2716D DeviceNet Module Installation Guide

Doc. No. 2716DIG
Revision A, October 2001

Contents

Notes to Readers

The *2716D DeviceNet Module Installation Guide* provides the following information:

- Installing the DeviceNet module.
- Connecting the module to a DeviceNet network.
- Configuring computer-controller communications.
- DeviceNet concepts such as scanning, data mapping, and objects.
- Software configuration -- BOI, Register Objects, and Register Set Objects.
- Messaging -- I/O and Explicit messaging and message fragmentation.
- Special DeviceNet registers -- how they function and how to use them.
- Troubleshooting -- how to diagnose problems by using LED status indicators, error codes, and diagnostic software.

Related Documents

The following documents contain additional information:

- For information on Quickstep, refer to the *Quickstep™ Language and Programming Guide* or the *Quickstep™ User Guide*.
- For information on your controller and its modules, refer to the appropriate Installation and Applications Guide.

- For information on the registers in your controller, refer to the *Register Reference Guide* (available at www.ctc-control.com).
- For information on the DeviceNet Configurator, refer to the *DeviceNet Configurator User Guide*.
- For information on Microsoft Windows or your PC, refer to the manuals provided by the vendor.

Book Conventions

The following conventions are used in this book:

ALL CAPS BOLDFACE	Identifies DOS, Windows, installation program file names.
Boldface	Indicates information you must enter, an action you must perform, or a selection you can make on a dialog box or menu.
Italics	Indicates a word requiring an appropriate substitution. For example, replace *filename* with an actual file name.
Text_Connected_With_Underlines	Indicates symbolic names used in Quickstep programs. Step Names are ALL_CAPITALS. Other symbolic names can be Initial_Capitals or lower_case.
SMALL CAPS	Identifies the names of Quickstep instructions in text.
`Courier font`	Identifies step names, comments, output changes, and Quickstep instructions appearing in the Quickstep editor.
Art Code - DN-24	Identifies the file name of a particular graphic image.

How to Contact Control Technology Corporation

Control Technology Corporation is located in Massachusetts, and we are open from 8:30 a.m. to 5:00 p.m. eastern time. Contact us at 1-508-435-9595 and 1-800-282-5008 or FAX 1-508-435-2373.

See us on the World Wide Web at www.ctc-control.com.

Your Comments

We welcome your suggestions and comments about this or any other Control Tech document. Comment forms are in the file called **BUGRPT.WRI**, which was installed in the **QSWIN21** directory during your Quickstep installation. You can also email comments to techpubs@control.com.

Chapter 1

Getting Started

Contents

System Description

Overview

The model 2716D DeviceNet Module adds DeviceNet network support to Control Technology's automation controllers. DeviceNet is a low-cost, open network standard that provides for reduced system complexity and significant reductions in wiring costs. DeviceNet allows different industrial devices such as a CTC controller and devices from other manufacturers (sensors, actuators, et al.) to work together on a single network. DeviceNet may also provide communications links between subsystems or system-level components. The results are improved control communications between devices and important device-level diagnostics. The 2716D can be configured as a DeviceNet master, DeviceNet slave, or as both master and slave on the same network.

Modes of Operation

In master mode, with the 2716D installed in a 2700 series automation controller, you can use Control Technology's DeviceNet Configurator software to create a network configuration and load it into the 2716D. The 2716D master module then establishes links to each device on the network and maps the device's I/O points and other resources locally for program access using Quickstep™.

The Configurator also has a monitor mode that identifies and interrogates any device on a DeviceNet network through the 2716D. This mode is especially useful when a device's documentation and/or electronic data sheet (EDS) are not readily available. Monitor mode can establish links, execute link commands, send and receive data, and generate a network traffic log.

In slave mode, with the 2716D located in any 2600 or 2700 series automation controller, you can map the I/O points and other controller resources to any number of commercially available DeviceNet master (or scanner) systems. The 2716D supports Bit-Strobe, Poll, Change-of-State (COS), Cyclic, and Explicit messaging. Baud rates (125K, 250K, and 500K) and node numbers can be changed with simple, on-board switches.

Flexible Architecture

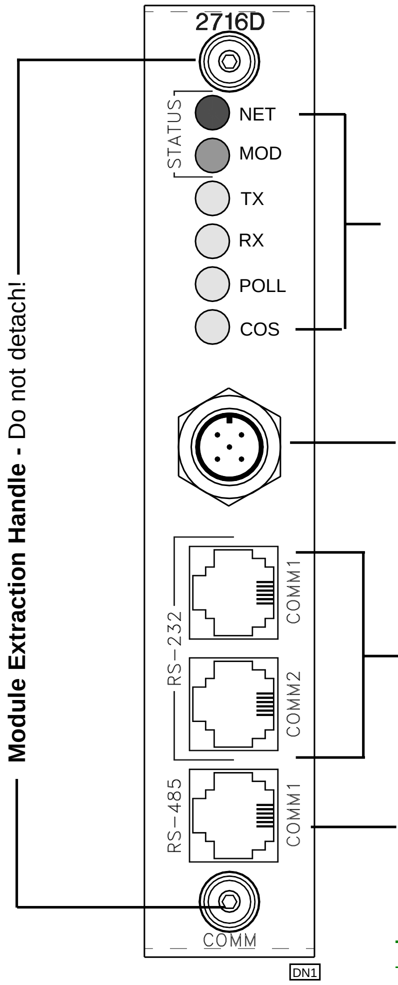
The 2716D DeviceNet module provides two RS-232 serial communication ports that support all current CTC protocols. Any of the controller's internal registers, flags, and other resources may be monitored or changed. Programming is accomplished with either of the serial ports. In addition, the module contains an RS-485 port that you can access by changing a jumper on the module's circuit board. This port replaces the module's first RS-232 port. Transmissions are limited to half-duplex mode (one direction at a time) and communication with multiple 2716D modules through a single port is not supported. These ports are opto-isolated from the controller's logic circuitry and from its I/O power system.

NOTE: You must use the 2716D's serial ports to communicate with the DeviceNet Configurator program. The 2716D currently cannot be programmed with any other communications ports.

Local CPU for Data Handling

The 2716D is equipped with a 32-bit processor, allowing operation of the DeviceNet network and both serial ports at full rated speed without encumbering the controller's CPU. Complete messages are assembled locally on the 2716D module and are then passed to the controller's processor for servicing.

LEDs and Communications Terminals

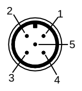


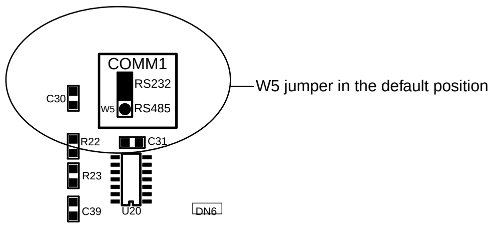
LED Status Indicators - Display status in red or green of DeviceNet, network, transmission and receipt of messages, and poll and change of state indicators between the module and the controller. Refer to the *LED Troubleshooting Tables* in *Chapter 4*, *Troubleshooting,* for more information.

DeviceNet connector - Links the module and controller to the DeviceNet network. The square notch is closest to pins 1 and 2. Refer to the pinout diagram below for more information.

RS-232 connectors - Use the same modular jack as the controller's on-board port. These connections are compatible with CTC's 2881-2883 communications cables as well as the cable supplied with your Quickstep software.

RS-485 connector - Configured for half-duplex operation; becomes active when the W5 jumper is in the RS-485 position. The port then becomes the COMM1 port on the module.

DeviceNet Connector	Pin #	Signal	Function	Wire Color
	1	Drain	Shield	Bare
	2	V+	Power Supply	Red
	3	V-	Common	Black
	4	CAN_H	Signal High	White
	5	CAN_L	Signal Low	Blue



2716D Specifications

Description	*Min.*	*Typical*	*Max.*	*Units*
Absolute Maximum Ratings				
Current draw from on-board +5V supply			250	mA DC
Ambient Temperature:				
Operating	0		+50	°C
Storage	-20		+80	°C
RS-232 Operating Characteristics				
RS-232 Transmitters		± 9	± 12	VDC
RS-232 Receivers	± 3		± 12	VDC
Common Mode Voltage Range	-10.0		+10.0	VDC
RS-485 Operating Characteristics				
RS-485 Common Mode Rejection	-7		+12	VDC
RS-485 Hysteresis		70		mVDC
Note: Combined impedance is less than one RS-485 load, up to 32 devices on a bus				
Power supply requirements (from controller)				
Logic Supply (5 volt from controller)		185	250	mA
Auxiliary Supply (24 volt from 24 volt bus)		85	170	mA
DeviceNet Power	11	24	28	VDC
DeviceNet Load		100	150	mA
DeviceNet Miswiring Protection (on any DeviceNet connection)		24		VDC

Hardware/Firmware Revision Levels

For master operation:

Model Number	Hardware Revision Level	Firmware Revision Level[2]
2716D Module	A	1.20
2701E CPU	A	2.25

For slave operation:

Model Number	Hardware Revision Level	Firmware Revision Level[2]
2716D Module	A	1.20
2701E CPU	A	2.25
2600XM Controller	C	2.0

NOTES:

1. You can confirm firmware revision levels by doing a register read in Quickstep's monitor program. For the 2716D, use register 13490, and for the 2600XM/2701E, use register 13003.

2. Firmware revision levels are not equivalent to standard decimal numbers. For example, firmware revision level 1.4 translates to:

Major Revision Level 1
Minor Revision Level 4

If this value changes to 1.20, it translates to:

Major Revision Level 1
Minor Revision Level 20

Board Handling Precautions

⚠ The module’s printed circuit board contains electrostatic discharge sensitive (ESD) devices. Improper handling could result in damage to the board. The following precautions are recommended when handling the board or before inserting it into the controller:

- Make sure you are grounded electrically by either using a wrist strap connected to an electrically grounded workstation or by physically touching the controller case or something electrically connected to the controller case.
- Avoid touching the leads or contacts of the circuit board and handle the board by its edges only.
- Transport circuit boards in protective, anti-static bags, bins or totes. Do not insert boards into materials such as plastic, polystyrene foam, clear plastic bags, bubble wrap, or plastic trays.

DeviceNet Hardware Configuration

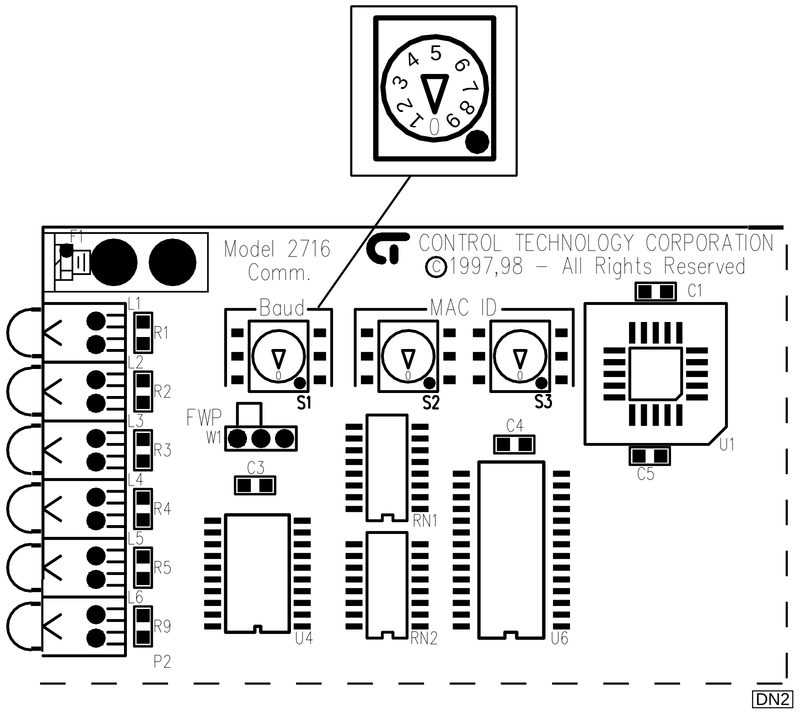
This section desribes how to set the MACID and baud rate by adjusting the DIP switches on the module's printed circuit board (PCB).

NOTES: The default baud rate is 125 kBd and the default MACID is 63. If these values are acceptable, skip this section and proceed to *Installing the 2716D Module*.

