Chapter 6

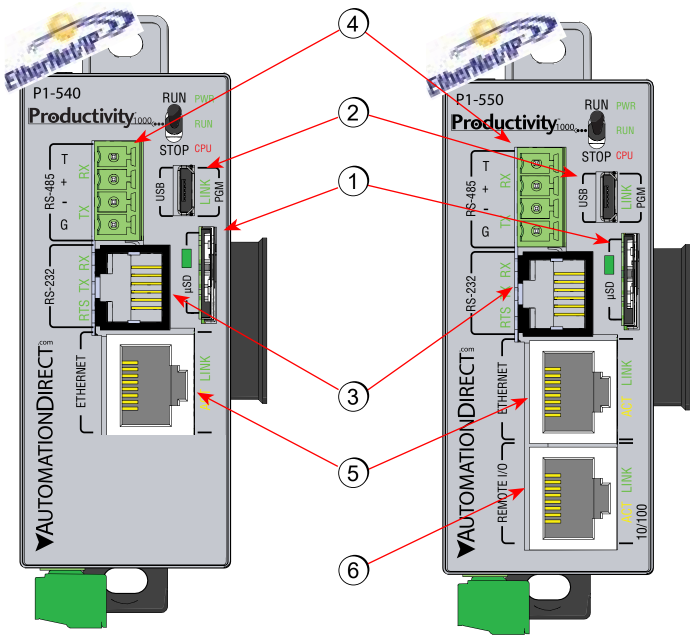
Communications

In This Chapter

Communications

Communication Ports

The AutomationDirect Productivity® 1000 CPUs are provided with several Communications Ports. Follow the associated numbered sections for a detailed description of each of these ports.



P1-540 **P1-550**

General Specifications	
Item #	Communication Port
1	microSD Slot
2	MicroUSB 2.0 Programming Port
3	RS232 Serial Port (RJ12)
4	RS485 Serial Port (TB Style)
5	10/100 MB Ethernet Port
6	10/100 MB Ethernet Port, Remote I/O

***NOTE:** The microUSB port is NOT compatible with older 1.0/1.1 full speed USB devices.*

Communication Ports, continued

❶ microSD Card:

The microSD card slot is provided for data logging capability. Files stored on the microSD card by a CPU or the Productivity Suite programming software are stored under a default name, so only one project may be handled at a time on a microSD card. Existing projects on the microSD card will be overwritten without a prompt.

Data Logging: The Data Logger tool allows setup of periodic or event-based data logging of tag and System Errors to the microSD card. Data Logger setup is accessed under the Monitor & Debug Menu. See Communications Connectivity section for more information.

❷ MicroUSB:

The microUSB 2.0 port uses a Type B connector. It is used for connection to a PC running the Productivity Suite programming software and Online monitoring of program.

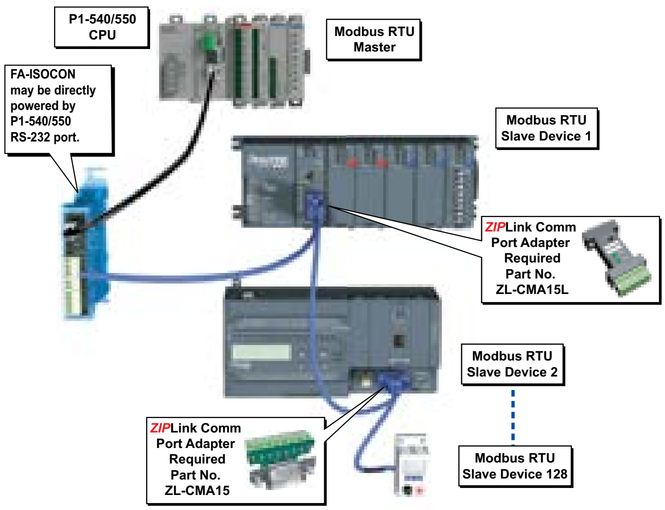
❸ RS-232:

The RS-232 port is an RJ-12 connector located on the lower right front of the CPU. This port can be used for:

- Modbus RTU Master connections.
- Modbus RTU Slave connections.
- ASCII Incoming and Outgoing communications.
- Custom Protocol Incoming and Outgoing communications.

Modbus RTU Master connections:

The RS-232 port is intended to be used for point-to-point connections but it is possible to connect up to 128 devices on a network if an RS-232 to RS-485/422 converter is connected to the port (such as a FA-ISOCON). This is accomplished by using the communications instructions in the ladder project (MRX, MWX, RX, WX). If 4-wire RS-485 or RS-422 communications is needed, using this port with an FA-ISOCON is the best method. See Communications Connectivity section in this manual for more information.


P1-540/550 CPU
Modbus RTU Master
FA-ISOCON may be directly powered by P1-540/550 RS-232 port.
Modbus RTU Slave Device 1
ZIPLink Comm Port Adapter Required Part No. ZL-CMA15L
Modbus RTU Slave Device 2
Modbus RTU Slave Device 128
ZIPLink Comm Port Adapter Required Part No. ZL-CMA15

Communication Ports, continued

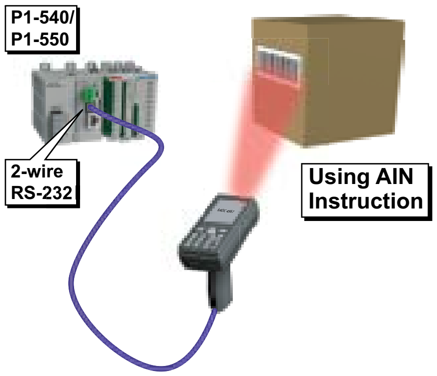
Modbus RTU Slave connections:

The RS-232 port is intended to be used for point-to-point connections but it is possible for the RS-232 port to be used on a Modbus RTU network by using a RS-232 to RS-485/422 converter. The port is addressable in the Hardware Configuration in the Productivity Suite programming software. It is important to note that the RS-232 port cannot be a Modbus RTU master and slave concurrently. If the port is set to Modbus RTU and there are no communications instructions (MRX, MWX, RX, WX) in the project, the CPU will automatically respond to Modbus requests from a Modbus master. See Communications Connectivity section for more information.

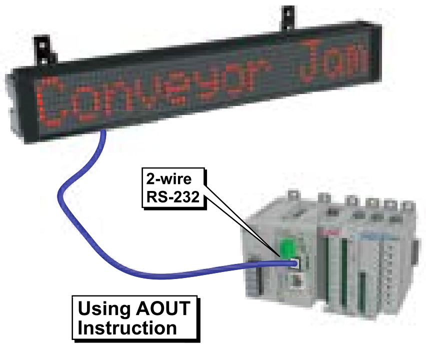
ASCII Incoming and Outgoing communications:

The RS-232 port can be used for sending and receiving non-sequenced String data. This feature is typically used for receiving bar code strings from a scanner or sending statistical data to a terminal or serial printer using the ASCII IN and ASCII OUT instructions. See Communications Connectivity section for more information.

RS-232 ASCII In Communication



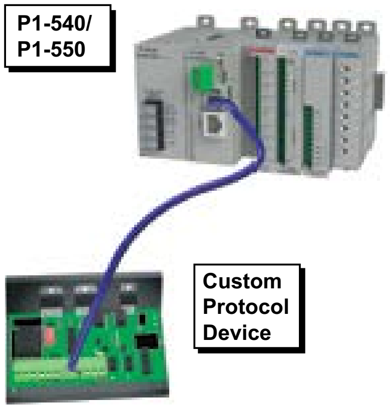
RS-232 ASCII Out Communication



Custom Protocol Incoming and Outgoing communications:

The RS-232 port can be used for sending and receiving non-sequenced byte arrays to various devices. This function is typically used for communicating with devices that don't support the Modbus protocol but have another serial communications protocol. This is accomplished by using the Custom Protocol In and Custom Protocol Out instructions. The RS-232 port is intended to be used for point-to-point connections but it is possible for the RS-232 port to be used on a multi-node network by using a RS-232 to RS-485/422 converter. See Communications Connectivity section for more information.

RS-232 Custom Protocol In and Out

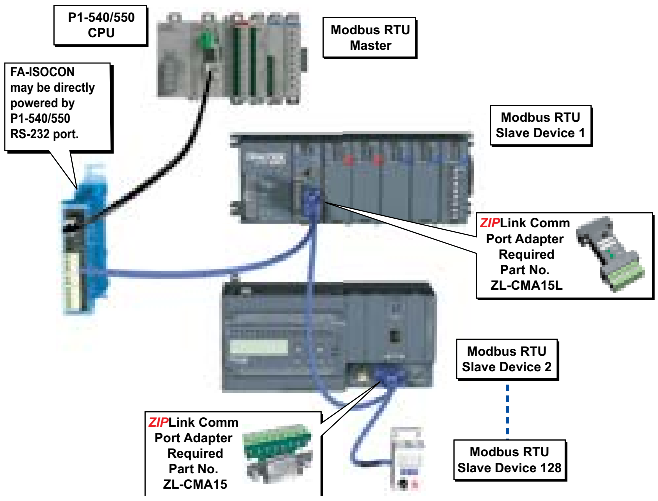


Communication Ports, continued

❹ The Modbus RTU Slave Connections:

The RS-485 network port is used for multi-node networks. The port is addressable in the Hardware Configuration in the Productivity Suite programming software. If the port is set to Modbus RTU and there are no communications instructions (MRX, MWX, RX, WX) in the project, the CPU will automatically respond to Modbus requests from a Modbus master. See Communications Connectivity section for more information.

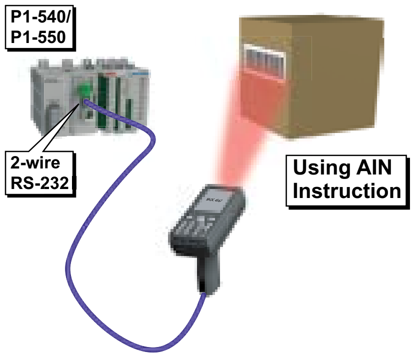
RS-485 Modbus RTU Slave Network Topology


P1-540/550 CPU
Modbus RTU Master
FA-ISOCON may be directly powered by P1-540/550 RS-232 port.
Modbus RTU Slave Device 1
ZIPLink Comm Port Adapter Required Part No. ZL-CMA15L
Modbus RTU Slave Device 2
ZIPLink Comm Port Adapter Required Part No. ZL-CMA15
Modbus RTU Slave Device 128

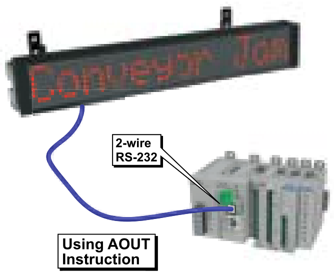
NOTE: *See respective PLC Manual for communication port cable pinouts.*

ASCII Incoming and Outgoing Communications: The RS-485 port can be used for sending and receiving non-sequenced String data. If long distances are required between the ASCII device and the CPU, the RS-485 port is the better selection because of its increased distance support (1,000 meters). ASCII communications are typically used for receiving bar code strings from a scanner or sending statistical data to a terminal or serial printer using the ASCII IN and ASCII OUT instructions. See Communications Connectivity section for more information.

RS-232 ASCII In Communication



RS-232 ASCII Out Communication



Communication Ports, continued

5 **Ethernet:**

The Ethernet port is 10/100Base-T Ethernet with an RJ-45 style connector. It is used for:

Connection to a PC running the Productivity Suite programming software.

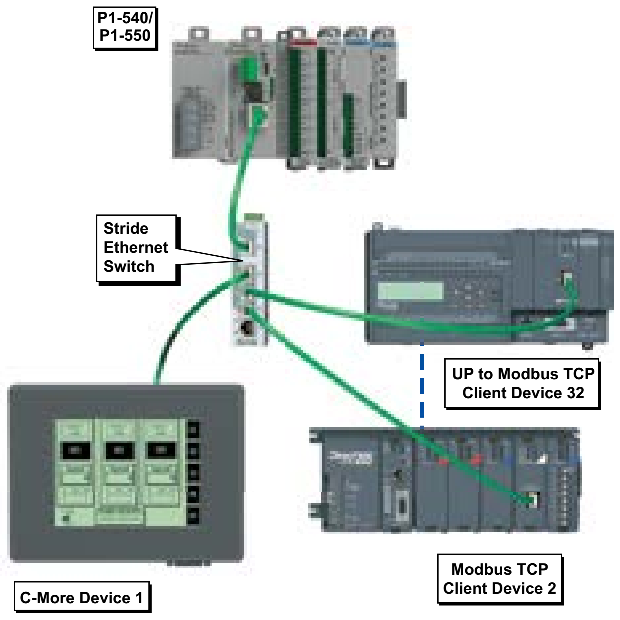
- Modbus TCP Client connections (Modbus requests sent from the CPU).
- Modbus TCP Server connections (Modbus requests received by the CPU).
- Custom Protocol over Ethernet
- ProNET
- EtherNet/IP Scanner (32 Adapters)
- EtherNet/IP Adapter (4 scanners) with 8 connections per device.
- Outgoing Email.

Modbus TCP Client Connections: The CPU can connect to 16 Modbus TCP server devices concurrently by means of communications instructions in the ladder program (MRX, MWX, RX, WX). It is possible to connect to more than 16 Modbus TCP server devices, but not concurrently.

This is accomplished by having communications instructions for more than 16 devices in the ladder program and controlling the enabling and disabling of the instructions so that only 16 devices are enabled at a given time. To connect to non Productivity® 1000 devices, use the MRX (Modbus Read) and MWX (Modbus Write) instructions.

The greatest difference in the RX versus the MRX is that with the RX, the Tag Name in the target CPU can be referenced directly and does not need a corresponding Modbus address. This is accomplished by mapping local and remote tagnames together within the local CPU's RX instruction. Once the instruction is set up to read a remote project, the "Tags of Remote Project" or "Array Tags of Remote Project" drop down lists will be accessible. Map the Tag of the Remote project to a Tag in the Local project to read this data.

Modbus TCP Client (RX-WX)



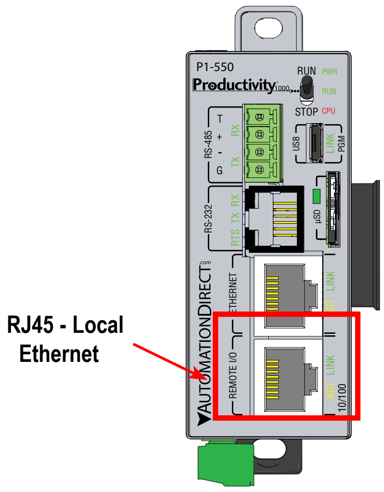
Communication Ports, continued

Modbus TCP Server Connections: The CPU can serve data back to 16 Modbus TCP Client devices concurrently. If 16 Modbus TCP Client devices are connected to the CPU, then any new TCP connection requests will be denied until one of the existing 16 devices drops its connection. If the Client device connecting to the CPU is not a Productivity® 1000 device, then a Modbus address must be assigned to the tag that is being requested. This is done in the Tag Database window. If the device connecting to the CPU is another P1000 CPU or C-more panel, no Modbus address is required.

Custom Protocol Incoming and Outgoing Communications: The Ethernet port can be used for sending and receiving non-sequenced byte arrays to various devices. This function is typically used for communicating with devices that don't support the Modbus protocol but have another custom Ethernet communications protocol. This is accomplished by configuring a "Custom Protocol Ethernet Device" using the hardware configuration and then using the "Custom Protocol Ethernet (CPE)" instruction. See Communications Connectivity section for more information.

