmissions of message if validity not acknowledged by recipient

DeviceNet Response Times

The time needed to send a DeviceNet message with 8 data bytes over the network at various baud rates is summarized in the figure below. **The response time of the Motion controller to read/load** a command is 0.2ms. All devices on a DeviceNet network segment must be operating at the same baud rate and each device must have a unique node address.

Figure 1.3. DeviceNet Response Times

Baud Rate	Message Time[1]	8 node Scan Cycle Time[2]
125K	0.888 ms	14.21 ms
250K	0.444 ms	7.10 ms
500K	0.222 ms	3.55 ms

Notes:

1. 8 data byte message including 3 bit interframe space
2. Transmission time for 16 messages of 8 data bytes

DeviceNet Bus Length

The maximum length of the bus is limited by the cable type, transfer rate, and number and accumulated length of drop lines. Individual droplines may not exceed 6 meters and are limited to one network node per drop. A node may, however, offer multiple ports. With thin cable the maximum bus length, regardless of data rate, is 100 meters. With thick cable used as the trunk line, the maximum bus length is shown in the following table.

Figure 1.4. DeviceNet Thick Cable Maximum Bus Length

Data Rate	Bus Length and Drop Length Restrictions
125Kbps	500m bus length and branches totaling < 156m
250Kbps	250m bus length and branches totaling < 78m
500Kbps	100m bus length and branches totaling < 39m

DeviceNet Bus Connectors

A DeviceNet cable plant (installed network cable) has two basic connection types: open and circular. The DeviceNet-equipped IMJ and Target ARS® Motion controllers use the 5-pin standard Phoenix open-style plug connector, which is available with inline terminal block wiring terminations and is suitable for environments without excessive humidity or vibration levels. Typically, thin wire connections are made to this connector. It is possible using thin wire cable to connect several Motion controller nodes together by inserting two conductors into each terminal pin, similar to a multi-drop wiring scheme. Terminating resistors may be added to the appropriate terminal pins. This arrangement provides an economical cable plant, typically within the same cabinet.

The second connection type uses a five-pole, circularly arranged connector. GE Fanuc's DeviceNet-equipped IMC products use this circular-style microconnector, which provides a robust

connection and is resistant to moisture and vibration. Trunk line cables, network T connectors, network power supply taps, thin wire drops and terminators are readily available in this form factor and in IP67 sealant ratings. This cable plant arrangement provides the greatest potential network length and is often used to connect cabinet to cabinet or cabinet to IP67 rated DeviceNet I/O devices.

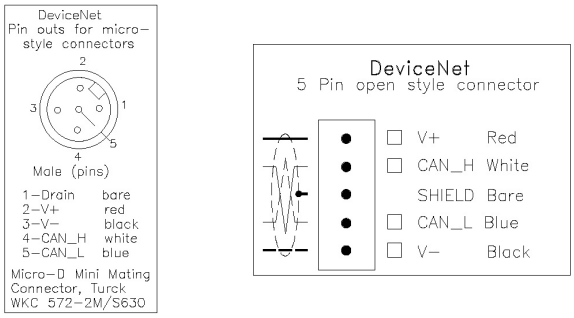


Figure 1.5. DeviceNet IMC Microconnector (left) and IMJ/Target ARS 5-Pin Open-style Connector (right) Pinouts

DeviceNet Bus Termination

Termination of a DeviceNet network is passive and includes one resistor at each end of the network trunk line, i.e., exactly two resistors per DeviceNet network. A terminating resistor is placed across the data communications at the CAN_L and CAN_H pins. The correct terminating resistor is a 121 ohm, 1% metal film, 0.25 Watt resistor.

DeviceNet networks must always be terminated to operate regardless of cable length.

DeviceNet Bus Power Supply and Grounding

DeviceNet networks require a power supply of 24 VDC (+/- 4%) at 16A maximum. With the use of thick cable, however, a maximum of 8A is permitted on a single network segment. This is possible if the power supply is placed at the center point of two network segments, thus supplying 8A to each segment. With the use of thin cable, a maximum of 3A is permitted.

Shielding is very important for a DeviceNet network and its devices. DeviceNet requires that all cable shields be tied together at each device connection. This is done by tying the bare wire of the network cable to pin 3 (Shield) of the open-style network connector or to pin 1 (Drain) of the circular-style microconnector. In a thin wire multi-drop configuration both shield wires landing at a node should be connected to the Shield or Drain pin. The shield should be tied to ground at one point in the network. This connection is best made at the network power supply.

The DeviceNet network power supply must also be grounded, but only at one point. The V- signal must be connected to a protective earth ground at the power supply only. If multiple power supplies are used, only one power supply must have V- connected to earth ground.

Introduction to DeviceNet Object Modeling

DeviceNet is an object-oriented network that uses some specialized terms to describe node behavior and the ways in which devices exchange information. For example, nodes communicate with each other via messaging connections. Each *node* consists of a collection of objects, or *object classes*. In turn, a node may contain more than one *instance* of an object class. An object has *attributes* and may provide *services*. To give these terms an everyday-use perspective, the example in the following figure illustrates how they might apply to a network of telephone lines in an office:

Figure 1.6. DeviceNet Terms Applied to a Network of Telephone Lines in an Office

Device	Node	Object Class	Services	Instances	Attributes	Attribute Values
Office	Individual telephone	Engineer	Get attribute Set attribute Set specifications Get design	A. Jones	Height Weight Age	6' 1" 185 lbs. 50
		Receptionist	Answer calls Transfer calls Register visitors	J. Doe	Height Weight Age	5' 9" 160 lbs. 60
		Accountant	Invoice clients Pay vendors Balance books	B. Smith	Height Weight Age	5' 7" 150 lbs. 45

Translating the terms in Figure 1.6 to a DeviceNet scenario is simple. Take a look at Figure 1.7. The DeviceNet trunk line is the device, the controllers are the nodes, and each oval represents an object class. Each object performs services. The connection object has two instances: the I/O connection and the explicit messaging connection.

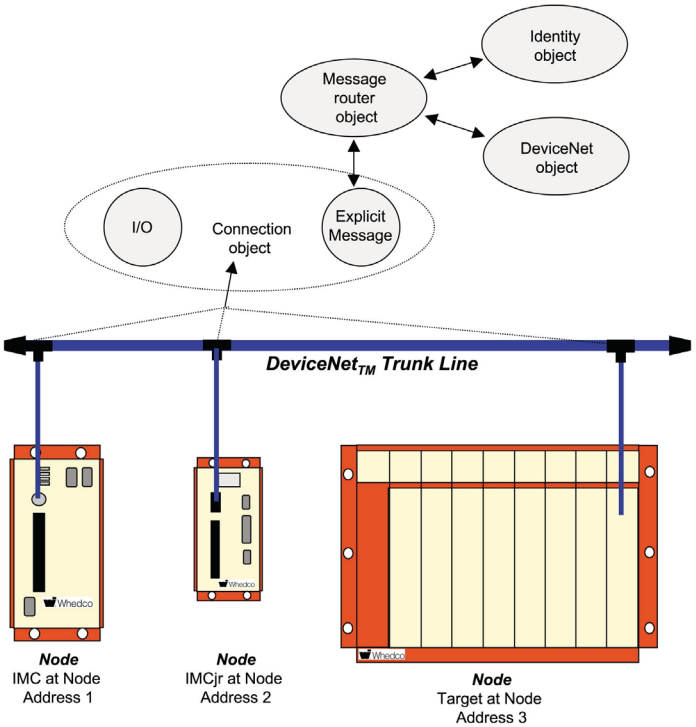


Figure 1.7. Object Model for GE Fanuc Motion Controllers

Network Size and Device Types

A single DeviceNet network can have 64 *nodes*, or device types. The DspMotion products conform to the following ODVA device types as specified in the *DeviceNet Specification, Release 2.0, Errata 3*:

IMC Series	Type 10_{16} Position Controller
IMC*jr* Series	Type 10_{16} Position Controller
Target ARS	Type $0C_{16}$ Communications Adapter

I/O messaging protocols may be defined in the DeviceNet specification by *Device Profiles*. The IMC and IMC*jr* products conform to the *Position Controller Profile*. For DeviceNet users, this conformance promotes interchangeability with other devices that qualify for the same profile.

The term *DspMotion controller* includes the IMC, IMC*jr* and the Target Motion controllers.

Generation D Real-Time Operating System (RTOS)

In addition to the DeviceNet conventions described herein, GE Fanuc Motion products use the Generation D Real-time Operating System (RTOS), which has been designed specifically for motion and machine control. The *Generation D RTOS Programming Manual*, GFK-2205 describes that operating system. Commands and registers specific to DeviceNet are included in detail in Appendix A of this manual. Chapter 3 provides instructions for sending the Generation D RTOS mnemonics via CCS for Windows and via the Explicit Messaging Connection.

Related Publications

For information on DspMotion controller set-up, configuration, and programming, refer to the following publications, which are available at http://www.gefanuc.com/support/plc/m-MotionSolutions.htm.

Generation D RTOS Programming Manual, GFK-2205

IMC Hardware Manual, GFK-2201

Target® ARS Field Service Manual, GFK-2200

IMCjr Hardware Manual, Pub 330

DeviceNet Reference Guide (for Early Firmware Revisions), Pub 305

For an in-depth DeviceNet resource, please consult the *DeviceNet Specification*, release 2.0, Errata 3, published by the Open DeviceNet Vendor Association (www.odva.org).

This document is applicable to DspMotion controllers with the following firmware revisions:

IMJ	>=	revision 2.1
IMC	>=	revision 3.1
Target	>=	revision 1.1

Documentation for earlier firmware revisions can be found in Pub 305, *DeviceNet Reference Guide*. This publication can be obtained from http://www.gefanuc.com/support/plc/m-MotionSolutions.htm.

Chapter 2 Getting Started

The contents of this chapter describe:

- DeviceNet Connection Checklist
- Complete Basic Set-up Procedure
- Configure GE Fanuc Motion controllers for DeviceNet operation
- Connect DeviceNet Hardware
- Connect DspMotion Controllers to a DeviceNet network
- Add slaves to scan list

DeviceNet Connection Checklist

GE Fanuc-supplied Components

One DspMotion controller with
DeviceNet per axis
One motor per axis
Cables
DC power to digital I/O
CCS for Windows version 5.1.1 or later

Figure 2.1: Two Members of the DspMotion Family of Controllers: IMCs and Motors

User-supplied Components

DC power
16-gauge wire to jumper I/O connectors
DeviceNet trunk line
One trunk line connector (T) per device
DeviceNet drop line cable and appropriate VAC power supply for each controller
Two terminating resistors for beginning and end of trunk line
PC or PLC with DeviceNet scanner (for master/slave architecture only).
Note: **scanner module must be compatible with UCMM-capable devices.**

Complete Basic Set-up Procedure

Before you connect and use your DspMotion controller on DeviceNet, take a few minutes to complete the Process for Basic Set-up located in the following publications:

IMC and Target ARS: *Generation D RTOS Programming Manual*, chapter 2, GFK-2205

IMC*jr*: *IMCjr Programming Supplement (or Generation D RTOS Programming Manual*, GFK-2205)

The set-up process in each document takes you step-by-step through each of the following items:

- Install CCS software
- Connect cables
- Jumper dedicated I/O (if applicable)
- Establish communication with the controller
- Complete basic equipment configuration
- Run the motor to verify correct set-up.

If you are using multiple DspMotion controllers, repeat the set-up for each of your products. When you have completed the set-up, leave your connections and jumpers in place—you're ready to build your DeviceNet system.

Configure DspMotion Controllers for DeviceNet

Each DspMotion controller requires some simple configuration before being used on DeviceNet. You have already established communication between your PC and controller through CCS for Windows. Now it's time for some DeviceNet-specific configuration.

The flowchart in figure 2-2 documents the process for configuring DspMotion controllers for DeviceNet. The remainder of this chapter expands upon each action in figure 2.2 with step-by-step instructions for each part of the process.

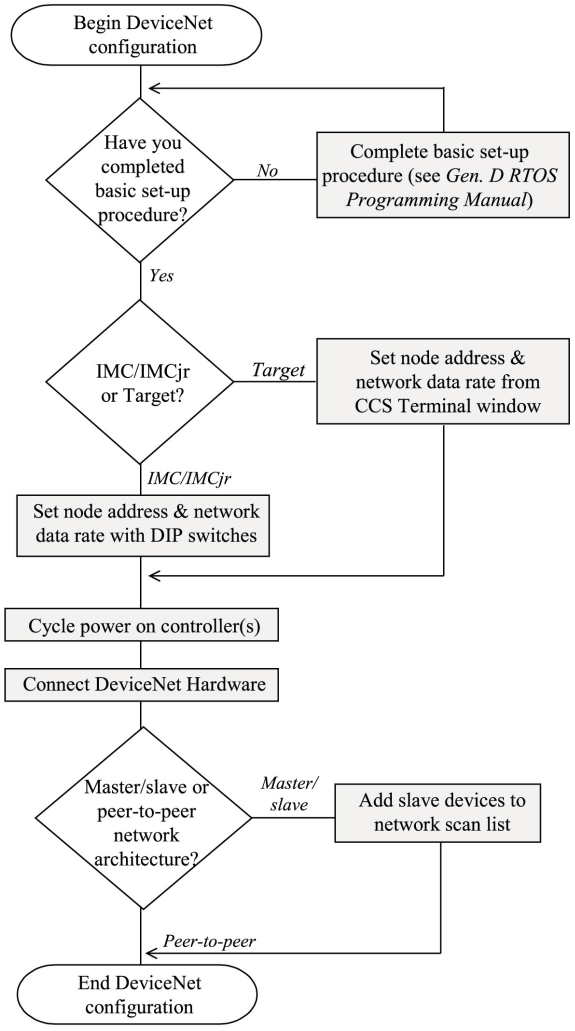


Figure 2.2: The Process for DeviceNet Configuration

The first step is to set the node address (i.e., MAC ID), which provides a unique network address, from 0 through 63, for each DeviceNet node. DspMotion controllers ship from the factory with the node address set to 63 and the network data rate set to 125K. IMC, IMC*jr*, and Target configurations are different—please use the procedure that is appropriate for your controller type.

IMC DeviceNet Configuration: Page 11??

IMC*jr* DeviceNet Configuration: Page 14

Target DeviceNet Configuration: Page 15

IMC DeviceNet Configuration

Step 1: Set the Node Address

When you completed the *Basic Set-up Procedure*, you used the DIP switches on the bottom of the IMC to set the unit serial address. When you use an IMC on DeviceNet, those same DIP switches change their function: you'll now use them to set the node address. If you wish to change the node address, ensure that IMC power is off. Use the DIP switches to set the node address to a network address indicated in the switch settings tables in figure 2.4 on the following page.

Figure 2.3 shows the location of the IMC DIP switches and the proper orientation for left and right switch settings.

Figure 2.3: Location of DIP Switches on Bottom of IMCs

The serial port address defaults to one when these DIP switches are used to set the node address.

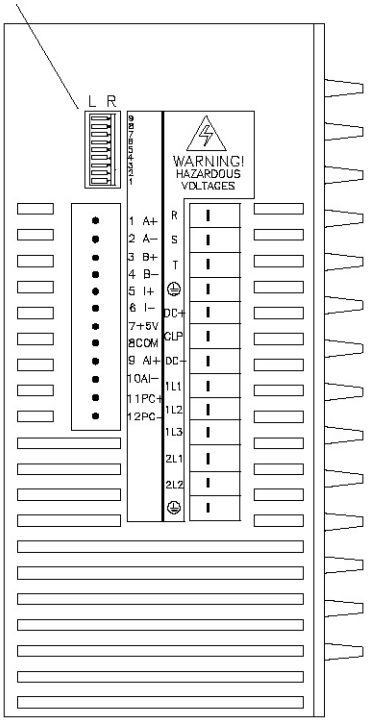


Figure 2.4: IMC DIP Switch Node Address (MAC ID) Settings (for Products Equipped with 9 DIP Switches)

Node Addr	Switch Locations							Node Addr	Switch Locations							Node Addr	Switch Locations						
	1	2	3	4	5	6	9		1	2	3	4	5	6	9		1	2	3	4	5	6	9
0	R	R	R	R	R	R	L	22	R	L	L	R	L	R	L	43	L	L	R	L	R	L	L
1	L	R	R	R	R	R	L	23	L	L	L	R	L	R	L	44	R	R	L	L	R	L	L
2	R	L	R	R	R	R	L	24	R	R	R	L	L	R	L	45	L	R	L	L	R	L	L
3	L	L	R	R	R	R	L	25	L	R	R	L	L	R	L	46	R	L	L	L	R	L	L
4	R	R	L	R	R	R	L	26	R	L	R	L	L	R	L	47	L	L	L	L	R	L	L
5	L	R	L	R	R	R	L	27	L	L	R	L	L	R	L	48	R	R	R	R	L	L	L
6	R	L	L	R	R	R	L	28	R	R	L	L	L	R	L	49	L	R	R	R	L	L	L
7	L	L	L	R	R	R	L	29	L	R	L	L	L	R	L	50	R	L	R	R	L	L	L
8	R	R	R	L	R	R	L	30	R	L	L	L	L	R	L	51	L	L	R	R	L	L	L
9	L	R	R	L	R	R	L	31	L	L	L	L	L	R	L	52	R	R	L	R	L	L	L
10	R	L	R	L	R	R	L	32	R	R	R	R	R	L	L	53	L	R	L	R	L	L	L
11	L	L	R	L	R	R	L	33	L	R	R	R	R	L	L	54	R	L	L	R	L	L	L
12	R	R	L	L	R	R	L	34	R	L	R	R	R	L	L	55	L	L	L	R	L	L	L
13	L	R	L	L	R	R	L	35	L	L	R	R	R	L	L	56	R	R	R	L	L	L	L
14	R	L	L	L	R	R	L	36	R	R	L	R	R	L	L	57	L	R	R	L	L	L	L
15	L	L	L	L	R	R	L	37	L	R	L	R	R	L	L	58	R	L	R	L	L	L	L
16	R	R	R	R	L	R	L	38	R	L	L	R	R	L	L	59	L	L	R	L	L	L	L
17	L	R	R	R	L	R	L	39	L	L	L	R	R	L	L	60	R	R	L	L	L	L	L
18	R	L	R	R	L	R	L	40	R	R	R	L	R	L	L	61	L	R	L	L	L	L	L
19	L	L	R	R	L	R	L	41	L	R	R	L	R	L	L	62	R	L	L	L	L	L	L
20	R	R	L	R	L	R	L	42	R	L	R	L	R	L	L	63	L	L	L	L	L	L	L
21	L	R	L	R	L	R	L																

Step 2: Set the Network Data Rate

DeviceNet can run at 125,250 or 500 Kbaud. To change the network data rate (i.e., baud rate), ensure IMC power is off. Then set the DIP switches as indicated in figure 2.5 below.

Note that when you completed the *Basic Set-up Procedure*, you used DIP switches to set the IMC serial baud rate. When you use an IMC on DeviceNet, those same DIP switches change their function: you'll now use them to set the *network data rate*.

Figure 2.5: IMC DIP Switch Settings for Network Data Rate (Models Equipped with 9 DIP Switches)

The serial port *baud rate defaults to 9600 when these DIP switches are used to set the network data rate.*

Network Data Rate	Switch Locations	
	7	8
125K	R	R
250K	L	R
500K	R	L

When you have configured all of your DspMotion controllers, proceed to page 16 to complete your DeviceNet set-up and connection.

IMC*jr* DeviceNet Configuration

Step 1: Set the Node Address

To change the node address (i.e., MAC ID), ensure that IMC*jr* power is off. Use the *NA* DIP switches located on the bottom of the IMC*jr* to set a MAC ID from 0-63 for each IMC*jr* (see switch settings table in figure 2.7). Figure 2.6 shows the location of the IMC*jr* DIP switches and the proper orientation for left and right switch settings.