The baud rate and MACID can also be set with configuration files. You can accomplish this by sending explicit messages over the DeviceNet network with a master device or through the 2716D's DeviceNet special registers. In case your controller's firmware is not version 2.25, you can use the module's serial port and CTCMon to access these registers.

Setting the Baud Rate

1. Locate the baud rate switch (S1) on the module's PCB. Refer to the illustration below.



2. Rotate S1 with a screwdriver until the correct number is indicated. Refer to the table below for switch settings and corresponding baud rates.

NOTE: 500 Kbd is the highest valid baud rate recognized by a DeviceNet network. If an invalid baud rate is specified, then the baud rate stored in non-volatile memory is used. If this stored rate is also invalid, then the default value of 125 kBd is used.

Switch Number	Baud Rate
0	125 kBd
1	250 kBd
2	500 kBd
3	Value stored in non-volatile memory
4	125 kBd
5	250 kBd
6	500 kBd
7	Value stored in non-volatile memory
8	125 kBd
9	250 kBd

Setting the MACID

1. Locate the two MACID switches (S2 and S3) on the module's PCB. Refer to the illustration on the previous page.

NOTES: S2 sets the tens digit and S3 sets the ones digit. For example, if you set S2 to 5 and S3 to 4, then your MACID is 54.

63 is the highest valid MACID recognized by a DeviceNet network. If you set the tens digit to 6, then the ones digit is limited to a range of 0 to 3.

If an invalid MACID is specified, then the ID stored in non-volatile memory is used. If this stored ID is also invalid, then a default value of 63 is used.

2. Set the MACID's tens digit to a value between 0 and 6 by rotating S2 with a screwdriver.

3. Set the MACID's ones digit to a value between 0 and 9 by rotating S3 with a screwdriver.

Installing the 2716D Module

The module fits into one of the slots of your automation controller. You can insert any combination of modules into the controller (subject to system limits) and can install them in any order. This is possible because the controller's CPU dynamically assigns such items as motor numbers, input numbers, and output numbers each time power is re-applied to the controller. These numbers are assigned from left-to-right across the controller.

NOTE: CTC recommends that you mount the 2716D module into the right-hand most slot of your controller and assign I/O addresses that do not conflict with local I/O assignments. For example, suppose you have a 2203 (16 inputs, 16 outputs) module installed in your controller as input number 8. When the controller powers up, it creates a resource map of all the installed modules. This map allows Quickstep to know what its resources are as well as how to address them. The controller only checks DeviceNet I/O assignments if it hasn't already found these addresses on another board. Therefore, since input 8 is already assigned to the 2203 module, it will be masked by the local I/O map and not added to the resource map.

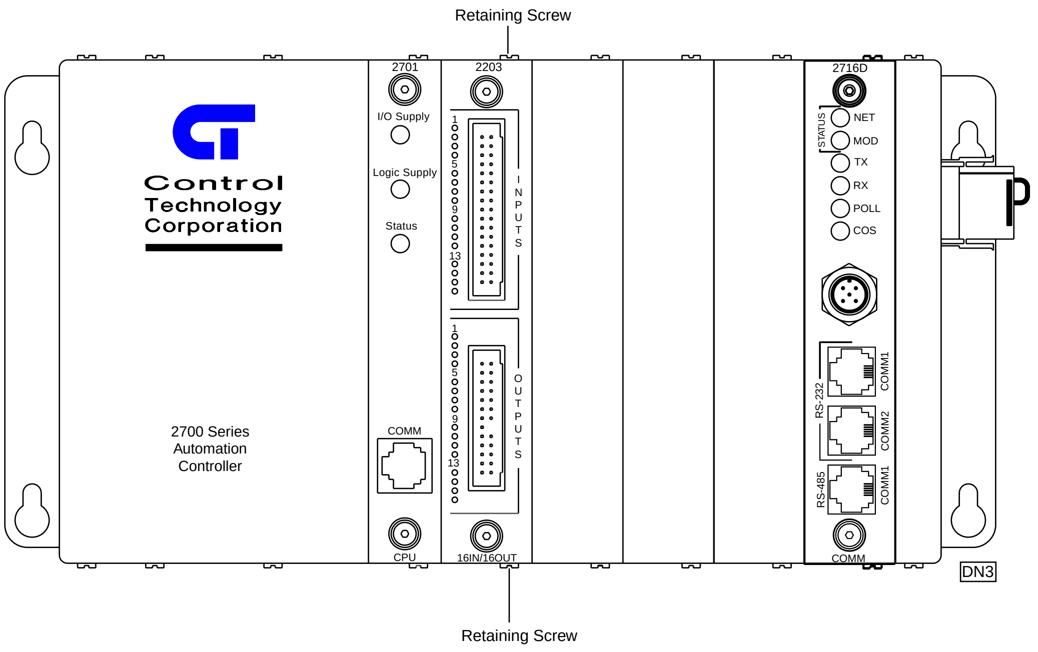
To install a module into the automation controller:

NOTE: Retain all hardware removed during this procedure.

1. Remove all AC and DC power, including any external supplies connected to the controller.
2. Locate an unused slot and remove two retaining screws from the top and bottom of its cover plate.
3. Slide the module into the slot and make sure that the circuit board slides into the nylon guides at the top and bottom of the controller case. Make sure that the card is oriented properly so that its labels are right-side-up.

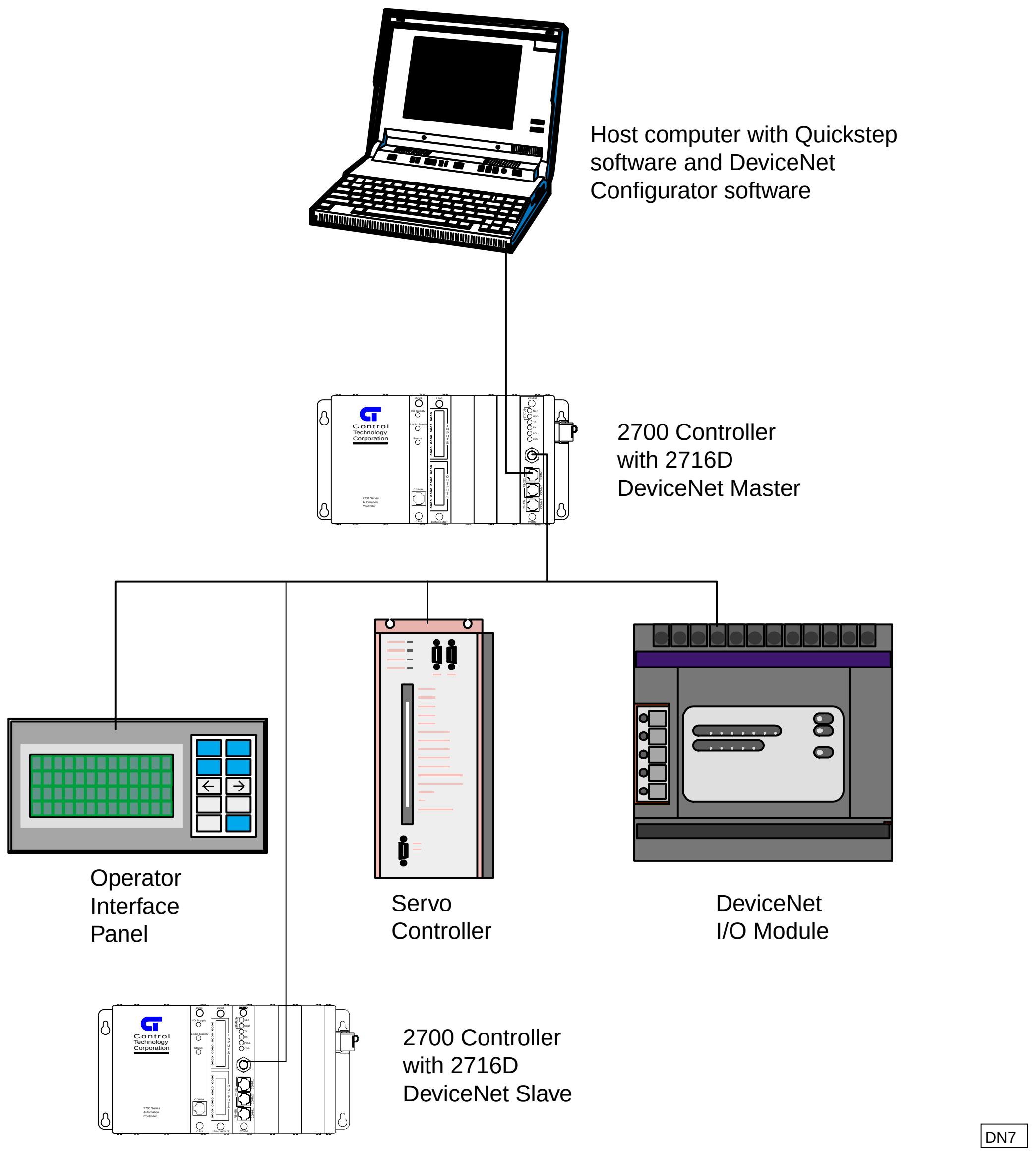
4. Press the module firmly into the controller. Make sure that the module's faceplate is flush with the adjacent sheet metal surface.
5. Re-install two retaining screws in the top and bottom of the new module.

Shown with the 2716D module installed in the right hand slot



Connecting the 2716D to a Network

After the module is installed in the controller slot, you are ready to connect it to a DeviceNet network. The illustration below shows a typical DeviceNet network using the 2716D as both a master and a slave device.


Host computer with Quickstep software and DeviceNet Configurator software
2700 Controller with 2716D DeviceNet Master
Operator Interface Panel
Servo Controller
DeviceNet I/O Module
2700 Controller with 2716D DeviceNet Slave
DN7

General Setup Information

Before you do any configuration or make any final connections to your network, you should consider the following steps:

1. Install a test DeviceNet network. If this is the first time you have used certain devices, you will probably want to create a test network with at least one of each device. This will allow you to test your scanner configuration and determine how you will configure each device. Once you have tested each device, you can replicate much of the configuration information for additional devices. When establishing your test network, ensure that the physical topology meets the DeviceNet specifications, including installation of required termination resistors.
2. Perform operational testing. Initial testing may involve only a few devices and a test network. During this phase of testing you will want to ensure that the scanner is properly connected to the devices and that the device data is mapped properly. Advanced testing will include all nodes on the network. During this phase you will ensure that network loading and response time meet the application requirements. Obviously, you will also be testing the application logic and system operation.

Configuring Master Devices

This section provides a general overview of the required steps for connecting a master device such as the 2716D to the DeviceNet network. If the network is more complex or you have sophisticated tools for configuring master devices, then some of the steps for configuring master devices may already be incorporated into the master device's configuration.

1. Set the baud rate and MACID. **For the 2716D, follow the instructions in the *DeviceNet Hardware Configuration* section in this chapter**. For other devices, refer to the device's documentation. Make sure that the slave device has a unique MACID and that its baud rate matches all other devices on the network.
2. Select a software tool to configure your devices. This program specifies which nodes will be scanned, how the devices will be accessed, and where DeviceNet data will be mapped within controller memory. **If you are using the 2716D as the master (scanner) device, you must use our DeviceNet Configurator program. The DeviceNet Configurator is expressly designed to work with the 2716D as a master device and will not work with other master devices**. For more information, refer to the *DeviceNet Configurator User Guide*.
3. Identify the DeviceNet devices that will be on your network. Each device vendor should provide you with information about the device's DeviceNet capabilities. This may take the form of written data or an electronic data sheet (.EDS file). You need to know the following device characteristics if you expect to have the 2716D (acting as master) verify them before connecting:
   - **Device Type -** The device type is a number code assigned by the ODVA.
   - **Device Vendor -** The device vendor is an ID assigned by the ODVA.
   - **Product Code -** The product code, which is assigned by the vendor, is a unique number that specifically identifies a type or model of device. For the 2716D, the vendor code is 2716.

- **Revision Level -** This is the revision level for a device's firmware.
- **Serial Number -** This is a product's serial number, but more importantly, the ODVA requires that a product's serial number **and** its device vendor ID be unique. These two numbers are used during the duplicate MACID detection routine to make sure there is a way to uniquely identify any device on the bus regardless of its MACID setting.

NOTE: Be certain that the .EDS file's version number or revision matches your actual device before using the file. If the revision number is more recent than your physical device, it may contain parameters that aren't supported by the device.