***NOTE:** See Communications: Port Configuration for port configuration; Communications: Connectivity for connection information and Communications: Ethernet for Ethernet setup.*

6 Local Ethernet: The Remote I/O Ethernet Port (Local Ethernet Network) is used for connections to 16 GS drives, 4 ProtosX couplers, 4 P1-RX remote slaves, and/or one PS-AMC module. Remote I/O is treated as local I/O by the CPU and is completely scan synchronous; except that PS-AMC modules run asynchronously with respect to the ladder scan, so AMC status bits should be used for interlocking logic if necessary. The I/O is automatically detected on power up.

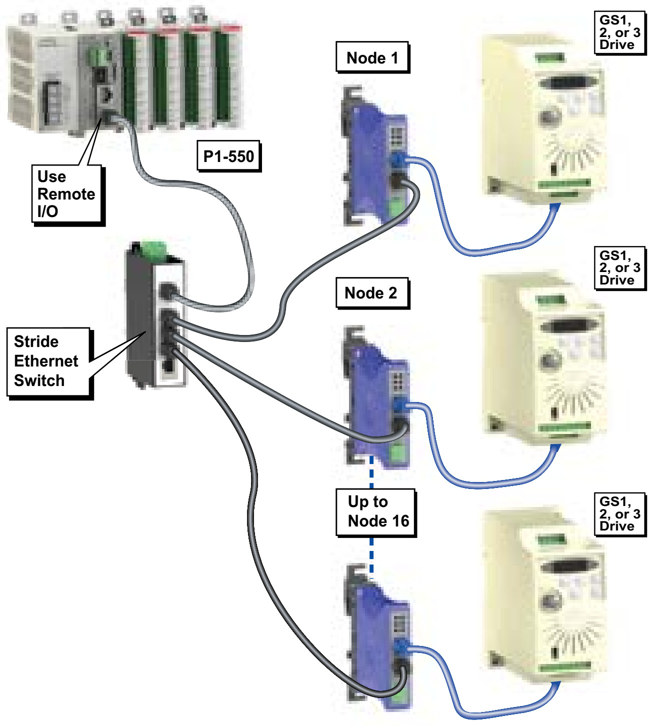


- GS Drive Devices: The P1-550 CPU can connect to 16 GS drives. It connects to a Modbus communication card added to GS4 or GS20 drives or via GS-EDRV100 modules for GS1, GS2 and GS3 series drives.
- The unique address for each GS4 or GS20 drive is set in the GSoft2 software or using the keypad on the GS4 or GS20 drive to set the last octet of the IP address.
- For the GS-EDRV100 modules, the unique address is set using the bank of DIP switches on the module.
- The P1-550 can connect to 4 ProtosX TCP couplers. The unique address for each ProtosX PX-TCP1 or PX-TCP2 module is set by assigning the last octet of the IP address at the DIP switches on the module. The P1-550 will auto-detect all GS-EDRV100 modules that have a unique address (configured by the bank of dipswitches on the module). The configuration can be managed in the Hardware Configuration in the Productivity Suite programming software. See Communications: Remote I/O and GS Drives for configuration information and Communications Connectivity for connection information.

Communication Ports (continued)

- PS-AMC module: The P1-550 CPU can connect to one PS-AMC module. The PS-AMC module configuration is managed in the Productivity Suite programming software. See the Help File for detailed information on module configuration and instructions used with the PS-AMC module. Note that PS-AMC modules run asynchronously with respect to the ladder scan, so AMC status bits should be used for interlocking logic if necessary.

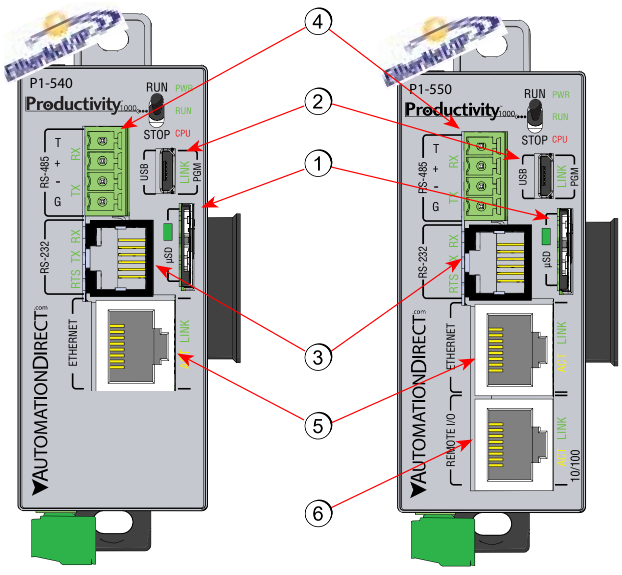
Remote I/O Example



Communications: Connectivity

CPU Port Connections

The AutomationDirect Productivity®1000 P1-540 and P1-550 CPUs are provided with several communications ports. The Connectivity for each of these ports is described in the following sections.



P1-540 **P1-550**

❶ microSD Card Slot

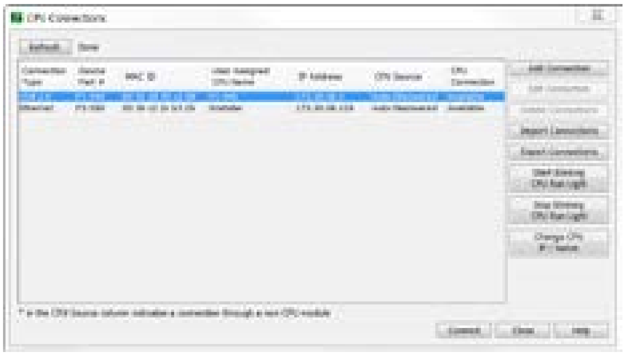
For program data logging (microSD card not included with processor).

❷ MicroUSB Port

This is a programming port with a USB 2.0 Type Micro B female connector. This port requires a MicroUSB Type A-Micro B cable (such as the USB-CBL-AMICB6 cable).

The MicroUSB Port is the simplest method of connecting the Productivity Suite Programming Software to the CPU. After the programming software has been installed, connect a USB A-Micro-B cable to the CPU and select the "Choose CPU" option. The dialog shown below will appear.

Highlight the CPU listed in the dialog box and click on "Connect". No configuration is required.



Connection Type	Device Part #	MAC ID	User Assigned CPU Name	IP Address	CPU Source	CPU Connection
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Ethernet	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Available

* in the CPU Source column indicates a connection through a non CPU module

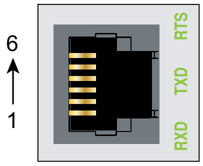
NOTE: *The MicroUSB port is NOT compatible with older 1.0/1.1 full speed USB devices.*

3 RS-232 Port:

Serial RS-232 multipurpose communications port with RJ12 connector.

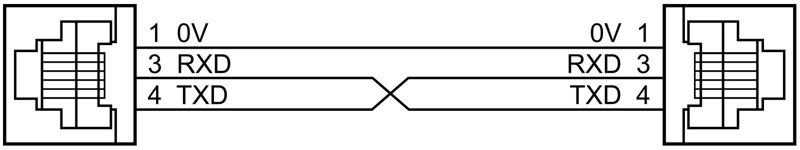
The RS-232 Port can be connected to Modbus RTU master or slave devices, as well as devices that output non-sequenced ASCII strings or characters. The manner in which these devices are wired to the CPU depends whether the device is considered to be Data Terminal Equipment (DTE) or Data Communications Equipment (DCE).

If two DTE devices are connected together, the RX and TX signals should cross or the RX of one device should go to the TX of the other device and the TX of one device should go to the RX of the other device (as shown below).



6-pin RJ12 Female Modular Connector

Pin #	Signal	
6	GND	Logic Ground
5	RTS	RS-232 Output
4	TXD	RS-232 Output
3	RXD	RS-232 Input
2	+5V	210mA Maximum
1	GND	Logic Ground



The CPU is considered a DTE device. Most Modbus or ASCII devices being connected to the CPU will also be considered a DTE device and will need to swap TX and RX, but you should always consult the documentation of that device to verify. If a communication device, such as a Modem, is placed between the CPU and another Modbus or ASCII device it will most likely require connecting the signals straight across (TX to TX and RX to RX). Again, this can differ from manufacturer to manufacturer so always consult the documentation before wiring the devices together.

The RTS signal on pin 5 of the RS-232 Port will turn on when the TX signal is turned on and the RTS signal will turn off when the TX signal turns off. The amount of time that the RTS signal turns on before the TX signal turns on and the amount of time that the RTS signal waits before turning off after the TX signal turns off is adjustable in the CPU Module Configuration for the RS-232 Port. The RTS signal is very often required for media converters, such as a RS-232 to RS-422/485 converter (much like the FA-ISOCON).

The RTS signal is sometimes required for use with radio modems as well (Key on and off control).

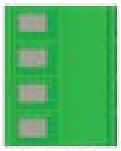
There is also +5VDC @ 210mA on pin 2 available for powering an external device such as the C-more Micro panel.

④ RS-485 Port

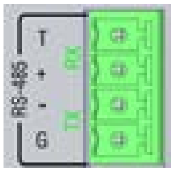
The RS-485 multipurpose serial communications port requires a removable 4-pin connector (See below). This port is useful for connecting multiple Modbus and ASCII devices on one network and/or connecting devices to the CPU at distances greater than 50 feet (RS-232 limit). The RS-485 standard supports distances of up to 1000 meters without requiring a repeater. The RS-485 Port on the CPU can support up to 50 devices, depending on load of each device (this assumes a 19K Ohm load for each device). This number can be increased by placing an RS-485 repeater on the network, if necessary.

This port only supports RS-485 2-wire connections. For 4-wire RS-485 or RS-422, a converter, such as an FA-ISOCON, should be used with the RS-232 Port.

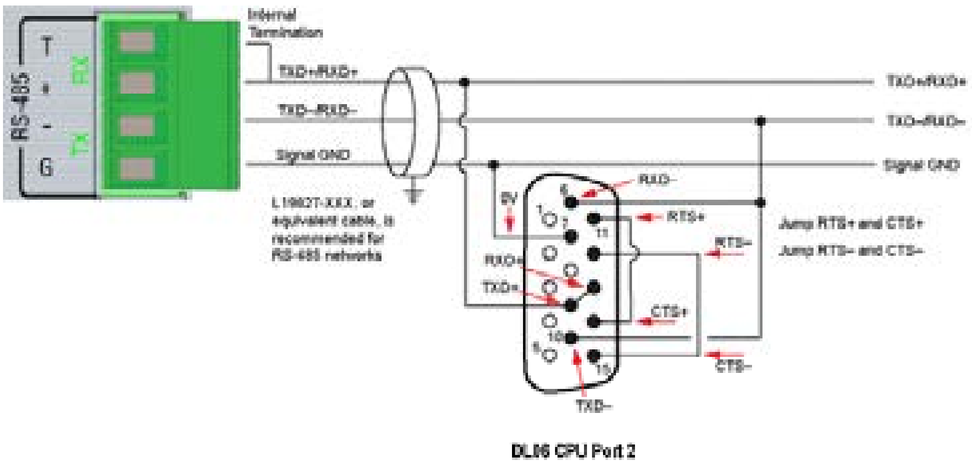
NOTE: *A 120 Ohm resistor is required at each end of the network for termination.*



Removable connector included. Spare connectors available (part no. P3-RS485CON-1).



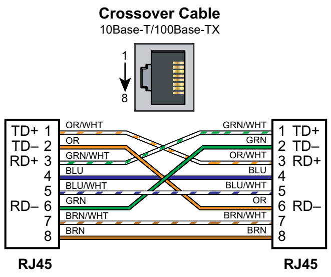
Pin	Signal
T	Termination
+	TXD+/RXD+
–	TXD-/RXD-
G	GND

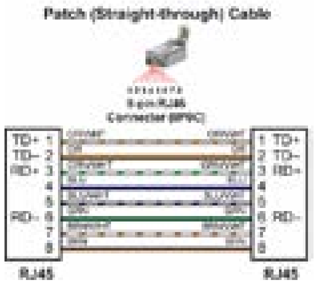


NOTE: *ZIPLink Comm Port Adapter Part No. ZL-CMA15 or ZL-CMA15L may be used to make the connection at DL06 or DL205 CPU Port 2.*

5 External Ethernet Port

The 10/100 Base-T Ethernet port with RJ45 connector is used for programming and Modbus TCP Client/Server functions.





General Information

Crossover cables can be used to directly connect two endpoint Ethernet devices such as a PC network interface card and the CPU. Crossover or patch (or Straight-through) cables can be used to directly connect endpoint Ethernet devices and the CPU.

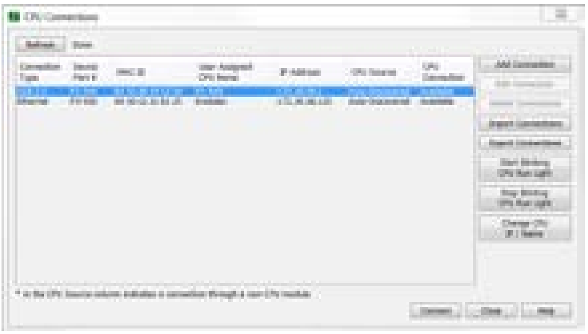
The maximum distance for one cable or segment is 100 meters (328 feet). If the distance required between 2 devices is greater than 100 meters, add an Ethernet switch to extend the distance. An Ethernet switch can be added every 100 meters (or less) almost indefinitely. Each Ethernet switch added will incur some latency (actual amount differs between switches and manufacturers). So if a very long distance is needed between 2 Ethernet devices, it may be better to convert to fiber optics.

The External Ethernet Port can be used as a programming port, a Modbus TCP Client (16 Servers) and Server (16 Clients), EtherNet/IP Scanner (32) and Adapter (4), Custom Protocol over Ethernet, ProNET.

The External Ethernet Port can also be used to send emails using the EMAIL instruction.

Create a Connection

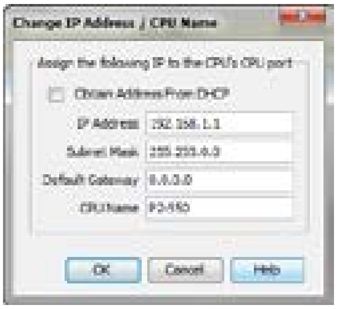
Productivity Suite allows you to connect to a directly connected PLC as well as securely connecting to a remote PLC connected to a StrideLinx VPN router. Tocommunicate with the Productivity Suite programming software, connect an Ethernet cable from the PC to the CPU External Ethernet Port. Once the software has been opened, click on CPU and select the "Choose CPU" option. The dialog shown below will appear.



5 Ethernet Port, continued

Highlight the CPU that you wish to connect to and press the “Connect” button. You may see in the CPU Connections dialog box CPUs that are not on the same subnet as your PC, but this does not mean you can connect to them. To connect to the CPU, you must configure either your PC or your CPU to be in the same subnet. You can easily change the Ethernet settings of the CPU by highlighting it and selecting the “Change CPU IP/Name” button (shown below). Or if you prefer, the PC Setup section of this chapter contains information on configuring the Ethernet settings of your PC.

6 Local Ethernet Port



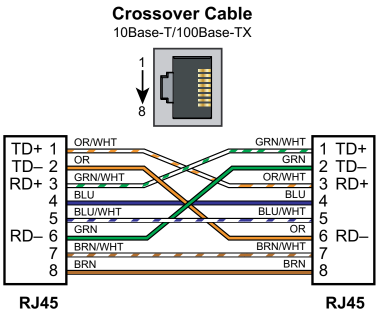
Change IP Address / CPU Name

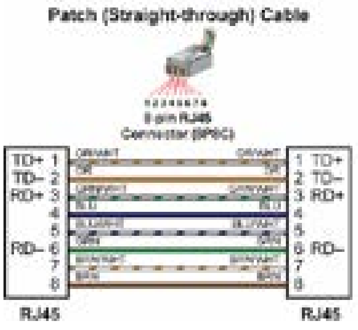
Assign the following IP to the CPU's CPU port

☐ Obtain Address From DHCP

IP Address: 192.168.1.1
Subnet Mask: 255.255.0.0
Default Gateway: 0.0.0.0
CPU Name: P2-550

- Local Ethernet RJ45 connector supports communication with GS drives via optional GS drive communication module ProtosX TCP couplers, P1-RX remote slaves, and a PS-AMC module. The P1 Ethernet ports are auto MDI/MDI-X so straight-thru or crossover cables may be used to connect devices.