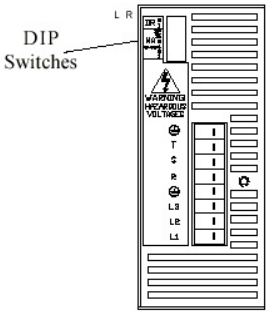


Figure 2.6: Location of DIP Switches on Bottom of IMC*jr*

Figure 2.7: IMC*jr* DIP Switch Node Address (Mac ID) Settings

Node Addr	*Switch Locations*						Node Addr	*Switch Locations*						Node Addr	*Switch Locations*					
	1	2	4	8	16	32		1	2	4	8	16	32		1	2	4	8	16	32
0	R	R	R	R	R	R	22	R	L	L	R	L	R	43	L	L	R	L	R	L
1	L	R	R	R	R	R	23	L	L	L	R	L	R	44	R	R	L	L	R	L
2	R	L	R	R	R	R	24	R	R	R	L	L	R	45	L	R	L	L	R	L
3	L	L	R	R	R	R	25	L	R	R	L	L	R	46	R	L	L	L	R	L
4	R	R	L	R	R	R	26	R	L	R	L	L	R	47	L	L	L	L	R	L
5	L	R	L	R	R	R	27	L	L	R	L	L	R	48	R	R	R	R	L	L
6	R	L	L	R	R	R	28	R	R	L	L	L	R	49	L	R	R	R	L	L
7	L	L	L	R	R	R	29	L	R	L	L	L	R	50	R	L	R	R	L	L
8	R	R	R	L	R	R	30	R	L	L	L	L	R	51	L	L	R	R	L	L
9	L	R	R	L	R	R	31	L	L	L	L	L	R	52	R	R	L	R	L	L
10	R	L	R	L	R	R	32	R	R	R	R	R	L	53	L	R	L	R	L	L
11	L	L	R	L	R	R	33	L	R	R	R	R	L	54	R	L	L	R	L	L
12	R	R	L	L	R	R	34	R	L	R	R	R	L	55	L	L	L	R	L	L
13	L	R	L	L	R	R	35	L	L	R	R	R	L	56	R	R	R	L	L	L
14	R	L	L	L	R	R	36	R	R	L	R	R	L	57	L	R	R	L	L	L
15	L	L	L	L	R	R	37	L	R	L	R	R	L	58	R	L	R	L	L	L
16	R	R	R	R	L	R	38	R	L	L	R	R	L	59	L	L	R	L	L	L
17	L	R	R	R	L	R	39	L	L	L	R	R	L	60	R	R	L	L	L	L
18	R	L	R	R	L	R	40	R	R	R	L	R	L	61	L	R	L	L	L	L
19	L	L	R	R	L	R	41	L	R	R	L	R	L	62	R	L	L	L	L	L
20	R	R	L	R	L	R	42	R	L	R	L	R	L	63	L	L	L	L	L	L
21	L	R	L	R	L	R														

Step 2: Set the Network Data Rate

To change the network data rate, ensure IMC*jr* power is off. Then flip *DR* DIP switches 1 and 2 on the bottom of the IMC*jr* to one of the settings provided in figure 2.8:

Figure 2.8: IMC*jr* DIP Switch Settings for Network Data Rate

Network Data Rate	Switch Locations	
	1	2
125K	R	R
250K	L	R
500K	R	L
N/A	L	L

When you have configured all of your DspMotion controllers, proceed to page 16?? to complete your DeviceNet set-up and connection.

Target DeviceNet Configuration

Step 1: Set Node Address & Network Data Rate

To change the Target's factory node address and data rate settings, you must be connected serially to a PC running CCS (see chapter 2, *Generation D RTOS Programming Manual* for Target set-up instructions). If the Target and a PC running CCS 6.0 or later are already connected to DeviceNet, you may configure the Target using DeviceNet rather than the serial connection.

Use the ADDN[1] and BAUDN[1] registers (see Appendix A for register descriptions and instructions) to set a new node address and data rate. For example:

```
ADDN = 1        (* sets node address to 1
BAUDN=250       (* sets network data rate to 250
```

After setting these registers, cycle the power to the Target to put your settings into effect.

When you have configured all of your DspMotion controllers, proceed to page 16 to complete your DeviceNet set-up and connection.

> ***Note:*** *These settings will be stored on the PCMCIA Flash EPROM card. Should you ever need to change the System Module, simply complete the following steps (refer to figure 2.9):*
>
> 1) turn off the power
> 2) remove the PCMCIA card from the old System Module
> 3) remove the old System Module from the rack
> 4) insert the new System Module into the rack
> 5) insert the PCMCIA Flash EPROM card into its slot in the new System Module
> 6) turn on the power to the rack.

[1] Generation D RTOS register or command.

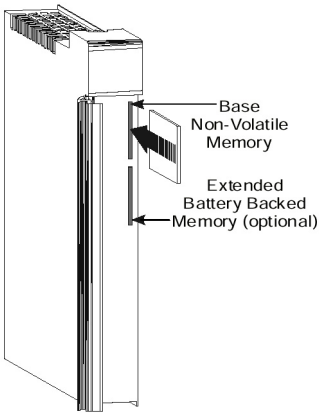


Figure 2.9: Installing the PCMCIA card in the Target System Module

Connect DeviceNet Hardware

Follow the instructions provided with your non-GE Fanuc DeviceNet hardware to connect your trunk line, drop lines, Ts, terminating resistors, and power supplies.

Connect DspMotion Controllers to DeviceNet

Each controller requires its own drop line. *IMCs* require drop lines with microconnectors. The *IMCjr* and the *Target* use drop lines with flying leads. Connection procedures for each type are given below.

IMC Users

Connect the microconnector end of the drop line cable to the DeviceNet port on the front of the IMC (see figure 2.10).

IMC*jr* Users

Connect the DeviceNet drop line flying leads to their appropriately labeled pins on the front of the IMC*jr* (see figure 2.11).

Target Users

Locate connector J9 in the *System Module* (see figure 2.12). Connect the DeviceNet drop line flying leads into their appropriately labeled pins in the J9 terminal strip (remove the connector for easy wiring).

Figure 2.10: IMC Connected to DeviceNet Drop Line Cable

Figure 2.11: IMCjr Connected to DeviceNet Drop Line Cable

Figure 2.12: Target Connected to DeviceNet Drop Line Cable

Add Slave Devices to Scan List

(not required for peer-to-peer communication)

About DeviceNet Scanners

DeviceNet systems using master/slave architecture need a scanner installed in the master PC/PLC. The scanner buffers all communication between the PC/PLC and any devices on the network in the order in which they have been added to the scan list. Your scanner ***must*** be able to communicate with slave devices that support the Unconnected Message Manager (UCMM).

To determine whether you have a UCMM-compatible scanner, verify with the manufacturer that their scanner uses the complete *Predefined Master/Slave Connection Set Allocation Procedure*, detailed in section 7.9.1, volume I of the DeviceNet Specification. If your scanner is UCMM-compatible, follow the manufacturer's instructions for proper installation and connection. If your scanner is not UCMM-compatible, you will need to purchase one that is.

Add GE Fanuc Slave Devices to Scan List

Some DeviceNet software configuration tools can use an Electronic Data Sheet (EDS) file for each DspMotion controller model. The EDS file is an ASCII file that tells the scan list what types of devices are connected to the network and how those devices send and receive messages.

EDS files are available from http://www.gefanuc.com/support/plc/f-MotionSolutions.htm. Follow the instructions provided with your DeviceNet commissioning software to load the EDS files and add your Motion devices to the scan list.

Apply DC Voltage to the DeviceNet Trunk Line

You must have DC voltage on the DeviceNet trunk line before you can talk to GE Fanuc Motion controllers. Chapter 3 provides information on communicating with GE Fanuc DeviceNet nodes.

Chapter 3 Communicating with DeviceNet Nodes

The contents of this chapter describe:

- Overview
- Master/Slave Network Architecture
  - Allocating the Master/Slave Connection Set
  - Message Types
  - I/O Command/Response Messages for the IMC and IMC*jr*
  - Homing Via the I/O Channel
  - I/O Handshake Sequences for the IMC and IMC*jr*
  - I/O Command/Response Messages for the Target ARS
  - Explicit Messages
- Peer-to-peer Network Architecture
  - Talking Peer-to-peer with CCS for Windows
  - Running a Peer-to-peer Application
- Distributed Control Network Architecture
  - Using Peer-to-peer and Master/Slave Communication in the Same System
  - Using Remote I/O with GE Fanuc Motion controller DeviceNet Products

Overview

DspMotion controllers provide the features you need for your DeviceNet system.

Serial Port Communication

DspMotion products allow you to communicate serially to GE Fanuc nodes while they are connected to a DeviceNet system. This means that at any time, you can bypass the network communication protocols and talk directly to your controllers to perform any task that the Generation D RTOS permits, including application program development, diagnostics…even setting tuning constants or other registers. Refer to the *Generation D RTOS Programming Manual* for information on maximizing the power of the operating system.

Nonserial Controller-to-controller Communication over DeviceNet

CCS for Windows, version 6.0 or later, makes it possible to use the Generation D RTOS to communicate from a network PC to any GE Fanuc Motion controller on the network. No serial connection or DeviceNet communication protocol is required. Program editing and diagnostics are as easy as typing simple Generation D RTOS mnemonics in the CCS Terminal window on the PC.

Running Resident Application Programs

DspMotion products allow you to run a complete Generation D RTOS program *and* communicate over DeviceNet. This means that your DeviceNet system can take advantage of high-performance DspMotion control and DeviceNet wiring, data handshaking and diagnostics. GE Fanuc Motion controller products include DeviceNet extensions to read and write to variables within your DspMotion controller, making the interface between your Generation D RTOS program and your master program straightforward, simple, and powerful.

Peer-to-peer Systems

DspMotion controllers allow multiple axes and some I/O devices from other vendors to communicate peer-to-peer over DeviceNet with no master. Each axis gets its own controller, and because they can share information, it's easy to create a high-performance, multiaxis system.

Distributed Control Systems

DspMotion controllers can simultaneously function as slaves to a master device and as peers to each other. With distributed control, you can manage system behavior through peer-to-peer communication or DeviceNet communication from the master device. You also get the power to optimize network traffic to further boost system performance.

DeviceNet Communication

GE Fanuc Motion controller DeviceNet systems provide several ways to communicate with DspMotion controllers and other network nodes. In fact, DspMotion technology makes it easy to use DeviceNet's I/O channel for system control—you can manipulate bits and bytes in the I/O message simply by turning them on and off as you would inputs and outputs. DspMotion products also contain DeviceNet objects that allow users to read and write registers and send commands via the explicit messaging connection. These communication methods are defined in the Master/Slave Network Architecture section, beginning on the following page.

The communication method that you use will depend upon the network architecture of the system. Figure 3.1, for example, lists the available message types.

Figure 3.1: DeviceNet System Communication Methods for DspMotion Controllers

Message Type	Medium	Used in…
Explicit Message Priority One[a]	UCMM	Peer-to-peer network architecture
I/O Priority Two[b]	Master-slave connection set	Master-slave network architecture
Explicit Message Priority Three[a]	Master-slave connection set	Master-slave network architecture
I/O and Explicit Messages	Master-slave connection set. UCMM	Distributed control (combination of master-slave and peer-to-peer network architecture)

[a] Explicit messages allow users to send commands and get/set registers of the Generation D RTOS. Explicit messages sent peer-to-peer have a higher priority and, therefore, are faster than explicit messages sent within a master/slave network architecture.

[b] I/O messages have priority over *priority three* explicit messages.

Conflict Resolution

DeviceNet resolves any communication conflicts on the bus to maintain message integrity. When two transmissions collide on the network, the device that has made the line active wins; the other stops transmitting. The collision does not interfere with the transmission of the higher priority message, and, therefore, maintains a high network bandwidth.

Master/Slave Network Architecture

DeviceNet has a *Predefined Master/Slave Connection Set* that consolidates the steps required to create and configure an application-to-application connection, letting you establish a communication environment that uses fewer network and device resources than other connection hierarchies. A master can have up to 63 slaves. A slave can have only one master.

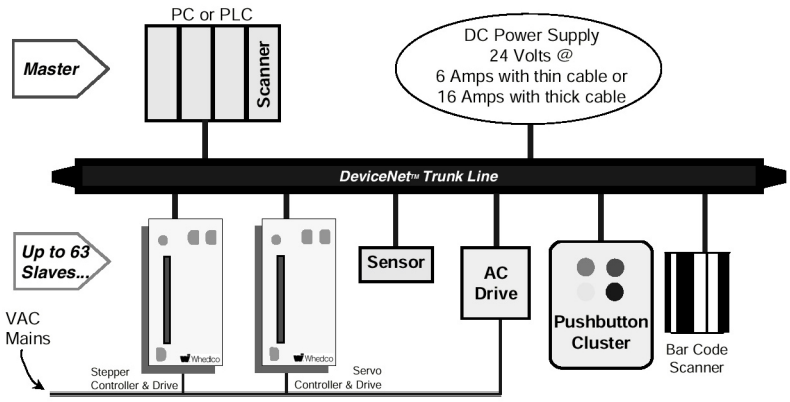


Figure 3.2: Master/Slave Network Architecture

Allocating the Master/Slave Connection Set

Before a device can use the *Predefined Master/Slave Connection Set*, it must become a slave to the master device, which allocates the *Predefined Master/Slave Connection Set*. Most manufacturers of DeviceNet scanners and commissioning software packages have given their products the power to simplify the master/slave allocation process.

Message Types

When used on DeviceNet, DspMotion Controllers support two message types within the Predefined Master/Slave Connection Set: *I/O command/response (implicit) messages* and *explicit messages*. Implicit messages have a predefined data content that eliminates the need to transmit identifiers along with the data. Explicit messages have no predefined data content and require headers to identify the type and meaning of the data.

Compared with explicit messages, implicit messages require less programming and network overhead. Explicit messages, however, are valuable because they allow the exchange of data not supported in the predefined data field of the I/O message. In master/slave architectures, explicit messages are typically used for configuration-type activities; and I/O messages are used for control (although it is possible to use explicit messages for control).

I/O Command/Response Messages

IMC and IMCjr

The *Position Controller Profile* in the DeviceNet specification defines and governs the format and content of the *I/O command/response* messages. Figures 3.3 and 3.4 show the formats of those messages for GE Fanuc position controllers (device type 10 hex).

Figure 3.3: Command Message Format for the IMC and IMC*jr*

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	Command Data 1							
2	Command Axis Number = 001_2				Command Message Type			
3	Command Data 2							
4	Command Data 3							
5	Command Data 4							
6	Command Data 5							
7	Command Data 6							

Figure 3.4: Descriptions of Command Message Format Bits for the IMC and IMC*jr*

Byte	Bit	Name	Action	RTOS Equivalent Action
0	0	Load Data/ Start Profile	Set from zero to one load command data. The transition of this bit from zero to one will also start a profile move when the command message type contained in the command message field is the message type that starts a profile move for the mode selected.	none
0	2	Incremental	This bit is used to define the position value as either absolute or incremental. 0 = absolute position value and 1 = incremental position value.	none
0	3	Direction(V. Mode)	This bit is used to control the direction of the motor in velocity mode. A 1 = forward or positive and 0 = reverse or negative.	none
0	4	Smooth Stop	This bit is used to bring the motor to a controlled stop at the currently implemented deceleration rate.	Setting this bit from zero to one causes the controller to execute the ST[1] command.
0	5	Hard Stop	This bit is used to bring the motor to an immediate stop.	Setting this bit from zero to one causes the controller to execute the HT[1] command.
0	6	Reg Arm	Clear capture edge bit of the I/O register, which enables a new position capture.	Setting this bit from zero to one causes bit 13 of the IO[1] register to clear.
0	7	Enable	This bit is used in the same manner as the enable input. Clearing this bit will fault the controller due to *enable lost*, and the currently executing motion profile will be aborted.	Setting this bit from zero to one causes the controller to execute the RSF[1] command.
2	0-4	Command Message Type	This field defines the Command Message Type.	none
2	5-7	Command Axis Number	These three bits will always be set to 001_2 to indicate axis 1.	none

[1] Generation D RTOS register or command.

Figure 3.5: Response Message Format for the IMC and IMC*jr*

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	Response Data 1							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Response Data 2							
5	Response Data 3							
6	Response Data 4							
7	Response Data 5							

Figure 3.6: Descriptions of Response Message Format Bits for the IMC and IMC*jr*

Byte	Bit	Name	Action	RTOS Equivalent Action
0	0	Profile in Progress	This bit indicates that a profile move is in progress.	Same as the motion generator enabled bit of the SRA[1] register
0	2	On Target Position	This bit indicates whether the motor is within the position band of the last targeted position. *(1 = Current position or target.)*	none
0	3	General Fault	This bit indicates the logical "or" of all fault conditions.	Same as the controller faulted bit of the SRS[1] register
0	4	Current Direction	This bit shows the current direction of the motor. If the motor is not moving, the bit will indicate the direction of the last commanded move. 0 = reverse or negative direction and 1 = forward or positive direction.	Same as the direction bit of the SRA[1] register
0	5	Home Level	This bit reflects the level of the home input.	Same as home input bit of the IO[1] register
0	6	Reg Level	This bit reflects the level of capture input.	Same as capture input bit of the IO[1] register
0	7	Enable	This bit indicates the state of the OK output. A 1 indicates the OK output is active.	Same as OK output bit of the IO[1] register
2	1	Fwd Limit	This bit indicates that the forward overtravel input is active.	Same as forward overtravel input bit of the IO[1] register
2	2	Rev Limit	This bit indicates that the reverse overtravel input is active.	Same as reverse overtravel input bit of the IO[1] register
2	3	Positive Limit	This bit indicates that the motor has attempted to travel past the programmed positive limit position. This bit remains valid until the motor is moved within the limits or the programmed limit value is set greater than the current position.	Same as (software overtravel bit AND direction bit) of the SRA[1] register
2	4	Negative Limit	This bit indicates that the motor has attempted to travel past the programmed negative limit position. This bit remains valid until the motor is moved within the limits or the programmed limit value is set less than the current position.	Same as (software overtravel bit AND [not of direction bit]) of SRA[1] register
2	5	FE Fault	This bit indicates that a following error fault has occurred. This fault occurs when the following error, or difference between the commanded and actual position, exceeds the programmed allowable following error.	Same as excessive following error bit of FC[1] register
2	7	Load Complete	This bit indicates that the command data contained in the command message has been successfully loaded into the device.	none
3	0-4	Response Message Type	This byte defines the Response Message Type.	none
3	5-7	Response Axis Number	These three bits will always be set to 001_2 to indicate axis 1.	none

[1] Generation D RTOS register or command.

Starting a Profile Move

A profile move is a move that uses Acceleration, Target Velocity, and Deceleration to run at a Target Velocity or to a Target Position. Whether the position controller device runs at a Target Velocity or to a Target Position depends on the *operating mode* (position controller object attribute 3), to which the position controller device is set. The position controller object attribute 3 sets the mode of the controller to the following:

0 = Position (default)
1 = Velocity
2 = Torque

A profile move starts when the Command Message Type for the specified mode is loaded and the Load Data/Start Profile bit transitions from zero to one. Figure 3.7 below shows the Command Message Type that starts a profile move for each mode.

Figure 3.7: Starting a Profile Move with Command Message Types

Mode (Attribute 3)	Command Message Type that Starts Motion
0 = Position	01 = Position (default)
1 = Velocity	02 = Velocity
2 = Torque	05 = Torque

Command Message Types for the IMC and IMCjr

The Command Message Type is defined by byte 02 of the message. Byte 00 is the same for all Command Message Types. Bytes 01 and 03 through 07 are defined by the Command Message Type code in byte 02. In message types 01 through 05, byte 03 defines the requested Response Axis Number and Response Message Type format. For message types 1A, 1B, and 1F hex, the requested Response Axis Number and Response Message Type is the same as the Command Axis Number and Command Message Type.

Figure 3.8: Command Message Type 01 hex Target Position

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	n/a							
2	Command Axis Number = 001_2			Command Message Type				
3	Response Axis Number = 001_2			Response Message Type				
4	Target Position Low Byte							
5	Target Position Low Middle Byte							
6	Target Position High Middle Byte							
7	Target Position High Byte							

Target Position – Command Message Type 01 hex

This double word defines the profile move's Target Position in pulses when the Load Data/Start Profile bit transitions from zero to one.

Figure 3.9: Command Message Type 02 hex Target Velocity

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	n/a							
2	Command Axis Number = 001_2			Command Message Type				
3	Response Axis Number = 001_2			Response Message Type				
4	Target Velocity Low Byte							
5	Target Velocity Low Middle Byte							
6	Target Velocity High Middle Byte							
7	Target Velocity High Byte							

Target Velocity – Command Message Type 02 hex

This double word defines the profile move's Target Velocity in pulses/sec. when the Load Data/Start Profile bit transitions from zero to one.

Figure 3.10: Command Message Type 03 hex Acceleration

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	n/a							
2	Command Axis Number = 001_2			Command Message Type				
3	Response Axis Number = 001_2			Response Message Type				
4	Acceleration Low Byte							
5	Acceleration Low Middle Byte							
6	Acceleration High Middle Byte							
7	Acceleration High Byte							

Acceleration – Command Message Type 03 hex

This double word defines the profile move's Acceleration in pulses/sec^2 when the Load Data/Start Profile bit transitions from zero to one.

Figure 3.11: Command Message Type 04 hex Deceleration

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	n/a							
2	Command Axis Number = 001_2			Command Message Type				
3	Response Axis Number = 001_2			Response Message Type				
4	Deceleration Low Byte							
5	Deceleration Low Middle Byte							
6	Deceleration High Middle Byte							
7	Deceleration High Byte							

Deceleration – Command Message Type 04 hex

This double word defines the profile move's Deceleration in pulses/sec^2 when the Load Data/Start Profile bit transitions from zero to one.

Figure 3.12: Command Message Type 05 hex Torque

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	n/a							
2	Command Axis Number = 001_2			Command Message Type				
3	Response Axis Number = 001_2			Response Message Type				
4	Torque Low Byte							
5	Torque Low Middle Byte							
6	Torque High Middle Byte							
7	Torque High Byte							

Torque – Command Message Type 05 hex

This double word is used to set the output torque (where 1000 = full continuous current setting) when the Load Data/Start Profile bit transitions from zero to one. The torque value will take effect only when in torque mode. (Position Controller Object Attribute 3 = 2.)