4. Choose the master device that will be responsible for sending configuration messages. **Some scanners such as the 2716D are equipped to send configuration messages when the system is initialized**. However, other master devices may need to have a separate network configuration module to send out the messages that configure the network for a particular operation or to provide access to specific information.
5. Determine the data format (I/O Assembly) that will be used by each device and decide how data will be mapped to the controller. The documentation for each device will indicate the format of the I/O messages it sends (produces) and receives (consumes). Many devices will allow you to select among several data formats. These selectable formats are called I/O assemblies. The particular I/O assembly that a device uses is established when the device is configured. Locate the required information in the module's DeviceNet Objects; information may exist in multiple locations.
If it exists in an Assembly Object, it always appears in another object.

   If you're using the 2716D as a master, you also need to determine if it can gather data from various locations within the device for sending messages and whether it allows incoming information to be distributed in the same way. Devices that are capable of extracting information from or sending information to any sequence of registers require additional configuration. The master also needs to know the format of the data such as whether it is an 8-bit decimal value or 8 digital inputs.

NOTES: Some devices may have various types of I/O modules attached to them. The type of module, a module's physical location, and the order in which it is attached to a device are all factors that can affect the format of the data.

Multiple sets of data may be required by different network functions. This includes information needed for normal operation (usually needed by a scanner) as well as any data required by any other master devices.

6. Determine the message types (Poll, Bit Strobe, COS/Cyclic) supported by each device that are available to transmit information. Before choosing any specific message types, determine your timing requirements (scan time) and how often you require updated information.

   If you are using the 2716D as a master, you also need to list the messages that are needed to gather information from various slave devices and to set values in the slave device Objects. Examine each message and determine what parts of the data are needed and whether to set any remaining data or ignore it.

7. Configure the DeviceNet devices. For each device being used you will need to set its MACID and baud rate. Every device must have a unique MACID. All baud rates must be the same. Some devices may use switches to set these parameters; others will require either a configuration program or configuration messages generated by the master device. Some devices will require additional configuration, such as setting the I/O assemblies used or setting up that device so it produces the information expected in steps 5 and 6.

   If you are using the 2716D as a master device, then you need to program the controller that is using the scanner data with configuration information specific to that scanner.

NOTE: Other master devices such as human-machine interfaces (HMIs). HMIs require you to program the display of information (graphics, text, colors, etc.) or even to program multiple pages of information and logs for trend and historical displays for more complex HMIs. Supervisory systems that are used as masters require acceptable warning and error limits and may possibly need diagnostic and troubleshooting sequences.

8. Select your cabling and power supply. Decide on the number of networks required and the number of nodes on each network. You need a separate 2716D module used as a scanner for each network and there can be multiple 2716D slaves on a single network. You also need to connect to each DeviceNet network through the DeviceNet connector on the 2716D's front panel. Finally, to configure the 2716D module, you need a serial cablem (CTC models 2880B and 2881 are recommended) to connect to a PC.

9. Set up the I/O messaging channel. Only one master device (usually a scanner) can receive I/O messages, so you must specify a device to receive this data. If other master devices need access to the same information, they must be capable of using Explicit messages. If not, you can configure the scanner so it transmits information to another module or even another master module that operates as a slave to the I/O configured scanner.

10. Determine your Explicit Messaging requirements. There might be areas on the network that can only be controlled with Explicit messages.

11. Use the DeviceNet Configurator to specify your Explicit messages. CTC recommends that you only work with one device at a time. Enter the required information for a device and test it on the network. If it works properly, save the configuration file and proceed to the next device.

NOTES: The network speed (125, 250 or 500 kBd) and scan time limit the amount of data you can include in messages for the scan. Cyclic/COS messages may allow the scanner to recognize infrequent changes without requiring a return message to the module during a scan.

You must also allow adequate idle time on the network so that infrequent Cyclic/COS messages can be processed along with any other messages needed by other functions on the network (HMI and Supervisory).

12. Download your configuration file to your master device. **If you are using the 2716D as a master**, you must use the DeviceNet Configurator to download the file. Make sure the MACID in your configuration file matches the 2716D's hardware settings.

Disabling Master Mode

To operate the 2716D as a slave, you need to disable master mode. You can accomplish this by downloading information to the 2716D with the DeviceNet Configurator in the following ways:

- Download a file without any slave devices
- Download a file where the MACID is different from the module’s hardware settings.

NOTE: Refer to the *DeviceNet Configurator User Guide* for more information on downloading data and using the Configurator.

Configuring Slave Devices

This section provides a general overview of the required steps for connecting a slave device such as the 2716D to the DeviceNet network.

1. Set the baud rate and MACID. **For the 2716D, follow the instructions in the *DeviceNet Hardware Configuration* section in this chapter**. For other devices, refer to the device's documentation. Make sure that the slave device has a unique MACID and that its baud rate matches all other devices on the network.

NOTES: Make sure the 2716D is in slave mode. Refer to *Disabling Master Mode* for instructions on disabling master mode and enabling slave mode.

Some devices use switches to set baud rate and MACID; others require either a configuration program or configuration messages generated by the master device. Some devices may require additional configuration, such as setting the I/O assemblies used.

2. Choose the master device that will be responsible for sending configuration messages. **Scanners such as the 2716D** are equipped to send configuration messages when the system is initialized. However, other master devices may need to have a separate network configuration module to send out the messages that configure the network for a particular operation or to provide access to specific information.
3. Select a software tool to configure your devices. Explicit messages are used to specify the required register objects (see step 4 below) and scan time of the slave device. **If you are using the 2716D as a master and have a second 2716D as a slave on the same network**, you should use CTC's DeviceNet Configurator program for this purpose. For more information, refer to the *DeviceNet Configurator User Guide*.

NOTES: **If you are using the 2716D as a slave**, you will need its .EDS file, which ships with the module and can also be downloaded from www.control.com/devicenet. If you are using other devices as slaves, you can obtain their EDS files from the ODVA at www.odva.org or from the manufacturer of the device.

Be certain that the .EDS file’s version number or revision matches your actual device before using the file. If the revision number is more recent than your physical device, it may contain parameters that aren’t supported by the device.

4. Determine the data format (I/O Assembly) used by the slave device and decide how data will be mapped to the controller. The documentation for each device will indicate the format of the I/O messages it sends (produces) and receives (consumes). Many devices will allow you to select among several data formats. These selectable formats are called I/O assemblies. The particular I/O assembly that a device uses is established when the device is configured. Locate the required information in the module’s DeviceNet Objects; information may exist in multiple locations. If it exists in an Assembly Object, it always appears in another object.
5. Determine the controller registers that will be available to the DeviceNet network. For more information on these registers, refer to *Chapter 3*, *DeviceNet Special Registers*.
6. Determine the message types (Poll, Bit Strobe, COS/Cyclic, etc.) supported by your slave device that are available to transmit information. Before choosing any specific message types, determine your timing requirements (scan time) and how often you require updated information.

7. Compile the list of messages needed to initialize this configuration. **If you are using the 2716D as a slave**, this can be as simple as a single message to set data in the Configuration Assembly Object. Some tools, such as the DeviceNet Configurator, can assist with this step by using information in the .eds (electronic data sheet) file or by using the Configuration Objects included with the device.

8. **Choose the controller registers for the 2716D** that will be used for messaging. The number of registers used per message affects the message size. For more information on these registers, refer to *Chapter 3*, *DeviceNet Special Registers*.

NOTE: **For the 2716D slave**, you should also consider other communications parameters (such as BOI), time-outs, and I/O configuration (default/error states for digital I/O, Analog I/O ranges, and proximity/photo-detector sensitivity). There may also be application-specific parameters such as Register and Register Set Objects. Some items are automatically set by the master device, are stored in non-volatile memory and set upon initialization, or are set during a reset sequence.

9. Select your cabling and power supply. **You can have multiple 2716D slaves** on a single network. You also need to connect to each DeviceNet network through the DeviceNet connector on the 2716D's front panel.

I/O Caveats

Because of certain DeviceNet limitations such as speed, CTC has the following list of caveats for I/O connections:

NOTES: **This section only applies to the 2716D in master mode.**

The first item on this list is associated with controller input functions. This causes the controller to monitor the inputs at a high rate (1 ms or less) or to check whether it's necessary to update the outputs faster than the normal execution scan time. DeviceNet I/O should not be used because it is not immediately accessible and a significant amount of time (large fraction of a ms) is needed to communicate with the 2716D module to get I/O status.

- Linkable Software Counters 1-8 - Digital inputs that are used to count pulses or transitions must be local to the controller containing the 2716D master module. DeviceNet inputs cannot be linked to the controller's internal software counters.
- Programmable Limit Switch (PLS) function - You should not assign a PLS output to the 2716D because the PLS operates at too great a speed.
- Pulse outputs - Outputs 1 and 2 in each CTC controller are dedicated to sending out an automatic stream of pulses with registers 5901-5908. These outputs must be located in the controller containing the 2716D master module.
- Software Faults and Register 13009 - Register 13009 is used to automatically turn off an output when a software fault occurs. This means that if a software fault occurs, the output specified in this register will be de-energized.
- Local I/O Assignments and DeviceNet I/O - Be very careful when you assign I/O mapping to your DeviceNet I/O. Your I/O assignments must not coincide with any local I/O in your controller. CTC recommends that you mount the 2716D module into the right-hand most slot of your controller and assign I/O numbers that start after any local I/O assignments. For example, if you have a 2203 16 input/16 output module installed on your system and have assigned I/O number 16 to your DeviceNet module, the controller will only see the local I/O and will not recognize conflicting I/O numbers on the 2716D module.

- Align digital I/O points in blocks - I/O addressing must be done in blocks of 8 bits, or 1 byte. Each block of 8 inputs or outputs must either be mapped to a single device or remain unmapped. For example, if you map a digital I/O point to address 17, then the remaining 7 bits may not be used for any other device. In addition, mappings must start with an I/O number that is a multiple of 8 plus 1 (1,9,17,25, etc.).
- Analog I/O limitations - Analog I/O is not recommended for use with DeviceNet because of resolution and scaling factors. For example, suppose you have an actual analog value of 312 that is represented by a number in the range 0-1,023. The controller and 2716D can scale this number, which seems to provide a greater degree of precision such as a number between 1-10,000. If you use 312, the closest analog value is (312/10,000)*1023=32 (rounded up from 31.9). If this number is converted back, you get 313 as a result. In addition, if a Quickstep program is checking the actual output value against the stored value to see if they're the same, it results in an error. This situation grows worse when the range is large or the number of bits is small.

Port Addressing

RS-232 Communications

Each communications port is designed to function independently and is automatically serviced on an interrupt basis. Quickstep program activity will in no way cause adverse effects to data integrity. CTC's communications protocols (DLLs) may be used on any port in the controller from a host computer or intelligent host terminal.

Configuring RS-232 and RS-485 Ports

Both the RS-232 and RS-485 ports may be configured to act as a host using currently supported message transmitting and receiving conventions. See the register reference list available on CTC's Web site. To select a port for host communications, you must store the port number to register 12000 before accessing any of the communications registers. The controller's on-board port is port No. 0. The first and second ports in the controller's rack are ports 1 and 2. Up to eleven ports are supported by the 2700 controller series.

Example:

```
[1] Store_Port_Number, Transmit _Row_10, Test_Port
    ;;; Select the second communications port in the controller.
    ;;; Send Row 10 of the data table out of the communications port.
    ;;; Test to see if port is busy.
   -------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
   -------------------------------------------------------------------------
    store 2 to Reg_12000
    store 10 to Reg_12001
    if Reg_12000=0 goto Next
```

NOTE: Register 12000 has different meanings when reading and writing to it.

Computer Based Programming and Communications

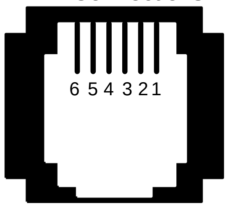
RS-232 Protocols

The RS-232 ports on the model 2716D provide a means for downloading Quickstep programs and data communications support.

The 2716D is equipped with built-in protocols that allow direct computer communications with its RS-232 ports. These protocols allow an external computer to directly interact with many of the controllers resources such as registers, inputs and outputs, and flags. The *CTC 32-bit Data Communications Functions Reference Guide,*which is available in the customer area of our website, describes these protocols. The 2716D can also be configured to act as a host to support communications with other external peripherals such as operation interface terminals, bar-code readers, printers, and other controllers.