- The maximum distance for one cable or segment is 100m (328ft). If the distance required between 2 devices is greater than 100m (328ft), add an Ethernet switch to extend the distance. An Ethernet switch can be added every 100m (328ft) or less, almost indefinitely. Each Ethernet switch added will incur some latency (actual amount differs between switches and manufacturers). So if a very long distance is needed between 2 Ethernet devices, it may be better to convert to a fiber-optic system.

ASCII and Custom Protocol Functionality

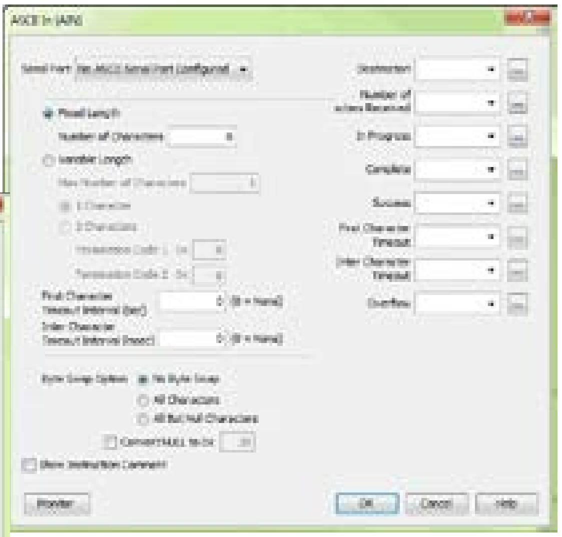
Besides Modbus RTU, there are two additional functions supported on the serial ports in the Productivity® 1000 system.

- The first function is the ability to send and receive text-based data with devices such as bar code readers and serial printers.
- The second function is the ability to communicate serially with other devices that do not support the Modbus protocol and lack a Productivity1000 driver.

ASCII Instructions

The ASCII In/Out instructions use the String data type to send or receive text-based data through the serial port. The String data type is only intended for use with the "printable character set". This can include numbers, letters or special characters.

With the ASCII In instruction, the CPU can receive a fixed length of characters or a variable length of characters with a termination code (an 'end of message' character).



ASCII In (AIN)

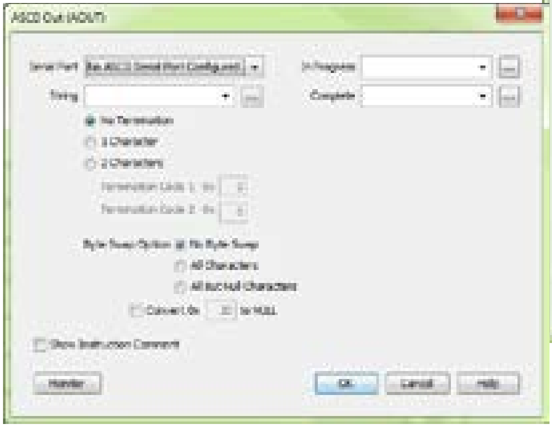
Serial Port: No ASCII Serial Port Configured
Destination
Number of Values Received
In Progress
Complete
Success
First Character Timeout
Inter Character Timeout
Overflow

[✓ Fixed Length]
Number of Characters: 8
Variable Length
Max Number of Characters: 1
[✓ 1 Character]
2 Characters
Termination Code 1: 0x 0
Termination Code 2: 0x 0
First Character Timeout Interval (sec): 0 (0 = None)
Inter Character Timeout Interval (msec): 0 (0 = None)

Byte Swap Option: [✓ No Byte Swap] All Characters / All But Null Characters
Convert NULL to 0x 20
Show Instruction Comment

Monitor | OK | Cancel | Help

The ASCII Out instruction sends text-based data out of the serial port to various devices for control, printing or display.



ASCII Out (AOUT)

Serial Port: No ASCII Serial Port Configured
String
In Progress
Complete

[✓ No Termination]
1 Character
2 Characters
Termination Code 1: 0x 0
Termination Code 2: 0x 0

Byte Swap Option: [✓ No Byte Swap] All Characters / All but Null Characters
Convert 0x 20 to NULL
Show Instruction Comment

Monitor | OK | Cancel | Help

ASCII and Custom Protocol Functionality, continued

While the ASCII In instruction and the ASCII Out instruction can both be used in a project, they are not intended to be used in conjunction with one another. In other words, it is not advisable to use the ASCII Out instruction to send a String to a device that will respond (if the response is needed) and to use the ASCII In instruction to try to receive this data.

The ASCII instruction limitations are:

1. AIN and AOUT cannot be enabled at the same time on the same serial port.
2. When the AOUT completes, the AIN cannot be enabled until the next logic scan.
3. AIN does not buffer data received while the AIN is not active. If a device responds too quickly, some of the response may be lost before the AIN instruction can start receiving data.

Custom Protocol Instructions

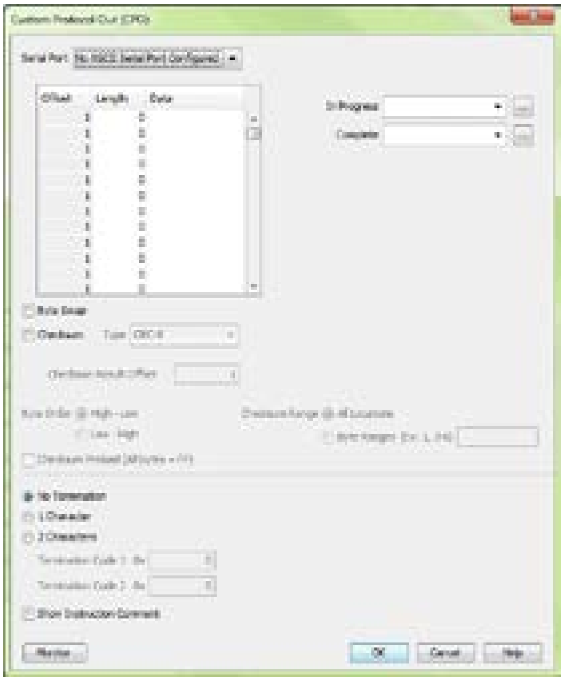
The Custom Protocol is a HEX based protocol used to communicate with devices that do not have the standard Modbus RTU Protocol. There are two instructions used with Custom Protocol communication:

- Custom Protocol Out (CPO)
- Custom Protocol In (CPI)

Custom Protocol Out

The Custom Protocol Out instruction allows the user to send a ‘byte formatted’ packet of data out of the CPU serial port.

Constant values and/or Tag values can be used as the source for data transmitted. There are several formatting options including Byte Swap and Checksum.



ASCII and Custom Protocol Functionality, continued

The Checksum option allows the user to select where in the packet the checksum should be inserted, what type of Checksum (CRC-8 bit, CRC-16 bit, CRC-32 bit, XOR-8 bit, XOR-16 bit and XOR 32 bit), which bytes of the data source should be used in the calculation of the checksum, what the byte order should be of the checksum (if greater than 8-bit) and how to preload the checksum calculation.

If the device requires a different Checksum calculation, this can be done outside of the instruction in other ladder code and the resulting Tag values can be inserted where appropriate in the packet.

Termination characters can also be specified when needed.

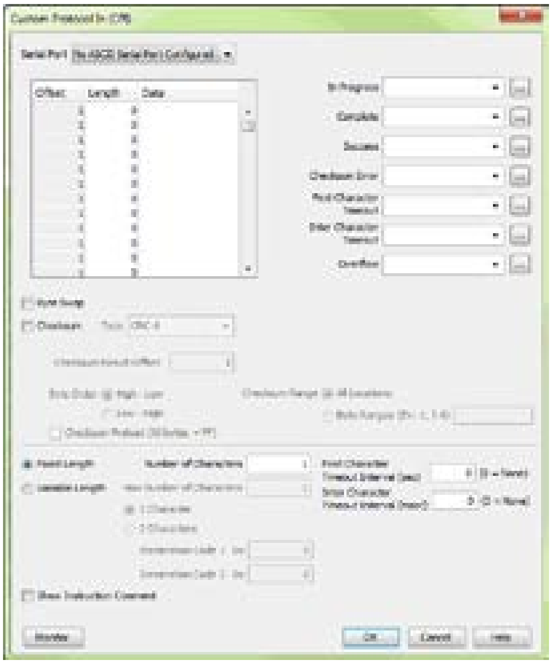
The Custom Protocol Out instruction is for transmission only. If information needs to be received from field devices, the Custom Protocol In instruction will have to be used. Unlike ASCII, the Custom Protocol will buffer the received data. When the Custom Protocol In instruction is executed, it will retrieve any data held in this buffer. Therefore, the lost responses found with ASCII communication do not occur with Custom Protocol communication.

Custom Protocol In

The Custom Protocol In instruction has similar formatting options to the Custom Protocol Out instruction.

The Custom Protocol In instruction will calculate the Checksum of the data packet received based on the criteria specified in the instruction and this will determine the state of the status bits assigned to the instruction. If the Checksum calculation passes based on the criteria specified in the instruction, the “Success” status bit will become true. If the Checksum calculation fails, the “Checksum Error” status bit will become true.

With the CPI instruction, the packet termination must be specified, either in terms of a termination character(s) or a packet length. If a Checksum is expected in the reply, be sure to include this in the Fixed Length value specified.



Communications: Ethernet

TCP and UDP Port Numbers

When doing TCP/IP and UDP/IP communications, there is a Source Port number and Destination Port number for every message. The Client device must be aware of the Destination Port Number(s) the Server device is expecting to see and the Server device must listen for this Destination Port number. After the Server device has received the message with the Destination Port Number it is listening on, it will formulate the return message (if the applications require this) with the Source Port Number from the message sent as its Destination Port Number.

It is important to understand a little about the Port numbering concept because many Ethernet devices, such as routers with firewalls, will block messages with Destination Port numbers that are not configured for that device. Listed below are the default Port Numbers used in the Productivity®1000 system. Some of these are configurable, allowing more flexibility when going through routers in many applications.

Port	Port Number (Decimal Format)	TCP or UDP	Configurable
Programming Software CPU Discovery	8888	UDP	No
Programming Software Connection and Project Transfer	9999	UDP	No
Modbus Client Connections (MRX, MWX, RX and WX instructions)	502	TCP	Yes
Modbus Server Connections	502	TCP	Yes
GS-Drive Discovery	28784	UDP	No
GS-Drive Connection	502	TCP	No
Remote I/O Discovery	8887	UDP	No
Remote I/O Connection	8887	UDP	No
Email Instruction	25	TCP	No
EtherNet/IP	44818	TCP	Yes
EtherNet/IP	2222	UDP	No*
ProNET	18888	UDP	Yes

** Adapters may choose to respond using another port number.*

IP Addressing and Subnetting

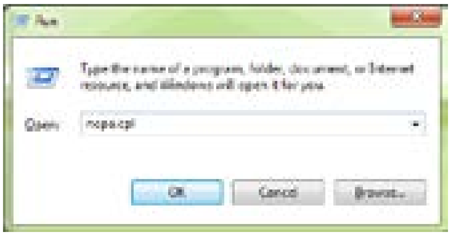
IP Addresses (used in conjunction with the Subnet Mask and Default Gateway address) are used for network routing. This allows for easy and logical separation of networks.

It is outside of the scope of this user manual to explain how IP Addresses and Subnet masks are configured for actual usage. There are many books, documents and tools (Subnet calculators) on the Internet that provide this information. Each facility and network will incorporate their own rules and guidelines for how their networks are to be configured.

PC Setup

For testing and verification purpose, it is recommended that the PC and the CPU be on an isolated Ethernet switch. Configure the PC's network interface card setting as described below.

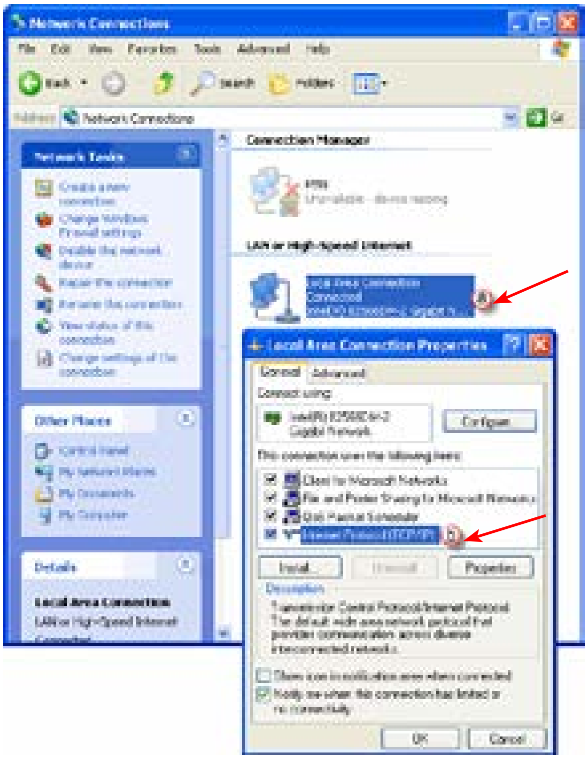
1. Go to Start, then Run, type ncpa.cpl in the Open field and click on OK to bring up the Network Connections dialog.


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open: ncpa.cpl
OK
Cancel
Browse...

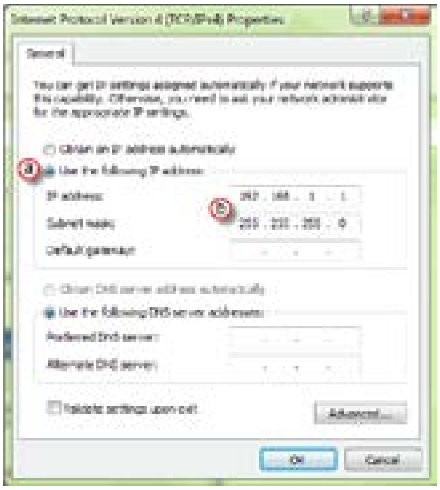
NOTE: *Many system settings on your computer require Administrative privileges. Consult with your IT department for necessary privileges and approvals.*

NOTE: *You should record initial settings prior to making any network configuration changes.*

2. Network Connections
   a. Right click on the Network interface shown in the Network Connections dialog and select Properties. If there is more than one Network Interface on the PC, be sure to choose the one connected to the Ethernet Switch with the CPU on it.
   b. From the Local Area Connection Properties window, highlight the Internet Protocol(TCP/IP) selection and click on Properties.


Network Connections
File Edit View Favorites Tools Advanced Help
Network Tasks
Other Places
Details
Connection Manager
LAN or High-Speed Internet
Local Area Connection
Local Area Connection Properties
General Advanced
Connect using:
Configure
This connection uses the following items:
Client for Microsoft Networks
File and Printer Sharing for Microsoft Networks
QoS Packet Scheduler
Internet Protocol (TCP/IP)
Install
Uninstall
Properties
Description
OK
Cancel

PC Setup, continued


Internet Protocol Version 4 (TCP/IPv4) Properties
General
Obtain an IP address automatically
a
Use the following IP address:
IP address:
b
Subnet mask:
Default gateway:
Obtain DNS server address automatically
Use the following DNS server addresses:
Preferred DNS server:
Alternate DNS server:
Validate settings upon exit
Advanced...
OK
Cancel

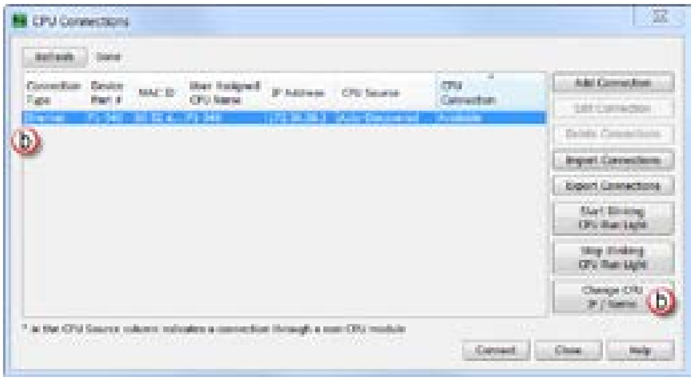
IP address: 192 . 168 . 1 . 1
Subnet mask: 255 . 255 . 255 . 0

3. Internet Protocol (TCP/IP) Properties.
   a. In the Properties window, select Use the following IP address.
   b. Enter an IP Address of 192.168.1.1 and Subnet Mask 255.255.255.0 and select OK. Select OK again on the Local Area Connection Properties window.