Figure 3.13: Command Message Type 1A hex Position Controller Supervisor

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	Position Controller Supervisor Attribute to Get							
2	Command Axis Number = 001_2			Command Message Type				
3	Position Controller Supervisor Attribute to Set							
4	Position Controller Supervisor Attribute Value Low Byte							
5	Position Controller Supervisor Attribute Value Low Middle Byte							
6	Position Controller Supervisor Attribute Value High Middle Byte							
7	Position Controller Supervisor Attribute Value High Byte							

Attribute Value – Command Message Type 1A hex

This double word defines the new value of the attribute to set when the Load Data/Start Profile bit transitions from zero to one.

Object Attribute to Get – Command Message Type 1A hex

This byte specifies the Position Controller Supervisor object attribute from which to get the value and return in the response message.

Object Attribute to Set – Command Message Type 1A hex

This byte specifies the Position Controller Supervisor object attribute to set to the new value defined by the attribute value when the Load Data/Start Profile bit transitions from zero to one.

Figure 3.14: Command Message Type 1B hex Position Controller

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	Position Controller Attribute to Get							
2	Command Axis Number = 001_2			Command Message Type				
3	Position Controller Attribute to Set							
4	Position Controller Attribute Value Low Byte							
5	Position Controller Attribute Value Low Middle Byte							
6	Position Controller Attribute Value High Middle Byte							
7	Position Controller Attribute Value High Byte							

Attribute Value – Command Message Type 1B hex

This double word defines the new value of the attribute to set when the Load Data/Start Profile bit transitions from zero to one.

Object Attribute to Get – Command Message Type 1B hex

This byte specifies the Position Controller object attribute from which to get the value and return in the response message.

Object Attribute to Set – Command Message Type 1B hex

This byte specifies the Position Controller object attribute to set to the new value defined by the attribute value when the Load Data/Start Profile bit transitions from zero to one.

Figure 3.15: Command Message Type 1F hex Parameter

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	n/a	Load Data/ Start Profile
1	Parameter Instance to Get							
2	Command Axis Number = 001_2			Command Message Type				
3	Parameter Instance to Set							
4	Parameter Value Low Byte							
5	Parameter Value Low Middle Byte							
6	Parameter Value High Middle Byte							
7	Parameter Value High Byte							

The Parameter Command Message Type allows you to read and write variables into the DspMotion controller using the I/O channel. Instances are divided among integer and floating point variables as indicated in figure 3.16.

Figure 3.16: Parameter Instances to Set/Get

Instance	Variable Equivalent
0	None
1 – 128	Integer Variables: 1 – 128
129 – 255	Floating Point Variables: 1 – 127

Parameter Instance to Get – Command Message Type 1F hex

This byte specifies a Parameter Instance from which to get and return the value of attribute 1 in the response message.

Parameter Instance to Set – Command Message Type 1F hex

This byte specifies the Parameter Instance whose value attribute will be set to the new value defined by the Parameter Value when the Load Data/Start Profile bit transitions from zero to one.

Parameter Value – Command Message Type 1F hex

This double word defines the new value of the Parameter Value attribute to set when the Load Data/Start Profile bit transitions from zero to one.

Response Message Types for the IMC and IMCjr

The response message type is defined by byte 03 of the message. Bytes 00, 02 and 03 are the same for all response message types. Bytes 01 and 04 through 07 are defined by the Response Message Type code in byte 03.

Figure 3.17: Response Message Type 01 hex Actual Position

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	n/a							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Actual Position Low Byte							
5	Actual Position Low Middle Byte							
6	Actual Position High Middle Byte							
7	Actual Position High Byte							

Actual Position – Response Message Type 01 hex

This double word reflects the actual position in pulses. If position feedback is not used, this word will report the commanded position.

Figure 3.18: Response Message Type 02 hex Commanded Position

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	n/a							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Commanded Position Low Byte							
5	Commanded Position Low Middle Byte							
6	Commanded Position High Middle Byte							
7	Commanded Position High Byte							

Commanded Position – Response Message Type 02 hex

This double word reflects the commanded position in pulses.

Figure 3.19: Response Message Type 03 hex Actual Velocity

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	n/a							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Actual Velocity Low Byte							
5	Actual Velocity Low Middle Byte							
6	Actual Velocity High Middle Byte							
7	Actual Velocity High Byte							

Actual Velocity – Response Message Type 03 hex

This double word reflects the actual velocity in pulses/second.

Figure 3.20: Response Message Type 05 hex Torque

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Hom e Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	n/a							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Torque Low Byte							
5	Torque Low Middle Byte							
6	Torque High Middle Byte							
7	Torque High Byte							

Torque – Response Message Type 05 hex

This double word reflects the actual torque where 1000 = full continuous current setting.

Figure 3.21: Response Message Type 08 hex Captured Registration Position

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	n/a							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Registration Position Low Byte							
5	Registration Position Low Middle Byte							
6	Registration Position High Middle Byte							
7	Registration Position High Byte							

Captured Registration Position – Response Message Type 08 hex

This double word reflects the captured registration position in pulses.

Figure 3.22: Response Message Type 14 hex Command/Response Error

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	Reserved = 0							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	General Error Code							
5	Additional Code							
6	Copy of Command Message Byte 2							
7	Copy of Command Message Byte 3							

General Error Code – Response Message Type 14 hex

This byte identifies the error that has been encountered. Figure 3.22.a on the following page summarizes specific behavior for the position controller profile.

Additional Code – Response Message Type 14 hex

This byte contains an object/service-specific value that further describes the error condition. If the responding object has no additional information to specify, then the value FF_{hex} is placed within this field.

Figure 3.22.a: General Error Codes for Position Controller Profile

General Error Code	Additional Code	Response	Semantics
08_{hex}	01_{hex}	Service Not Supported	Command Message type not supported. Additional code 01 takes precedence over additional code 02.
	02_{hex}	Service Not Supported.	Response message type not supported.
05_{hex}	01_{hex}	Path Destination Unknown	A consumed axis number was requested that does not exist in the drive.
	02_{hex}	Path Destination Unknown	A produced axis number was requested that does not exist in the drive.
09_{hex}	FF_{hex}	Invalid Attribute Value	Load value is out of range.
OE_{hex}	FF_{hex}	Attribute not Settable	A request to modify a non-modifiable attribute was received.
11_{hex}	FF_{hex}	Reply Data too Large	The data requested is more than four bytes.
13_{hex}	FF_{hex}	Not Enough Data	I/O command message contains fewer than 8 bytes.
14_{hex}	01	Attribute Not Supported	Attribute to set specified in request is not supported.
	02	Attribute Not Supported	Attribute to get specified in request is not supported.

Figure 3.23: Response Message Type 1A hex Position Controller Supervisor Attribute

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	Position Controller Supervisor Attribute to Get							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Position Controller Supervisor Attribute Value Low Byte							
5	Position Controller Supervisor Attribute Value Low Middle Byte							
6	Position Controller Supervisor Attribute Value High Middle Byte							
7	Position Controller Supervisor Attribute Value High Byte							

Attribute Value – Response Message Type 1A hex

This double word reflects the value of the Position Controller Supervisor attribute to get.

Object Attribute to Get – Response Message Type 1A hex

This byte specifies the Position Controller Supervisor object attribute from which to get the value.

Figure 3.24: Response Message Type 1B hex Position Controller Attribute

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	Position Controller Attribute to Get							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Position Controller Attribute Value Low Byte							
5	Position Controller Attribute Value Low Middle Byte							
6	Position Controller Attribute Value High Middle Byte							
7	Position Controller Attribute Value High Byte							

Attribute Value – Response Message Type 1B hex

This double word reflects the value of the Position Controller attribute to get.

Object Attribute to Get – Response Message Type 1B hex

This byte specifies the Position Controller object attribute from which to get the value.

Figure 3.25: Response Message Type 1F hex Parameter

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	n/a	Profile in Progress
1	Parameter Instance to Get							
2	Load Complete	n/a	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	n/a
3	Response Axis Number = 001_2			Response Message Type				
4	Parameter Value Low Byte							
5	Parameter Value Low Middle Byte							
6	Parameter Value High Middle Byte							
7	Parameter Value High Byte							

The Parameter Response Message Type allows you to read variables from the DspMotion controller using the I/O channel. Instances are divided among integer and floating point variables as indicated in figure 3.26.

Figure 3.26: Parameter Instances to Set/Get

Instance	Variable Equivalent
0	None
1 – 128	Integer Variables: 1 – 128
129 – 255	Floating Point Variables: 1 – 127

Parameter Instance to Get – Response Message Type 1F hex

This byte specifies a Parameter Instance from which to get and return the value of attribute 1 in the response message.

Parameter Value – Response Message Type 1A hex

This double word reflects the value of the Parameter Value attribute to get.

Homing via the I/O Channel

The procedure for homing position controllers via the I/O channel is described below:

1. **Set mode attribute** (Position Controller attribute 3) to one for velocity mode.
2. **Set acceleration and deceleration attributes** to the appropriate values for the homing move.
3. **Arm the homing function** using one of the following methods:
    a. set the home arm attribute (Position Controller Supervisor attribute 12) for home to home input to one;
    b. set the index arm attribute (Position Controller Supervisor attribute 15) for home to index to one.
4. **Set direction for the home move.** Set the direction attribute (Position Controller attribute 23) to one for forward or zero for reverse. Note that this attribute is automatically set by bit 3 of byte 0 of the I/O command message.
5. **Initiate the home move.** Load the target velocity using command message type 02. Note: If the home arm is set, the home move velocity is determined by the target velocity. If, however, the index arm is set, the home move velocity is fixed at 4,096 pulses per second.
6. **Wait for profile in progress attribute (Position Controller attribute 11) to be zero.** Note: this attribute is reflected in bit 0 of byte 0 of the I/O response message.
7. **Confirm that the home trigger has occurred** — check for the appropriate home arm or index arm attribute to be zero.
8. **If you wish to home to index on home input**, perform steps 1–7 with home arm set; and then repeat steps 3–7 with index arm set.
9. **Set actual position attribute** (Position Controller attribute 13) to desired home position.

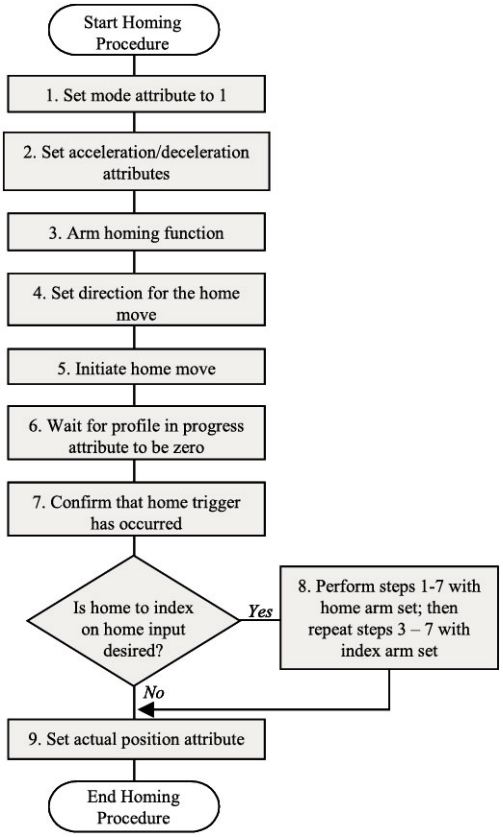


Figure 3.27: Procedure for Homing Via the I/O Channel

I/O Handshake Sequences for IMC and IMCjr

The Load Data/Start Profile bit (i.e., bit 0 of byte 0) and Load Data Complete bit are used to synchronize data transfers between the master and the IMC or IMC*jr*. The position controller acts on the data contained in an input message only when the Load Data/Start Profile bit is true. In each data transaction example, reset the Load Data/Start Profile bit before writing new data into the scanner output table. Write a complete data set into the scanner output table before setting the Load Data/Start Profile bit. Figures 3.28 and 3.29 below flowchart the procedures for Client Data Loading and Client Profile Move.

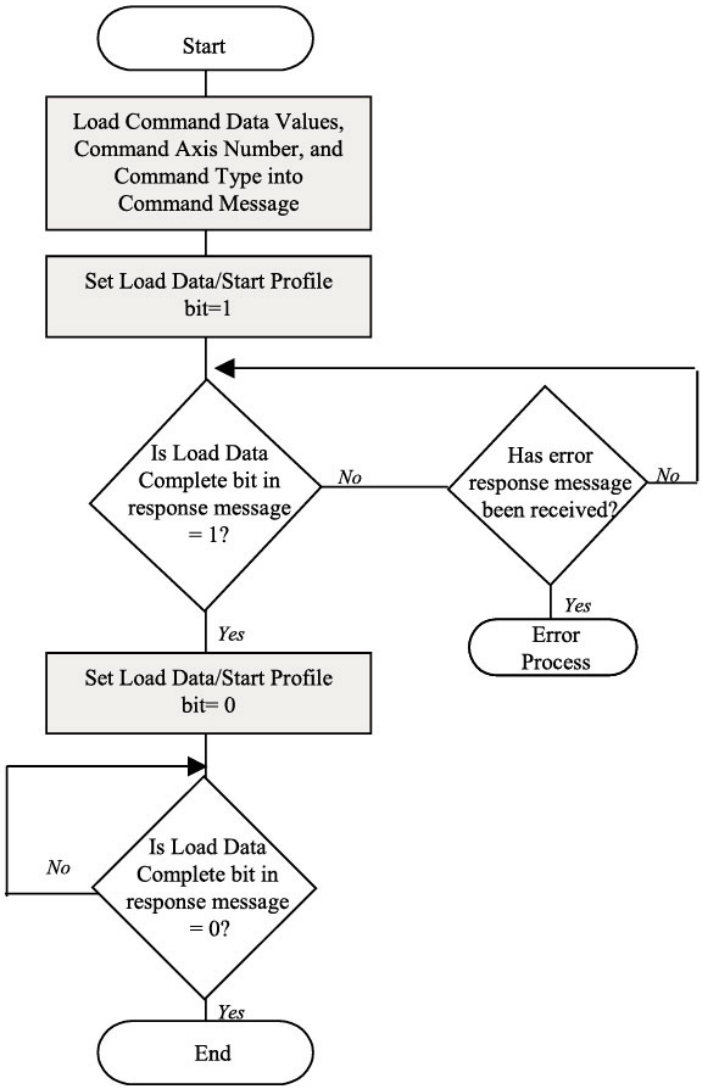


Figure 3.28: Client Data Loading Procedure

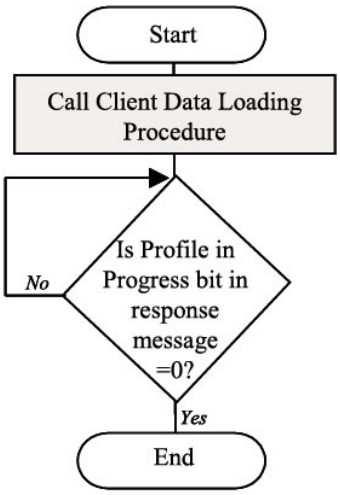


Figure 3.29: Client Profile Move

I/O Command/Response Messages for the Target ARS

In Target ARS systems, the DeviceNet I/O connection is used exclusively to get and set variables. The I/O message is a fragmented, 34-byte message that allows the user to get or set eight, 32-bit variables in one message transmission. Figure 3.30 identifies the command/response messages for the communications adapter (device type 0C hex).

Figure 3.30: Command/Response Messages for Target ARS

Byte	Description	Byte	Description
0	Variable n Data Low Byte	17	Variable n+4 Data Low Middle Byte
1	Variable n Data Low Middle Byte	18	Variable n+4 Data High Middle Byte
2	Variable n Data High Middle Byte	19	Variable n+4 Data High Byte
3	Variable n Data High Byte	20	Variable n+5 Data Low Byte
4	Variable n+1 Data Low Byte	21	Variable n+5 Data Low Middle Byte
5	Variable n+1 Data Low Middle Byte	22	Variable n+5 Data High Middle Byte
6	Variable n+1 Data High Middle Byte	23	Variable n+5 Data High Byte
7	Variable n+1 Data High Byte	24	Variable n+6 Data Low Byte
8	Variable n+2 Data Low Byte	25	Variable n+6 Data Low Middle Byte
9	Variable n+2 Data Low Middle Byte	26	Variable n+6 Data High Middle Byte
10	Variable n+2 Data High Middle Byte	27	Variable n+6 Data High Byte
11	Variable n+2 Data High Byte	28	Variable n+7 Data Low Byte
12	Variable n+3 Data Low Byte	29	Variable n+7 Data Low Middle Byte
13	Variable n+3 Data Low Middle Byte	30	Variable n+7 Data High Middle Byte
14	Variable n+3 Data High Middle Byte	31	Variable n+7 Data High Byte
15	Variable n+3 Data High Byte	32	Variable Instances to Set
16	Variable n+4 Data Low Byte	33	Variable Instances to Get

The controller will set or get variable instances based on the value (x) of bytes 32 and 33. The table in figure 3.31 defines those variable instances:

Figure 3.31: Variable Instances to Set/Get for Target ARS

Value (x)	Variable Instances (n) to Set/Get
0	None
1 – 128	Integer Variables: n = 8(x-1) + 1
129 – 255	Floating Point Variables: n = 8(x-129) + 1

In DeviceNet systems, you can assume that unless the I/O connection fails when you set a variable instance, the variable instance has been set correctly. If, however, you would like to confirm that a variable has been set correctly, set bytes 32 and 33 to equal values and read the variable's value from the Target.

Sending Explicit Messages

The *Explicit Messaging Connection* is a generic, multipurpose communication path between two *nodes*. Explicit messages travel from client to server. The *client* originates a message, or request; the *server* reacts to the message with a response. The client's DeviceNet services usually generate message identifiers and headers automatically.

Explicit messages use the DeviceNet *objects* that reside within the DspMotion controller. Those objects support *services* that allow you to send commands and get/set attributes to and from your DspMotion controller over DeviceNet. The services perform as described below:

Send Command Service Sends a command to the DspMotion controller. If an error in sending the command is detected, an error response is returned. Otherwise, a successful *Send Command Response* is returned with the requested data.

Set Attribute Single Service Modifies an attribute value within the DspMotion controller. The service validates any attribute data before it accepts the modification. If an error is detected, an error response is returned; otherwise, a successful Set Attribute Single response is returned.

Get Attribute Single Service Causes the object to return the contents of the specified attribute to the requester. If an error is detected, an error response is returned; otherwise, a successful Get Attribute Single response is returned along with the requested attribute data.

Figure 3.32 lists the parameters that are specified within the service data field of service requests and successful service responses.

Figure 3.32: Service Data Field Parameters

Service	Service Data	Name	Data Type	Description of Parameter
Send Command Service	*Data for Request*	Immediate Mode Choice	BOOL	Immediate mode commands are to be executed when nonzero.
		Command	SHORT_STRING	Controller command of 0-240 characters. Does not include unit address or carriage return.
	Service Data for Success Response	Command Response	SHORT_STRING	Response to the controller command. String contains the response you would get had you sent the command to the controller via serial port.
Set Attribute Single Service	*Data for Request*	Attribute ID	USINT	Identifies the attribute to be set.
		Attribute Data	Object/class Attribute-specific Struct	Contains the value to which the specified attribute is to be modified.
Get Attribute Single Service	*Data for Request*	Attribute ID	USINT	Identifies the attribute to be read/returned.
	Service Data for Success Response	Attribute Data	Object/class Attribute-specific Struct	Contains the requested attribute data.

Assembly Object Instances

In the IMC and IMC*jr* controllers, 192 assembly instances are defined to access a group of 32 integer or floating point variables with one explicit message. These instances allow for more efficient transfer of large amounts of data over DeviceNet to/from the IMC and IMC*jr* controllers. Figure 3.33 identifies the structure of the data in the assembly instances.