RS-232 Connections

Modular Jack
Pin Connections



1 - Not Used
2 - TxD outbound
3 - Ground
4 - Ground
5 - RxD inbound
6 - Not Used

CT1

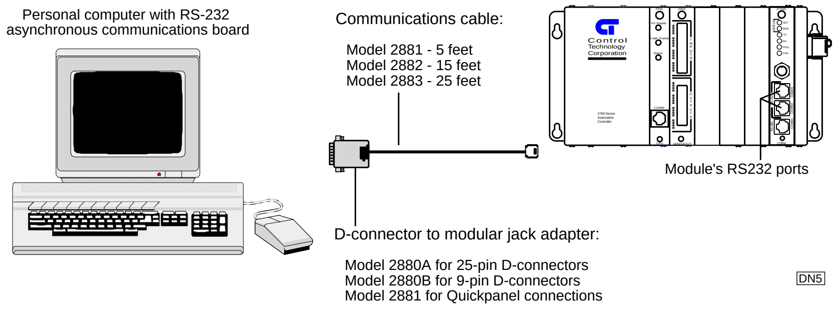
Connections to the module's RS-232 port is made through a modular jack (labeled COMM1 or COMM2) on the module's front panel. This jack carries the receive signal, two grounds, and the transmit signal for the communications channel. The pin connection diagram below illustrates the wiring of the jack. Only the center four conductors of a six or eight conductor jack are used.

A series of standard Control Technology cables are available for connecting to this jack (see the diagram on the next page). As an alternative, many commonly available telephone cables may be substituted.

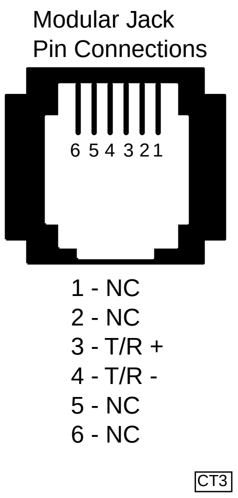
NOTE: Do not connect the module to a telephone line.

Connecting to a D-Connector

RS-232 ports on computers are usually configured through 25-pin (DB25) or 9-pin (DB9) D-type connectors. There is a standard for wiring such connectors, which is followed by most PC manufacturers.


Personal computer with RS-232 asynchronous communications board
Communications cable:
Model 2881 - 5 feet
Model 2882 - 15 feet
Model 2883 - 25 feet
Control Technology Corporation
Module's RS232 ports
D-connector to modular jack adapter:
Model 2880A for 25-pin D-connectors
Model 2880B for 9-pin D-connectors
Model 2881 for Quickpanel connections
DN5

RS-485 Connections



Connections to the module's RS-485 port are made through a modular jack (labeled COMM 1) on the module's front panel (jumper must be set properly). This jack carries the receive signal and the transmit signal for the communications channel (half duplex communications). The pin connection diagram illustrates the wiring of the jack. Only the center two connectors of a six or eight conductor jack are used.

Chapter 2

DeviceNet Technical Overview

Contents

DeviceNet Concepts

Advantages

DeviceNet is a low-cost communications link that connects industrial devices (such as a CTC controller) to a network and eliminates expensive hardwiring. This results in improved communication between devices as well as important device-level diagnostics that aren't easily accessible or available through hardwired I/O interfaces. In addition, since DeviceNet is an open network standard, it allows many different types of devices from numerous manufacturers to work together on a single network.

CAN

DeviceNet is based on a broadcast-oriented, communications protocol called CAN (Controller Area Network). This protocol was originally developed for the European automotive market to replace expensive wire harnesses with low-cost network cable. As a result, CAN has a fast response time and is highly reliable. Consumer and commercial demand for CAN was a key factor in lowering the price and increasing the performance of CAN. With CAN, any node may transmit if the bus is not busy. If two or more nodes begin transmitting at the same time, the message with the lowest CAN ID will complete the transmission. DeviceNet adds a layer above CAN that allows logical connections to exist among nodes and defines message formats. A single DeviceNet node may have up to 64 nodes, each with a unique address (MAC ID). DeviceNet supports baud rates of 125, 250, and 500 KB. As the baud rate increases, the maximum allowable distance of cable between any two devices decreases. There is only one baud rate allowed per network, and all devices must operate at the same baud rate. The table below lists some of the major features of DeviceNet.

Feature	Description	
Network Size	Up to 64 nodes	
Network Length	Selectable end-to-end network distance varies with speed	
	Baud Rate	Distance
	125 Kbps	500 m (1640 ft)
	250 Kbps	250 m (820 ft)
	500 Kbps	100 m (328 ft)
Data Packets	0-8 bytes	
Bus Topology	Linear (trunkline/dropline): power and signal on the same network cable	

Bus Addressing	Peer-to-Peer with Multicast (one-to-many);
	Multi-Master and Master/Slave special case;
	polled or change-of-state (exception-based)	

System Features	Removal and replacement of devices from the network under power	

Features

DeviceNet allows for explicit message connections and I/O message connections. Explicit messages are typically used for device configuration and diagnostics and are more generic and flexible than I/O messages. I/O messages are used for transferring control information, have less protocol overhead, and are more efficient than Explicit messages. Most devices use the predefined Master Slave Connection Set messages for I/O messaging. This set provides for strobed, polled, change of state, or cyclic transmissions. When a strobed connection is used, the master sends one strobe message to which all enabled slaves respond. With a polled connection, the master sends individual poll messages to each device and expects a response from each device. Change-of-State or Cyclic connections do not require a command from the master. A slave device using either a change of state (COS) or Cyclic connection sends data when the device detects a change in the monitored value(s). These messages restart the interval timer and are used to prevent rapidly changing data from generating multiple messages and overloading the network. A slave device with either a cyclic or COS connection sends data on a fixed (configurable) time interval and the COS connection will also send a message if a change in the data is detected before the time interval is up.

Theory of Operation

The 2716D module uses a 32-bit processor, which allows operation of the DeviceNet network and both serial ports at full rated speed without encumbering the controller’s CPU. A shared memory interface allows the processor to communicate with the controller. The DeviceNet Scanner processor transfers data between the memory interface and the DeviceNet network. Configuration data passed to the DeviceNet Scanner processor at startup defines which devices will be accessed, how often they will be scanned, and where each device’s data will be stored in the memory interface.

Scanning Characteristics

The DeviceNet scan operates asynchronously with the controller scan. Thus, the 2716D can be reading data from DeviceNet nodes while the controller is solving logic. When the controller reads data from the module, it receives the most recent data that has been read. Because the scans are asynchronous, the DeviceNet scan may be faster or slower than the controller scan.

The scanner configuration includes a Scan List, which identifies all devices that the scanner should scan. When the scanner is started, it will attempt to connect to each device on the Scan List. Upon connecting to each device, the scanner will obtain identity object attribute data from the device and compare this data against user specified values. If the selected attributes match, the scanner will begin I/O communications with the device.

The rate at which individual devices are polled is configured by the user. This interval is the elapsed time between the start of one scan and the start of the following scan. The scanner begins a scan by sending out all strobe and poll messages defined in the configuration. As devices answer, their responses are placed in an input buffer. When all polls have been sent, the scanner retrieves the responses from the input buffer and maps this data to the I/O as specified in the configuration. Assuming the scan interval is longer than the time required to perform these operations, the module will remain idle until it is time to start the next scan.

NOTE: You must set the scan time to a value large enough to allow all messages to be sent and responses processed. If more time is required to send the polls and process the response than allotted in the scan time interval, the module will extend the scan long enough to process the responses.

Data Mapping

The process of transferring data between DeviceNet messages and controller data elements is called data mapping. DeviceNet data is contained in messages that consist of one or more eight-bit bytes. The device vendor determines the content of the message. The vendor may use an ODVA standard data assembly or use a vendor specific format.

For example, assume a proximity sensor transmits one byte of data where the first bit contains the Sensor State (on or off) and the second bit contains the Device Status (OK or Fault). Data maps are used to specify the controller data elements that will contain the Sensor State and the Device Status. Using the DeviceNet Configurator, you can map the Sensor State bit to Input 49 and the Device Status bit to Input 50.

DeviceNet Objects

The DeviceNet module provides an inexpensive communications link between the DeviceNet network and a CTC controller. This link is made possible through a “window” created by the module’s software configuration. This configuration is stored in the module’s non-volatile memory and is retained if the module is reset.

The module has pre-configured DeviceNet “objects” which allow the network devices to gain access to the controller’s registers. These objects are abstract representations of particular components. In this case, the objects are specified as CTC Register Objects and CTC Register Set Objects. Register Objects and Register Set Objects are defined by the device profile that is resident in the module’s software. There are 5 CTC Register Objects and 5 CTC Register Set Objects available on the DeviceNet interface.

Register Set Objects can be configured to select both the start register and the number of registers required for data transfer. These objects provide access of up to a range of 16 registers within a CTC controller. The first three register set objects are used to process Group 2 I/O messages but are also used for explicit messaging. Setting the number of registers (or start register to 0) will change the size of the Poll message and should be configured before using the messages. Register Objects are always 1 register wide (32 bits/4 bytes of data), are specified only by register number, and are available for explicit message access to standard controller registers. The remaining register set objects and all register objects are set to an idle state when the system is initialized.

NOTE: Refer to the *Software Configuration* section for more information on register objects, register set objects, and specific DeviceNet parameters.

Messages

All interactions between Master and Slave devices are based on messages. There are two basic message types: I/O messages and explicit messages. I/O messages provide dedicated, special-purpose communications between a producing application and one or more consuming applications. Explicit messages provide generic, multi-purpose communication paths between two devices and are used to perform a particular task and report the results of performing the task. For more information on message types, refer to *Messaging*.

Master Device Types

Master devices establish connections on the network and are responsible for controlling the data sent over a connection. They are capable of acting as masters on one connection and slaves on a different connection.

Scanners

Scanners continuously scan the network to receive data from inputs on multiple slave devices and make this information available to a controller such as a PLC or a computer running dedicated control software. The controller uses this information to update its output values and transmits this information to the scanner. The scanner then sends the data to various slave devices. Scanners generally have a fixed set of messages and devices that are monitored and updated within a fixed period (the scan time). Most scanners use I/O messaging.

Human-Machine Interface

A Human-Machine Interface (HMI) is any way that you interact with a system. This interface may include simple pushbuttons, switches, lights, and any text or graphical display (such as a touchscreen).

HMIs monitor data in multiple slave devices and use this information to determine what you see on a display. The HMI may also use inputs that you enter to update data in slave devices. However, the HMI does not necessarily communicate with all devices at any one time and may only communicate with devices whose I/O is currently displayed. The interface may contain a subset of the system inputs that are continuously monitored, but it usually doesn’t update all its I/O data during a fixed cycle. Although an HMI can use I/O messaging, it usually employs Explicit Messaging Services.

Supervisors

Supervisory systems monitor specific data in some slave devices and are capable of receiving or setting specific data during troubleshooting. Supervisors do not have to be active on a continual basis. These devices can remain idle until a problem occurs and may only be needed when starting the network.

The supervisor provides additional ways to inspect and evaluate the status of individual devices on the network. The data it transmits and receives depends on the state of operation. For example, supervisors might be used to configure machines for different operations in situations where the supervisor has greater storage capacity than a controller. Supervisors can also track operations and make historical data available to outside systems. In general, supervisors use Explicit Messaging.

Software Configuration

This section describes parameters that are stored in volatile memory and can only be set by software.

Bus-Off Interrupt (BOI)

This action corresponds to an interrupt state that disables the DeviceNet interface because of continuous streams of corrupted data. Data can be corrupted by large amounts of noise on the line, electronics/wiring problems, and connecting or disconnecting network devices while they're transmitting.

A value of 0 causes the module to disconnect itself from the network until the system is reset. A value of 1 causes the module to automatically reset the interface and attempt to re-establish bus communications. The parameter's default value is 0, but you can change this value by writing to the BOI Action Attribute of the DeviceNet Object (Object code 3, Instance 1, Attribute 3).

Register Objects and Register Set Objects

This section describes the various objects residing in the 2716D module.

Poll/Change of State (COS)/Cyclic Response Output Data Start Register

This is the start register in the first CTC Register Set Object (Object code 80, Instance 1, Attribute 1). It is the first register whose data is transmitted in response to a Poll I/O command or a COS/Cyclic I/O message. The module also monitors this data when a COS connection is established so that a message is sent when a change of state is detected.