CPU Setup

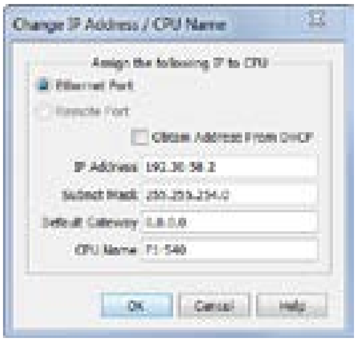
Now configure the CPU’s network IP setting as shown below.

1. Select CPU from the Productivity Suite Software Main Menu and then select Choose CPU from the drop down menu.
2. The CPU Connections window will open as shown below.
   a. Click to highlight the CPU connected to the Ethernet switch.
   b. Select the “Change CPU IP/Name” button.



CPU Setup, continued

3. The Change IP Address/CPU Name window will open as shown below.
   - Enter an IP Address of 192.168.1.2 and Subnet Mask 255.255.0.0 for the CPU's network IP setting and select OK.


Change IP Address / CPU Name

Assign the following IP to CPU

Ethernet Port

Remote Port

Obtain Address From DHCP

IP Address: 192.30.58.2

Subnet Mask: 255.255.254.0

Default Gateway: 0.0.0.0

CPU Name: P1-540

The CPU is now configured with the correct IP Address for connectivity with the PC. The IP Address and Subnet Mask settings will very likely differ from what will be used in the actual application. Consult the Network Administrator of the facility where the CPU will be installed to get the appropriate settings for that network.

TCP Connection Behavior with Modbus TCP and Network Instructions

When performing communications over TCP, a Connection must be established before the applications can transfer data. The connection is typically maintained until the application decides that the connection is no longer needed and then the connection will be severed. Frequent connects and disconnects are not efficient for the Client or the Server and can add unnecessary network traffic. But maintaining connections needlessly is also costly to the Client and Server in terms of processing and memory so this should also be avoided.

The CPU allows user control of Client connections through enabling and disabling the rungs containing Modbus and Network instructions. The MRX, MWX, RX and WX instructions have two options for sending messages: Automatic Poll and Manual Poll.

Automatic Poll sends out messages at a specified rate. When enabled, the instruction performs a TCP connect with the Server device. Once the connection is established, the instruction messages are sent at the rate entered in the poll rate field. This continues until the instruction is disabled. The TCP connection will automatically be severed five seconds after the instruction is disabled.

Manual Poll sends out a message each time the instruction is enabled. Enabling the instruction performs a TCP connect with the Server device and sends the message one time. The TCP connection will automatically be severed five seconds after receiving the reply from the Server device. If the instruction gets another positive edge enable within the five seconds, the message will be sent and the disconnect of the TCP connection will be delayed by an additional five seconds.

Communications Modbus Functionality

Master/Client Function Code and Data Type Support

The following table lists the Modbus data type, the function code and the CPU source data type that is supported when the CPU is the Client or Master on a Modbus TCP or serial connection.

Modbus Client/Master Support (Using MRX and MWX Instructions)

Function Code	Function Name	Modbus 984 Addressing (Zero Based)	Modbus 984 Addressing	Productivity®1000 Tag Types (Data designation or source)
01	Read Coil Status	000000 - 065535	000001 - 065536	Discrete Output (DO)
				Boolean (C)
				Boolean System (SBRW)
02	Read Coil Status	100000 - 165535	100001 - 165536	Discrete Input (DI)
				Boolean (C)
				Boolean System (SBRW)
03	Read Holding Registers	400000 - 465535	400001 - 465536	Integer 8 bit Unsigned (U8)
				Integer 16 bit (S16)
				Integer 16 bit Unsigned (U16)
				Integer 16 bit BCD (B16)
				Integer 32 bit (S32)
				Integer 32 bit BCD (B32)
				Integer 32 bit Float (F32)
				Integer 16 bit System (SWRW)
04	Read Input Registers	300000 - 365535	300001 -365536	Integer 8 bit Unsigned (U8)
				Integer 16 bit (S16)
				Integer 16 bit Unsigned (U16)
				Integer 16 bit BCD (B16)
				Integer 32 bit (S32)
				Integer 32 bit BCD (B32)
				Integer 32 bit Float (F32)
				Integer 16 bit System (SWRW)
05	Write Single Coil	000000 - 065535	000001 - 065536	Discrete Input (DI)
				Discrete Output (DO)
				Boolean (C)
				Boolean System (SBRW)
				Boolean System Read Only (SBR)

Modbus Client/Master Support (Using MRX and MWX Instructions) (continued)				
Function Code	Function Name	Modbus 984 Addressing (Zero Based)	Modbus 984 Addressing	Productivity®1000 Tag Types (Data designation or source)
06	Write Single Register	400000 - 465535	400001 - 465536	Integer 8 bit Unsigned (U8)
				Integer 16 bit (S16)
				Integer 16 bit Unsigned (U16)
				Integer 16 bit BCD (B16)
				Integer 32 bit (S32)
				Integer 32 bit BCD (B32)
				Integer 32 bit Float (F32)
				Integer 16 bit System (SWRW)
				Integer 16 bit System Read Only (SWR)
15	Write Multiple Coils	000000 - 065535	000001 - 065536	Discrete Input (DI)
				Discrete Output (DO)
				Boolean (C)
				Boolean System (SBRW)
				Boolean System Read Only (SBR)
16	Write Multiple Registers	400000 - 465535	400001 - 465536	Integer 8 bit Unsigned (U8)
				Integer 16 bit (S16)
				Integer 16 bit Unsigned (U16)
				Integer 16 bit BCD (B16)
				Integer 32 bit (S32)
				Integer 32 bit BCD (B32)
				Integer 32 bit Float (F32)
				Integer 16 bit System (SWRW)
				Integer 16 bit System Read Only (SWR)

Slave/Server Function Code and Data Type Support

The following table lists the Modbus data type, the function code and the CPU source data type that is supported when the CPU is the Server or Slave on a Modbus TCP or serial connection.

Modbus Server/Slave Support			
Function Code	Function Name	Modbus 984 Addressing	Productivity®1000 Tag Types (Data designation or source)
01	Read Coil Status	000001 - 065536	Discrete Output (DO)
			Boolean (C)
			Boolean System (SBRW)
02	Read Coil Status	100001 - 165536	Discrete Input (DI)
			Boolean System Read Only (SBR)
03	Read Holding Registers	400001 - 465536	Integer 8 bit Unsigned (U8)
			Integer 16 bit (S16)
			Integer 16 bit Unsigned (U16)
			Integer 16 bit BCD (B16)
			Integer 32 bit (S32)
			Integer 32 bit BCD (B32)
			Integer 32 bit Float (F32)
			Integer 16 bit System (SWRW)
			String
04	Read Input Registers	300001 -365536	Analog Input, Integer 32 bit (AIS32)
			Analog Input, Float 32 bit (AIF32)
			Integer 16 bit System Read Only (SWR)
05	Write Single Coil	000001 - 065536	Discrete Output (DO)
			Boolean (C)
			Boolean System (SBRW)
06	Write Single Register	400001 - 465536	Integer 8 bit Unsigned (U8)
			Integer 16 bit (S16)
			Integer 16 bit Unsigned (U16)
			Integer 16 bit BCD (B16)
			Integer 32 bit (S32)
			Integer 32 bit BCD (B32)
			Integer 32 bit Float (F32)
			Integer 16 bit System (SWRW)
			Integer 16 bit System Read Only (SBR)
			String
15	Write Multiple Coils	000001 - 065536	Discrete Output (DO)
			Boolean (C)
			Boolean System (SBRW)

Modbus Server/Slave Support (continued)			
Function Code	Function Name	Modbus 984 Addressing	Productivity®1000 Tag Types (Data designation or source)
16	Write Multiple Registers	400001 - 465536	Integer 8 bit Unsigned (U8)
			Integer 16 bit (S16)
			Integer 16 bit Unsigned (U16)
			Integer 16 bit BCD (B16)
			Integer 32 bit (S32)
			Integer 32 bit BCD (B32)
			Integer 32 bit Float (F32)
			Integer 16 bit System (SWRW)
			Integer 16 bit System Read Only (SBR)
			String

Assigning Modbus Addresses to Tags

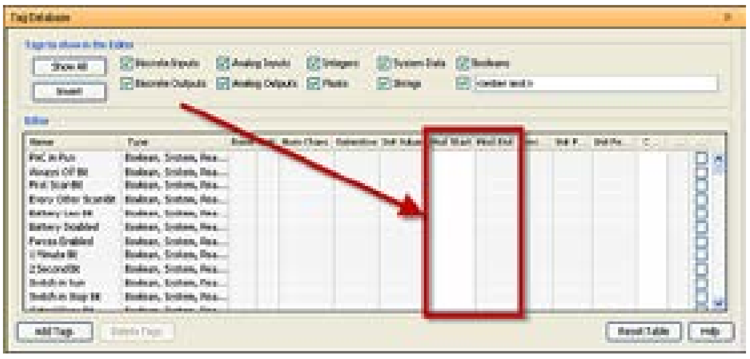
There are many different data types in the CPU. Because of this, the Modbus addresses need to be mapped to the various tag data types in the CPU.

There are two ways to map Modbus addresses to Tags in the Programming software:

- Modbus mapping in Tag Database window.
- Modbus mapping when creating Tags.

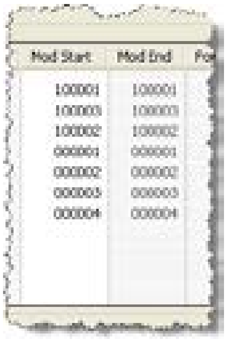
1. Modbus mapping in Tag Database window:

   - There are only two data sizes in the Modbus protocol: bits and words. In the CPU, there are multiple size types, so it is sometimes necessary to map multiple Modbus addresses to a single Tag entity. There are also array data structures in the CPU. When Modbus addresses are mapped to arrays, they will be mapped as a contiguous block of addresses. This is, in fact, the most efficient method to handle Modbus communications.
   - In the Tag Database window, there are two columns named “Mod Start” and “Mod End”. To map a Modbus address to a tag in the Tag Database window, simply double-click in the Mod Start field for the Tag.



Assigning Modbus Addresses, continued

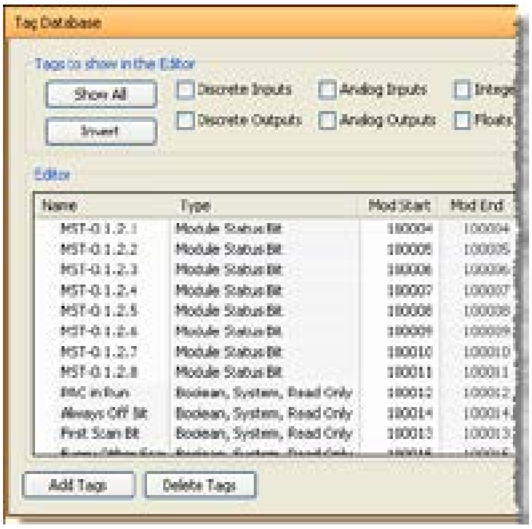
- When this is done, two values will appear in the field The left most value is the Modbus data type. This is fixed based upon the tag data type. The chart below indicates the four different Modbus data types in the 984 addressing scheme.



Mod Start	Mod End
100001	100001
100003	100003
100002	100002
000001	000001
000002	000002
000003	000003
000004	000004

Address Identifier	Modbus 984 Address Type
0xxxxx	Coil (Read/Write bit)
1xxxxx	Input (Read Only bit)
3xxxxx	Input Register (Read Only 16 bit word)
4xxxxx	Holding Register (Read/Write 16 bit word)

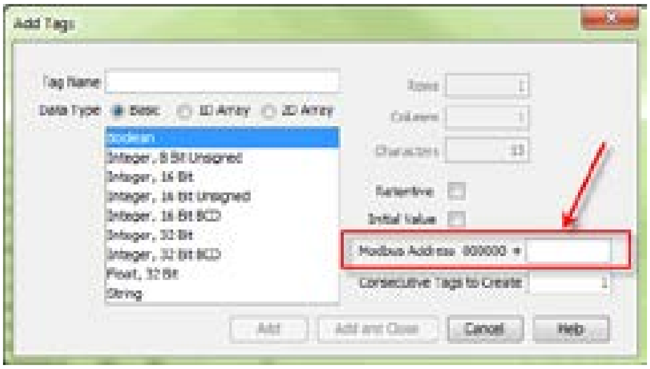
The right most value in the "Mod Start" field is the address offset (range is from 1 – 65535). You can accept the value that is pre-filled or the value can be changed. The software automatically pre-fills the address offset with the next available address.

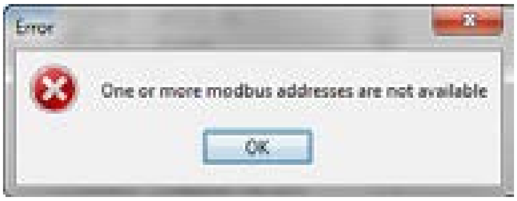


Name	Type	Mod Start	Mod End
MST-0.1.2.1	Module Status Bit	100004	100004
MST-0.1.2.2	Module Status Bit	100005	100005
MST-0.1.2.3	Module Status Bit	100006	100006
MST-0.1.2.4	Module Status Bit	100007	100007
MST-0.1.2.5	Module Status Bit	100008	100008
MST-0.1.2.6	Module Status Bit	100009	100009
MST-0.1.2.7	Module Status Bit	100010	100010
MST-0.1.2.8	Module Status Bit	100011	100011
PAC in Run	Boolean, System, Read Only	100012	100012
Always Off Bit	Boolean, System, Read Only	100014	100014
First Scan Bit	Boolean, System, Read Only	100013	100013

Assigning Modbus Addresses, continued

2. Modbus mapping when creating Tags:
   - Modbus addresses can be assigned to Tags as they are created in the Tag Database.
   - Type in the Modbus offset value when entering the Tag Name and Data Type.
   - If the address is already assigned, a warning message will appear.



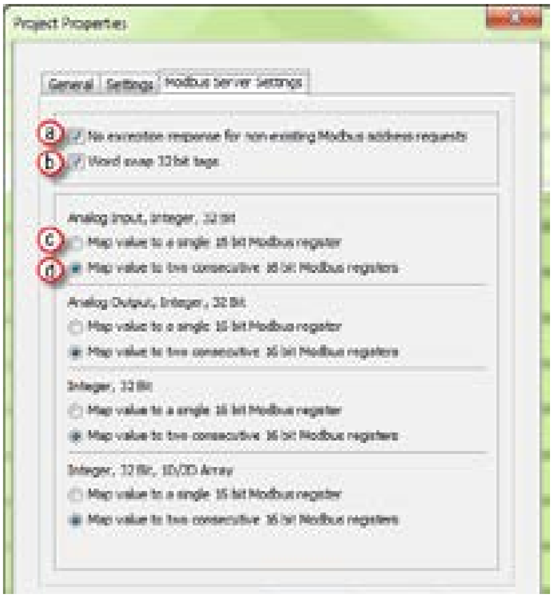


One or more modbus addresses are not available

Modbus Options

The Modbus protocol does not have a specific method outlined for data types outside of bits and 16-bit words. Most systems now have 32-bit data types. In order to transport 32-bit data types across Modbus, they must be placed into two Modbus 16-bit registers. Unfortunately, some devices do not support this method, so sometimes incompatibilities in the order in which the 16-bit high word and low word are handled between devices persist.