Figure 3.33: Data Attribute of Assembly Object Instances

Byte	Description	Byte	Description	Byte	Description
0	Variable n Data Low Byte	24	Variable n+6 Data Low Byte	48	Variable n+12 Data Low Byte
1	Variable n Data Low Middle Byte	25	Variable n+6 Data Low Middle Byte	49	Variable n+12 Data Low Middle Byte
2	Variable n Data High Middle Byte	26	Variable n+6 Data High Middle Byte	50	Variable n+12 Data High Middle Byte
3	Variable n Data High Byte	27	Variable n+6 Data High Byte	51	Variable n+12 Data High Byte
4	Variable n+1 Data Low Byte	28	Variable n+7 Data Low Byte	52	Variable n+13 Data Low Byte
5	Variable n+1 Data Low Middle Byte	29	Variable n+7 Data Low Middle Byte	53	Variable n+13 Data Low Middle Byte
6	Variable n+1 Data High Middle Byte	30	Variable n+7 Data High Middle Byte	54	Variable n+13 Data High Middle Byte
7	Variable n+1 Data High Byte	31	Variable n+7 Data High Byte	55	Variable n+13 Data High Byte
8	Variable n+2 Data Low Byte	32	Variable n+8 Data Low Byte	56	Variable n+14 Data Low Byte
9	Variable n+2 Data Low Middle Byte	33	Variable n+8 Data Low Middle Byte	57	Variable n+14 Data Low Middle Byte
10	Variable n+2 Data High Middle Byte	34	Variable n+8 Data High Middle Byte	58	Variable n+14 Data High Middle Byte
11	Variable n+2 Data High Byte	35	Variable n+8 Data High Byte	59	Variable n+14 Data High Byte
12	Variable n+3 Data Low Byte	36	Variable n+9 Data Low Byte	60	Variable n+15 Data Low Byte
13	Variable n+3 Data Low Middle Byte	37	Variable n+9 Data Low Middle Byte	61	Variable n+15 Data Low Middle Byte
14	Variable n+3 Data High Middle Byte	38	Variable n+9 Data High Middle Byte	62	Variable n+15 Data High Middle Byte
15	Variable n+3 Data High Byte	39	Variable n+9 Data High Byte	63	Variable n+15 Data High Byte
16	Variable n+4 Data Low Byte	40	Variable n+10 Data Low Byte	64	Variable n+16 Data Low Byte
17	Variable n+4 Data Low Middle Byte	41	Variable n+10 Data Low Middle Byte	65	Variable n+16 Data Low Middle Byte
18	Variable n+4 Data High Middle Byte	42	Variable n+10 Data High Middle Byte	66	Variable n+16 Data High Middle Byte
19	Variable n+4 Data High Byte	43	Variable n+10 Data High Byte	67	Variable n+16 Data High Byte
20	Variable n+5 Data Low Byte	44	Variable n+11 Data Low Byte	68	Variable n+17 Data Low Byte
21	Variable n+5 Data Low Middle Byte	45	Variable n+11 Data Low Middle Byte	69	Variable n+17 Data Low Middle Byte
22	Variable n+5 Data High Middle Byte	46	Variable n+11 Data High Middle Byte	70	Variable n+17 Data High Middle Byte
23	Variable n+5 Data High Byte	47	Variable n+11 Data High Byte	71	Variable n+17 Data High Byte

Table continues on the following page

Figure 3.33: Data Attribute of Assembly Object Instances (continued)

Byte	Description	Byte	Description	Byte	Description
72	Variable n+18 Data Low Byte	92	Variable n+23 Data Low Byte	112	Variable n+28 Data Low Byte
73	Variable n+18 Data Low Middle Byte	93	Variable n+23 Data Low Middle Byte	113	Variable n+28 Data Low Middle Byte
74	Variable n+18 Data High Middle Byte	94	Variable n+23 Data High Middle Byte	114	Variable n+28 Data High Middle Byte
75	Variable n+18 Data High Byte	95	Variable n+23 Data High Byte	115	Variable n+28 Data High Byte
76	Variable n+19 Data Low Byte	96	Variable n+24 Data Low Byte	116	Variable n+29 Data Low Byte
77	Variable n+19 Data Low Middle Byte	97	Variable n+24 Data Low Middle Byte	117	Variable n+29 Data Low Middle Byte
78	Variable n+19 Data High Middle Byte	98	Variable n+24 Data High Middle Byte	118	Variable n+29 Data High Middle Byte
79	Variable n+19 Data High Byte	99	Variable n+24 Data High Byte	119	Variable n+29 Data High Byte
80	Variable n+20 Data Low Byte	100	Variable n+25 Data Low Byte	120	Variable n+30 Data Low Byte
81	Variable n+20 Data Low Middle Byte	101	Variable n+25 Data Low Middle Byte	121	Variable n+30 Data Low Middle Byte
82	Variable n+20 Data High Middle Byte	102	Variable n+25 Data High Middle Byte	122	Variable n+30 Data High Middle Byte
83	Variable n+20 Data High Byte	103	Variable n+25 Data High Byte	123	Variable n+30 Data High Byte
84	Variable n+21 Data Low Byte	104	Variable n+26 Data Low Byte	124	Variable n+31 Data Low Byte
85	Variable n+21 Data Low Middle Byte	105	Variable n+26 Data Low Middle Byte	125	Variable n+31 Data Low Middle Byte
86	Variable n+21 Data High Middle Byte	106	Variable n+26 Data High Middle Byte	126	Variable n+31 Data High Middle Byte
87	Variable n+21 Data High Byte	107	Variable n+26 Data High Byte	127	Variable n+31 Data High Byte
88	Variable n+22 Data Low Byte	108	Variable n+27 Data Low Byte		
89	Variable n+22 Data Low Middle Byte	109	Variable n+27 Data Low Middle Byte		
90	Variable n+22 Data High Middle Byte	110	Variable n+27 Data High Middle Byte		
91	Variable n+22 Data High Byte	111	Variable n+27 Data High Byte		

The assembly object will set or get variable instances based on the instance (x) of the assembly object accessed. The table in figure 3.34 defines those variable instances.

Figure 3.34: Variable Instances to Set/Get for IMC and IMC*jr*

Assembly Object Instance (x)	Variable Instances (n) to Set/Get
768 – 895	Integer Variables: n = 32(x-768) + 1
896 – 959	Floating Point Variables: n = 32(x-896) + 1

DeviceNet Objects for Explicit Messaging

The remainder of this section shows how to use the *objects* in the DspMotion controller and provides examples of how to use them with explicit messaging over DeviceNet. The table in figure 3.35 identifies those objects and the services and attributes that they provide.

Figure 3.35 includes *System* and *Variable* (i.e., Boolean, Integer, Floating Point, and String) *Objects* that allow you to send commands to the controller and get/set controller variables via explicit messaging. Those objects are unique to GE Fanuc Motion controller products and have

been designed to let you get the most of your DeviceNet controllers. This is especially important because unlike other DeviceNet motion controllers, GE Fanuc Motion controller products can run complete application programs in conjunction with DeviceNet communication. Use figure 3.35 and the examples that follow as your guide, and try the examples with your own equipment—you'll see just how easy it is to send explicit messages to DspMotion controllers.

Figure 3.35: DspMotion Controller DeviceNet Objects

Object & Class ID (Hex)	Instance *(Generation D RTOS Equivalent)*	Service	Service Code (Hex)	Attr. ID	Attribute Description	Attribute Type	Attribute Values
Identity 01	1	Reset	05	—	Type of reset	USINT	0, 1
	1	Get Attr Single	0E	1	Vendor ID	UINT	38
	1	Get Attr Single	0E	2	Device type	UINT	12 -- Target® 16 -- IMC, IMC*jr*
	1	Get Attr Single	0E	3	Product code	UINT	0-65,535
	1	Get Attr Single	0E	4	Revision	UINT, UINT	1-4, 10-99
	1	Get Attr Single	0E	5	Status	WORD	0-65,535
	1	Get Attr Single	0E	6	Serial number	UDINT	$0-(2^{32}-1)$
	1	Get Attr Single	0E	7	Product name	SHORT_STRING	Model number
DeviceNet 03	0	Get Attr Single	0E	1	Revision	UINT	2
	1	Get Attr Single	0E	5	Allocation info	BYTE, USINT	0-255, 0-63, 255
	1	Allocate M/S Connection Set	4B	—	—	BYTE, USINT	0-255, 0-63
	1	Release M/S Connection Set	4C	—	—	BYTE	0-255
Assembly 04 ***(Target® only)***	1	Get/Set Attr Single	0E/10	3	Data	ARRAY of BYTE	See figures 3.30 & 3.31
	2	Get Attr Single	0E	3	Data	ARRAY of BYTE	See figures 3.30 & 3.31
Assembly 04 ***(IMC and IMCjr only)***	768 - 959	Get/Set Attr Single	0E/10	3	Data	ARRAY of BYTE	See figures 3.33 & 3.34

Table continues on the following page

Figure 3.35: DspMotion Controller DeviceNet Objects (continued)

Object & Class ID (Hex)	Instance *(Generation D RTOS Equivalent)*	Service	Service Code (Hex)	Attr. ID	Attribute Description	Attribute Type	Attribute Values
Connection 05	0	Create	08	—	—	—	—
	0	Delete	09	—	—	—	—
	1-6 or 1-16[a]	Reset	05	—	—	—	—
	1-6 or 1-16[a]	Delete	09	—	—	—	—
	1-6 or 1-16[a]	Apply Attributes	0D	—	—	—	—
	1-6 or 1-16[a]	Get Attr Single	0E	1	State	USINT	0-5
	1-6 or 1-16[a]	Get Attr Single	0E	2	Instance type	USINT	0, 1
	1-6 or 1-16[a]	Get Attr Single	0E	3	Transport class trigger	BYTE	23_{16}, 83_{16}
	1-6 or 1-16[a]	Get Attr Single	0E	4	Produced connection ID	UINT	0-7F0$_{16}$, FFFF$_{16}$
	1-6 or 1-16[a]	Get/Set Attr Single	0E/10	5	Consumed connection ID	UINT	0-7F0$_{16}$, FFFF$_{16}$
	1-6 or 1-16[a]	Get Attr Single	0E	6	Initial comm. characteristics	BYTE	0, 1, 21_{16}, 33_{16}
	1-6 or 1-16[a]	Get Attr Single	0E	7	Produced connection size	UINT	256, 8, 34
	1-6 or 1-16[a]	Get Attr Single	0E	8	Consumed connection size	UINT	256, 8, 34
	1-6 or 1-16[a]	Get/Set Attr Single	0E/10	9	Expected packet rate	UINT	0-65,535
	1-6 or 1-16[a]	Get/Set Attr Single	0E/10	12	Watchdog timeout action	UINT	0, 1, 3
	1-6 or 1-16[a]	Get Attr Single	0E	13	Produced connection path length	UINT	0, 6
	1-6 or 1-16[a]	Get Attr Single	0E	14	Produced connection path	EPATH	Empty, 202424003021_{16}
	1-6 or 1-16[a]	Get Attr Single	0E	15	Consumed connection path length	UINT	0, 6
	1-6 or 1-16[a]	Get Attr Single	0E	16	Consumed connection path	EPATH	Empty, 202424003020_{16}
Discrete Input Point 08 ***(IMC and IMCjr only)***	0	Get Attr Single	0E	1	Revision	UINT	2
	1-12 IMC 1-14 IMCjr *(DIn)*	Get Attr Single	0E	3	Input point value	BOOL	0, 1
Discrete Output Point 09 ***(IMC and IMCjr only)***	7-12 IMC 9-14 IMCjr *(DOn)*	Get/Set Attr Single	0E/10	3	Output point value	BOOL	0, 1

Table continues on the following page

[a] Instance 1–6 applies to the IMC and IMC*jr*. Instance 1–16 applies to the Target® ARS.

Figure 3.35: DspMotion Controller DeviceNet Objects (continued)

Object & Class ID (Hex)	Instance *(Generation D RTOS Equivalent)*	Service	Service Code (Hex)	Attr. ID	Attribute Description	Attribute Type	Attribute Values
Analog Input Point 0A ***(IMC and IMCjr only)***	0	Get Attr Single	0E	1	Revision	UINT	2
	1 IMC 1, 2 IMC*jr* *(AI, AIn)*	Get Attr Single	0E	3	Analog input value	INT	-10,000 = -10 Volts to 10,000 = 10 Volts
Analog Output Point 0B ***(IMC and IMCjr only)***	1 *(AO)*	Get/Set Attr Single	0E/10	3	Analog output value	INT	-10,000 = -10 Volts to 10,000 = 10 Volts
	1	Get Attr Single	0E	7	Output range	USINT	3 = -10 Volts to 10 Volts
Parameter 0F ***(IMC and IMCjr only)***	0	Get Attr Single	0E	2	Maximum instance	UINT	255
	0	Get Attr Single	0E	8	Parameter class descriptor	WORD	1
	0	Get Attr Single	0E	9	Configuration assembly instance	UINT	0
	1-128	Get/Set Attr Single	0E/10	1	Parameter value	DINT	-2,147,483,648 to 2,147,483,647
	129-255	Get/Set Attr Single	0E/10	1	Parameter value	REAL	1.5×10^{-39} to 1.7×10^{38} (absolute value)
	1-255	Get Attr Single	0E	2	Link path size	USINT	0
	1-255	Get Attr Single	0E	3	Link path	ARRAY	Empty
	1-255	Get Attr Single	0E	4	Descriptor	WORD	0
	1-128	Get Attr Single	0E	5	Data type	EPATH	$C4_{16}$ = DINT
	129-255	Get Attr Single	0E	5	Data type	EPATH	CA_{16} = REAL
	1-255	Get Attr Single	0E	6	Data size	USINT	4

Table continues on the following page

Figure 3.35: DspMotion Controller DeviceNet Objects (continued)

Object & Class ID (Hex)	Instance *(Generation D RTOS*	Service	Service Code (Hex)	Attr. ID	Attribute Description	Attribute Type	Attribute Values
Position Controller Supervisor 24_{16} *(IMC and IMCjr only)*	0	Get Attr Single	0E	1	Revision	UINT	2
	0	Get/Set Attr Single	0E/10	32	Consumed command message	ARRAY of BYTE	See figures 3.8 - 3.15
	0	Get Attr Single	0E	33	Produced response message	ARRAY of BYTE	See figures 3.17 - 3.25
	1	Get Attr Single	0E	1	Number of attributes	USINT	12
	1	Get Attr Single	0E	2	Attribute list	ARRAY of USINT	1-3, 5-7, 12, 15, 16, 21, 22, 24
	1	Get Attr Single	0E	3	Axis number	USINT	1
	1	Get Attr Single	0E	5	General fault	BOOL	0, 1 = fault condition exists
	1	Get Attr Single	0E	6	Command message type	USINT	1-5, 26, 27, 31
	1	Get Attr Single	0E	7	Response message type	USINT	1-3, 5, 8, 20, 26, 27, 31
	1	Get/Set Attr Single	0E/10	12	Home arm	BOOL	1 = armed; 0 = trigger has occurred
	1	Get/Set Attr Single	0E/10	15	Index arm	BOOL	1 = armed; 0 = trigger has occurred
	1	Get Attr Single	0E	16	Home input level	BOOL	0 = low; 1 = high
	1	Get/Set Attr Single	OE/10	21	Registration arm	BOOL	1 = armed; 0 = trigger has occurred
	1	Get Attr Single	0E	22	Registration input level	BOOL	0 = low; 1 = high
	1 *(PCA)*	Get Attr Single	0E	24	Registration position	DINT	+/-2,000,000,000
Position Controller 25_{16} *(IMC and IMCjr only*	0	Get Attr Single	0E	1	Revision	UINT	2
	1	Get Attr Single	0E	1	Number of attributes	USINT	47
	1	Get Attr Single	0E	2	Attribute list	ARRAY of USINT	1-3, 6-15, 17-21, 23-25, 30-32, 36, 40, 45, 47, 48 50, 51, 54-58, 100-110
	1	Get/Set Attr Single	0E/10	3	Mode	USINT	0 = position; 1 = velocity; 2 = torque
	1 *(MPI, MPA)*	Get/Set Attr Single	0E/10	6	Target position	DINT	+/-2,000,000,000
	1 *(MVL)*	Get/Set Attr Single	0E/10	7	Target velocity	DINT	1-16,000,000
	1 *(MAC)*	Get/Set Attr Single	0E/10	8	Acceleration	DINT	100-1,000,000,000
	1 *(MDC)*	Get/Set Attr Single	0E/10	9	Deceleration	DINT	100-1,000,000,000
	1	Get/Set Attr Single	0E/10	10	Incremental position flag	BOOL	0 = absolute 1 = incremental
	1	Get/Set Attr Single	0E/10	11	Load data/ profile handshake	BOOL	0, 1
	1	Get Attr Single	0E	12	On target position	BOOL	0, 1

Table continues on the following page

Figure 3.35: DspMotion Controller DeviceNet Objects (continued)

Object & Class ID (Hex)	Instance *(Generation D RTOS Equivalent)*	Service	Service Code (Hex)	Attr. ID	Attribute Description	Attribute Type	Attribute Values
	1 *(PSA)*	Get/Set Attr Single	0E/10	13	Actual position	DINT	+/-2,000,000,000
	1 *(VLA)*	Get Attr Single	0E	14	Actual velocity	DINT	+/-16,000,000
	1 *(PSC)*	Get Attr Single	0E	15	Commanded position	DINT	+/-2,000,000,000
	1	Get/Set Attr Single	0E/10	17	Enable	BOOL	0 = disable 1 = enable drive
	1	Get/Set Attr Single	0E/10	18	Profile type	USINT	0 = trapezoidal; 1 = s-curve
	1 *(MJK)*	Get/Set Attr Single	0E/10	19	Profile gain	DINT	0-100
	1 *(ST)*	Get/Set Attr Single	0E/10	20	Smooth stop	BOOL	0, 1 = stop
	1 *(HT)*	Get/Set Attr Single	0E/10	21	Hard stop	BOOL	0, 1 = halt
	1	Get/Set Attr Single	0E/10	23	Instantaneous direction	BOOL	0 = negative or reverse; 1 = positive or forward
	1 *(DIR)*	Get/Set Attr Single	0E/10	24	Reference direction	BOOL	0 = forward is CW 1 = forward is CCW
	1 *(TLC)*	Get/Set Attr Single	0E/10	25	Torque	DINT	+/-1,000
	1 *(KP)*	Get/Set Attr Single	0E/10	30	Proportional gain	UINT	0 - 8,000
Position	1 *(KI)*	Get/Set Attr Single	0E/10	31	Integral gain	UINT	0 - 64,000
Controller	1 *(KD)*	Get/Set Attr Single	0E/10	32	Derivative gain	UINT	0 - 8,000
25_{16}	1 *(KA)*	Get/Set Attr Single	0E/10	36	Accel feed forward	UINT	0 - 64,000
(IMC and	1 *(FR)*	Get/Set Attr Single	0E/10	40	Feedback resolution	DINT	500 - 1,000,000
IMCjr only)	1 *(FEB)*	Get/Set Attr Single	0E/10	45	Max following error	DINT	0 - 16,000
	1	Get Attr Single	0E	47	Following error fault	BOOL	0, 1
	1	Get Attr Single	0E	48	Actual following error	DINT	0 - 16,000
	1	Get Attr Single	0E	50	Forward limit	BOOL	0, 1
	1	Get Attr Single	0E	51	Reverse limit	BOOL	0, 1
	1 *(OTF)*	Get/Set Attr Single	0E/10	54	Positive soft limit position	DINT	+/-2,100,000,000
	1 *(OTR)*	Get/Set Attr Single	0E/10	55	Negative soft limit position	DINT	+/-2,100,000,000
	1	Get Attr Single	0E	56	Positive limit triggered	BOOL	0, 1
	1	Get Attr Single	0E	57	Negative limit triggered	BOOL	0, 1
	1	Get Attr Single	0E	58	Load data complete	BOOL	0, 1
	1 *(FC)*	Get Attr Single	0E	100	Fault code	UDINT	0 - $FFFFFFFF_{16}$
	1 *(CURC)*	Get/Set Attr Single	0E/10	101	Continuous current	UINT	1-1,000
	1 *(CURS)*	Get/Set Attr Single	0E/10	102	Power save current	UINT	0-1,000
	1 *(CURP)*	Get/Set Attr Single	0E/10	103	Peak current	UINT	1-1,000
	1 *(CMR)*	Get/Set Attr Single	0E/10	104	Commutation ratio	UINT	1-16
	1 *(CMO)*	Get/Set Attr Single	0E/10	105	Commutation offset	INT	+/-1,800
	1 *(IPB)*	Get/Set Attr Single	0E/10	106	In-position band	UINT	0-16,000
	1 *(PWE)*	Get/Set Attr Single	0E/10	107	Position wrap	BOOL	0, 1 = enabled
	1 *(KT)*	Get/Set Attr Single	0E/10	108	Filter time constant	UINT	0-5
	1 *(KL)*	Get/Set Attr Single	0E/10	109	Motor inductance	UINT	1-100
	1 *(KM)*	Get/Set Attr Single	0E/10	110	Motor number	UINT	0-20

Table continues on the following page

Figure 3.35: DspMotion Controller DeviceNet Objects (continued)

Object & Class ID (Hex)	Instance *(Generation D RTOS Equivalent)*	Service	Service Code (Hex)	Attr. ID	Attribute Description	Attribute Type	Attribute Values
System 64	1	Send Command	32	FF_{16}	Generation D RTOS Command	SHORT_STRING	Generation D RTOS acceptable command
Boolean Variable 65	*n (VBn)*	Get/Set Attr Single	0E/10	1	Value	BOOL	0, 1
Integer Variable 66	*n (VIn)*	Get/Set Attr Single	0E/10	1	Value	DINT	-2,147,483,648 to +2,147,483,647
|Floating Point Variable 67	*n (VFn)*	Get/Set Attr Single	0E/10	1	Value	REAL	1.5×10^{-39} to 1.7×10^{38} (absolute value)
String Variable 68	*n (VSn)*	Get/Set Attr Single	0E/10	1	Value	SHORT_STRING	Character string, 0-127 characters long

n = variable instance.
Note: The *Get/Set Attribute Single* services parse faster and are more efficient than the *Send Command* service because they take fewer bytes to send and receive.

Definitions of Position Controller Supervisor Object Instance Attributes

Instance 1 attributes of the Position Controller Supervisor object are defined in the following list. The number preceding the attribute name is the attribute ID, which corresponds with the attribute IDs provided in figure 3.35.

1 ***Number of Attributes.*** The total number of attributes supported by this object in this device.