You can set this parameter to any value between 0 and 32,767. A value of 0 indicates that no register number is required. If this state occurs, then the response to a Poll/COS message contains no data. All other values, including invalid register numbers, are passed through the module to the CTC controller. This parameter's default value is 10001 with the first 32 digital outputs grouped as a 32-bit value.

Poll/COS/Cyclic Response Output Data Number of Registers

This parameter specifies the number of registers in the first CTC Register Object (Object code 80, Instance 1, Attribute 2). The module transmits information from these registers in response to a Poll I/O command or a COS/Cyclic I/O message.

You can set this parameter to any number between 0 and 16. A value of 0 indicates that no data is available. The physical size of the data is 4 times larger than the specified parameter (4 data bytes per register). Therefore, if the specified value is greater than 2, then data cannot be sent in a single message packet and a fragmented message protocol is required. If the Poll/COS/Cyclic data start register attribute has not changed, then the number of registers attribute defaults to 2, which allows access to the first 64 digital outputs.

Poll Command Input Data Start Register

This is the start register in the second CTC Register Set Object (Object code 80, Instance 2, Attribute 1) and is the first register that gets updated when the module receives data from a Poll I/O command. The module also monitors this data when a poll connection is established, which allows it to detect when the CTC controller makes changes to the data.

NOTE: If new data is received while the controller is trying to update existing data, then a software conflict may occur.

You can set this parameter to any value between 0 and 32,767. A value of 0 indicates that a register number is not required, while all other values, including invalid register numbers, are transferred to the CTC controller. The default value is 11001, with the first 32 digital inputs grouped as a 32-bit value.

Poll Command Input Data Number of Registers

This parameter specifies the number of registers in the second CTC Register Object (Object code 80, Instance 2, Attribute 2) which are updated when data is received from a Poll I/O command.

You can set this parameter to any number between 0 and 16. A value of 0 indicates that no data is available. If the Poll command data start register attribute has not changed, then the number of registers attribute defaults to 2, which allows access to the first 64 digital outputs. If you specify a number larger than 2, then data is not sent in a single packet and fragmented message protocol is required. This occurs because the physical size of data is 4 times greater than the specified value (4 data bytes per register).

NOTE: If the module receives a Poll message with either insufficient or additional data, then the controller registers are not updated.

Bit Strobe Response Data Start Register

This parameter is the start register in the third CTC Register Set Object (Object code 80, Instance 3, Attribute 1). This is the first register that is transmitted in response to a Bit Strobe I/O message whose device MAC ID bit is set to 1.

You can set this parameter to any value between 0 and 32,767. A value of 0 indicates that a register is not required, while all other values, including invalid register numbers, are transferred to the CTC controller. The default value is 11001, with the first 32 digital inputs grouped as a 32-bit value.

Bit Strobe Response Data Number of Registers

This parameter specifies the number of registers in the third CTC Register Set Object (Object code 80, Instance 3, Attribute 2) which are transmitted in response to a Bit Strobe I/O message whose device MACID bit is set to 1.

The parameter value is constrained to a number between 0 and 2 because of the restriction on the size of Bit Strobe data. A value of 0 indicates that no data is available. Although the physical size of the data is 4 times greater than the parameter value (4 data bytes per register), the bit strobe restriction of 2 registers allows the data to be sent in single message packets. If the bit strobe data start register attribute has not changed, then the number of registers attribute defaults to 2, which allows access to the first 64 digital inputs.

Register Request Timing

The module continuously requests updated register information from the controller so that current data is available to the DeviceNet network. This situation places an additional burden on the controller. The Register Request Timing parameter alleviates this problem by specifying a minimum delay (in ms) between each register value inquiry. If controller and module traffic is unusually high, then the update cycle might be longer than the specified delay.

It is part of the class instance in both the CTC Register Object class and the CTC Register Set Object class (Object codes 80 and 81, Instance 0. Attribute 3). You can set this parameter to a value between 0 and 20,000 (20 seconds) with a default value of 20 ms.

Messaging

This section describes the advantages and disadvantages of using different message types.

I/O Messaging

I/O messages consist of a small set of messages that may contain a fixed amount of data. For example, Bit-Strobe messages can only transmit one bit of data from a master to a slave device and can only receive 8 bytes (64 bits) of data from each slave. Poll messages are used to send and receive significant amounts of data between masters and slaves and are only limited by bandwidth constraints. Cyclic/COS messages are also used to transmit large amounts of data from slaves to masters when data changes or at regular intervals.

I/O messages have low overhead. Messages of 8 bytes or less have no communications overhead. Larger packets are less efficient because they require one byte of overhead plus an extra message packet for every 7 data bytes. For example, if you have 8 data bytes or less, the message is not fragmented and only requires the usual CAN header and trailer. However, if you are sending 9 bytes of data, then the message is fragmented and an additional fragmentation protocol byte must be included in the first CAN message, which only leaves room for 7 data bytes. The remaining 2 data bytes require a 2nd CAN message with a CAN header, CAN trailer, and an additional fragmentation byte. Therefore, it takes 3 additional bytes to complete a 9-byte message. Although longer messages have a smaller percentage of overhead, they still consume network bandwidth.

NOTE: For more information on fragmented messages, refer to the *Message Fragmentation* section.

If you’re using a more complex device with configurable I/O, you might be able to change the message contents when I/O is added, but the message data structure is still determined by the manufacturer. For example, for the 2716D, you are able to configure the message size and message contents, but the message length must be a multiple of 4 bytes and cannot exceed 64 bytes.

I/O messages are also Group 2 messages, and the DeviceNet specification states that a device is limited to only one Group 2 connection. Therefore, the device is only allowed to exchange I/O messages with a single master device. This means that if you use I/O messaging with a scanner on your network, then HMIs and Supervisory masters are not allowed to communicate using I/O messaging with any of the nodes being scanned with I/O messaging. These master devices are allowed to use Explicit messaging if a device supports UCMM.

The various types of I/O messages are described in the following paragraphs.

Bit-Strobe Message

Bit-Strobe commands can rapidly transmit small amounts of data between a master and its bit-strobed slaves. Because of the need to send/receive data quickly, the command response messages are limited to 64 bits (8 bytes) in length. Each bit corresponds to a MACID (0-63) supported on the network. Although the master device transmits all 64 bits, the network does not have to contain 64 devices. In addition, the message is only “consumed” by devices that are configured for Bit/Strobe messages from that particular master device. You can configure slave devices to either ignore the Bit-Strobe command, consume the command and its output data, or consume the command and use it to trigger a response. Commands are ignored until a bit-strobe connection is established. The master then uses the connection to update data on the slave device and to receive data from the slave in the response (if configured).

Poll Message

Poll commands can move any amount of I/O data between a master and its polled slaves and back again if required. Unlike the Bit-Strobe command, which broadcasts its message to the entire network, the Poll command is directed to one specific slave device. You can configure the slave device to either ignore the poll command and its output data or consume the command and use it to trigger a poll response. Poll commands, like Bit-Strobe commands, are also used by the master to update data on the slave device and to receive data from the slave in response.

Change of State/Cyclic Message

Cyclic/COS messages are used for slave-to-master transmission and are not in response to a request from the master. Although this message can consist of unlimited amounts of I/O data that is directed towards a single, master device, maximum message size and message content may be limited by the design of your device. In many cases, these items cannot be altered because some manufacturers of simple devices with fixed I/O have pre-set the contents of messages as well as message length.

Both COS and Cyclic messages can send messages at a fixed (configurable) time interval as well as when data has changed. When data changes and messages are sent out, the interval timer is restarted, which determines the maximum time between COS messages. To prevent rapid data changes from overloading the network, the COS connection allows you to define a second interval. This interval prevents the device from sending additional messages before the production inhibit timer has elapsed. This also allows you to define the minimum time between COS messages and also ignores changes to the data during this time. When you establish a connection to the network, you can configure the module to automatically send the next message without an acknowledgment or you can postpone sending another message until the module receives an acknowledgment from the master device.

COS/Cyclic messages can also be combined with a Poll message to send data in response to scanner requests, at specified intervals, or when the data changes. The response to the Poll command will also re-start the interval timer mentioned above.

Explicit Messaging

Explicit messages allow a master device to retrieve and even change any of the DeviceNet objects within a module. The master also has access to all the data on the DeviceNet network and can issue commands to a module and its subsystems. However, these advantages place additional overhead on the message such as specifying the operation requested and the required objects in the module for this operation. Explicit messaging may also be provided by using UCMM services, which allow multiple connections to any module. The maximum number of connections and number of operations depend on the module's implementation. This information is provided with the module's specifications or with the statement of compliance.

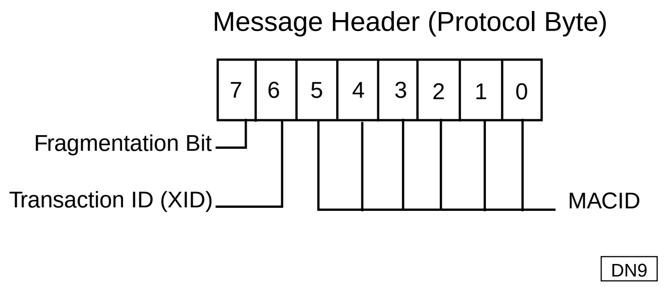
Another disadvantage of using Explicit messaging is that data is limited to the data in a particular object. If the required data does not reside within a single object, then multiple messages are needed to gather the data. Although it may be possible to avoid this problem by defining additional objects (Dynamic Assemblies) in your setup that are a collection of data from several objects within the system, this type of assembly object is not widely supported by most devices.

Message Fragmentation

This section compares I/O and Explicit message fragmentation services.

Protocol Byte

The protocol byte is a message header that is **only** used in the explicit message fragmented data field format. It is not used with I/O message fragmentation. The protocol byte is structured as illustrated below and as described in the following paragraphs.



Bit 7 indicates whether a message is fragmented. If this bit is set to 1, then the message is fragmented and the message header will be followed by a fragmentation protocol byte, which is discussed later in this section.

Bit 6 is the transaction ID, or XID. This ID is used to match up a message response with its original command message. For example, suppose you send out two messages to a device and receive a single reply. Without the XID, there's no way of knowing which message corresponds to the response. The XID bit is turned on and off as each message is transmitted. The first has an XID of 0 and the second has an XID of 1. This makes it easy to pair a message with its response. The XID is one bit in length and has only two states, so it is only used for a maximum of 2 messages at a time. This feature is useful for master devices where the timeout setting is smaller than the response time. The remaining 6 bits of the protocol byte correspond to a device's MACID. This protocol byte is repeated for all fragments of the message.

Fragmentation Protocol Byte

Both explicit messages and I/O messages contain a fragmentation byte that consists of a start, middle, or end fragment. The lower 6 bits is the sequence number. Sequence numbers are the fragment numbers within a message. The fragment number starts with 0 and increases with each fragment, which ensures that no fragments are lost or repeated.

For explicit messages, the message recipient must acknowledge each message fragment before the next fragment is transmitted. This acknowledgment message sets both high bits to 1 and returns the sequence number in the lower 6 bits. If no acknowledgment is received within the fragment time-out, then the fragment is resent.

For I/O messages, there is no acknowledgment received for fragmented messages, so each fragment is sent only once. The time-out for I/O messages is based on a complete message being received, so if a fragment is lost, the entire message is treated as lost and the transaction should time out.