In order to alleviate this situation, there are some options for handling this in the programming software. To find the Modbus Address options, go to File and click on Project Properties and then click on the “Modbus Server Settings” tab.



[✓] No exception response for non-existing Modbus address requests
[✓] Word swap 32 bit tags

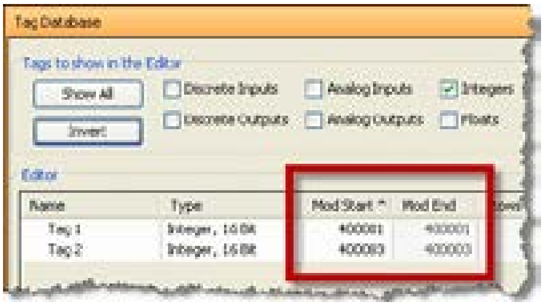
Analog Input, Integer, 32 Bit
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

Analog Output, Integer, 32 Bit
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

Integer, 32 Bit
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

Integer, 32 Bit, 1D/2D Array
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

ⓐ No exception response for non-existing Modbus address requests: Because the Modbus addresses can be manually assigned to tags, it is possible that gaps can occur in the Modbus address mapping. For example: Tag 1 has Modbus address 400001 assigned to it and Tag 2 has Modbus address 400003 assigned to it.

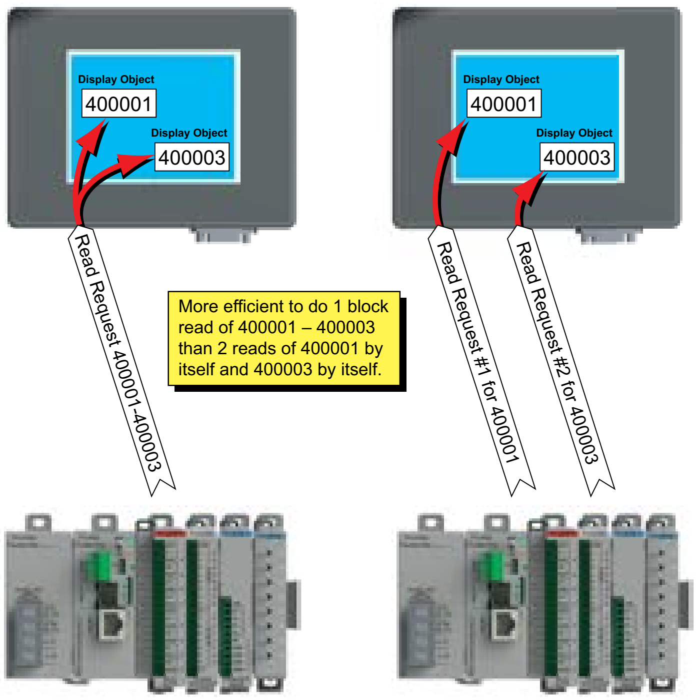


[ ] Discrete Inputs [ ] Analog Inputs [✓] Integers
[ ] Discrete Outputs [ ] Analog Outputs [ ] Floats

Name	Type	Mod Start ^	Mod End
Tag 1	Integer, 16 Bit	400001	400001
Tag 2	Integer, 16 Bit	400003	400003

Modbus Options, continued

Most Modbus Master/Client devices will attempt to optimize their data requests to a Modbus Slave/Server device by requesting blocks of data instead of individual registers. In the case mentioned previously, most Modbus masters would send one read request starting at 400001 and a size of three instead of sending two read requests starting at 400001 with size one and 400003 with size one as shown below.


Display Object
400001
Display Object
400003
Display Object
400001
Display Object
400003
Read Request 400001-400003
More efficient to do 1 block read of 400001 – 400003 than 2 reads of 400001 by itself and 400003 by itself.
Read Request #1 for 400001
Read Request #2 for 400003

In the example shown above on left, a Modbus Slave/Server device should give an exception response since there is no Modbus Address of 400002 in the device. This method can cause a lot of inefficiencies. By selecting the “No exception response for non-existing Modbus address requests” option, the CPU will not give an exception response to the request. Note that if Modbus address 400002 by itself were requested it would give an exception response.

b **Word swap 32 bit tags:** (S-32, AIS-32, AOS-32, F-32, FI-32, FO-32):

Word swap allows the word order of 32-bit tags to be changed when sending the values across Modbus. The default selection is on, which returns the data low word first.

Tag 1 (Integer, 32-Bit) = 305,419,896 (hex = 0x12345678)

Tag1 Modbus address = 400001, 400002

	Low Word First	High Word Last
Modbus reply for Tag 1 (Word Swap ON) = 01 03 04	56 78	12 34

	High Word First	Low Word Last
Modbus reply for Tag 1 (Word Swap OFF) = 01 03 04	12 34	56 78

Modbus Options, continued

ⓒ Map value to a single 16 bit Modbus register:

This option allows for compatibility with devices that do not support 32-bit Modbus functionality. This option can be selected individually for the Analog Input and Output Signed 32 data types and the Internal Signed 32 data types, including the array form of these data types. This function is only useful when the value contained in a 32-bit tag does not exceed a signed 15-bit value (32,765).

Tag 1 (Integer, 32-Bit) = 22136 (hex = 0x00005678)

With "Map value to a single 16 bit Modbus register" turned OFF =

Tag 1 Modbus address = 400001, 400002

Modbus reply for Tag1 (Word Swap ON) = 01 03 04 56 78 00 00

With "Map value to a single 16 bit Modbus register" turned ON =

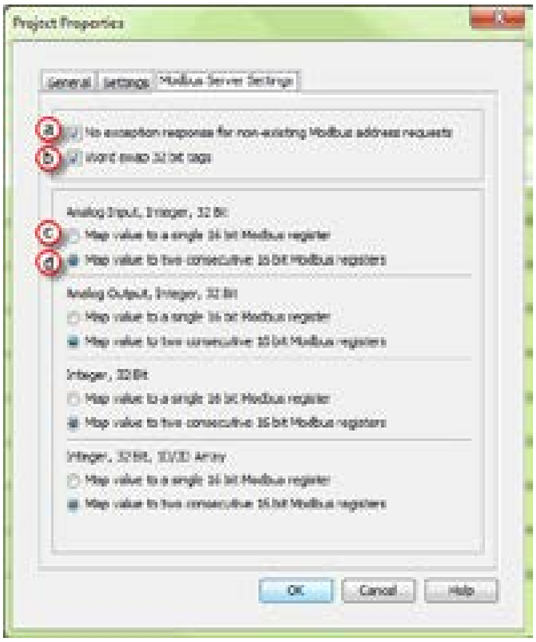
Tag 1 Modbus address = 400001

Modbus reply for Tag1 = 01 03 02 56 78

ⓓ Map value to two consecutive 16-bit Modbus registers:

Allows for 32-bit data types to be mapped to two consecutive 16-bit registers. This option is selected as default.

All of the options in the "Modbus Address" tab of the Project Properties only apply to the Modbus Slave/Server functionality. Similar options are available for the Modbus Master/ Client functions as well and are available in the MRX and MWX Modbus instructions.



a [✓] No exception response for non-existing Modbus address requests
b [✓] Word swap 32 bit tags

Analog Input, Integer, 32 Bit
c ○ Map value to a single 16 bit Modbus register
d ● Map value to two consecutive 16 bit Modbus registers

Analog Output, Integer, 32 Bit
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

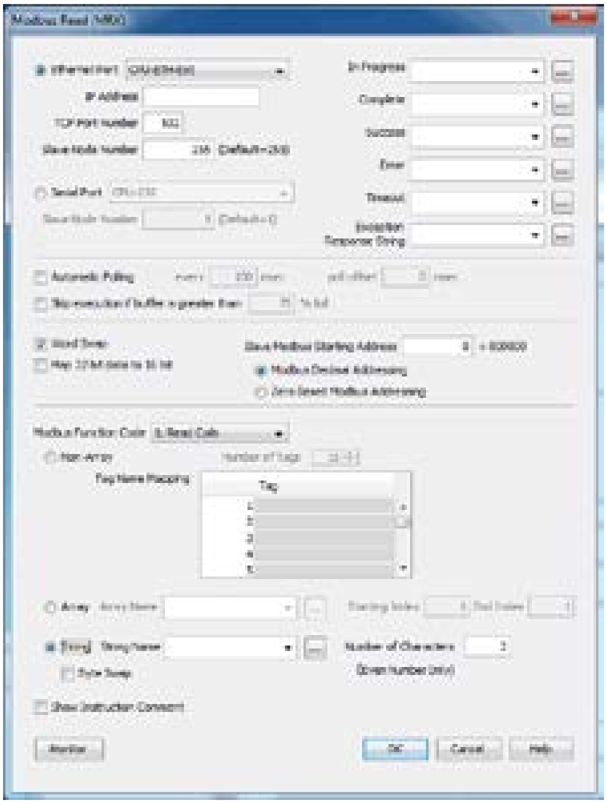
Integer, 32 Bit
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

Integer, 32 Bit, 1D/2D Array
○ Map value to a single 16 bit Modbus register
● Map value to two consecutive 16 bit Modbus registers

Modbus Instructions

To read or set data in other Modbus Slave/Server devices, there are two instructions available in the programming software, Modbus Read and Modbus Write.

- The Modbus Read (MRX) instruction is used to read data from other Modbus devices into Tags of the CPU.


Modbus Read (MRX)

Ethernet Port [illegible]
IP Address
TCP Port Number 502
Slave Node Number 255 (Default=255)
Serial Port [illegible]
Slave Node Number 1 (Default=1)
In Progress
Complete
Success
Error
Timeout
Exception Response String
Automatic Polling [illegible] msec [illegible]
Skip execution if buffer is greater than [illegible] % full
[✓] Word Swap
Map 32 bit data to 16 bit
Slave Modbus Starting Address 1 = 000001
Modbus Decimal Addressing
Zero Based Modbus Addressing
Modbus Function Code 1: Read Coils
Non-Array
Number of Tags
Tag Name Mapping

	Tag
1	
2	
3	
4	
5	

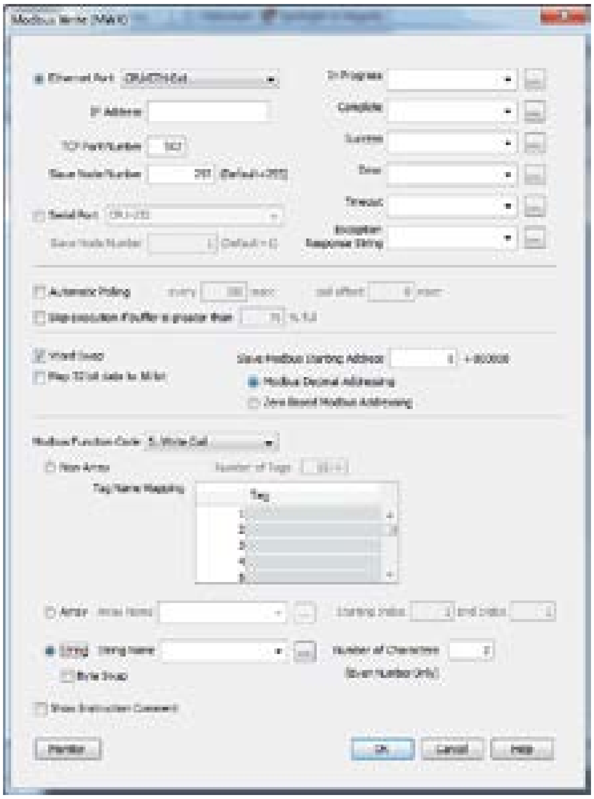
Array
Array Name
Starting Index
End Index
String
String Name
Number of Characters
(Even Number Only)
Byte Swap
Show Instruction Comment
Monitor
OK
Cancel
Help

- The MRX instruction can be used for Modbus TCP or Modbus RTU. There are several status bits that can be used to determine whether the read message was successful and if it was not, the reason why.

Modbus Instructions, continued

There is an “Automatic Polling” feature in the instruction to make it easier to read a device on a pre-determined poll rate. There is also a “poll offset” field that can be used when simultaneous instructions are enabled with the Automatic Polling feature to help stagger the flow of messages being sent to the network.

- The Modbus Write (MWX) instruction is very similar in layout and configuration to the MRX instruction. It is used to write values to a Modbus device from the tags in the CPU.



Modbus Write (MWX)

Ethernet Port [illegible] · IP Address · TCP Port Number 502 · Slave Node Number 255 (Default=255)

Serial Port [illegible] · Slave Node Number 1 (Default=1)

In Progress · Complete · Success · Error · Timeout · Exception Response String

Automatic Polling every 100 msec · poll offset 0 msec

Skip execution if buffer is greater than 75 % full

[✓] Word Swap

Map 32 bit data to 16 bit

Slave Modbus Starting Address 1 (+400000)

[✓ Modbus Decimal Addressing] · Zero Based Modbus Addressing

Modbus Function Code 5, Write Coil

Non Array · Number of Tags · Tag Name Mapping

	Tag
1	
2	
3	
4	
5	

Array · Array Name · Starting Index 1 · End Index 1

[✓ String] · String Name · Number of Characters 2 (Even Number Only)

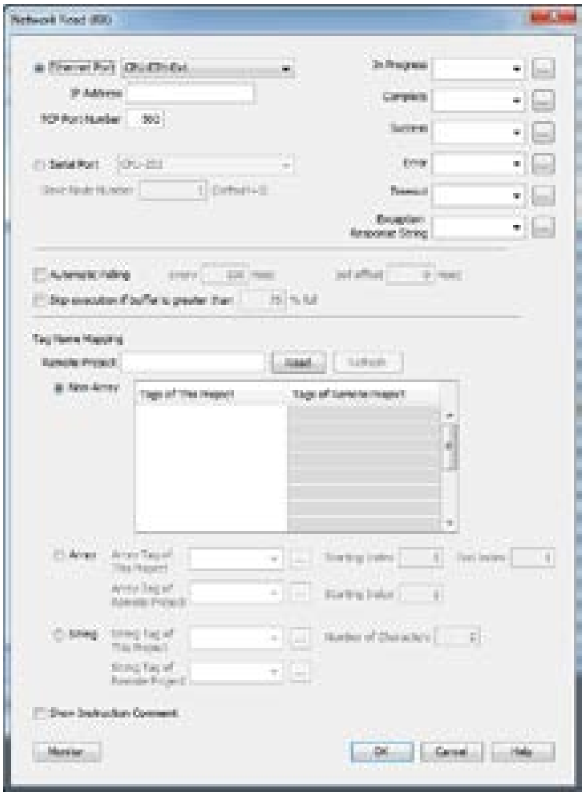
Byte Swap

Show Instruction Comment

- The MWX operates very similarly to the MRX instruction. There are also many status bits to indicate the success or reason for failure when sending a message.
- The Automatic Polling option is also available to the MWX instruction, although greater care should be taken when using this feature in this instruction. This is explained in better detail in the “Message Queue” section.