2 ***Attribute List.*** Returns an array with a list of the attributes supported by this object in this device.

3 ***Axis Number.*** Returns the axis number, which is the same as the instance for this object. The axis number is the same as the instance number for all of the axis objects: position controller supervisor, position controller, drive, motor data, and block sequencer.

5 ***General Fault.*** General Fault flag. This bit is the logical OR of all fault condition attribute flags in the device. This bit is reset when the fault condition is removed.

6 ***Command Message Type.*** The command message type that is being sent by the controlling device. Valid Message Type codes are 1 to 1F hex: 1=Type 01 hex ; 2=Type 02 hex, etc.

7 ***Response Message Type.*** The response message type that is returned to the controlling device. Valid Message Type codes are 1 to 1F hex: 1=Type 01 hex; 2=Type 02 hex, etc.

12 ***Home Arm.*** Home trigger arm flag is used to arm the Home input.

15 ***Index Arm.*** Index trigger arm flag is used to arm the Index input.

16 ***Home Input Level.*** Actual level of the Home input.

21 ***Registration Arm.*** Registration trigger arm flag is used to arm the Registration input.

22 ***Registration Input Level.*** Actual level of the registration input.

24 ***Registration Position.*** Registration trigger position reflects the position at the time the Registration input is triggered.

Definitions of Position Controller Object Instance Attributes

Instance 1 attributes of the Position Controller object are defined in the following list. The number preceding the attribute name is the attribute ID, which corresponds with the attribute IDs provided in figure 3.35.

1 ***Number of Attributes.*** Returns the total number of attributes supported by this object in this device.

2 ***Attribute List.*** Returns an array with a list of the attributes supported by this object in this device.

3 ***Mode.*** Operating mode.

6 ***Target Position.*** Profile move position defined in pulses.

7 ***Target Velocity.*** Profile velocity defined in pulses/second.

8 ***Acceleration.*** Profile Acceleration rate defined in pulses/second^2.

9 ***Deceleration.*** Profile Deceleration rate defined in pulses/second^2.

10 ***Incremental Position Flag.*** If set to 0 the target position (attribute 6) will be interpreted as absolute. If set to 1 the target position will be interpreted as incremental.

11 ***Load Data/Profile Handshake.*** Used to Load Command Data, Start a Profile Move, and indicate that a Profile Move is in progress.

12 ***On Target Position.*** On target position flag indicates that the motor is within the in-position band (Attribute 106) distance to the target position and there is no motion. Reads Set when the Target position equals the actual position within limits. Will clear if target position is changed.

13 ***Actual Position.*** Actual Absolute position value equals the real position in pulses. Set to redefine Actual Position.

14 ***Actual Velocity.*** Actual Velocity in pulses/sec.

15 ***Commanded Position.*** This value equals the instantaneous calculated position.

17 ***Enable.*** Enable state of controller.

18 ***Profile Type.*** Profile Type code defines the type of move profile: 0 = trapezoid; 1 = S-curve.

19 ***Profile Gain.*** This attribute provides a gain value for non-trapezoidal profiles such as S-Curve profiling.

20 ***Smooth Stop.*** Smooth stop motor. Set to force immediate deceleration to zero velocity at programmed deceleration rate.

21 ***Hard Stop.*** Hard stop motor. Set to force immediate halt of motor.

23 ***Instantaneous Direction.*** 0 = negative or reverse direction and 1= positive or forward. This value can be set in Velocity mode to change direction.

24 ***Reference Direction.*** Defines direction. 0 = forward direction is clockwise; 1 = forward direction is counterclockwise as viewed from the load end of the motor shaft.

25 ***Torque.*** Output torque. Set this value to change the torque (Torque mode only) or read the current torque. 0 = no torque output. 1000 = maximum continuous current setting.

30 ***Proportional Gain.*** The position loop proportional control gain is used to multiply the following error to control the position of the axis. Set automatically by the AUTOTUNE command.

31 ***Integral Gain.*** The position loop integral control gain is used to multiply the time integral of the following error to control the position of the axis. Set automatically by the AUTOTUNE command.

32 ***Derivative Gain.*** The position loop derivative control gain is used to multiply the time derivative of the following error to control the position of the axis. Set automatically by the AUTOTUNE command.

36 ***Accel Feed Forward.*** The acceleration feedforward constant is used to reduce following error during acceleration or deceleration.

40 ***Feedback Resolution.*** Feedback resolution in pulses. Number of actual position feedback pulses in one revolution of the motor. This value is set to a positive number only.

45 ***Max Following Error.*** Maximum allowable following error. If difference between actual and commanded position exceeds this value, following error flag is set. Set value to a positive number only.

47 ***Following Error Fault.*** Following error occurrence flag. Set when a following error occurs.

48 ***Actual Following Error.*** Actual Following Error. This value is the actual amount of Following Error in position feedback pulses.

50 ***Forward Limit.*** Motion is not allowed in the positive direction when active. Set when the forward overtravel is active.

51 ***Reverse Limit.*** Motion is not allowed in the negative direction when active. Set when a reverse overtravel is active.

54 ***Positive Soft Limit Position.*** Soft limit positive boundary defined in pulses.

55 ***Negative Soft Limit Position.*** Soft limit negative boundary defined in pulses.

56 ***Positive Limit Triggered.*** Soft Forward limit occurrence flag. Set when a positive soft limit stop occurs.

57 ***Negative Limit Triggered.*** Soft Reverse limit occurrence flag. Set when a negative soft limit stop occurs.

58 ***Load Data Complete.*** Indicates that valid data for a valid I/O command message type has been loaded into the position controller device.

100 ***Fault Code.*** Identifies what type of system fault has taken place.

101 ***Continuous Current.*** The continuous current limits the current that the drive will continuously supply to the motor. It is a percentage of the maximum continuous current rating of the drive.

102 ***Power Save Current.*** The power save current is used to reduce motor heating when the axis is stopped. While the axis is in position, the continuous current value, CURC, is reduced to the percentage loaded into CURS. Stepping motor controllers only.

103 ***Peak Current.*** Limits the peak value of the current that the drive will supply to the motor. It is a percentage of the maximum peak current rating of the drive. Servo motor controllers only.

104 ***Commutation Ratio.*** Motor poles to resolver poles commutation ratio is one of the motor constants needed to operate a resolver feedback servo motor. This value, along with the value of CMO, can be set automatically by the MOTORSET command.

105 ***Commutation Offset.*** Commutation angle offset of the motor is set by the motor manufacturer. This value, along with the value of CMR, can be set automatically by the MOTORSET command.

106 ***In-Position Band.*** Defines the maximum amount of position error that the axis can have and still be in position.

107 ***Position Wrap.*** Determines whether position register wrap is enabled. If PWE is set to 1, position register wrap is enabled; and if PWE is set to 0, it is disabled.

108 ***Filter Time Constant.*** Filter time constant is used to eliminate dither. Set automatically by the AUTOTUNE command.

109 ***Motor Inductance.*** Tunes the digital current controller to the attached motor.

110 ***Motor Number.*** Tunes the controller to provide optimum performance for the attached stepper motor.

Explicit Message Examples

The explicit message examples that follow were generated from a typical commissioning software program. Other DeviceNet commissioning software products have a screen similar to the Basic Configuration operator interface screen in figure 3.36. Use this screen to generate your explicit messages.

Get Attribute Single Service

The *Get Attribute Single* service lets an explicit message retrieve the values of any of the DeviceNet object instances (refer to figure 3.35) that support the *Get Attribute Single* service. This example uses the Get Attribute Single service to get the string ***Hello*** stored in string variable 3 from the controller.

The device requesting service (by sending the command) is the client, with a MAC ID=00. The device responding to the command is the server, with a MAC ID=01. Figure 3.36 shows the data the user must enter. The *Service Code*, *Class*, *Instance*, and *Attribute* all come from the table of DspMotion Controller DeviceNet Objects in figure 3.35.

Click the *Send Request* button to send the command over DeviceNet to the controller. The successful response reveals the value of string variable 3 to be ***Hello***.

The *Basic Configuration* screen shows the simplified view of what occurs when you send a command. When you click that *Send Request* button, the client and server are actually exchanging data behind the scenes.

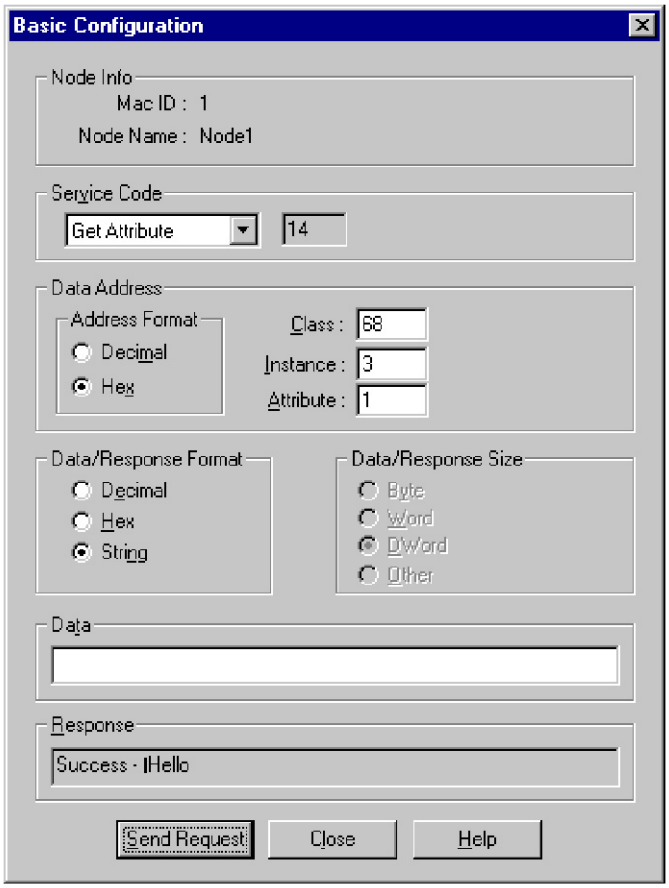
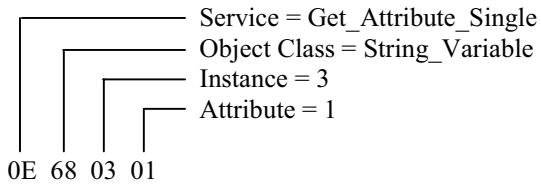


Figure 3.36: Using the Explicit Messaging Connection to Get Attribute from a DspMotion Controller

The data (hex) passed by application to DeviceNet are:



Set Attribute Single Service

The *Set Attribute Single* service allows an explicit message to set the value of any of the DeviceNet object instances (refer to figure 3.35) that support the *Set Attribute Single* service.

This example uses the *Set Attribute Single* service to set integer variable 20 (14 hex) to 2,000,000,000 (77359400 hex). The device requesting service (by sending the command) is the client, with a MAC ID=00. The device responding to the command is the server, with a MAC ID=01. Figure 3.37 shows the data the user must enter. The *Service Code*, *Class*, *Instance*, and *Attribute* all come from the table of DspMotion Controller DeviceNet Objects in figure 3.35. Click the *Send Request* button to send the command over DeviceNet to the controller. The response tells us that the explicit message was successful.

The *Basic Configuration* screen shows the simplified view of what occurs when you send a command. When you click that *Send Request* button, the client and server are actually exchanging a series of fragmented requests and responses.

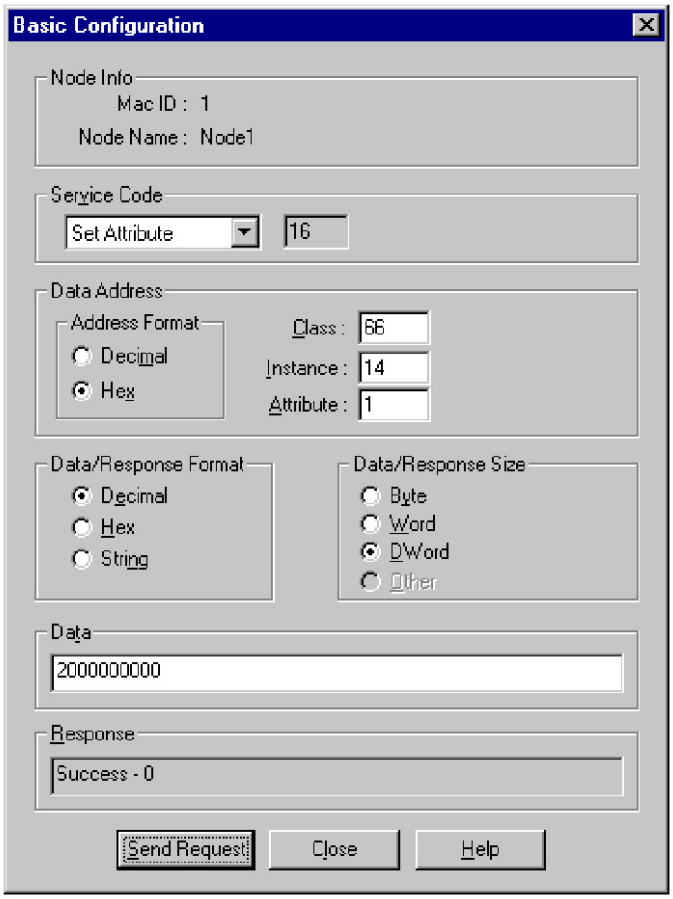
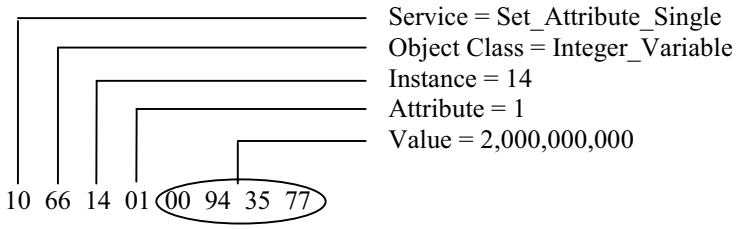


Figure 3.37: Using the Explicit Messaging Connection to Set Attribute from DspMotion Controller

The data (hex) passed by application to DeviceNet are:

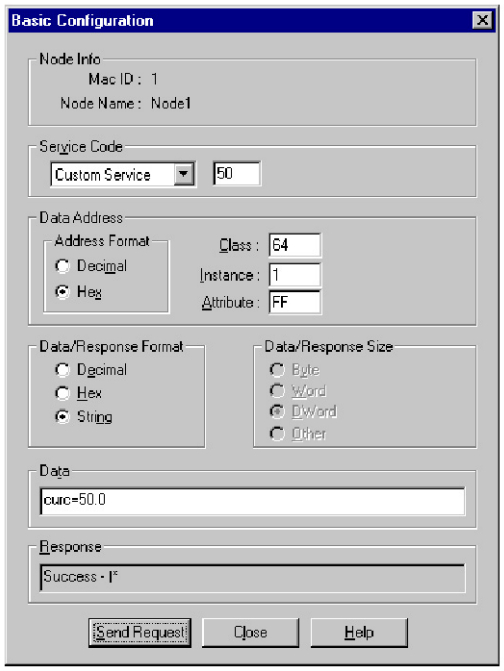


Send Command Service

The *Send Command* service allows you to send the registers and commands in the Generation D RTOS as explicit messages in immediate mode to your DspMotion controller. DeviceNet places no limits on the controller's capabilities in a network environment—you can still create, store, and execute programs within your DspMotion controller, just as if you were communicating to the controller via its serial port.

This example uses the *Send Command Service* to set the CURC (continuous current) register value to 50% by sending the string CURC=50.0 as an explicit message. The string can be from 0-240 characters. It should not include a unit address or carriage return.

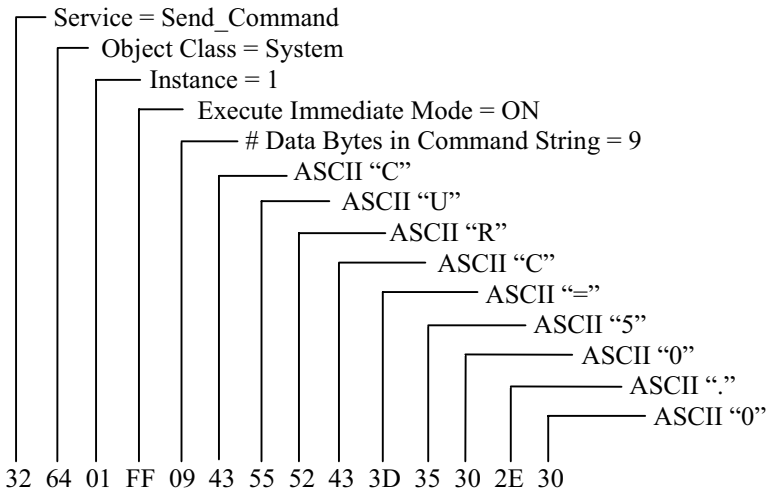
The device requesting service (by sending the command) is the client, with a MAC ID=00. The device responding to the command is the server, with a MAC ID=01. Figure 3.38 shows the data the user must enter. Click the *Send Request* button to send the command over DeviceNet to the controller. The response tells us that the explicit message was successful—the CURC register is set to 50.0 percent.



The send command service requires FF_{16} in the attribute field for immediate mode commands.

Figure 3.38: Basic Configuration Screen Used to Send Explicit Messages over DeviceNet

The *Basic Configuration* screen shows the simplified view of what occurs when you send a command. When you click that *Send Request* button, the client and server are actually exchanging a series of fragmented requests and responses. If the request includes any error, you will receive an error response. The data (hex) passed by application to DeviceNet are:



Peer-to-Peer Network Architecture

Using the available message types, DspMotion controllers can communicate to each other and to other UCMM-capable devices over DeviceNet in a peer-to-peer fashion. This is useful in multiaxis systems because axes can share information to improve system performance. Peer-to-peer communication allows DspMotion controllers to perform the following functions:

- Send commands to other controllers
- Receive commands from other controllers
- Load registers and variables from one controller into another controller
- Read registers and variables contained in another controller
- Exchange status information between controllers
- Read digital/analog inputs
- Control digital/analog outputs.

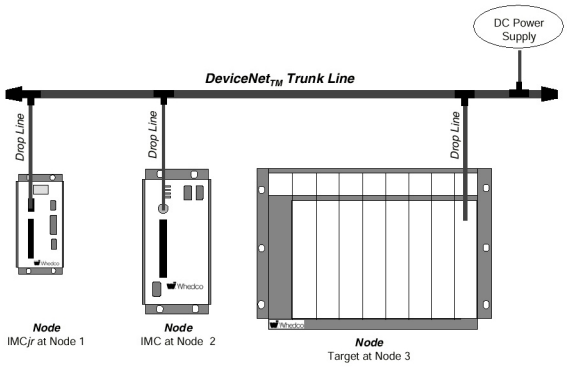


Figure 3.39: Peer-to-Peer Network Architecture

Talking Peer-to-Peer with CCS for Windows

GE Fanuc *CCS for Windows* supports program development and peer-to-peer communication for GE Fanuc Motion products connected on DeviceNet. In a peer-to-peer architecture, system designers can allow their DspMotion controllers to talk to each other over the network using standard Generation D RTOS mnemonics. ***No knowledge of DeviceNet communication protocol is required!*** And CCS accommodates either serial or DeviceNet connection.

To communicate from CCS version 6.0 or later to DspMotion controllers on a peer-to-peer network, just make a serial connection to one Motion unit anywhere in the network. If your PC is equipped with a DeviceNet network card (i.e., SST5136-DNP-PCMCIA, SST5136-DNP-PCI, or DIP052), you can talk to Motion controllers simply by connecting a drop line between the PC and the network trunkline.

Once connected, you can talk directly to any Motion controller on the network just as if you were connected to its serial port. Just select the controller's node address (MAC ID) as figure 3.41 shows. Then use the Generation D RTOS mnemonics to send/receive commands or load/read registers from one controller to another.

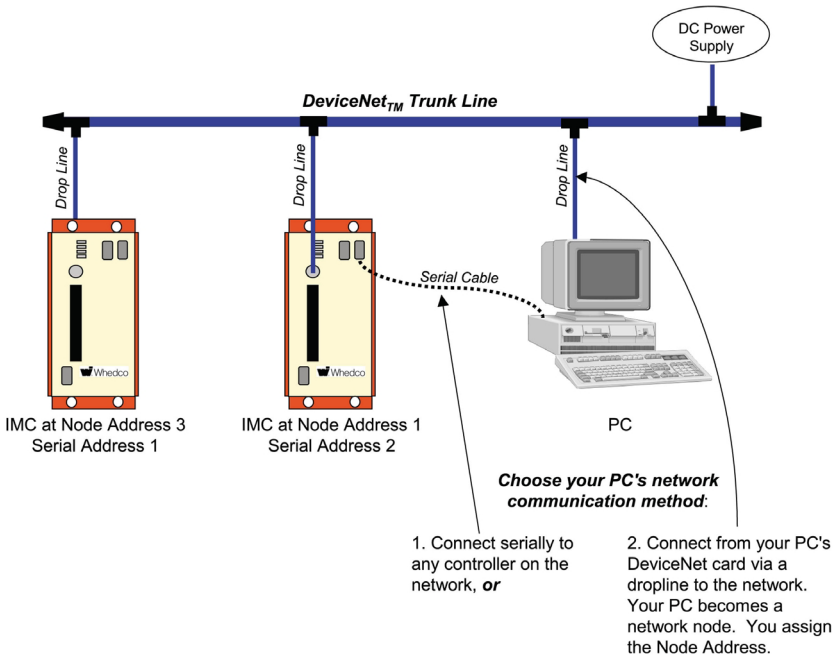


Figure 3.40: CCS Communication Options:
1. Serial to Controller to Network; and
2. Directly from PC to DeviceNet via Dropline

Peer-to-peer via Serial Connection to Network

To communicate serially to GE Fanuc Motion controllers on the network, connect a serial cable from your PC to the serial port on the controller. Run CCS for Windows version 6.0 or later, and configure the **Options** > **Communication Setup** screen (see figure 3.41, left screen) for serial communication. Select the *Serial* communication type, *COM Port*, *Baud Rate*, and click **OK**. Then configure the **Options** > **Controller Settings** screen (see figure 3.41, right screen); select the *Controller Type* of the GE Fanuc Motion product to receive communication; check *Network*; select the *Network Address* of the controller you want to address; and click **OK**. Note that the Controller Address indicates the serial address of the controller to which your serial cable is connected. Figure 3.40 shows the node address, serial address of the network nodes used to generate these examples.