Additional Bytes

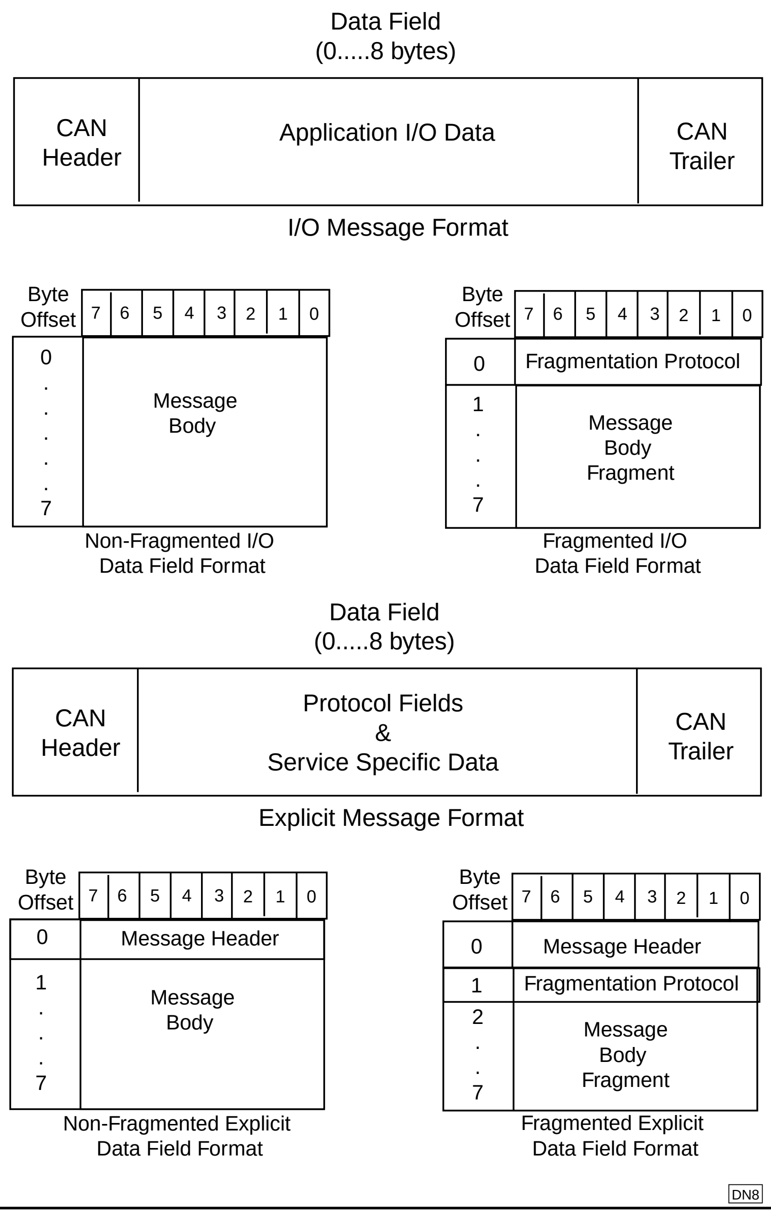
The next byte in the message is the service code. The three most useful codes are Read Single Attribute (0EH), Write Single Attribute (10H), and Reset (05H). In the response, if the most significant bit is set, then the service was successful. If the service failed, then the error response has a special service code that indicates this.

Service codes are followed by object, instance, and attribute numbers. Objects and instances are either 8 or 16 bits and are dependent on the type of connection. 16-bit values are structured with the low byte followed by the high byte.

Write commands are sequenced so that the data follows the object, instance, and attribute information. Reset commands may not require the attribute although attributes can be used to specify the type of reset.

Message Overhead

Message fragmentation services are used when messages exceed 8 bytes in length. The fragmented data field formats for I/O and Explicit messaging are shown below.



You may notice that the Explicit message data field contains an extra protocol byte. This results in additional overhead and means that explicit message fragments only contain 6 data bytes while I/O message fragments contain 7 data bytes. Explicit messages carry even more overhead because they require an acknowledgment for each message fragment; I/O messages do not.

Each fragment, or CAN message, contains up to a maximum of 8 bytes of data but may also contain as little as 0 bytes. DeviceNet does not allow for 0 byte fragments because of the requirement for a fragmentation protocol byte. The very last message fragment has at least 2 bytes: the fragmentation protocol byte and the data byte that forced the message to splinter.

NOTE: 0 byte non-fragmented messages are permitted.

A single CAN packet without data consists of a CAN header that includes the CAN message ID, the checksum, and the trailer. The header contains 19 bits of data, of which 11 are assigned to the message ID, and the trailer contains 28 data bits. Therefore, the total CAN overhead is 47 bits.

All this overhead means that an 8 byte message contains a total of 111 bits (47 CAN bits + 64 data bits). If we assume that we are using the fastest 500 kBd baud rate and each bit requires 2 µS to transmit, then the total transmission time is 222 µS. If you are at 125 kBd, then the total time is 888 µS.

If we change the I/O message to 9 data bytes, then fragmentation services are required and the overhead accumulates. You might think that the 1st data packet contains all 8 data bytes, but in reality, the 8th data byte is shifted to the second packet because the first packet now requires a fragmentation protocol byte.

The second data packet consists of the CAN header and CAN trailer plus the fragmentation protocol byte, 8th data byte, and 9th data byte. From the previous example, we know that the CAN overhead is 47 data bits. So, if you add the protocol byte and the two data bytes, you get a total of 71 bits (47 + 24). The transmission time for the second packet is 142 µS (71 x 2 µS/bit). If you add this to the 222 µS for the first packet, you have a total transmission time of 364 µS for both packets. The percentage of overhead decreases with increasing message size but is still significant.

For explicit messages, even non-fragmented messages contain a message header in the first data packet, so only 7 data bytes are allowed. Once you need 8 data bytes, fragmentation services are required. The second fragment has 4 bytes, and only 2 of these are data bytes. These 4 bytes are the message header (protocol byte), fragmentation protocol byte, 7th data byte (pushed out of the first packet by the addition of a fragmentation byte), plus the 8th data byte. The fragment is 79 bits and has a transmission time of 158 µS. Each fragment also requires an acknowledgment, so this adds 2 acknowledgment messages that are 3 bytes in length.

As you can see, with either type of messaging, adding even one additional data byte places considerable overhead on the DeviceNet network.

Chapter 3

DeviceNet Special Registers

Contents

DeviceNet Special Registers

The 2716D uses registers 13400 to 13599 for DeviceNet operations. These registers operate as follows:

13250-13399 I/O Data Registers-R/W

These registers provide access to specific sections of an I/O message that can vary from 1-32 bits in length. The DeviceNet Configurator has an I/O mapping feature that allows you to specify register reads and writes and determines how 13250-13399 are used.

Register reads are mapped as inputs and register writes are mapped as outputs. Any data stored in a read register is overwritten when the 2716D receives its next I/O message. Unmapped registers are undefined and return a value of -1. Older versions of CTCMon interpret a -1 as a nonexistent register and display this value as an asterisk (*). Mapped registers with nonfunctioning connections return a value of 0.

13400-13463 Individual Module Status and Retry Time - R/W

NOTES: Each hexadecimal digit is equivalent to 4 bits.

Quickstep does not handle hexadecimal numbers. Therefore, decimal equivalents are provided for registers 13400-13463 and register 13464.

2700 series controllers must have firmware v2.20 or later to directly access the DeviceNet special registers from Quickstep. However, if your firmware is an earlier version than v2.20, you can still access these registers for configuration by connecting to one of the 2716D’s serial ports with CTCMon. This allows you to read and change (if possible) the registers from CTCMon.

The upper 16 bits are used for individual module status and the lower 16 bits are used for retry time (read). Scanner Retry Time (Write) value is in seconds and a write of 0 will immediately retry once if there is an error. The table on the next page lists these values in hexadecimal notation and provides decimal equivalents for your convenience.

High Bits Hexadecimal Notation	Decimal Equivalent	Module Status Upper 16 Bits
0x00010000	65536	Poll connection configured correctly
0x00020000	131072	Bit strobe connection configured correctly
0x00040000	262144	COS/Cyclic connection configured correctly
0x00080000	524288	Explicit connection configured correctly
0x00100000	1048576	Poll timeout detected
0x00200000	2097152	Bit strobe timeout detected
0x00400000	4194304	COS/Cyclic timeout detected
0x00800000	8388608	Explicit timeout detected
0x01000000	16777216	Module defined
0x02000000	33554432	Module configured
0x04000000	67108864	Reserved for future use
0x08000000	134217728	Reserved for future use
0x10000000	268435456	Module connection or I/O access failure
0x20000000	536870912	Reserved for future use
0x40000000	1073741824	I/O mapping failure
0x80000000	2147483648	Module failure

The following Quickstep example fetches the retry time:

[1] Get_Retry_Time

```
 ;;; This statement extracts the retry time for module MACID 1
 ;;; from the module status register. The retry time will be left in
 ;;;  the Status_1 general purpose register.
   ---------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
    store Mod_1_Status_R13401 and 65535 to Status_1
 ....Other Quickstep statements....
```

This next Quickstep example checks on whether a module is configured and also checks for software faults:

[1] Check_Module_Configuration

```
;;; These statements extract the Module Configured flag for
;;; module MACID 1 from the module status register and
;;; check if the Module has been configured. If the module has
;;; been configured, then the program will continue with the step
;;; labeled Module_1_Configured. There should be other statements
;;; to handle the state when the Module is not Configured.

   ---------------------------------------------------------------------------
```

```
    <NO CHANGE IN DIGITAL OUTPUTS>
   --------------------------------------------------------------------------
    store Mod_1_Status_R13401 and 33554432 to Status_1
    if Status_1=33554432 goto Module_1_Configured
....Other Quickstep statements....

[2] Check_for_Software_Faults
;;; These statements extract the Module Fault flag for
;;; module MACID 1 from the module status register and check
;;; if there are any errors in the Module operation. If there has been
;;; an error with the module, then the program will continue with the
;;; step labeled Module_1_Fault. There may be other statements to
;;; handle the state when the Module is operating properly or to check
;;; for faults in other modules.

   --------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
   --------------------------------------------------------------------------
    store Mod_1_Status_R13401 and 2147483648 to Status_1
    if Status_1=2147483648 goto Module_1_Configured
....Other Quickstep statements....

[3] Module_1_Configured
   --------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
   --------------------------------------------------------------------------
```

13464 Scanner Status - R/W

Register 13464 provides status information for the scanner. The table below lists these values in hexadecimal notation and provides decimal equivalents for your convenience.

Low Bits Hexadecimal Notation	Decimal Equivalent	Scanner Status Lower 16 Bits
0x00000001	1	Downloading or initializing configuration
0x00000002	2	Storing configuration in flash memory
0x00000004	4	Valid configuration processed
0x00000008	8	Reserved for future use
0x00000010	16	Reserved for future use
0x00000020	32	Reserved for future use
0x00000040	64	Reserved for future use
0x00000080	128	Scanner disabled
0x00000100	256	Scanner configured
0x00000200	512	Scanner establishing connections; starting scan
0x00000400	1024	Scanner closing connections; stopping scan
0x00000800	2048	Scanner running
0x00001000	4096	Configuration of one or more devices has failed
0x00002000	8192	Reserved for future use
0x00004000	16384	Scanner has failed
0x00008000	32768	Scanner has an error

13465 Scanner Module Status - R/W

When a module’s MACID (0-63) is written to this register, it is read as a 1 if the module is properly configured and is scanning with no errors. A value of 0 means that the module is not configured, is not present on the network, or has an error.

13466 Digital Input Status - R/W

When a digital input number from 1-1024 is written to this register, it is read as a 1 if digital input data is valid and as a 0 if the data is invalid. Data is valid when the input is configured as a DeviceNet input. In addition, the connection in use must be active with no errors (data is being updated).

13467 Digital Output Status - R/W

When a digital output number from 1-1024 is written to this register, it is read as a 1 if digital output data is valid and as a 0 if the data is invalid. Data is valid when the output is configured as a DeviceNet output. In addition, the connection in use must be active with no errors (data is being updated).

13468 Analog Input Status - R/W

When an analog input number from 1-128 is written to this register, it is read as a 1 if analog input data is valid and as a 0 if the data is invalid. Data is valid when the input is configured as a DeviceNet input. In addition, the connection in use must be active with no errors (data is being updated).

13469 Analog Output Status - R/W

When an analog output number from 1-128 is written to this register, it is read as a 1 if analog output data is valid and as a 0 if the data is invalid. Data is valid when the input is configured as a DeviceNet output. In addition, the connection in use must be active with no errors (data is being updated).

13480 Scanner MACID - Read Only

This register is used to list the current active DeviceNet setting of the scanner’s MACID.

13481 Scanner Baud Rate - Read Only

This register is used to specify the current active DeviceNet setting of the scanner’s baud rate. 0=125kBd, 1=250kBd, 2=500kBd.

13482 Configuration Switch - Read Only

This register stores the configuration switch settings.The MACID is stored in the lower 8 bits and the baud rate is stored in the next byte.

13483 Serial Number - Read Only

This register stores the module's serial number.

13490 Software (Firmware) Version - Read Only

This register lists the software (firmware) version. The format is major revision * 100 + minor revision.

NOTE: For more information on registers 13491-13496, refer to the *Software Configuration* section in *Chapter 2, DeviceNet Technical Overview*.

13491 Start Register - Bit Strobe Slave Response

This register stores the starting register number for the bit strobe slave response.