Network Instructions

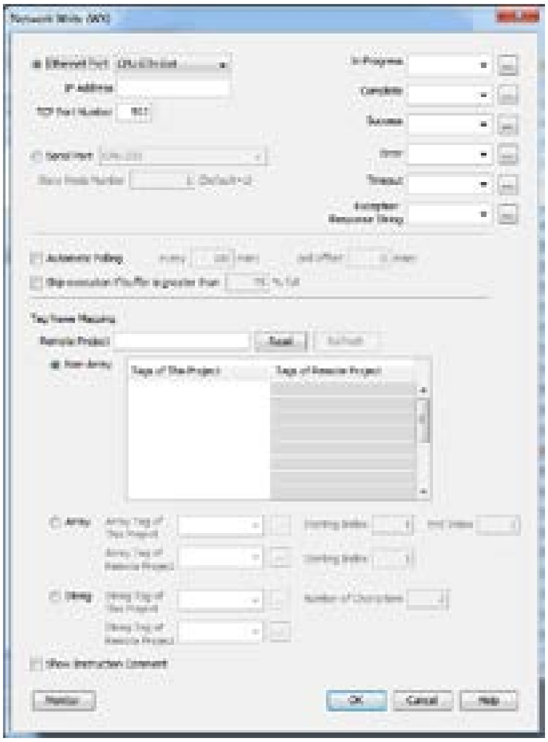
The Network Read (RX) and Network Write (WX) instructions are used to communicate to other CPU's. They are very similar in operation to the MRX and MWX instructions but they target Tag Names instead of Modbus addresses in the other CPU. There is also a significant performance gain in using the RX and WX instructions when communicating to other CPU's as opposed to using the MRX and MWX instructions.



The same status bits are available in the RX instruction as in the MRX instruction and operate in the same manner. The greatest difference in the RX versus the MRX is that with the RX, the Tag Name in the target CPU can be referenced directly and does not need a corresponding Modbus address. The way this is accomplished is by mapping local and remote tagnames together within the local CPU's RX instruction. Once the instruction is set up to read a remote project, the "Tags of Remote Project" or "Array Tags of Remote Project" drop down lists will be accessible. Map the Tag of the Remote project to a Tag in the Local project to read this data.

Network Instructions, continued

The WX instruction operates in the same manner except that the data from the Local tags will be written into the Tags of the remote project. No Modbus mapping is required.



NOTE: *The PC programming software project for the Remote CPU must be accessible by the PC running the programming software for the Local project.*

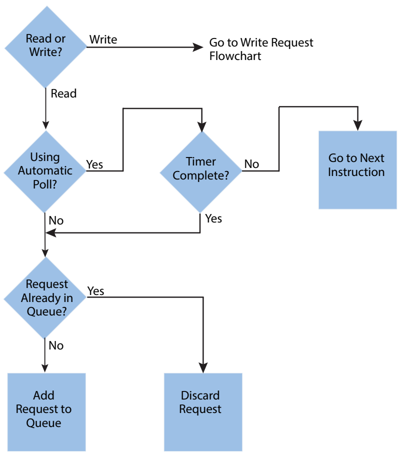
Automatic Poll versus Manual Polling and Interlocking

In many cases when performing multiple communications requests to other devices, the message flow must be explicitly controlled in ladder code so that a message is not sent while another one is in operation. This usually requires writing 'interlocking' code between the instructions which typically involves the use of timers and shift registers, etc. Sometimes this is necessary because of the application but in other cases where the CPU just wants to read changing values from other devices and the frequency of that update is not critical it would be much more efficient to skip the unnecessary code complexity of interlocking.

The desire to make it easier to communicate to other devices brought about the "Automatic Polling" feature and the "Message Queue" in the CPU. The Automatic Polling feature allows the user to choose the rate at which messages are sent without having to use a separate timer and enabling logic. The 'Message Queue' allows the user to stage the messages from the ladder code to go out to each physical communications port without requiring interlocking logic.

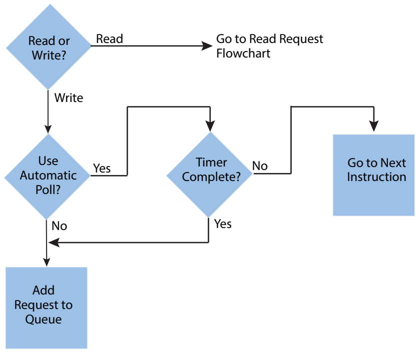
Network Instructions, continued

Read Request Flowchart



The implementation of how the message queue works is slightly different based on whether the request is a read request or a write request.

Write Request Flowchart



Write requests will fill the queue much faster than read requests. That's why it is advisable to carefully choose when doing write requests whether to use the "Automatic Poll" feature or to manually send write requests only when needed (data to write has changed). When designing a system, it is important to know the total time it takes to send a request and get a reply for each target device. The Poll time should be longer than this time. The longer the poll time can be, within tolerance of the application, the better the overall network performance. So for efficiency in programming and for the best possible performance for the system, conservative poll rates should be used when utilizing the "Automatic Poll" feature.

There is also a "Poll offset" field in the communications instructions. This helps prevent the instructions from being queued all at the same time. When the CPU project starts, a master timer begins. The ladder scan will look to see if the instruction is enabled. If it is enabled, it will begin the Automatic Poll timer at the specified poll offset value from the master time clock.

Message Queue

If the application requires more explicit, orderly control of each message sent to the devices, turn off the “Automatic Poll” feature. Using the instruction’s status bits, logically control each message as required.

All of the above explains how messages get into the “queue”. There are several factors involved with how each queue (1 for each physical port) is emptied.

- Serial port queues: The serial port queues empty slower than the Ethernet port queues, not just because of the hardware speed itself but because of the nature of serial communications. Each request sent must wait for a response or a timeout (whichever comes first). Once the reply is received for a request or a timeout has occurred, the next item in the list can be sent. So the response time of the slave devices on the network will largely affect the speed at which the queue fills and empties.
- Ethernet port queues: The Ethernet port queue can empty faster because when sending requests to multiple devices, the CPU does not have to wait on a response from one device before sending a request to another device due to the inherent nature of the Ethernet hardware. However, sending multiple requests to the same Ethernet device does necessitate that the CPU waits for a response from the first request before sending another request to that same device.

Another difference in the Ethernet port queue versus the Serial port queue spawns from the TCP ‘connection’ based behavior of Modbus TCP. If a TCP connection is lost to a device and there are still requests in the queue for that device, those requests will be dropped from the queue. There are three ways this can happen:

1. If a TCP timeout occurs (server device fails to respond within specified timeout value), the TCP connection is lost.
2. If the server device closes the connection, then all of the requests will be dropped.
3. And, finally, if all rungs with communications instructions to a device are disabled for five seconds, the CPU will drop the TCP connection for that device in order to free up valuable resources that could be used elsewhere in the system.

This is another factor that should be considered when designing the system. If it is imperative that no message be lost when communicating to a device, each instruction should be explicitly handled one by one (interlocking logic).

EtherNet/IP for the Productivity Series

Terminology Definitions

A lot of terminology associated with EtherNet/IP is not always clear. Some of these terms are listed below along with their respective definitions.

1. Scanner: This is the term used to describe the device that initiates the EtherNet/IP sessions. The Scanner is sometimes referred to as the “Originator” as well. In more standard Ethernet terms, the Scanner would often be called the “Client”.
2. Adapter: This is the device that responds to the EtherNet/IP communications that are initiated by the Scanner. The Adapter is also known as the “Target” as well. Typically, the Adapter is an Ethernet “Server”.
3. Object: In EtherNet/IP, an Object is a representation of a defined set of Ethernet connections, behaviors, services and data attributes. There are standard objects and there are custom defined objects as well. See Object Modeling example below.
4. Class: A Class is a set of Objects that are related in some fashion. See Object Modeling example below.
5. Instance: An Instance is an actual, usable manifestation of an Object. See Object Modeling example below.
6. Attributes: Attributes are the specific items within an Object Class. The category of Attributes should be the same for all Instances of an Object but the actual Attribute itself might vary. See Object Modeling example below.
7. Connection Point: A Connection Point value is the “Class Code” reference for a data block. This value is required for access to input and output data in IO Messaging. It is typically defined for each input and output data block by the Adapter device manufacturer.
8. IO Messaging: IO Messaging (also called “Implicit Messaging”) is a method of reading and writing blocks of data without defining the Connection Point and size for each block transfer. The Connection Point, size and transfer rate (RPI) are defined at the beginning and then the data blocks are transferred at the specified intervals.
9. Explicit Messaging: This method of reading or writing data requires that each message defines the type of data and size of data needed for each request.

Object Modeling Example:

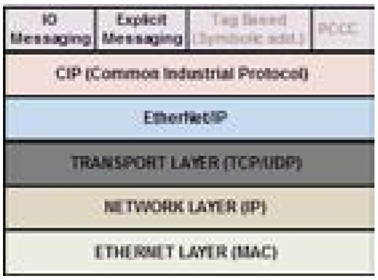
Class ------- Definition of Automobile

Attributes -- Make, Model, etc...

Object ------ A Ford Mustang

Instance ----Sally’s Ford Mustang

Network Layer Chart



The diagram above illustrates the OSI seven layer model and how EtherNet/IP fits into this model. In general, there are three basic layers for sending and receiving data in the EtherNet/IP protocol:

- EtherNet/IP layer (Register Session, etc...)
- CIP layer (CIP Forward Open, etc...)
- The uppermost layer,which contains several different types of messaging.

The ODVA (Open DeviceNet Vendor Association) specification defines many different types of messaging that reside on the CIP layer. Two types of messaging supported in the phase 1 release of the Productivity Series EtherNet/IP protocol are I/O Messaging and Explicit Messaging. I/O Messaging is accomplished through a Class 1 Connection and Explicit Messaging can be accomplished through a Class 3 Connection or an Unconnected Message.

Tag Based Messaging (used for reading and writing values to Allen Bradley Control and CompactLogix PLCs) and PCCC (used for reading and writing values to Allen Bradley MicroLogix and SLC PLCs) are planned for subsequent phases of this protocol.

EtherNet/IP Data

When doing I/O Messaging, the data that is transported is defined as “Input” data and “Output” data. Don’t confuse this type of data with what most PLCs define as Input data and Output data. In most PLCs, Inputs are typically associated with an Input module that reads points from real word devices. Outputs are typically associated with an Output module that turns off and on real word devices.

In I/O Messaging, Input data is data that is sent from the target device back to the Originator or to multiple devices that are listening (multicast messages). Output data is data that is sent from the Target device. This data may or may not be connected to real word devices. That is completely dependent upon the Adapter device. For example: When the Productivity® 1000 is configured as an EtherNet/IP Adapter device, the Input data and Output data is defined in internal data arrays and does not directly tie to any Input and Output point to the real world. If it is desired to tie these array elements to real word devices, that must be accomplished in code by Copy commands (or other instructions).

NOTE: *The Scanner (originator) in the P1000 will only accept messages from an Adapter (target) device with an established connection with a Scanner. The Adapter (target) in the P1000 will respond back to a Scanner (originator) in the method (Multicast or Unicast) that is sent in the forward open message from the Scanner (originator).*

Class 1 and Class 3 Connections

What are they and how are they best used?

- Class 1 Connection is the transport mechanism that IO Messaging uses to send data. The basic concept is that data is sent in one direction: the Originator sends Output data in a Unicast UDP message to the Target and the Target sends Input data in either a Unicast message back to the Originator or Multicast UDP messages to multiple devices. The Input data and Output data messages have no relationship to each other. This method works well for Remote I/O type data and is very efficient due to little overhead and reduced handshaking messages on the wire. Class 3 Connection is one of the mechanisms that Explicit messaging uses. Class 3 messaging uses TCP messages unlike Class 1. Each Class 3 request has a header that defines the type of data requested as well as the size requested. It allows for more flexibility in messaging but does create additional overhead.

NOTE: *Explicit messaging can be accomplished with unconnected messages as well for more infrequent requests. Explicit messaging is a slower performing method of communications but it typically allows for more flexibility and control when the situation requires it.*

When can the P1000 CPU use Class 1 or Class 3 Connections?

- Class 1 and Class 3 Connections can be accomplished with the Productivity®1000 CPU as an Adapter or as a Scanner or both simultaneously.

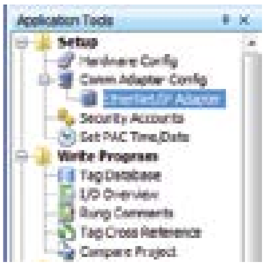
How many connections can the Productivity1000 support for EtherNet/IP?

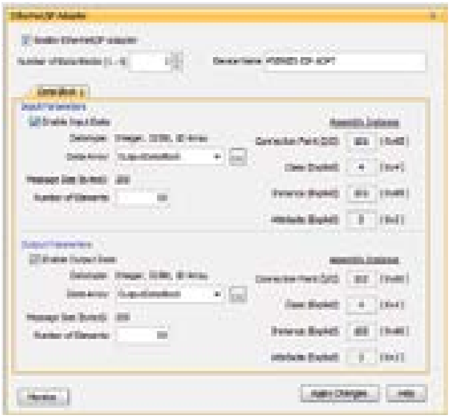
- 4 - TCP
- 4 - EtherNet IP
- 4 - CIP (Up to 4 CIP connections are allowed per EtherNet/IP connection. Therefore, if one device can support 4 CIP connections then you can have up to a total of 16 CIP connections using 4 devices)

Setup Example 1: Productivity®1000 as EtherNet/IP Adapter

The Adapter setup is accomplished through the EtherNet/IP Adapter setup under the Comm Adapter Config section of the Setup menu as seen on right.

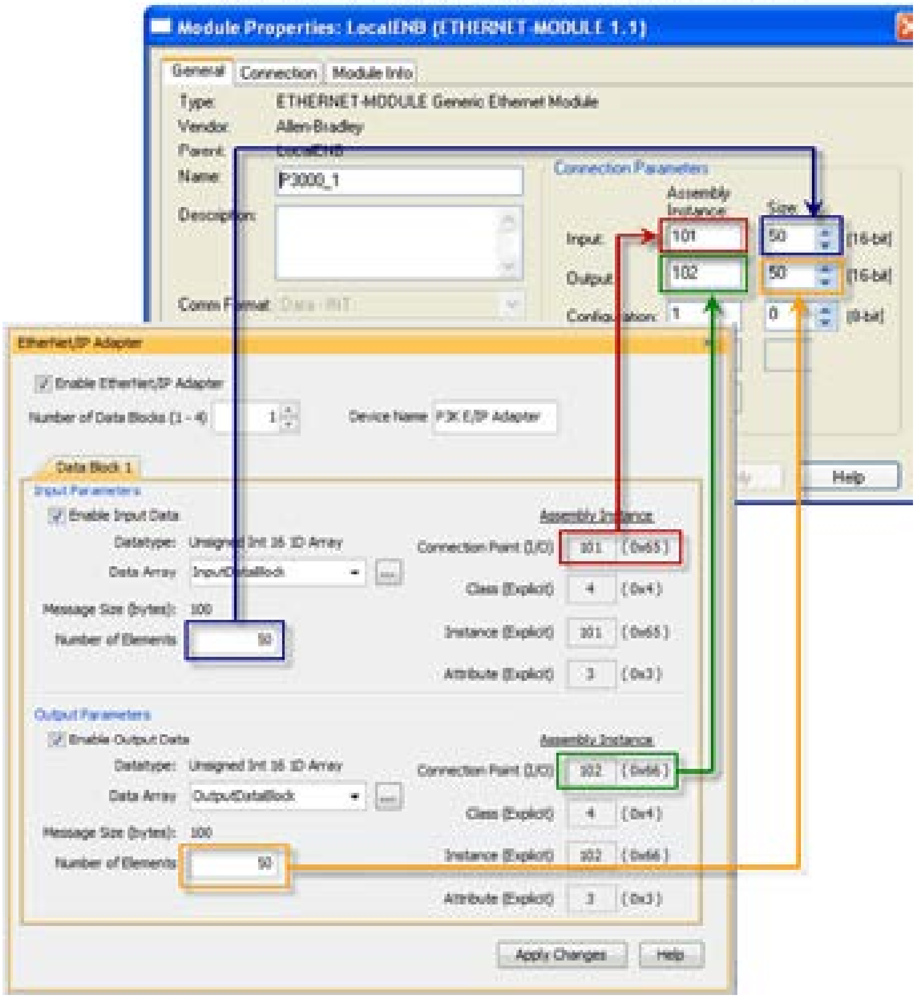
When the EtherNet/IP Adapter is selected from the menu, the window shown here will open.