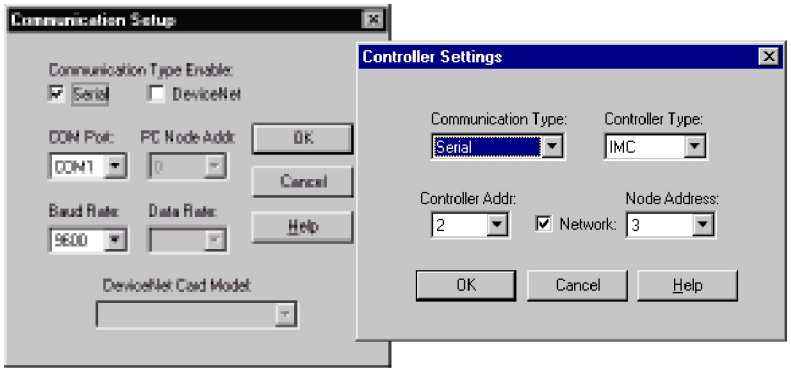


Figure 3.41: Communicate through Serial Connection to the Network

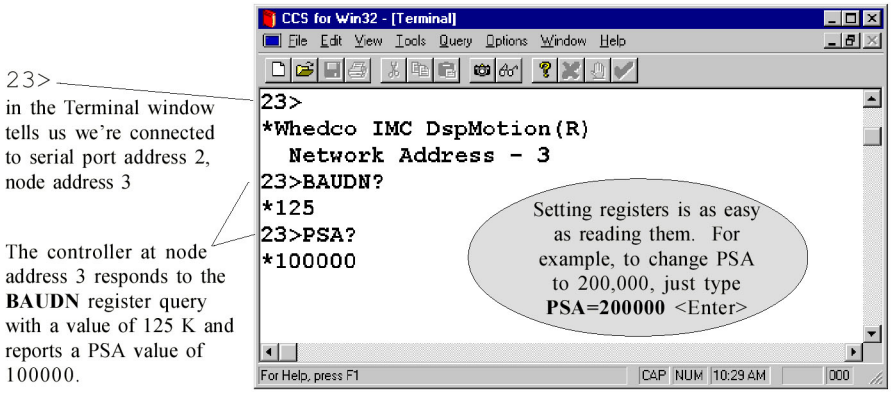


Figure 3.42: Reading Registers in Node Address 3 via Peer-to-peer Network Communication

Peer-to-peer via DeviceNet Dropline Connection to Network

If your PC is equipped with a DeviceNet network card (i.e., SST5136-DNP-PCMCIA, SST5136-DNP-PCI, or DIP052), you can assign a node address to your PC and talk to GE Fanuc Motion controllers directly over DeviceNet without having to connect a serial cable. Once you have run a dropline to connect your PC's DeviceNet card to the network trunk line, run CCS for Windows version 6.0 or later.

Configure the **Options** > **Communication Setup** screen (see figure 3.43, left screen) for DeviceNet communication. Enable DeviceNet as your *Communication Type*. Select the *COM Port*, and *Baud Rate*. Set the *PC Node Addr* to give your PC a network address (your PC is the network node from which you will be sending commands). Select the *Data Rate* for network communication, choose your *DeviceNet Card Model*, and click **OK**.

Now configure the **Options** > **Controller Settings** screen (see figure 3.43, right screen). Select DeviceNet as the *Communication Type*, choose the *Controller Type* of the GE Fanuc Motion product to receive communication; select the *Node Address* of the controller you want to address; and click **OK**. Figure 3.40 shows the node addresses of the IMCs used to generate these examples.

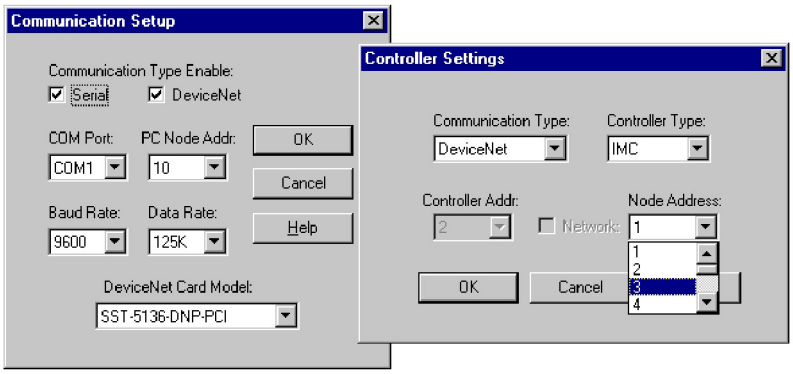


Figure 3.43: Communicate through DeviceNet Connection to the Network

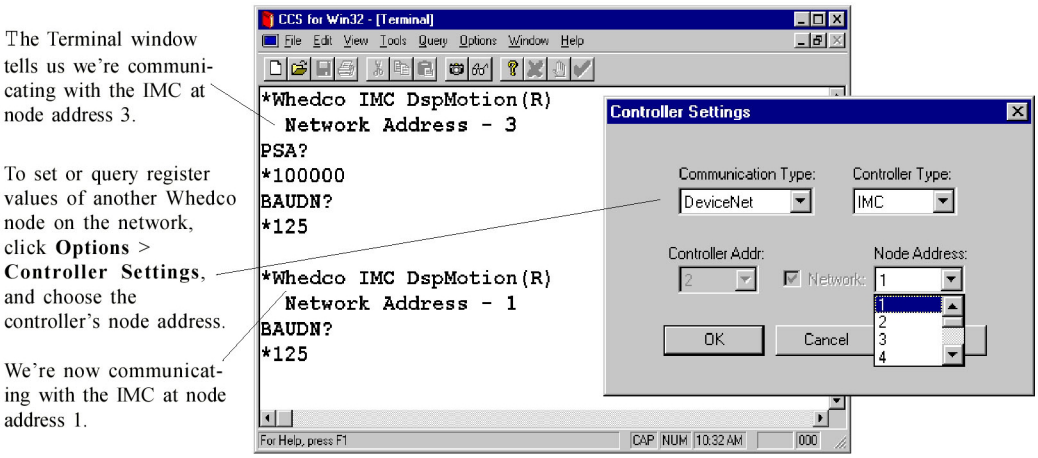


Figure 3.44: Reading Registers from Node Addresses 3 and 1 via Nonserial Peer-to-Peer Network Communication

Shortcuts with OUTN

With a serial connection between one controller and your PC, it is possible to send a command to a DspMotion controller on DeviceNet without selecting a new network address as we did in the previous example. Using the **OUTN** command, you can address any other valid command to the node address of the desired controller. In figure 3.45, the IMC at node address 1 has loaded the **MVL** register of the controller at node address 3 with a velocity of 10 axis units/second. After changing the node address to 3, we queried the value of MVL in the IMC at Node 3—sure enough, the IMC responds *10, telling us that MVL is set to 10.

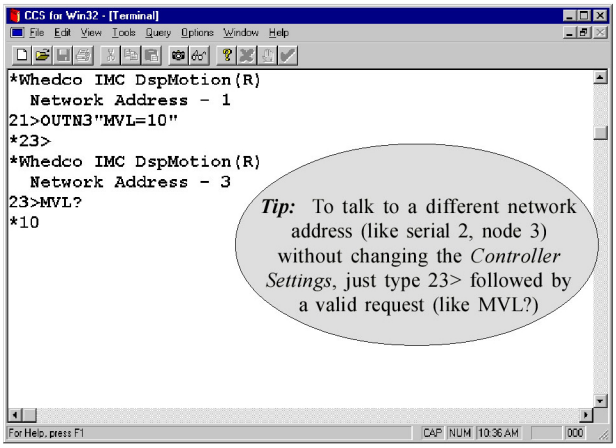


Figure 3.45: Using the OUTN Command

Running a peer-to-peer application is easy. Take a look at figure 3.46, which explains the Generation D RTOS syntax of the command issued in the previous example.

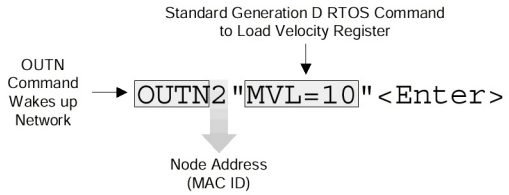


Figure 3.46: Syntax to Output a Command to a Node Address

Running a Peer-to-peer Application

In the following excerpts from a DeviceNet application program, motion sequences are controlled between IMCs over DeviceNet. The optional GE Fanuc Operator Interface can display position information from both axes at the same time.

```
(*Program1
        OUTS3                       (*outputs screen 3
        UPS=3                       (*updates data fields for screen 3
10      STM1=1                      (*sets timer 1=1 second
        WAIT TM1                    (*waits for 1 second
        PSA=0                       (*sets axis 1 absolute position=0
        STM1=2                      (*sets timer 1=2 seconds
        WAIT TM1                    (*waits for 2 seconds
        EXM1                        (*executes motion block 1
        WAIT NOT MB                 (*waits for motion block 1 to end
        STM1=1                      (*sets timer 1=1 second
        WAIT TM1                    (*waits for 1 second
        OUTN63“PSA=0”               (*sets over DeviceNet axis 2 absolute position=0
        STM1=2                      (*sets timer 1=2 seconds
        WAIT TM1                    (*waits for 2 seconds
        OUTN63“MAC=2500000”         (*sets axis 2 acceleration=2,500,000 pulses/sec2
        OUTN63“MVL=250000”          (*sets axis 2 velocity=250,000 pulses/sec
        OUTN63“MPI=250000”          (*sets axis 2 incremental move distance
        OUTN63“RPI”                 (*commands axis 2 to run incremental move
                                    (*distance=250,000
15      OUTN63“VB1=IP”              (*sets Boolean variable 1 in axis 2
                                    (*true when axis 2 is In Position
        IF NOT VBN63.1 GOTO15       (*gotos label 15 until axis 2 is In Position
                                    (*i.e., Boolean variable 1 is true
20      GOTO10                      (*gotos label 10 in Program 1 and repeats
        END
```

```
(*Program 2 runs concurrently with Program 1 and updates the Operator Interface
(*display with the current position of axis 2
        STM2=.2                     (*sets timer 2=200 milliseconds
10      WAIT TM2                    (*waits for 200 milliseconds
        OUTN63“VF1=PSA”             (*set floating point variable 1 in axis 2=absolute
                                    (*position of axis 2
        GOTO10                      (*gotos label 10 in Program 2 and repeats
        END
```

Distributed Control Network Architecture

Using Peer-to-peer and Master/Slave Communication in the Same System

DeviceNet supports multiple, concurrent communication hierarchies. Peer-to-peer and master-slave hierarchies can coexist on the same network; and so can multiple message types. This means that DspMotion controllers on a peer-to-peer network are ideal for distributed control applications with self-contained automation tools.

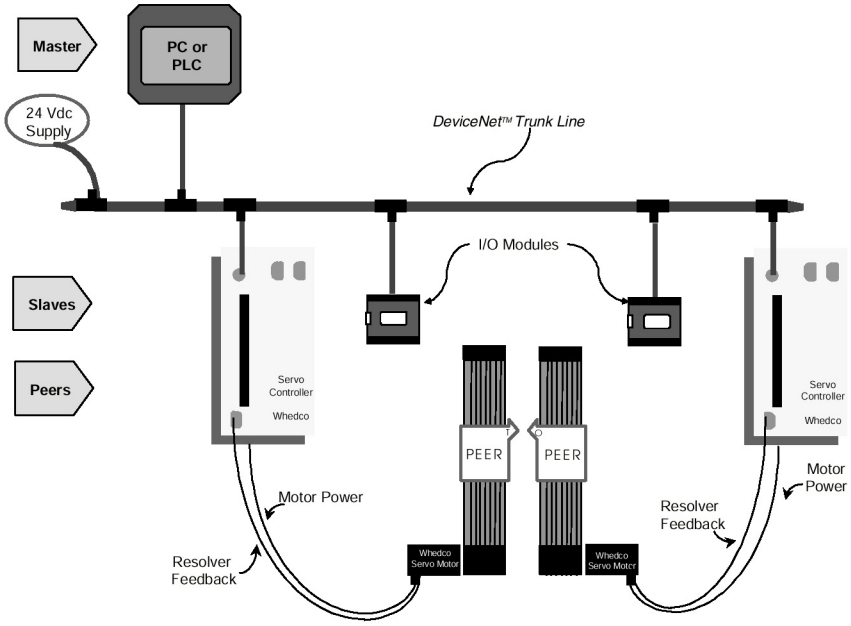


Figure 3.47: Distributed Control Architecture

Being already DeviceNet-enabled, these self-contained tools support a plug-and-play automation architecture in which the operation of the tool is not dependent on the plant network, but connectivity to that network allows dynamic interaction between the tool and the plant network. Tool behavior (e.g., target position) can be changed based on inputs from the network. In addition, status information can be shared among the tool, supervisory controls, and human-machine interfaces also on the network.

Using Remote I/O in a DeviceNet Peer-to-peer System

In some systems, it may be desirable to use remote I/O modules for inputs to GE Fanuc Motion products. DspMotion controllers can use explicit messages to communicate with UCMM-capable remote Digital I/O Modules. Figure 3.48 shows this procedure.

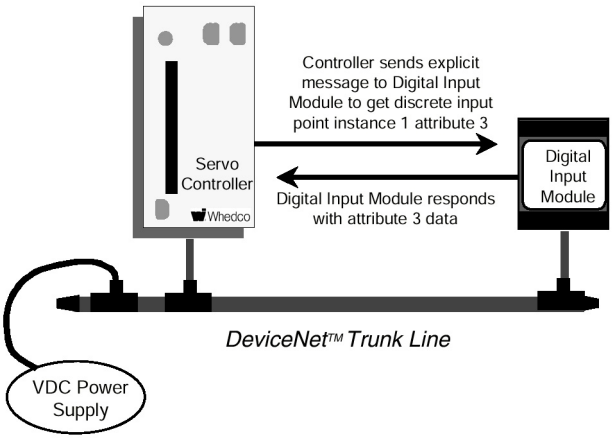


Figure 3.48: Communication between DspMotion Controller and UCMM Remote I/O Module

DspMotion products have several general-purpose inputs and outputs that can be used directly in the DspMotion control program for process control. The registers for those inputs and outputs, and their maximum I/O counts, are listed in the table in figure 3.49.

Figure 3.49: Generation D RTOS I/O Registers—Maximum Counts

Type	RTOS Register	Maximum # of Nodes	Maximum # per Node
Analog Input	AIN	63	64
Analog Output	AON	63	64
Digital Input	DIN	63	1,024
Digital Output	DON	63	1,024

Consult GE Fanuc for compatibility requirements of I/O modules.

The following example shows a section of a DspMotion program which, upon seeing digital input (DIN) 3 of node 20 from a remote input on the network, makes an absolute move.

IMC or IMC*jr*	**Target**	
Program2	Program 2	(*begin program 2
STM1=.03	STM1=.03	(*set timer 1 for 30 millisecond timeout
1 WAIT(TM1)	1 WAIT(TM1)	
IF NOT DIN20.3 GOTO 1	IF NOT DIN20.3 GOTO 1	
		(*if digital input 3 of node 20 is not true,
		(*goto label 1
RPA	RPA1	(*run motor to absolute position
END	END	(*end program 2

Digital input 3 of node 20

Chapter 4 Frequently Asked Questions

I can't communicate with my GE Fanuc DeviceNet nodes?

You must have DC voltage on the DeviceNet trunkline before you can talk to GE Fanuc Motion controllers.

My controller is in a faulted state—how do I attempt to reset the fault?

IMC or IMC*jr*

Using I/O Messaging: Set the Enable bit (i.e., bit 7, byte 0) in the I/O Command Message false; then set the enable bit true (see figure 3.3). This will cause the controller to execute the RSF[1] command. Note that the hardwired enable input to the controller must also be true.

All DspMotion Controllers

Using Explicit Messaging, Send Command Service: Use the System Object, class ID 64 hex, instance 1, to send the RSF[1] or RSFALL[1] command to the controller as a short string (see figure 3.35). Note that the hardwired enable input to the controller must also be true.

All DspMotion Controllers

Peer-to-peer: Send the RSF[1] or RSFALL[1] command to the controller using the OUTN[1] command or via the Terminal window of CCS (see figures 3.40 and 3.41). Note that the hardwired enable input to the controller must also be true.

How do I poll the IMC or IMC*jr* without setting a data value?

Set the Load Data bit false in the I/O Command Message.

[1] Generation D RTOS register or command.

I get *no response* from the GE Fanuc node when I try to allocate the predefined master-slave connection set?

Consult your scanner manufacturer to verify that the scanner module you are using works with UCMM-capable devices.

What does the Generation D RTOS Message *Resource Not* Available mean for a DeviceNet system?

The message ***Resource Not Available*** means that the DeviceNet communication did not occur correctly for some reason. Use CCS to query the FCN[1] register (i.e., type FCN? <Enter>) for a more specific network fault code message. See Appendix B for a list of the potential fault and status messages and possible solutions.

My system response times are slow!

In a master/slave hierarchy, the scanner handles all communication; and only one message can travel at a time. When you use GE Fanuc Motion controllers as slaves, they must wait their turn in the scan list, along with all the other network devices. This can lead to response times that are a little slow for some systems. To get back to full DspMotion processing speed, create a distributed control architecture. Just connect your GE Fanuc Motion controllers in a peer-to-peer fashion so that they can talk to each other without using the scan list. The master PC/PLC can still retain control over the non-GE Fanuc devices on the network. Figure 3.1 in Chapter 3 describes the message priorities.

How do I change the deceleration rate in the I/O command message?

Use the I/O Command Message Type 04 hex, Deceleration, (see figure 3.11) to set the motion deceleration. The deceleration does not default to the acceleration value.

What units do DspMotion controllers use to report position values in the I/O response message?

All units are in position pulses.

When using explicit messaging to retrieve a variable, do I get VI or VIN?

VI accesses variables of the controller in which the program resides. ***VIN*** accesses variables of any other controller on the network.

How are message priorities assigned to a series of OUTN commands?

These commands use message group 1 exclusively, and it is not possible to assign message priorities or groups. If you send the OUTN command in a program, it waits for a response before it

executes (wait length depends on network activity). To best manage your machine control, we recommend that you follow the programming practices described in the Generation D RTOS Programming Manual, for example:

1. Use program 1 for your main control
2. Call other programs and motion blocks from program 1
3. Put OUTN commands in a program that doesn't contain time-critical functions.

How do GE Fanuc Motion controllers manage the UCMM connection?

Each GE Fanuc Motion controller can open 3 client connections and respond to 3 server connections simultaneously. The controllers automatically handle connection management. For example, if all connections are open and node x tries to open another one, the controller will check to see whether any of the connections are available. If the controller finds an open but unused connection, it will reassign the connection to node x. If all connections are being used, it will issue an *Out of Connection* fault. If you are using many devices on a network, use the NCO command to reassign connections.

What does NCO do?

NCO is a read-only register that lets you attempt to make a connection without faulting the controller if the connection is unsuccessful. When you use NCO in a program, be sure that your next line of code makes a decision about what to do next if a connection isn't available (e.g., try to connect again).

How many server connections does the master/slave connection consume?

The master/slave connection uses 2 server connections: 1 for I/O and 1 for explicit messaging. The three client connections remain available.