13492 Register Count - Bit Strobe Slave Response

This register stores the register count for the bit strobe slave response.

13493 Start Register - Poll/COS Slave Response - R/W

This register stores the starting register number for the poll/COS slave response.

13494 Register Count - Poll/COS Slave Response - R/W

This register stores the register count for the poll/COS slave response.

13495 Start Register - Poll Slave Input - R/W

This register stores the starting register number for the poll slave input message.

13496 Register Count - Poll Slave Input (Slave Mode) - R/W

This register stores the register count for the poll slave input message.

13499 Update Cycle (ms) for Slave Mode, 0=Disable Slave

This register stores the update cycle time (in milliseconds) or scan rate for the slave registers 13491-13496. A value of zero disables slave mode.

13500-13589 Data Registers for Explicit Messages/General Purpose

These registers are used for data storage, the source of data to be included in explicit messages to various DeviceNet devices, or in responses to these explicit messages.

13590 Module ID for Explicit Messaging

This register stores the module ID number for explicit messaging.

13591 Message Number and Register Status - R/W

In write mode, this register writes out the message number to send. In read mode, it lists the register status: 0 = Idle, < 0 = Busy, 1 to 254 = Number of bytes Response Received (including protocol and service code bytes). If there is no connection, it will read as 255. This register is cleared by a write to 13591, which sends a new message.

13592 Requested and Actual Explicit Message Format - R/W

Write mode sets the requested explicit message format (0=8/8, 1=8/16, 2=16/16, 3=16/8). This format specifies the byte length for class IDs and instance IDs. For example, 8/8 indicates that the class ID is one byte long and the instance ID is one byte long. Setting this value will establish a connection if there was not one defined for that module. If the connection exists and the value is different, then a range error will be returned. Read mode sets the actual message format in low byte, bit 8 = 1 when a link is established. If there is no connection, it will read as 255.

13593 Data Index for Registers 13594 to 13599

Register 13593 functions as an index, or pointer, that works in conjunction with the controller's buffer. The buffer is a region in memory that temporarily stores the contents of an Explicit Message as it goes out to a device or when it's received from a device. Controller registers are then used to provide easy Quickstep access to the data.

This register is capable of selecting the buffer location for data. Separate registers (13597-13599) are then provided so Quickstep can access various types of data available in the buffer. If you write data to registers 13594-13599, the index register will change its value accordingly. For registers 13597-13599, data access is relative to the start of the buffer and is based on the value stored in register 13593. For example, if you want to write data to register 13597, data is written as a single byte at the buffer location specified in 13593, then 1 is added to the existing register value. For registers 13598 and 13599, 2 and 4 are added to the register value, which corresponds to byte lengths of 2 and 4 bytes. For other registers, the index is set to the appropriate value. If you want to read the contents of a register, then 13593 is not affected. If no connection exists, then attempts to access this register results in an error.

When dealing with registers 13594-13596, the index register behaves differently. The first byte of the message (Index = 0) is the DeviceNet protocol header byte and will be overwritten by the appropriate data when transmitting a message. The next byte is the service code, followed by the DeviceNet Object Class (1 byte when the message format is 8/8 and 8/16 or 2 bytes when the format is 16/8 or 16/16). The next 1 or 2 bytes are the DeviceNet Object Instance (1 byte for 8/8 and 16/8 formats, 2 for 8/16 and 16/16) followed by a 1 byte Attribute number, followed by the data. For the response, the service code will be in the second byte (Index = 1) and the response data will start in the third byte.

NOTE: Although registers 13594-13596 are specifically designed to handle service codes, class IDs, and instance values, you can also use registers 13597-13599. However, if you use 13597-13599, you need to check the connection type. You must also make sure that class IDs and instance values are the appropriate size (byte or word) and that they correspond to the connection type specified in Register 13592.

13594 Service Code - Explicit Message

This register stores the service code for an explicit message.

13595 Class ID value - Explicit Message

This register stores the class ID value for an explicit message. The size (8/16 bits) depends on the format specified in Register 13592.

13596 Instance value - Explicit message

This register stores the instance value for an explicit message. The size (8/16 bits) depends on the format specified in Register 13592.

13597 Selected Data as a Signed Byte - Extended to 32 bits

This register stores selected data as a signed byte that can be extended up to 32 bits in read mode. In write mode, only the lowest 8 data bits are used; the upper 24 bits are discarded.

13598 Selected Data as a Signed Word (16 bits) - Extended to 32 bits

This register stores selected data as a signed word that can be extended up to 32 bits in read mode. In write mode, only the lowest 16 bits are used; the upper 16 bits are discarded.

13599 Selected Data as a Signed Long Integer (32 bit)

This register stores selected data as a signed long integer which can be extended up to 32 bits.

Using DeviceNet Special Registers

This section describes how to use the special DeviceNet registers

NOTE: The special DeviceNet registers discussed below are only used for Explicit messages.

Registers 13590-13599 are used to view the details of messages and are specifically designed to work with Explicit messages. First, you set register 13593 to the specific byte you want to examine. For example, set 13593 to a value of 1 if you want to access the service code. Then, you can either read this byte value or write it as a byte to registers 13594 or 13597. Byte lengths up to 2 bytes such as a class ID or instance value can be used with registers 13595 (class ID up to 2 bytes), 13596 (instance value up to 2 bytes), or 13598. Byte values up to 4 bytes can be used with register 13599.

As mentioned above, registers 13594-13596 are set up to handle service codes, class IDs, and instances as directed by the connection type specified in register 13592. If you build an Explicit message yourself, storing these values to the appropriate registers changes the index register (13593) to the correct location in your message so that 13597 is ready to store an attribute byte. If you store (write) a 0 to register 13591, then this automatically sends the message in the buffer.

Once a response is received to your message, then the index in register 13593 is set to 2, which skips past the protocol byte and response code byte and points directly to the first byte of response data. If you want to read this data, then depending on the size of the data bytes, you can directly access this information in registers 13597-13599. Of course, you can also examine the protocol byte or the response code byte by setting the 13593 index to 0 or 1. The response code is the service code plus 128 (80H) for success or 148 (94H) for failure, and you can access this data in register 13594.

If you want to set up a connection without using the DeviceNet Configurator, you have to set the connection type with register 13592 (0 = 8/8, 1 = 8/16, 2 = 16/16, 3 = 16/8). If a connection was already established during the scanner's startup sequence, then the actual connection type can be read out of this register, even though it may not match the connection you originally specified. The high bit = 1 indicates that a link has been established; 255 indicates that the connection has failed.

If you do use the Configurator to set up your connections, it will attempt to fix an improperly configured connection type but only works in situations where the specified connection type is either the same size or a smaller size than the actual connection. For instance, if you define a connection type as 8/8 but the actual connection is 8/16, then the Configurator will change it to 8/16 before sending a message. However, if you've defined a 16/16 message, then it can only be sent on a 16/16 connection.

Chapter 4

Troubleshooting

Contents

Troubleshooting

This section describes some of the problems you may encounter during installation and suggests possible solutions. Typical problems include:

- 2716D module is not configured in the controller’s I/O table
- DeviceNet power is absent
- Devices are improperly configured or installed
- Scan list configuration errors
- I/O mapping errors
- Two devices have the same MACID
- Different baud rates on the network
- No other devices on the network

NOTE: The information in this chapter provides general troubleshooting information that covers many common scenarios. If you still require assistance, contact CTC Technical Support at 1-800-282-5008 (x300) or visit our Web site at http://www.ctc-control.com.

LED Troubleshooting Tables

If you are having installation problems, you can use the LEDs to determine the general cause of a problem.

NOTE: Some LED states are normal conditions and do not indicate any problems.

Network (NET) LED Status Table

Color	State	Reason	Corrective Action
None	OFF	Module is off-line because DUP_MAC_ID test is incomplete, no power is applied, or no other devices are on the network.	Check all DeviceNet cabling. All cables should be securely attached and properly wired. Make sure power is applied to the controller and that the module is correctly installed in the controller. You can also check the MOD LED status and see if it is OFF. Finally, put another device on the network. NOTE: One way to test for power is to disconnect the DeviceNet cable from the module's front panel. This should cause the LED to change to solid red (see below).
Green	Solid	Module is operational and has established connections.	N/A
	Flashing	Module is operational but has not established connections. Module is not included in the controller's I/O table. Scanner program is invalid. This may occur when attempting to update the scanner module's firmware.	N/A Use the DeviceNet Configurator to configure the module in the controller's I/O table. Download another copy of the configuration to the scanner using the DeviceNet Configurator software.
Red	Solid	Failed communications – sent off-line by an error. Duplicate MACID. A device on the network has the same MACID as the scanner.	Check all DeviceNet cabling. All cables should be securely attached and properly wired. Change the MACID of the device or scanner with the DeviceNet Configurator software.

Network (NET) LED Status Table (Continued)

Color	State	Reason	Corrective Action
Red	Solid	Invalid scanner baud rate setting. The scanner's baud rate is different from the devices out on the network. Invalid device baud rate setting. At least one device on the network has a baud rate that differs from the rest of the devices.	Change the scanner's baud rate with the DeviceNet Configurator software or check the DIP switch settings on the module. Change the device's baud rate with the DeviceNet Configurator software or check the DIP switch settings on the module.
Red	Flashing	I/O connections have timed out. Device is not operational.	Check all DeviceNet cabling. All cables should be securely attached and properly wired. Check the device's status LEDs. Make sure that DeviceNet power and signal cabling are properly connected.

Device (MOD) LED Status Table

Color	State	Reason	Corrective Action
None	OFF	No controller power is applied to module.	Apply power to the module.
Green	Solid	Module is operating correctly.	N/A
	Flashing	Module is in stand-by mode because it needs commissioning or because of configuration problems.	Download another copy of the configuration to the scanner using the DeviceNet Configurator software.
Red	Solid	Module has an unrecoverable fault and may need to be replaced. This fault could be a fatal error in the scanner's processor.	Cycle the power. If the problem persists, download another copy of the scanner software.
	Flashing	Module has a recoverable fault. Error in the scanner's configuration.	Download another copy of the configuration to the scanner using the DeviceNet Configurator software. Download another copy of the configuration to the scanner using the DeviceNet Configurator software. If the problem persists, download another copy of the scanner software.
Red-Green	Flashing	Module is in self-test mode.	Contact CTC Technical Support.

General Troubleshooting Tables

This section provides general troubleshooting remedies.

Serial Communications

Symptoms	Possible Causes	Corrective Action
Configurator reports a communications timeout.	Defective cable.	Verify the integrity of the cable. If the cable can be used to access the controller through the serial port, then it should be OK.
	Invalid communications parameter setting in the configuration program.	Set the communications parameters to: Baud = 19200 Parity = None Data Bits = 8 Stop Bits = 1

One or more devices are inaccessible to the scanner

Module LED condition	Possible Causes	Corrective Action
LED conditions are normal and no device errors are evident.	Device is not in the scan list.	Review the configuration file to ensure that an entry exists for the device. If not, add the device, create new maps, download the new configuration to the scanner, and restart the scanner.
	Device is not active in the scan list.	Review the scan list to ensure that the device is active in the scan. If not, activate it in the configuration, download the new configuration to the scanner, and restart the scanner.
	No map has been created for the device.	Review the scan list to ensure that at least one data map exists for the device. If not, create at least one map, download the new configuration to the scanner, and restart the scanner.
	Device is not mapped to the proper location in the controller.	Review the data map for the device.

Error Codes for Explicit Messaging

The table below lists error codes returned in the responses to Explicit Messages. When you get these errors:

1. Make sure your message is valid and contains the correct service code, attribute, instance, or object for the specific device receiving the data.
2. Make sure the data you are sending is valid.
3. The type of information you are sending must be appropriate for the DeviceNet application in question.
4. Check the device's documentation for more information on error codes.

Error Codes

Error Code (Hexadecimal)	Error Name	Description
00-01	Reserved by DeviceNet	
02	Resource unavailable	Resources needed for an object to perform a requested service were unavailable.
03-07	Reserved by DeviceNet	
08	Service not supported	Requested service was unimplemented or undefined for a particular object class/instance.
09	Invalid attribute value	Invalid attribute data has been detected.
0A	Reserved by DeviceNet	
0B	Already in requested mode/state	Object is already in the mode/state being requested by the service.
0C	Object state conflict	Object cannot perform a requested service in its current mode/state.
0D	Reserved by DeviceNet	
0E	Attribute cannot be set	Request was received to change an attribute that cannot be modified.
0F	Privilege violation	A permission/privilege check has failed.
10	Device state conflict	A device's current mode/state prohibits the execution of a requested service.
11	Reply data too large	Data in the response buffer is set for transmission but is larger than the allocated response buffer.