Fill in the required parameters and once configured these parameters will be used to configure the Scanner side as shown in the examples below. The first example shows how to setup a Class 1 IO Message connection from a 3rd party EtherNet/IP Scanner device (an Allen Bradley PLC).

RS Logix5000 Control/CompactLogix Generic Ethernet Device Setup



Module Properties: LocalENB (ETHERNET-MODULE 1.1)

Type: ETHERNET-MODULE Generic Ethernet Module
Vendor: Allen-Bradley
Parent: LocalENB
Name: P3000_1
Description:
Comm Format: Data - INT

Connection Parameters

	Assembly Instance	Size	
Input:	101	50	(16-bit)
Output:	102	50	(16-bit)
Configuration:	1	0	(8-bit)

EtherNet/IP Adapter

☑ Enable EtherNet/IP Adapter
Number of Data Blocks (1 - 4): 1
Device Name: P3K E/IP Adapter

Data Block 1

Input Parameters
☑ Enable Input Data
Datatype: Unsigned Int 16 1D Array
Data Array: InputDataBlock
Message Size (bytes): 100
Number of Elements: 50

	Assembly Instance	
Connection Point (I/O)	101	(0x65)
Class (Explicit)	4	(0x4)
Instance (Explicit)	101	(0x65)
Attribute (Explicit)	3	(0x3)

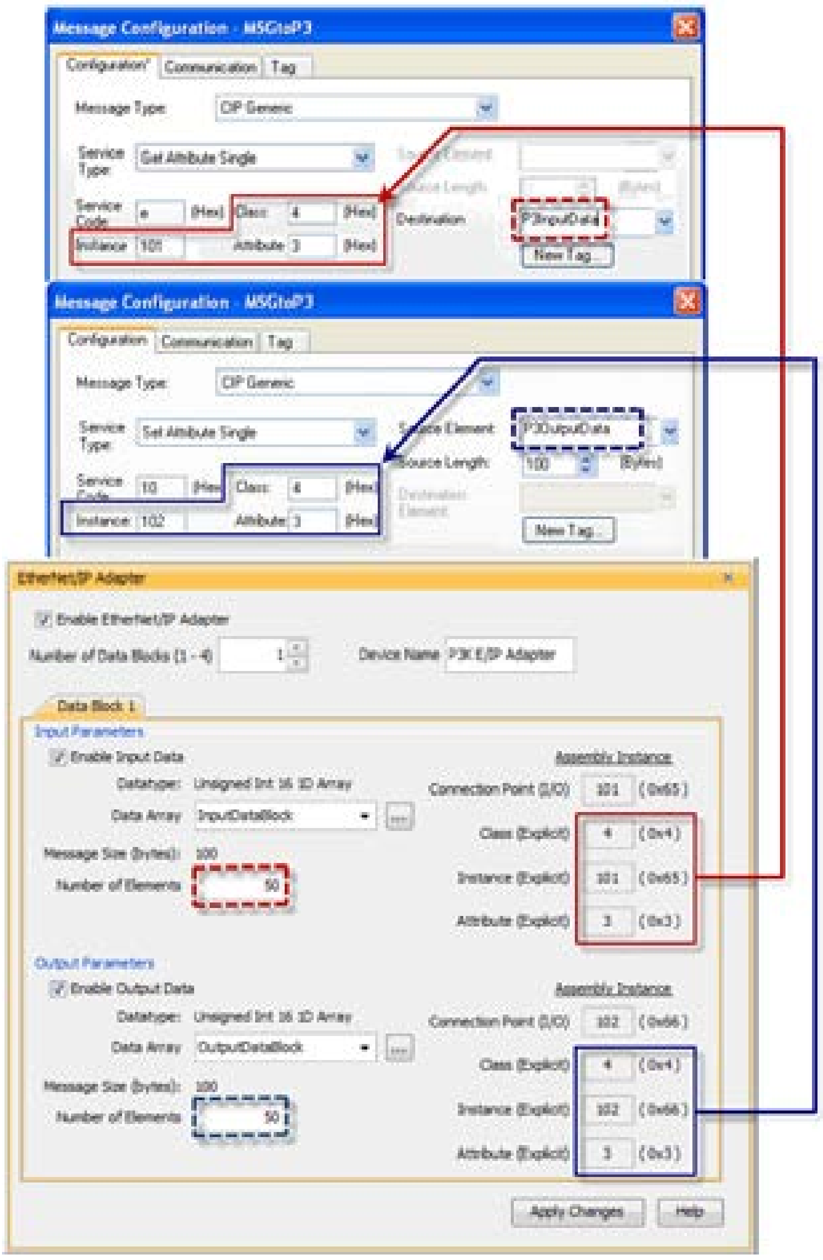
Output Parameters
☑ Enable Output Data
Datatype: Unsigned Int 16 1D Array
Data Array: OutputDataBlock
Message Size (bytes): 100
Number of Elements: 50

	Assembly Instance	
Connection Point (I/O)	102	(0x66)
Class (Explicit)	4	(0x4)
Instance (Explicit)	102	(0x66)
Attribute (Explicit)	3	(0x3)

The following example shows how a Class 3 Explicit Message might be accomplished from a 3rd party device (Allen Bradley PLC). As you can see the Input Data must be retrieved in one connection or message and the output data in another. Remember that Class 3 messaging is not as efficient in protocol messaging as Class 1 but it does allow for granular control.

NOTE: *In this example, size configuration is not shown on the Scanner side. The tag created for the Destination must be large enough to contain the data requested (shown with dashed boxes).*

RS Logix 5000 MSG instruction for Control/CompactLogix



Message Type: CIP Generic
Service Type: Get Attribute Single
Service Code: e (Hex) Class: 4 (Hex)
Instance: 101 Attribute: 3 (Hex)
Destination: P3InputData

Message Type: CIP Generic
Service Type: Set Attribute Single
Service Code: 10 (Hex) Class: 4 (Hex)
Instance: 102 Attribute: 3 (Hex)
Source Element: P3OutputData
Source Length: 100 (Bytes)
Destination Element:

✓ Enable EtherNet/IP Adapter
Number of Data Blocks (1 - 4): 1
Device Name: P3K E/IP Adapter

Input Parameters
✓ Enable Input Data
Datatype: Unsigned Int 16 1D Array
Data Array: InputDataBlock
Message Size (bytes): 100
Number of Elements: 50

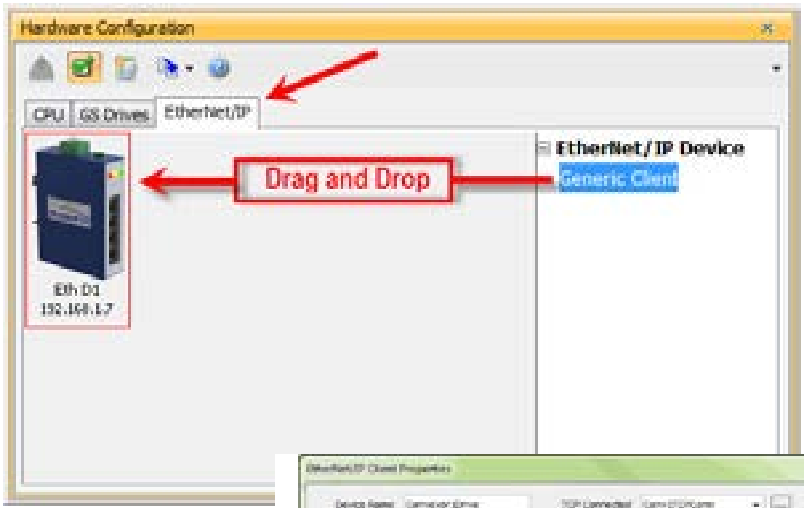
	Assembly Instance	
Connection Point (I/O)	101	(0x65)
Class (Explicit)	4	(0x4)
Instance (Explicit)	101	(0x65)
Attribute (Explicit)	3	(0x3)

Output Parameters
✓ Enable Output Data
Datatype: Unsigned Int 16 1D Array
Data Array: OutputDataBlock
Message Size (bytes): 100
Number of Elements: 50

	Assembly Instance	
Connection Point (I/O)	102	(0x66)
Class (Explicit)	4	(0x4)
Instance (Explicit)	102	(0x66)
Attribute (Explicit)	3	(0x3)

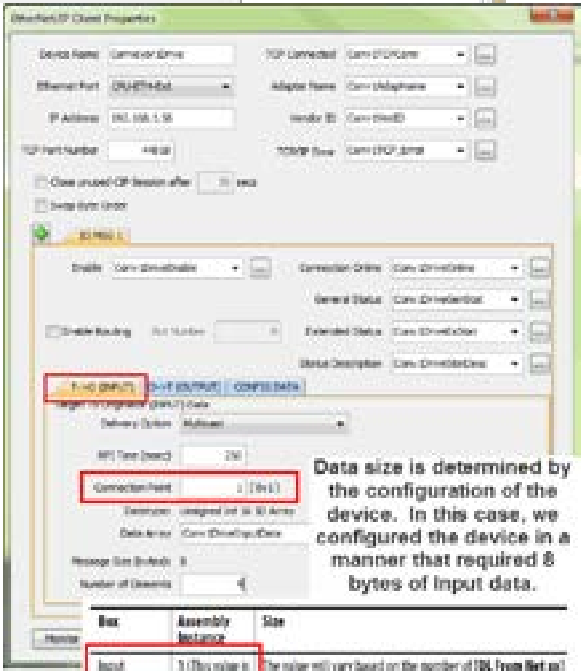
Example 2: Productivity®1000 as EtherNet/IP Scanner

This example shows how to connect the Productivity1000 Scanner function to an EtherNet/IP adapter device using Class 1 I/O Messaging. First, create an EtherNet/IP device in the Hardware Configuration as seen below:



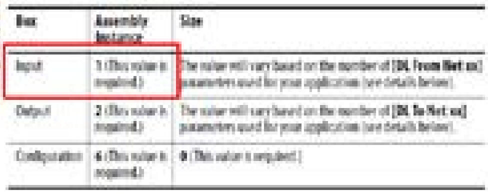
Configure the parameters to match the settings of the Adapter device. The image on right shows the setup of the Input data.

The size, in this case, is dynamic to the configuration of the device. For this particular example, we configured the device in a manner that allows it to publish 8 bytes of data for Input. Many devices will have a fixed configuration that should be published in the manufacturer's documentation.



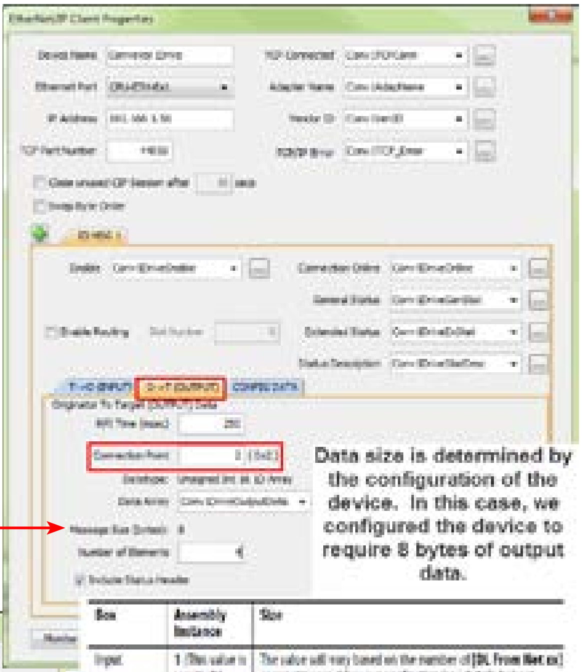
Device Name: [illegible]
Ethernet Port: [illegible]
IP Address: [illegible]
TCP Port Number: 44818
TCP Connected: [illegible]
Adapter Name: [illegible]
Vendor ID: [illegible]
TCP/IP Error: [illegible]
Close unused CIP Session after [illegible] secs
Swap Byte Order
Enable: [illegible]
Connection Online: [illegible]
General Status: [illegible]
Extended Status: [illegible]
Status Description: [illegible]
Enable Routing
Slot Number: 0
Delivery Option: Multicast
RPI Time (msec): 250
Connection Point: 1 [0x1]
Datatype: Unsigned Int 16 32 Array
Data Array: [illegible]
Message Size (in Bytes): 8
Number of Elements: 4

Data size is determined by the configuration of the device. In this case, we configured the device in a manner that required 8 bytes of input data.



Bus	Assembly Instance	Size
Input	1 (This value is required.)	The value will vary based on the number of [DL From Net xx] parameters used for your application (see details below).
Output	2 (This value is required.)	The value will vary based on the number of [DL To Net xx] parameters used for your application (see details below).
Configuration	6 (This value is required.)	0 (This value is required.)

The Output data must also be configured. Its data is also dynamic based upon the configuration. In our example, we configured the device in a manner that caused it to require 8 bytes of Output data.



EtherNet/IP Client Properties

Device Name: Conveyor Drive
IP Address: 192.168.1.50
TCP Port Number: 44818
Close unused CIP Session after 0 secs
Swap Byte Order

IO MSG 1

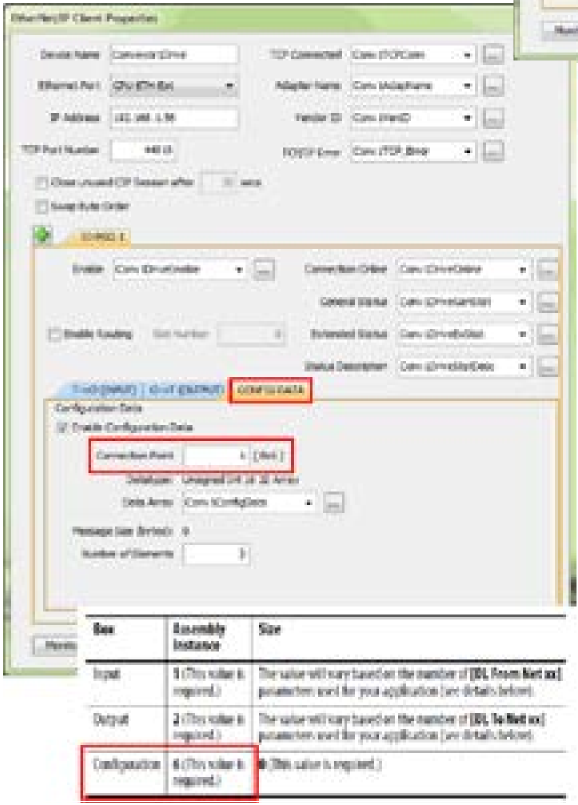
T->O (INPUT) | O->T (OUTPUT) | CONFIG DATA

Originator To Target (OUTPUT) Data
RPI Time (msec): 250
Connection Point: 2 (0x2)
Message Size (Bytes): 8
Number of Elements: 4
Include Status Header

Data size is determined by the configuration of the device. In this case, we configured the device to require 8 bytes of output data.

Box	Assembly Instance	Size
Input	1 (This value is required.)	The value will vary based on the number of [DL From Net xx] parameters used for your application (see details below).
Output	2 (This value is required.)	The value will vary based on the number of [DL To Net xx] parameters used for your application (see details below).
Configuration	6 (This value is required.)	0 (This value is required.)

The image on left shows the setup for the Configuration data. The Configuration data, for most devices, is a fixed size. Some devices will require that the Configuration data Connection Point be included in the Forward Open message (as shown on left) even if the size is 0. Some devices will require that the Configuration data Connection Point not be in the Forward Open and the checkbox option in the image below would need to be de-selected.