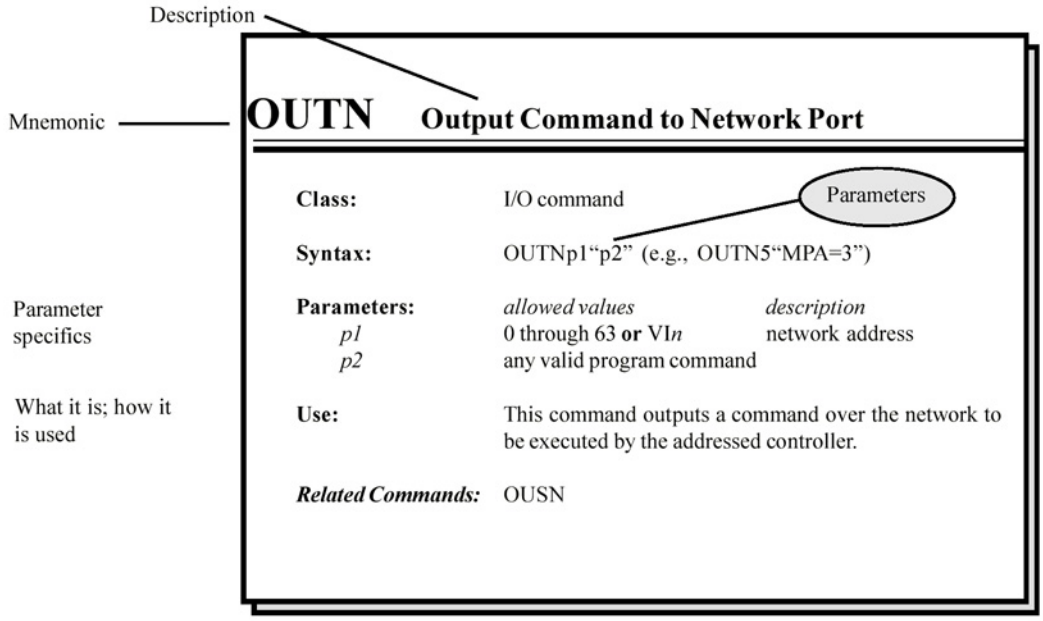
Appendix A

Generation D RTOS Network Registers and Commands

The network registers and commands in Appendix A are alphabetized and formatted according to the template shown below. Note that not all of the fields (i.e., *Type*, *Restrictions*, etc.) apply to every mnemonic. All registers and commands in this appendix apply to the entire DspMotion® family of DeviceNet controllers. In some cases, a particular feature may apply only to a specific product—those distinctions are noted with the following symbols:

I	Applies to IMC
⊙	Applies to Target®
jr	Applies to IMC*jr*

This appendix contains only those Generation D RTOS commands and registers that apply to DeviceNet systems. For all non-network registers and commands, consult CCS online help or *The Generation D RTOS Programming Manual*, Appendix A, and the *IMCjr Programming Supplement*, Appendix A.



ADDN Address of Network Port

Class: System Register

Type: Integer

Syntax: ADDN

Range:

default	63
minimum	0
maximum	63

Restrictions: Cannot be assigned in programs or motion blocks. Read only for IMC*jr*.

Use: The address of the network port is a number used to identify the network port. This setting is also referred to as the network address or node address.

Remarks:

I If DIP switch 9 is set to the right, the ADDN register value determines the address of the network port. If, however, DIP switch 9 is set to the left, DIP switches 1-6 determine the address of the network port. See figure 2.4 for DIP switch settings for the IMC network address.

I ⊙ You must cycle the power to the controller before an ADDN register setting will take effect.

Related Registers: BAUDN

AIN Network Analog Input

Class:	I/O Register	
Type:	Integer	
Syntax:	AIN*p1.p2* (e.g., AIN28.2 AINVI5.6 AINVI2.VI7)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	1 through 64 **or** VI*n*	analog input number
Range:		
minimum	-32,768	
maximum	32,767	
	Note that these minimum and maximum values are a function of your analog input device. Refer to the documentation for your analog input device to determine how to map its values to your DspMotion® controller.	
Restrictions:	Read only. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	The network analog input is a general-purpose input used for process control.	
Related Registers:	AON	

AON Network Analog Output

Class: I/O Register

Type: Integer

Syntax: AON*p1.p2* (e.g., AON15.7 AONVI2.4 AONVI5.VI2)

Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	1 through 64 **or** VI*n*	analog output number

Range:

minimum	-32,768
maximum	32,767

Note that these minimum and maximum values are a function of your analog output device. Refer to the documentation for your analog output device to determine how to map its values to your DspMotion® controller.

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: The network analog output is a general-purpose output used for process control.

Related Registers: AIN

BAUDN Baud Rate of Network Port

Class: System Register

Type: Integer

Syntax: BAUDN

Range:

default	125 kbit/s
allowed values	125, 250, 500

Restrictions: Cannot be assigned in programs or motion blocks. Read only for IMC*jr*.

Use: The baud rate of the network port is the rate at which bit transfer takes place to and from the network port.

Remarks:

I If the IMC's DIP switch 9 is set to the right, the BAUDN register value determines the network port baud rate. If, however, DIP switch 9 is set to the left, DIP switches 7 and 8 determine the network port baud rate. See figure 2.4 for DIP switch settings for the IMC network baud rate.

Related Registers: ADDN

CNC Close Network Connection

Class:	System Command	
Syntax:	CNC*p1*	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command closes the network connection to the device addressed at *p1*.	
Related Registers:	NCO	

CURCN Network Continuous Current

Class: Axis Register

Type: Floating Point

Syntax: CURCN*p1* (e.g., CURCN0 CURCN63 CURCNVI2)

Parameters:	*allowed values*	*description*
p1	1 through 63 **or** VI*n*	network address

Range:

units	%
minimum	1.0
maximum	100.0

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: This command accesses attribute 101 of the position controller object to limit the current that the drive of the device addressed at *p1* will continuously supply to the motor. It is a percentage of the maximum continuous current rating of the drive.

Remarks: The equation for CURC = motor continuous current rating / drive continuous current rating x 100%. Continuous current ratings are listed on the drive and motor product labels. Use the following values for CURCN:

IMC/IMJ Stepper Unit

Motor	*5 Amps*
1221-_-A-E-_	70
1231-_-A-E-_	62
1324-_-A-E-_	100
1324-_-D-E-_	54
1337-_-A-E-_	82
1337-_-D-E-_	82
1350-_-A-E-_	100
1350-_-D-E-_	80
1362-_-A-E-_	80
1454-_-A-E-_	100
1480-_-A-E-S	100

IMC Servo Unit CURC Values

Motor	*3 Amps*	*6 Amps*	*12 Amps*	*24 Amps*
3S22-G	46	23	n/a	n/a
3S32-G	96	48	24	n/a
3S33-G	100	53	26	n/a
3S33-H	100	100	53	26
3S34-G	100	50	25	n/a
3S35-G	96	48	24	n/a
3S43-G	96	48	24	n/a
3S43-H	100	93	46	23
3S45-G	100	91	45	22
3S45-H	100	100	91	45
3S46-G	100	91	45	22
3S46-H	100	100	91	45
3S63-G	100	100	91	45
3S65-G	100	100	89	44
3S67-G	100	100	94	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

CURCN Network Continuous Current (continued)

IMC*jr* Servo Unit CURC Values

Motor	*3 Amps*	*7.2 Amps*	*16 Amps*	*28 Amps*
3N21-H	100	42	n/a	n/a
3N22-H	100	42	n/a	n/a
3N24-G	87	36	n/a	n/a
3N31-H	100	46	21	n/a
3N32-G	100	42	n/a	n/a
3N32-H	100	86	39	22
3N33-G	93	39	n/a	n/a
3N33-H	100	79	36	20
3S22-G	50	21	n/a	n/a
3S32-G	100	42	n/a	n/a
3S33-G	100	44	20	n/a
3S34-G	100	42	n/a	n/a
3S35-G	100	42	n/a	n/a
3S43-G	97	40	n/a	n/a
3S43-H	100	79	36	20
3S45-G	100	76	34	20
3S45-H	100	100	69	39
3S46-G	100	76	34	20
3S46-H	100	100	69	40
3S63-G	100	100	69	40
3S63-H	100	100	100	79
3S65-G	100	100	68	39
3S65-H	100	100	100	77
3S67-G	100	100	71	40
3S84-G	100	100	100	100
3S86-G	100	100	100	100
3S88-G	100	100	100	100
3S8A-G	100	100	100	88

CURCN Network Continuous Current (continued)

Target ARS Servo Motor CURC Values

Motor	1 Servo Module	2 Servo Modules	3 Servo Modules	4 Servo Modules
3S22-G	23	n/a	n/a	n/a
3S32-G	48	24	n/a	n/a
3S33-G	53	26	n/a	n/a
3S33-H	100	53	35	26
3S34-G	50	25	n/a	n/a
3S35-G	48	24	n/a	n/a
3S43-G	48	24	n/a	n/a
3S43-H	93	46	31	23
3S45-G	91	45	30	22
3S45-H	100	91	61	45
3S46-G	91	45	30	22
3S46-H	100	91	61	45
3S63-G	100	91	61	45
3S65-G	100	89	59	44
3S67-G	100	94	63	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

Related Registers: CURPN, CURSN, CURC, CURP, CURS, TLC

CURSN Network Power Save Current

Class:	Axis Register	
Type:	Floating Point	
Syntax:	CURSN*p1* (e.g., CURSN0 CURSN63 CURSNVI5)	
Parameters:	*allowed values*	*description*
p1	1 through 63 **or** VI*n*	network address
Range:		
units	%	
minimum	0.0	
maximum	100.0	

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: This command accesses attribute 102 of the position controller object to reduce motor heating when the axis driven by the device addressed at *p1* has stopped. While the axis is in position, the network continuous current value, CURCN, is reduced to the percentage loaded into CURSN. For example, if CURCN63=50 and CURSN63=20, the value of CURCN will be reduced to 10 percent while the axis is in position.

Related Registers: CURCN, CURC, CURS

DIN Network Digital Input

Class: I/O Register

Type: Integer, Boolean

Syntax: DIN*p1.p2* (e.g., DIN5 DINVI4 DIN3.28 DINVI3.16 DINVI6.VI9)

Parameters:

	allowed values	*description*
p1	1 through 63 **or** VI*n*	network address
p2	none **or** 1 through 1,024 **or** VI*n*	digital input number

Range:

allowed values 0 through $FFFFFFFF_{16}$ **or** 0 and 1

Note that these minimum and maximum values are a function of your digital input device. Refer to the documentation for your digital input device to determine how to map its values to your DspMotion® controller.

Restrictions: Read only. Cannot be accessed in immediate mode over a DeviceNet connection.

Use: The network digital input register contains the values of the network digital inputs. The digital inputs are general-purpose inputs used for process control.

Remarks:

1. When the DIN*p1.p2*? command is executed, the value of the digital input *p2* of the network digital input device addressed at *p1* will be given as a Boolean number.

2. When DIN*p1*? is executed, up to four bytes of the digital inputs of the assigned assembly object instance of the network digital input device addressed at *p1* will be reported as a hexadecimal number. The four bytes reported are determined by the DINO register.

Examples:

```
DINA17=101,2      (* set assembly object instance 101 and 2 bytes)
DIN17?            (* report value of network digital inputs)
16#13AC
```

Related Registers: DINA, DINO, DON

DINA Network Digital Input Register Assignment

Class:	I/O Register	
Syntax:	DINA*p1* (e.g., DINA3 DINA63)	
Parameters:	*allowed values*	*description*
p1	0 through 63	network address
Range:		
default	1,1	
allowed values	1 through 255	assembly object instance
	1 through 128	number of bytes
Restrictions:	Not allowed in expressions.	
Use:	This register is used to assign the assembly object instance number and number of bytes to be used by the DIN register to read the network digital inputs from the device addressed at *p1*. The first number of the assignment is the assembly object instance and the second number is the quantity of bytes used by the assembly object instance data attribute.	
Example:		
DINA11=101,2	(* set assembly object instance 101 and 2 bytes)	
Related Registers:	DIN, DINO	

DINO Network Digital Input Register Offset

Class:	I/O Register	
Syntax:	DINO*p1* (e.g., DINO3 DINO63)	
Parameters:	*allowed values*	*description*
p1	0 through 63	network address
Range:		
default	0	
allowed values	0 through 124	byte offset
Use:	This register defines the starting byte offset into the data returned by the assembly instance accessed by DIN. The DIN register can return a maximum of four bytes. When the number of bytes returned by the assembly instance is greater than four bytes, the DINO register specifies the starting byte of the four bytes returned by DIN.	
Example:		
DINO11=100	(* set network digital input register offset of device at address 11 to 100)	
Related Registers:	DIN, DINA	

DIRN Network Direction of Motor

Class:	Axis Register	
Syntax:	DIRN*p1* (e.g., DIRN0 DIRN63 DIRNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
Range:		
allowed values	CW, CCW	
Restrictions:	Not allowed in motion blocks. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command accesses attribute 24 of the position controller object to define the direction of the motor assigned to the axis for forward moves. If DIR is set to CW, a forward move by the motor is clockwise, facing the motor shaft. If DIR is set to CCW, a forward move by the motor is counterclockwise, facing the motor shaft.	

DON Network Digital Output

Class: I/O Register

Type: Integer, Boolean

Syntax: DON*p1.p2* (e.g., DON5 DONVI4 DON3.28 DONVI3.16 DONVI6.VI9)

Parameters:

	allowed values	*description*
p1	0 through 63 **or** VI*n*	network address
p2	none **or** 1 through 1,024 **or** VI*n*	digital output number

Range:

allowed values 0 through $FFFFFFFF_{16}$ or 0 and 1

Note that these minimum and maximum values are a function of your digital output device. Refer to the documentation for your digital output device to determine how to map its values to your DspMotion® controller.

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: The network digital output register contains the values of the network digital outputs. The digital outputs are general-purpose outputs used for process control.

Remarks:

1. When the DON*p1.p2*? command is executed, the value of the digital output *p2* of the network digital output device addressed at *p1* will be given as a Boolean number.

2. When DON*p1*? is executed, up to four bytes of the digital outputs of the assigned assembly object instance of the network digital output device addressed at *p1* will be reported as a hexadecimal number. The four bytes reported are determined by the DONO register.

Examples:

```
DONA17=101,2      (* set assembly object instance 101 and 2 bytes)
DON17?            (* report value of network digital outputs)
16#13AC
```

Related Registers: DONA, DONO, DIN

DONA Network Digital Output Register Assignment

Class:	I/O Register	
Syntax:	DONA*p1* (e.g., DONA3 DONA63)	
Parameters:	*allowed values*	*description*
p1	0 through 63	network address
Range:		
default	1,1	
allowed values	1 through 255	assembly object instance
	1 through 128	number of bytes
Restrictions:	Not allowed in expressions.	
Use:	This register is used to assign the assembly object instance number and number of bytes to be used by the DON register to get or set the network digital outputs of the device addressed at *p1*. The first number of the assignment is the assembly object instance and the second number is the quantity of bytes used by the assembly object instance data attribute.	
Example:		
DONA11=101,2	(* set assembly object instance 101 and 2 bytes)	
Related Registers:	DON	

DONO Network Digital Output Register Offset

Class:	I/O Register	
Syntax:	DONO*p1* (e.g., DONO3 DONO63)	
Parameters:	*allowed values*	*description*
p1	0 through 63	network address
Range:		
default	0	
allowed values	0 through 124	byte offset
Use:	This register defines the starting byte offset into the data required for the assembly instance accessed by DON. The DON register can access a maximum of four bytes. When the number of bytes required by the assembly instance is greater than four bytes, the DONO register specifies the starting byte of the four bytes accessed by DIN. The rest of the bytes of the assembly instance are unchanged.	
Example:		
DONO11=100	(* set network digital output register offset of device at address 11 to 100)	
Related Registers:	DON, DONA	

FCN Network Fault Code

Class: System Register

Type: Integer, Boolean

Syntax: FCN*p1* (e.g., FCN FCN4 FCNVI6)

Parameters:	*allowed values*	*description*
p1	none **or** 0 through 15 **or** VI*n*	network fault code register bit number

Range:

allowed values	0 through $FFFF_{16}$ **or** 0 and 1

Restrictions: Read only.

Use: The network fault code register is used to identify what type of network fault has taken place.

Remarks:

1. When the FCN? command is executed, the network fault code register value will be given as an English statement. If no fault has occurred, the message given is *Network Functional*.

2. If the computer interface format is enabled, and the FCN? command is executed, the network fault code register value will be given as an integer number. If no fault has occurred, the network fault code register is set to 0. The possibilities are listed below:

bit	*message*	*bit*	*message*
0	Network Off-line	8	Not Enough Data
1	Addressed Device Not Present	9	Too Much Data
2	Addressed Device Out of Connections	10	Device State Conflict
3	Connection Deleted Unexpectedly	11	I/O Scan Time-Out
4	Time-out On Response	12	Invalid Attribute Value
5	Not Requested Response	13	Attribute Not Supported
6	Error Response	14	Object Does Not Exist
7	Resource Unavailable	15	Reserved

FCNN Network Device Fault Code

Class: System Register

Type: Integer, Boolean

Syntax: FCNN*p1.p2* (e.g., FCNN0 FCNN63.31 FCNN2.VI3 FCNNVI5.VI3)

Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
p2	none **or** 0 through 31 **or** VI*n*	network device fault code register bit number

Range:

allowed values	0 through $FFFFFFFF_{16}$ **or** 0 and 1

Restrictions: Read only. Cannot be accessed in immediate mode over a DeviceNet connection.

Use: The network device fault code register accesses attribute 100 of the position controller object to identify what type of fault has taken place in the device addressed at *p1*.

Remarks:

1. When the FCNN? command is executed, the fault code register value will be given as an English statement. If no fault has occurred, the message given is *Controller functional*.

2. If the computer interface format is enabled, and the FCNN? command is executed, the fault code register value will be given as an integer number. If no fault has occurred, the fault code register is set to 0. The possibilities are listed below:

bit	*message*
0	Power Failure
1	Reserved
2	Software Fault
3	Lost Enable
4	Digital Output Fault
5	Invalid Command in String
6	Transmit Buffer Overflow
7	Resource Not Available
8	Invalid Variable Pointer
9	Mathematical Overflow
10	Mathematical Data Error
11	Value Out of Range
12	String Too Long
13	Nonexistent Label
14	Gosub Stack Underflow
15	Gosub Stack Overflow
16	Invalid Motion
17	Reserved
18	Reserved
19	Network Power Failure
20	Duplicate Network Address
21	Excessive Following Error
22	Excessive Command Increment
23	Position Register Overflow
24	Resolver Feedback Lost
25	Motor Power Over-Voltage
26	(*3 - 7 Amp*) Motor Power Clamp Excessive Duty Cycle—Under-Voltage (*12–28 Amp*) Motor Power Under-Voltage
27	(*3 & 4 Amp IMJ*) Reserved (*3 & 6 Amp IMC; 7 Amp IMJ*) Motor Power Clamp Over-Current Fault (*12–28 Amp*) Motor Power Clamp Excessive Duty Cycle
28	Motor Over-Current Fault
29	Motor Over-Temperature
30	Controller Over-Temperature
31	Network Communication Error

HTN Network Halt

Class:	Motion Command
Syntax:	HTN*p1* (e.g., HTN0 HTN63 HTNVI5)
Parameters:	*allowed values* *description*
p1	0 through 63 **or** VI*n* network node address
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.
Use:	The network halt command accesses attribute 21 of the position controller object to immediately halt all motion for the axis addressed at *p1*.
Remarks:	This command should be used only at low velocities or in extreme situations because the sudden stop may damage mechanical components in the system.
Related Commands:	HT, ST, STN

IPN Network In Position

Class:	System Register	
Type:	Boolean	
Syntax:	IPN*p1* (e.g., IPN0 IPN63 IPNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
allowed values	0, 1	
Restrictions:	Read only. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	The network in position register accesses attribute 12 of the position controller object to determine whether the axis addressed at *p1* is in position. If the axis is in position, then IPN will be 1; and if the axis is not in position, then IPN will be 0.	
Related Registers:	IP, IPALL, IPB, SRA	

KSN Network Stall Velocity Threshold

Class:	Axis Register	
Type:	Integer	
Syntax:	KSN*p1* (e.g., KSN0 KSN63 KSNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	pulses	
minimum	50,000 pulses	
maximum	16,000,000 pulses	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	The network stall velocity register accesses attribute 111 of the position controller object to set the minimum velocity at which stall detection will start to work.	
Remarks:	The Position Controller attribute that this register accesses exists in GE Fanuc Motor Cubes™ (model numbers CUB-1216, CUB-1221, and CUB-1231) but not in the IMC and IMJ products.	
Related Registers:	KVN	

KVN Network Bus Voltage

Class:	Axis Register	
Type:	Integer	
Syntax:	KVN*p1* (e.g., KVN0 KVN63 KVNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	volts	
minimum	0	
maximum	50	

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: The network bus voltage register accesses attribute 112 of the position controller object to set the bus voltage for the device addressed at *p1*.

Remarks: The Position Controller attribute that this register accesses exists in Stepping Motor Cubes (model numbers CUB-1216, CUB-1221, and CUB-1231) but not in the IMC and IMJ products.