Error Codes (Continued)

Error Code (Hexadecimal)	Error Name	Description
12	Reserved by DeviceNet	
13	Not enough data	The requested service did not supply enough data to perform the specified operation.
14	Attribute not supported	Requested attribute is not supported.
15	Too much data	The service supplied more data than expected.
16	Object does not exist	Specified object does not exist in the device
17	Reserved by DeviceNet	
18	Attribute data was not stored	An object's attribute data was not saved before a service was requested.
19	Store operation failure	An object's attribute data was not saved because an attempt to save this information failed.
1A-1E	Reserved by DeviceNet	
1F	Vendor specific error	An error specifically related to a vendor has occurred. The additional code field of the error response defines this error. Use of this general error code should only be performed when none of the codes in this table or within an object class accurately describe the error.
20	Invalid parameter	Requested parameter is invalid. This code applies when a parameter does not meet the requirements of this specification and/or the requirements defined in an application object specification.
21-27	Future extensions	Reserved by DeviceNet for future extensions.

Error Codes (Continued)

Error Code (Hexadecimal)	Error Name	Description
28	Invalid member ID	Requested member ID does not exist in the specified class/instance/attribute.
29	Member cannot be set	Request to modify a member is invalid because the member cannot be modified.
2A-CF	Future extensions	Reserved by DeviceNet for future extensions.
D0-FF	Reserved for object class	This range of error codes is used to indicate object class and service errors. Using this range should only be performed when none of the error codes in this table accurately reflect the error. An additional field code is available that further describes any general error code.

Glossary

Glossary

Assembly
A collection of data that consists of several objects within a device.

Asynchronous Operation
An operation that proceeds independently of any timing mechanism, such as a clock. For example, two modems communicating asynchronously rely upon each sending the other start and stop signals in order to pace the exchange of information.

Attributes
Data that helps define an object by specifying a node address (MAC ID), object class identifier, instance number, and attribute number.

Baud Rate
The speed at which a modem can transmit data. The baud rate is the number of events, or signal changes, that occur in one second—not the number of bits per second (bps) transmitted. In high-speed digital communications, one event can actually encode more than one bit, and modems are more accurately described in terms of bits per second than baud rate. For example, a so-called 9,600-baud modem actually operates at 2,400 baud but transmits 9,600 bits per second by encoding 4 bits per event (2,400 × 4 = 9,600) and thus is a 9,600-bps modem.

Bit Strobe
A multicast master message that all supporting slaves use as a trigger to return input data. This message type is efficient and useful, especially for sensor type devices that produce input data but do not consume output data.

Bus-Off Interrupt (BOI)
This action corresponds to an interrupt state that disables the DeviceNet interface because of continuous streams of corrupted data. Data can be corrupted by large amounts of noise on the line, electronics/wiring problems, and connecting or disconnecting network devices while they’re transmitting.

CAN
Controller Area Network. A broadcast-oriented communications protocol with high reliability and fast response.

Change of State (COS)
A feature on a slave device that sends data whenever a change has occurred or at a user-configurable heartbeat rate, whichever occurs first.

Channel

A path or link through which information passes between two devices. A channel can be either internal or external to a microcomputer.

Class

In object-oriented programming, a generalized category that describes a group of more specific items, called *objects*, that can exist within it. A class is a descriptive tool used in a program to define a set of attributes or a set of services (actions available to other parts of the program) that characterize any member (object) of the class. Program classes are comparable in concept to the categories that people use to organize information about their world, such as *animal, vegetable,* and *mineral,* that define the types of entities they include and the ways those entities behave. The definition of classes in object-oriented programming is comparable to the definition of types in languages such as C and Pascal.

Connection

A logical communications link between two devices on a DeviceNet network. Connection types are defined by the types of messages that are transmitted across the link and are either explicit or I/O. The number and types of connections you can establish depends on the design and programming of the DeviceNet device.

Cyclic

A feature that only sends data at a user-configurable heartbeat rate. Both the master and slave produce data independently based on the value of the heartbeat rate.

Data Mapping

Defines how data is moved between the DeviceNet network and data elements that are accessed by controller logic. Once data is mapped, the controller can access this data as if it is standard controller I/O.

Debounce

Situation where a set of contacts closes or opens and bounces between states before settling down to one value.

DeviceNet

A low-cost, open network standard that connects industrial devices (such as limit switches, sensors, etc.) to a network and eliminates expensive hardwiring. This direct connectivity provides improved communication between devices as well as important device-level diagnostics not easily accessible or available through hardwired I/O interfaces.

Device Profile

A collection of standard and application-specific objects that define a device as viewed from the network. Generally, this data includes a definition of the device’s object model, its I/O format, and any configurable parameters.

Device Type

Indicates the ODVA defined device profile that a device complies with. A vendor may include additional objects and/or attributes beyond the specified minimums.

Dropline

A connection from a trunkline to an individual controller or other device.

Dynamic-Linked Library (DLL)

A feature of the Microsoft Windows family of operating systems that allows executable routines to be stored separately as files with DLL extensions and to be loaded only when needed by a program. A dynamic-link library has several advantages. First, it does not consume any memory until it is used. Second, because a dynamic-link library is a separate file, a programmer can make corrections or improvements to only that module without affecting the operation of the calling program or any other dynamic-link library. Finally, a programmer can use the same dynamic-link library with other programs.

Electronic Data Sheet (EDS File)

A file format that allows a vendor to release product-specific information to other vendors. For example, part of the EDS file for the 2716D contains the following type of information:

```
$ CTC 2716 DeviceNet Interface Electronic Data Sheet

[File]

DescText = "CTC2716D DeviceNet adapter";
CreateDate = 06-26-98;
CreateTime    = 09:56:14;
ModDate = 06-26-98;
ModTime    = 09:56:14;
Revision      = 1.1;

[Device]

VendCode    = 254;                    $ Vendor Code
ProdType    = 0;                      $ Product Type
ProdCode    = 2716;                   $ Product Code
```

```
MajRev      = 1;                              $ Major Rev
MinRev      = 1;                              $ Minor Rev
VendName    = "Control Technology Corporation";
ProdTypeStr = "Generic";
ProdName    = "CTC 2716 DeviceNet";
Catalog     = "";
```

Explicit Messages

Messages directed at specific objects within a device that can determine the current value of the object’s attributes or change the value of these attributes. These messages are typically used for device configuration and diagnostics and are more generic and flexible than I/O messages.

Heartbeat Rate

An adjustable, background rate used with change-of-state/cyclic messages to detect that a producer is still in an active state on a network. Data is sent when it changes or when the heartbeat timer expires.

Human-Machine Interface (HMI)

The set of commands, displays, controls, and hardware devices enabling the human user and the computer system to exchange information. With DeviceNet, HMIs monitor data in multiple slave devices and use this information to determine what you see on a display.

I/O Messages

Messages that carry operational data based on the module’s configuration which are limited to specific sources and destinations within a device. These messages are used for transferring control information, have less protocol overhead, and are more efficient than Explicit messages.

IP Address

Short for Internet Protocol address. A 32-bit (4-byte) binary number that uniquely identifies a host (computer) connected to the Internet to other Internet hosts, for the purposes of communication through the transfer of packets. An IP address is expressed in “dotted quad” format, consisting of the decimal values of its four bytes, separated with periods; for example, 127.0.0.1. The first one, two, or three bytes of the IP address, assigned by InterNIC Registration Services, identify the network the host is connected to; the remaining bits identify the host itself. The 32 bits of all 4 bytes together can signify almost 2^{32}, or roughly 4 billion, hosts (a few small ranges within that set of numbers are not used).

Inhibit Time

Amount of time that a device must maintain a COS condition.

Input Buffer

A portion of computer memory set aside for temporary storage of information arriving for processing.

Input Data

Data that is produced by a DeviceNet device, collected by a scanner, and made available to a controller.

Instance

An object, in object-oriented programming, in relation to the class to which it belongs. For example, an object *myList* that belongs to a class *List* is an instance of the class *List*. An instance might be used to define different categories of the same class in situations where the same set of functions is required but for different reasons or in different ways. Every instance shares a common data structure for the characteristics but has its own set of this data and the individual values in the data structure are often different between the instances. Each instance can then have different operating characteristics.

Interoperability

Refers to components of computer systems that are able to function in different environments. For example, Microsoft's NT operating system is interoperable on Intel, DEC Alpha, and other CPUs. Another example is the SCSI standard for disk drives and other peripheral devices that allows them to interoperate with different operating systems. With software, interoperability occurs when programs are able to share data and resources.

Layer

The protocol or protocols operating at a particular level within a protocol suite, such as IP within the TCP/IP suite. Each layer is responsible for providing specific services or functions for computers exchanging information over a communications network (such as the layers outlined in the ISO/OSI model) and information is passed from one layer to the next. Although different suites have varying numbers of levels, generally the highest layer deals with software interactions at the application level, and the lowest governs hardware-level connections between different machines.

Link

See *Connection*.

Master/Slave
A network protocol defining the communications link between the controlling device (master) and the device (slave) responding to its commands.

Media Access Control (MAC) ID
A unique node address (0-63) on a network.

Node
A device that has a single address on the network and is capable of communicating with other network devices.

Object
An abstract representation of a particular component within a product.

Open DeviceNet Vendor Association (ODVA)
ODVA is an independent supplier organization which manages the DeviceNet Specification and supports the worldwide growth of DeviceNet. ODVA works with vendors and provides assistance through developer tools, developer training, compliance testing and marketing activities. ODVA publishes the DeviceNet product catalog and supports vendor Special Interest Groups in developing Device Profiles for specific classes of products.

Open Standard
A publicly available set of specifications describing the characteristics of a hardware device or software program. Open standards are published to encourage interoperability and thereby help popularize new technologies.

Output Data
Data produced by the controller that is written to the module’s memory and sent out to DeviceNet devices.

Packet
A unit of information transmitted as a whole from one device to another on a network. In packet-switching networks, a transmission unit of fixed maximum size that consists of binary digits representing both data and a header containing an identification number, source and destination addresses, and sometimes error-control data.

Parameter

In programming, a value that is given to a variable, either at the beginning of an operation or before an expression is evaluated by a program. Until the operation is completed, a parameter is effectively treated as a constant value by the program. A parameter can be text, a number, or an argument name assigned to a value that is passed from one routine to another. Parameters are used as a means of customizing program operation.

Poll

A message type that is initiated by the master device. It contains output data for a slave device which returns input data in a poll response.

Producer/Consumer

Setup where a client device (typically a master) produces requests and consumes responses from a server device (typically a slave) that consumes requests and produces responses.

Production Inhibit Time

A COS configuration setting that is the minimum time between messages. This setting helps prevent excessive bus traffic.

Scan Time

The rate at which individual devices are polled. This interval is the elapsed time between the start of one scan and the start of the next scan.

Scanner

A device (such as the 2716D) which is capable of functioning as a master. This includes such functions as the pre-defined master/slave connection set and explicit peer-to-peer messages (UCMM). The scanner continuously scans a DeviceNet network to receive input data from multiple slaves and makes this data available to a controller.

Service Code

The service code is a standard code for all devices that is used to build messages. Codes such as **Get_Attribute_Single, Set_Attribute_Single, Reset** are the most commonly used codes.

Services

Requests you can make on a network.

Shared Memory

Memory accessed by more than one program in a multitasking environment. Also, a portion of memory used by parallel-processor computer systems to exchange information. In CTC controllers, shared memory is also referred to as bi-port RAM.

Supervisors

Monitor specific data in slave devices and are also capable of receiving specific data during troubleshooting.

Trunkline

The backbone of a network. It is the longest continuous wiring path from one end of a network to another. In general, the trunkline has the largest cable type and runs from one terminator.

UCMM

Unconnected Message Manager. Provides for the dynamic establishment of explicit messaging connections.

Bibliography

Bibliography

Freedman, Alan. (1996). *The Computer Desktop Encyclopedia*. Amacom.

Hansen, Brad. (1999). *The Dictionary of Computing and Digital Media, Terms and Acronyms*. ABF Content.

Microsoft Corporation. (1998). *Microsoft Press Computer Dictionary, 3rd Edition*. Microsoft Press.

Index

Index