Related Registers: KSN

MACN Network Motion Acceleration

Class:	Motion Register	
Type:	Integer	
Syntax:	MACN*p1* (e.g., MACN0 MACN63 MACNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	pulses/sec^2	
minimum	100 pulses/sec^2	
maximum	1,000,000,000 pulses/sec^2	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	The MACN register accesses attribute 8 of the position controller object to define an acceleration rate for the axis addressed at *p1*. Define the deceleration rate separately with MDCN.	
Related Registers:	MAC, MDC, MDCN	

MDCN Network Motion Deceleration

Class:	Motion Register	
Type:	Integer	
Syntax:	MDCN*p1* (e.g., MDCN0 MDCN63 MDCNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	pulses/sec^2	
minimum	100 pulses/sec^2	
maximum	1,000,000,000 pulses/sec^2	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	The MDCN register accesses attribute 9 of the position controller object to define a deceleration rate for the axis addressed at *p1*.	
Related Registers:	MAC, MACN, MDC	

MPN Network Move Position

Class:	Motion Register	
Type:	Integer	
Syntax:	MPN*p1* (e.g., MPN0 MPN63 MPNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	The MPN register accesses attribute 6 of the position controller object to define the move position of the axis addressed at *p1*.	
Related Registers:	MPI, MPA	

MVLN Network Motion Velocity

Class:	Motion Register	
Type:	Integer	
Syntax:	MVLN*p1* (e.g., MVLN0 MVLN63)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	pulses/sec	
minimum	1 pulse/sec	
maximum	16,000,000 pulses/sec	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This register accesses attribute 7 of the position controller object to define the motion velocity of the axis.	
Related Registers:	MAC, MACN, MVL	

NCO Network Connection Open

Class:	System Register	
Type:	Boolean	
Syntax:	NCO*p1* (NCO0 NCO63 NCOVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
Range:		
allowed values	0, 1	
Restrictions:	Read only. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This register returns a one when network connection is established with the device addressed at *p1*. If a connection cannot be established, then this register returns a zero.	
Related Registers:	NET	
Related Commands:	CNC	

NET Network Connection Available

Class:	System Register
Type:	Boolean
Syntax:	NET
Range:	
allowed values	0,1
Restrictions:	Read only.
Use:	This register is used to determine when a network access can be made.
Related Registers:	NCO, SRS

OUSN Output Command to Network Port with Status

Class: I/O Command

Syntax: OUSN*p1*"*p2*" (e.g., OUSN5"MPA=3")

Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	any valid program command	

Restrictions: Allowed only in programs.

Use: This command outputs a command over the network to be executed by the addressed controller.

Remarks: If the command sent to the addressed controller is not accepted, then bit 8 in the program status register will be set to 1, which means *Invalid Command Acknowledgment*.

Related Commands: OUTN

OUTN Output Command to Network Port

Class:	I/O Command	
Syntax:	OUTN*p1*"*p2*" (e.g., OUTN5"MPA=3")	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	any valid program command	
Use:	This command outputs a command over the network to be executed by the addressed controller.	
Related Commands:	OUSN	

PIPN Network Profile in Progress

Class:	System Register	
Type:	Boolean	
Syntax:	PIPN*p1* (PIPN0 PIPN63 PIPNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
Range:		
allowed values	0, 1	
Restrictions:	Read only. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This register returns a one when a network profile is in progress for the device addressed at *p1*. If a network profile is not in progress, then this register returns a zero.	
Related Registers:	IPN	

PSAN Network Axis Position

Class:	Axis Register	
Type:	Integer	
Syntax:	PSAN*p1* (e.g., PSAN0 PSAN63 PSANVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
units	pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This register accesses attribute 13 of the position controller object to define the actual position of the axis.	

PSCN Network Command Position

Class:	Axis Register	
Type:	Integer	
Syntax:	PSCN*p1* (e.g., PSCN0 PSCN63 PSCNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
Range:		
units	pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	
Restrictions:	Read only. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This register accesses attribute 15 of the position controller object to determine the command position of the axis. The command position is the controller's required position for the axis. The difference between this and the axis position, PSAN, is called the following error, FE.	
Related Registers:	PSA, PSAN, FE	

RDN Network Run Direction Flag

Class:	Motion Register	
Type:	Boolean	
Syntax:	RDN*p1* (RDN0 RDN63 RDNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
allowed values	0, 1	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command accesses attribute 23 of the position controller object to set the direction of motion of the axis addressed at *p1* when the RMN*p1* register is set to 1 (velocity mode). 1 = forward motion; 0 = reverse motion.	
Related Registers:	RMN	
Related Commands:	RPN, RVR, RVF	

REVN Network Device Revision

Class:	Diagnostic Command
Syntax:	REVN*p1* (REVN0 REVN63)
Parameters:	*allowed values* *description*
p1	0 through 63 network node address
Restrictions:	Not allowed in programs or motion blocks. Cannot be accessed in immediate mode over a DeviceNet connection.
Use:	This command accesses attributes 4 and 7 of the identity object to report the model number and firmware revision of the device addressed at *p1*.
Related Commands:	REVISION

RIN Network Run Incremental Flag

Class:	Motion Register	
Type:	Boolean	
Syntax:	RIN*p1* (RIN0 RIN63)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Range:		
allowed values	0, 1	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command accesses attribute 10 of the position controller object to set the absolute or incremental position mode of the axis addressed at *p1*. 0 = absolute position mode; 1 = incremental position mode.	
Related Registers:	RMN	
Related Commands:	RPA, RPI, RPN	

RMN Network Run Mode

Class:	Motion Register	
Type:	Integer	
Syntax:	RMN*p1* (RMN0 RMN63 RMNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 or VIn	network node address
Range:		
allowed values	0, 1, 2	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command accesses attribute 3 of the position controller object to set the motion mode of the axis addressed at *p1*. 0 = position mode; 1 = velocity mode; and 2 = torque mode.	
Related Registers:	RDN, RIN	
Related Commands:	RPA, RPI, RPN, RVF, RVR	

RPN Run Profile of Network Device

Class:	Motion Command
Syntax:	RPN*p1* (RPN0 RPN63)
Parameters:	*allowed values* *description*
p1	0 through 63 **or** VI*n* network node address
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.
Use:	This command accesses attribute 11 of the position controller object to cause the axis addressed at *p1* to perform one of the following functions: Run to absolute position when RMN=0 and RIN=0 Run to incremental position when RMN=0 and RIN=1 Run to velocity forward when RMN=1 and RDN=1 Run to velocity reverse when RMN=1 and RDN=0 Run to torque when RMN = 2
Related Registers:	RIN, RDN, RMN
Related Commands:	RPA, RPI, RVF, RVR

RSFN Reset Network Faults

Class:	System Command
Syntax:	RSFN*p1* (RSFN RSFN0 RSFN63 RSFNVI5)
Parameters:	*allowed values* — *description*
p1	none **or** 0 through 63 — network node address **or** VI*n*
Restrictions:	Not allowed in motion blocks. Cannot be accessed in immediate mode over a DeviceNet connection.
Use:	When *p1* is not specified, this command resets the network faults of the controller. When *p1* is specified, this command accesses attribute 17 of the position controller object to reset faults of the controller addressed at *p1*.
Related Commands:	RSF, RSFS, RSFALL
Related Registers:	FCN, FCNN

SCAN Maximum Scan Time

Class: System Register

Syntax: SCAN

Range:

units	seconds
default	0
minimum	0.00
maximum	1.00

Restrictions: Not allowed in programs, motion blocks, or expressions.

Use: Use to define the maximum time allowed between updates of the I/O connection of the controller. If the I/O connection is not updated in time, then the controller will fault due to *Network Communication Error* and the FCN register will have bit 11 set (*I/O Scan Time-Out*). If SCAN is set to zero, then no check of the update time is performed.

Example:

```
SCAN=.05        (* set maximum scan time to 50 milliseconds)
```

SNI Scanned Network Input

Class: I/O Register

Type: Integer, Boolean

Syntax: SNI*p1.p2*

Parameters:

	allowed values	*description*
p1	1 through 4 **or** VI*n*	scanned register number
p2	none **or** 1 through 32 **or** VI*n*	digital input number

Range:

allowed values	0 through $FFFFFFFF_{16}$ or 0 and 1

Restrictions: Read only.

Use: The scanned network input register contains the values of the network digital inputs of the node at the address assigned by SNIA. The digital inputs are general-purpose inputs used for process control.

Remarks: The controller cannot read the network inputs using this register unless the node is in a scan list of a scanner installed on the same network. This register allows the controller to read network inputs without additional network traffic. The controller looks at the message sent over the network from the I/O device to the scanner module. Only nonfragmented I/O messages can be monitored, and only the first four bytes of the I/O message are reported.

Examples:

```
SNIA1=24          (* set node address for scanned input one to 24)
SNI1?             (* report value of scanned inputs)
*16#135B60F7
```

SNIA Scanned Network Input Address

Class:	I/O Register	
Type:	Integer	
Syntax:	SNIA*p1*	
Parameters:	*allowed values*	*description*
p1	1 through 4 **or** VI*n*	scanned register number
Range:		
default	0	
allowed values	0 through 63 **or** VIn	
Use:	This register is used to assign the address of the node whose inputs are scanned for the scanned register *p1*.	
Example:		
SNIA1=24	(* set node address for scanned input one to 24)	

STFN Network Set Fault

Class:	System Command	
Syntax:	STFN*p1* (STFN0 STFN63 STFNVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network node address
Restrictions:	Not allowed in motion blocks. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command accesses attribute 17 of the position controller object to fault the controller addressed at *p1*.	
Related Commands:	RSF, RSFN, STF	
Related Registers:	FC, FCNN	

STN Network Stop

Class:	Motion Command
Syntax:	STN*p1* (e.g., STN0 STN63 STNVI5)
Parameters:	*allowed values* *description*
p1	0 through 63 **or** Vi*n* network node address
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.
Use:	The network stop command accesses attribute 20 of the position controller object to stop all motion for the axis addressed at *p1*.
Remarks:	This command, once executed, will immediately decelerate the axis at the deceleration loaded.
Related Commands:	HT, HTN, ST

VBN Network Boolean Variable

Class:	Variable Register	
Type:	Boolean	
Syntax:	VBN*p1.p2* (e.g., VBN1.1 VBNVI2.VI5)	
Parameters:	*allowed values*	*description*
p1	1 through 63 **or** VI*n*	network address
p2	1 through 256 **or** VI*n*	Boolean variable number
Range:		
allowed values	0, 1	
Restrictions:	Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	Network Boolean variables are used mainly in conditional statements of programs, such as IF...GOTO (conditional goto) and WAIT (wait for expression to be true). They can also be used to load register values.	
Examples:		
VBN5.1=VI1>0	(* set network Boolean variable one of controller addressed at five to 1 if integer variable one is greater than zero)	
VBNVI20.3=VB1	(* set network Boolean variable three of controller addressed at VI20 to 1)	
VBN3.VI2=VI1<5	(* set network Boolean variable VI2 of controller addressed at three to 1 if integer variable one is less than five)	
VBN3.VI2?	(* report network Boolean variable VI2 of controller addressed at three)	

VFN Network Floating Point Variable

Class: Variable Register

Type: Floating point

Syntax: VFN*p1.p2* (e.g., VFN1.1 VFNVI2.VI5)

Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	1 through 2,048 **or** VI*n*	floating point variable number

Range:

minimum	1.5×10^{-39} (absolute value)
maximum	1.7×10^{38} (absolute value)

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: Network floating point variables are used in variable expressions and to load register values.

Remarks:

1. The numerical value for the maximum of parameter *p2* shown above is assuming that the floating point variable allocation, VFA, of the addressed controller is set to 2,048. If VFA is set to a value other than 2,048, the maximum of *p2* will change.

2. In order to access the extended floating point variables, the indirect addressing scheme (i.e., VFN*n*.VI*n*) must be used.

Examples:

VFN5.1=5.776	(* set network floating point variable one of controller addressed at five to 5.776)
VI1=3500	(* set integer variable to 3,500)
VFVI1=SQR(2.*VFN5.1)	(* set floating point variable VI1 [i.e., 3,500] to square root of 2 times 5.776)
VFNVI20.2=PSA/5.	(* set network floating point variable two of controller addressed at VI20 to axis position divided by 5)
VFN3.VI1?	(* report network floating point variable VI1 of controller addressed at three)

Related Registers: VFA, VFEA, VI

VIN Network Integer Variable

Class: Variable Register

Type: Integer

Syntax: VIN*p1.p2* (e.g., VIN1.1, VINVI2.VI5)

Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	1 through 4,096 **or** VI*n*	integer variable number

Range:

minimum	-2,147,483,648
maximum	2,147,483,647

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: Network integer variables are used in variable expressions and to load register values.

Remarks:

1. The numerical value for the maximum of parameter *p2* shown above is assuming that the floating point variable allocation, VFA, of the address controller is set to 0. If VFA is set to a value other than 0, the maximum of *p2* will change.

2. In order to access the extended integer variables, the indirect addressing scheme (i.e., VIN*n*.VI*n*) must be used.

Examples:

VI1=5000	(* set integer variable one to 5,000)
VIN5.2=-330	(* set network integer variable two of controller addressed at five to -330)
VIN2.VI1=VI1+VIN5.2	(* set network integer variable VI1 [i.e., integer variable 5,000) of controller addressed at two to 5,000 plus -330]
VINVI20.3=PSR*2	(* set network integer variable three to PSR times 2, [i.e., resolver position times 2])
VINVI20.3?	(* report network integer variable three of controller addressed at VI20)

Related Registers: VFA, VFEA, VI

VLAN Network Axis Velocity

Class:	Axis Register	
Type:	Integer	
Syntax:	VLAN*p1* (e.g., VLAN0 VLAN63 VLANVI5)	
Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
Range:		
units	pulses/sec	
minimum	-16,000,000 pulses/sec	
maximum	16,000,000 pulses/sec	
Restrictions:	Read only. Cannot be accessed in immediate mode over a DeviceNet connection.	
Use:	This command accesses attribute 14 of the position controller object to determine the actual velocity of the axis.	
Related Registers:	VLA	

VSN Network String Variable

Class: Variable Register

Type: String

Syntax: VSN*p1.p2* (e.g., VSN1.1, VSNVI2.VI5)

Parameters:	*allowed values*	*description*
p1	0 through 63 **or** VI*n*	network address
p2	1 through 144 **or** VI*n*	string variable number

Range:

allowed values any string, 0 through 127 characters long, enclosed in quotes

Restrictions: Cannot be accessed in immediate mode over a DeviceNet connection.

Use: Network string variables are used mainly to load strings and in input/output commands such as GET, PUT, IN, and OUT as a means of user interface.

Remarks: If the extended memory card is available, then *p2* can be up to 272 for a Target controller.

Examples:

VSN5.1=“$20”+“$R”	(* set network string variable one of controller addressed at five to a space followed by a carriage return)
VI1=2	(* set integer variable one to 2)
VSN3.VI1=“Done”	(* set network string variable VI1 [i.e., string variable two] of controller +VSN5.1 addressed at three to “Done” followed by a space and a carriage return)
VSNVI20.2?	(* report network string variable two of controller addressed at VI20)

Related Commands: GET, PUT, IN, OUT, VS

Appendix B

Generation D RTOS DeviceNet Fault and Status Messages

How to Read Generation D RTOS Fault and Status Messages

Register fault and status messages give users an easy way to diagnose any problems in their systems. To query any of the registers presented herein, establish communication with your controller from the CCS Terminal window. Then type *reg*? and press the <Enter> key. The controller will respond with the appropriate message.

DspMotion controllers will return command fault and status messages any time the operator tries to perform a task that is not allowed. Command messages also indicate the status of operations and prompt the user for any required action.

This appendix contains only those Generation D RTOS command and register fault and status messages that apply to DeviceNet systems. This section begins with a list of the command messages. Register messages follow in alphabetical order. Messages apply to all DspMotion controllers unless otherwise indicated. To find non-DeviceNet fault and status messages, please turn to *The Generation D RTOS Programming Manual*, Appendices D, E, and F.

Command Messages

Number	Command Message	Possible Cause(s)	Possible Solution(s)
7	NETWORK ADDRESS OUT OF RANGE	The network address entered is less than 0 or greater than 63.	Re-enter the address making sure that it is a number 0 through 63.
28	RESOURCE NOT AVAILABLE	The addressed network controller is not online or had a problem.	Check network connections. Check network address and baud rate. Check FCN for more information.
		◦ An attempt was made to execute a command specific to a Target module, but the module is not available.	◦ Check system configuration.

FC -- *IMC* and *IMCjr* System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
5	Invalid Command in String	The program attempted to execute the EXVS command, but the command stored in the string variable was not recognized by the system. The program attempted to execute the OUTN command, but the command sent over the network was not recognized by the recipient.	The command is misspelled. Re-enter the correctly spelled command in the program editor. The command is invalid. Re-enter the valid command in the program editor.
7	Resource Not Available	The addressed network controller is not online or had a problem.	Check network connections. Check network address and baud rate. Check FCN for more information.
19	Network Power Failure	The network connector is disconnected, or the network power is below the minimum voltage.	Reconnect the network connector. Inspect the network power source and replace if required.
20	Duplicate Network Address	More than one device at the same MAC ID.	Assign each device a unique address.
31	Network Communication Error	Network is not properly configured	Check network configuration

FCN -- Network Fault Code Register			
Bit	**Network Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
0	Network Off-line	Network cable is disconnected or there is no power to the network. No network card. No other node on network.	Connect network cable. Examine network for proper power distribution.
1	Addressed Device Not Present	Device not present on network or address not correct for intended device.	Connect device to network. Change address to match device address.
2	Addressed Device Out of Connections	Maximum number of server connections presently open.	One or more client devices must delete connection to this server.
3	Connection Deleted Unexpectedly	Connection deleted by transmission error or timeout.	Examine network cable for proper connections and terminations. Examine device for proper connection and operation.
4	Time-out On Response	Server connection deleted or server timed out (250 ms).	Examine transmission media and server. Examine network cable for proper connections and terminations.
5	Not Requested Response	Server transmission error.	Examine transmission media and server. Examine network cable for proper connections and terminations.
6	Error Response	Server error response.	Verify object/instance/attribute/service exists for device addressed.
7	Resource Unavailable	Server device doesn't have required resource. May be caused by getting or setting a variable not allocated in server.	Make certain that server has resources available.
8	Not Enough Data	Mismatched data type between client and server.	Ensure client and server data types match.
9	Too Much Data	Mismatched data type between client and server.	Ensure client and server data types match.
10	Device State Conflict	Controller is in program mode.	Exit the line editor.
11	I/O Scan Timeout	Controller or digital inputs not scanned within specified interval.	Increase allotted scan time in SCAN register. Update scan time in scanner.

FCN -- Network Fault Code Register (continued)			
Bit	**Network Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
12	Invalid Attribute Value	Attribute value entered is out of range.	Enter a value within the allowed range. Check attribute values supported by the addressed device.
13	Attribute Not Supported	Attribute is not supported by this object.	Check attributes supported by the addressed device.
14	Object Does Not Exist	Object is not supported by this node.	Check objects supported by the addressed device.
15	Reserved		
All bits set to 0	Network functional	The network is not faulted.	Continue with normal operation.

FCS -- *Target*® System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
7	Resource Not Available	An attempt was made to execute a command specific to a module, but the module is not available. The addressed network controller is not online.	Double check the configuration of the system. Check the network connections, network address, and baud rate.
18	Duplicate Network Address	More than one device at the same MAC ID.	Assign each device a unique address.
19	Network Power Failure	The network connector is disconnected, or the network power is below the minimum voltage.	Reconnect the network connector. Inspect the network power source and replace if required.
27	Network Communication Error	Network is not configured properly.	Check network configuration.

FI -- *IMC* and *IMCjr* System Fault Input Register		
Bit	**Fault Input Message**	**Possible Solution(s)**
7	Network power failure input active	The network is disconnected or the network power source is below the minimum voltage.

IOS -- *Target*® System I/O Register		
Bit	**System I/O Message**	**Description**
12	Network power failure input active	The network is disconnected, or the network power source is below the minimum voltage.

SRP -- Program Status Register		
Bit	**Program Status Message**	**Description**
8	Invalid command acknowledgment	The OUSN command was executed, and the responding device didn't accept the command as valid.

SRS -- System Status Register		
Bit	**System Status Message**	**Description**
6	Network connection available	There is a connection available for communication.
7	Network on-line	The network is ready to communicate.

Appendix C ASCII Codes

Use the following chart as your guide to decimal-hex-character conversions for your DeviceNet system.

Decimal	Hex	Character
0	00	ctrl@ NUL
1	01	ctrlA SOH
2	02	CtrlB STX
3	03	CtrlC ETX
4	04	CtrlD EOT
5	05	CtrlE ENQ
6	06	CtrlF ACK
7	07	CtrlG BEL
8	08	CtrlH BS
9	09	CtrlI HT
10	0A	CtrlJ LF
11	0B	CtrlK VT
12	0C	CtrlL FF
13	0D	CtrlM CR
14	0E	CtrlN SO
15	0F	CtrlO SI
16	10	CtrlP DLE
17	11	CtrlQ DC1
18	12	CtrlR DC2
19	13	CtrlS DC3
20	14	CtrlT DC4
21	15	CtrlU NAK
22	16	CtrlV SYN
23	17	CtrlW ETB
24	18	CtrlX CAN
25	19	CtrlY EM
26	1A	CtrlZ SUB
27	1B	Ctrl[ESC
28	1C	Ctrl\ FS
29	1D	Ctrl] GS
30	1E	Ctrl^ RS
31	1F	Ctrl_ US
32	20	Space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*

Decimal	Hex	Character
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U

Decimal	Hex	Character
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	I
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	-
127	7F	DEL

S

T

U