EtherNet/IP Client Properties

Device Name: Conveyor Drive
IP Address: 192.168.1.50
TCP Port Number: 44818
Close unused CIP Session after 0 secs
Swap Byte Order

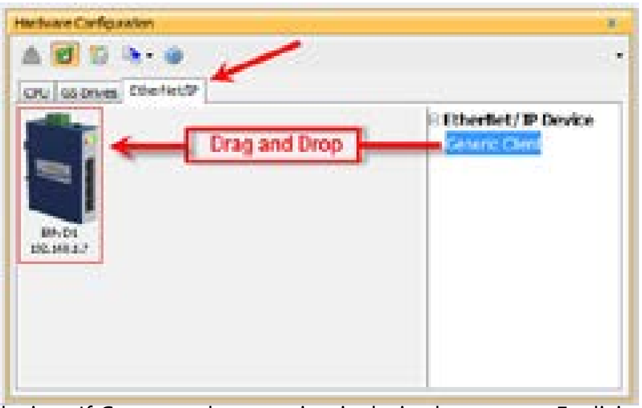
IO MSG 1

T->O (INPUT) | O->T (OUTPUT) | CONFIG DATA

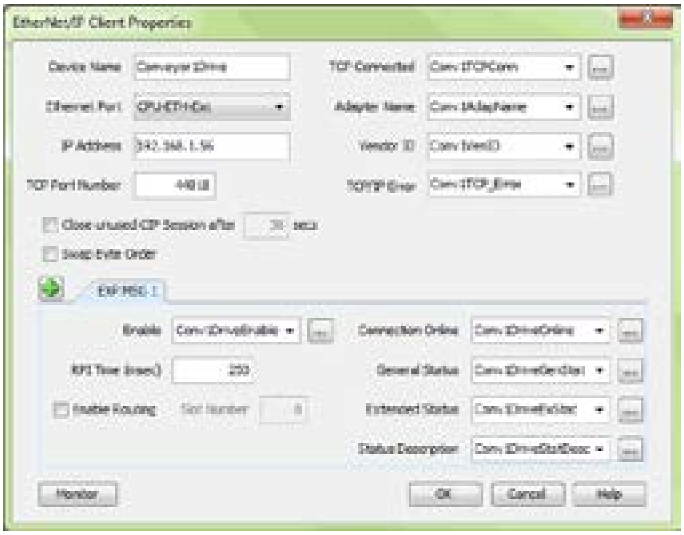
Configuration Data
Enable Configuration Data
Connection Point: 6 (0x6)
Message Size (Bytes): 0
Number of Elements: 0

Box	Assembly Instance	Size
Input	1 (This value is required.)	The value will vary based on the number of [DL From Net xx] parameters used for your application (see details below).
Output	2 (This value is required.)	The value will vary based on the number of [DL To Net xx] parameters used for your application (see details below).
Configuration	6 (This value is required.)	0 (This value is required.)

The following example shows how to connect the Productivity1000 Scanner function to an EtherNet/IP Adapter device using Class 3 Explicit Messaging. As with IO Messaging, an EtherNet/IP device must be created in the Hardware Configuration as seen below:



Explicit Messages can be performed in 2 ways: Unconnected or Connected (Class 3). The advantage of using Unconnected messaging is it allows more discrete control of each request. The disadvantage of Unconnected messaging is that Unconnected messages have a lower priority and will take longer to get serviced on some devices. Connected messages get serviced faster since there is a connection established to the device. If Connected messaging is desired, create an Explicit Message tab as shown in the image above. If Unconnected messaging is desired, do not create an Explicit Message tab. Only fill out the information in the upper portion of the EtherNet/IP Client Properties window.


EtherNet/IP Client Properties

Device Name	Conveyor Drive	TCP Connected	Conv.1TCPConn
Ethernet Port	CPU-ETH-Ext	Adapter Name	Conv.1AdapName
IP Address	192.168.1.56	Vendor ID	Conv.1VenID
TCP Port Number	44818	TCP/IP Error	Conv.1TCP_Error

Close unused CIP Session after 30 secs

Swap Byte Order

Expl MSG 1

Enable	Conv.1DriveEnable	Connection Online	Conv.1DriveOnline
RPI Time (msec)	250	General Status	Conv.1DriveGenStat
Enable Routing	Slot Number 0	Extended Status	Conv.1DriveExStat
		Status Description	Conv.1DriveStatDesc

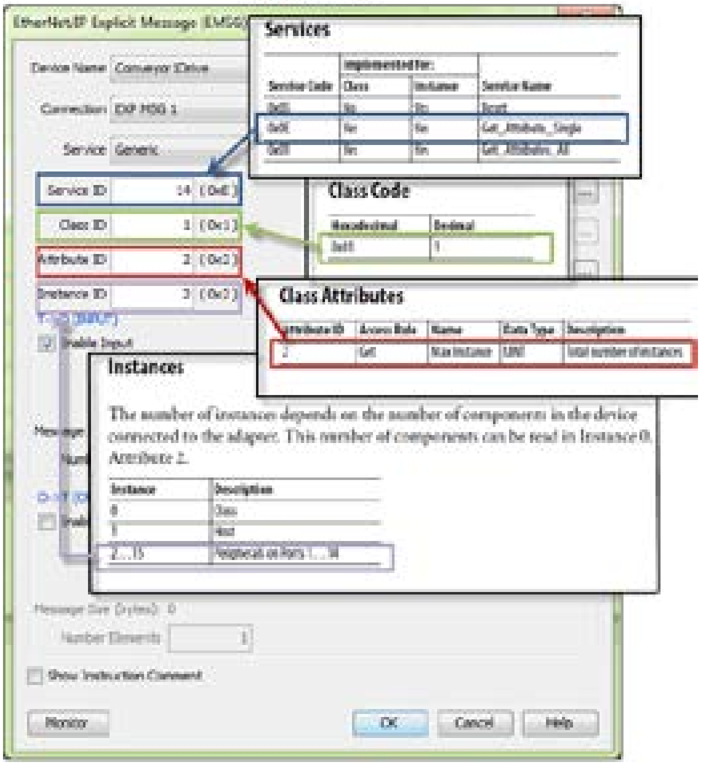
Once the desired parameters have been entered, the device may now be referenced in the Explicit Message Instruction. If Unconnected messaging has been selected, choose the Unconnected MSG option in the Connection drop down box. If Connected messaging has been selected, choose the Explicit Message that was configured in the EtherNet/IP Client Properties window in the Connection drop down box. The rest of the settings should be matched to the specifications documented by the manufacturer.

An example for requesting the Identity of a device is shown below. The data array configured for this function must be sufficient in size to hold the returned data from the device for this object. Data can also be written to the device if it supports an object for this purpose. If data is being written, enable the Output selection and specify the data array and size required by that device's object.

Troubleshooting Tips:

1. Use the diagnostic tags in the Hardware Configuration and Explicit Message Instruction: As explained previously in the Network Layer Chart section, there are multiple layers of messaging involved with EtherNet/IP. If it appears that the Productivity1000 is not communicating with another EtherNet/IP device, there are diagnostic tags available to narrow down which layer of the protocol is preventing successful communications.
   a. At the TCP layer, there is a TCP Connected field that will expose the status of the TCP/IP connection when a tag is populated in this field.

Identity Object



EtherNet/IP Explicit Message (EMSG)

Device Name: Conveyor Drive
Connection: EXP MSG 1
Service: Generic
Service ID: 14 (0xE)
Class ID: 1 (0x1)
Attribute ID: 2 (0x2)
Instance ID: 2 (0x2)
☑ Enable Input
Message Size (bytes): 0
Number Elements: 1
☐ Show Instruction Comment

Services

Service Code	Implemented for: Class	Instance	Service Name
0x05	[illegible]	[illegible]	Reset
0x0E	[illegible]	[illegible]	Get_Attribute_Single
0x01	[illegible]	[illegible]	Get_Attributes_All

Class Code

Hexadecimal	Decimal
0x01	1

Class Attributes

Attribute ID	Access Rule	Name	Data Type	Description
2	Get	Max Instance	UINT	Total number of instances

Instances

The number of instances depends on the number of components in the device connected to the adapter. This number of components can be read in Instance 0, Attribute 2.

Instance	Description
0	Class
1	Host
2...15	Peripherals on Ports 1...14

   b. There is an Adapter Name field for a String tag and a Vendor ID field for an Integer tag. Both of these fields can help to identify whether the Productivity1000 is connected to the correct device or not.
   c. At the CIP layer, there is a Connection Online field for a Boolean tag.

d. There are three additional fields to help determine why the CIP session might not be successful: General Status for an Integer tag, Extended Status for an Integer Data Array and Status Description for a String tag.

2. Use the TCP connected tag:

First check the TCP Connected tag. If the connection has been enabled (by turning on the tag configured in the Enable field or triggering an Explicit Message instruction with an Unconnected MSG specified) and the TCP Connected tag is not true, check the following items:

a. Cabling. Ensure that all of the cables are connected and in good shape. In most cases, the Ethernet port that the cable is connected to should indicate a Link Good LED. Ensure that any interim Ethernet switches are powered up and functioning and that the end device is powered up and functional.

b. IP address and correct subnet. Check that the IP address entered into the IP Address field is the correct address for the device that you are connecting to. Also check that the EtherNet/IP device's IP address and subnet mask is compatible with the IP address and subnet mask of the Productivity1000. If there are any routers in between the two, ensure that a proper default gateway that matches the router's IP address is configured. If you are unfamiliar with proper IP addressing and subnet configuration, consult with the network administrator for guidance.

c. TCP Port number. The default listening TCP port number for EtherNet/IP is 44818. Check that the target device is listening on this specific port number. If it is not, change the value in TCP Port Number field to the appropriate value. If there are interim router devices that are using port forwarding, ensure that the router is properly configured for this setup.

NOTE: *Attempting to do IO Messaging across routers (different subnets) is unlikely to be successful. IO Messaging uses multicast messaging in many cases and the Port number is not necessarily fixed when the IO Messaging is established (the Forward Open message has the ability to 'negotiate' the port number used for the IO Messages).*

d. Adapter Name and Vendor ID. If the network contains many EtherNet/IP devices and these devices may not necessarily be connected to the Productivity1000, it may be a good safeguard to check the Adapter Name and Vendor ID returned and verify that these devices are the correct devices to which it is connected.

3. Use the Connection Online and Error tags:

If the TCP Connected tag is true and the Adapter Name and Vendor ID look correct, the next tags to look at are the Connection Online, the General Status, the Extended Status and the Status Description.

If the Enable tag is true and the Connection Online tag is not true, check the General Status value along with the Extended Status value(s) and the Status Description. If the General Status value and the Extended Status value(s) are part of the defined errors from the ODVA specification, the Status Description should also return a more descriptive String. Once these errors are known, it may be possible to very simply make the adjustment in the settings to correct the issue.

If it is not obvious from the description, first check the manufacturer's documentation for corrective action in this particular scenario.

If the manufacturer's documentation doesn't give corrective action, check the EtherNet/IP Error Code List in this chapter for possible solutions.

NOTE: *This may not always solve the problem as each device manufacturer may publish the error for slightly different reasons.*

If the Connection Online tag is true and the data being received is different than what is expected, verify that the correct Connection Point values and/or Class, Instance, Attribute values are configured. There may be multiple areas of available data in that device. Verify that the correct data types are being used for both sides. If the data types are mismatched, this may make the data 'appear' to be incorrect.

Another great tool that can be used is Wireshark. Wireshark is a free network analyzer tool that can be downloaded from www.wireshark.com.

NOTE: *Using this tool implies some knowledge of how networking protocols function. Using Wireshark will also require that you have a true Ethernet hub (not an unmanaged switch) or a managed switch with Port mirroring capability.*

You may also use the following basic steps to check your EtherNet/IP Setup.

EtherNet/IP I/O Message Troubleshooting:

1. Does the IP Address set up in the Scanner match the Adapter IP Address?
2. Is the enable tag entered into the Scanner turned ON?
3. Does the connection point entered into the I/O Message Data Block match the connection point of the Adapter?
4. Does the number of elements match the Adapter?
5. Does the data type match the Adapter?
6. Steps 4 & 5 are important because the number of bytes being read from or written to the Adapter have to match the Adapter bytes allocated.

EtherNet/IP Explicit Message Troubleshooting:

1. Does the IP Address set up in the Scanner match the Adapter IP Address?
2. Is the enable tag entered into the Scanner turned ON when not using the Unconnected MSG connection type?
3. Make sure the logic for the EtherNet/IP Explicit Message (EMSG) is TRUE so the instruction is enabled.
4. When using Get or Set single attributes in the Service field make sure the Instance ID matches the Instance ID of the Adapter.
5. When using Generic in the Service field make sure the Service ID, Class ID, Attribute ID and Instance ID match the Adapter settings.
6. Does the number of elements match the Adapter?
7. Does the data type match the Adapter?

Steps 6 & 7 are important because the number of bytes being read from or written to the Adapter have to match the Adapter bytes allocated.

ProNET

Productivity Network (ProNET) provides the ability to share data with other P-Series CPU's, This can easily be accomplished using the Productivity Network (PNET) setup in the Hardware Configuration window used to join a data sharing network consisting of other P-Series controllers.

Each member of the data sharing network receives data from all of the other P-Series controllers on that data sharing network. Each node can optionally send data to the other nodes of the data sharing network by electing to "publish" data.

The ProNET configuration uses UDP broadcast packets to publish the blocks of data to the network. One caveat with the use of broadcast packets is that it limits the scope of the shared data network to the local broadcast domain.

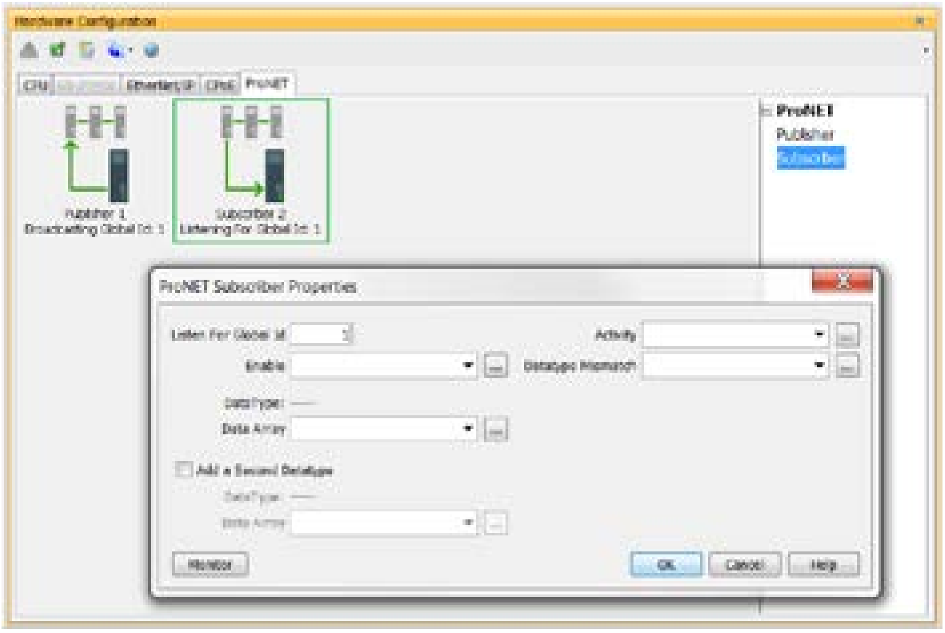
ProNET uses the verbs 'publishing' and 'subscribing' to describe how the controller data is exchanged with other P-Series controllers on the data sharing network.

Publishing is analogous to sending data, and is done only if ProNET is configured to 'publish' one or more of its assigned tags. If so configured, the P-Series controller will broadcast a packet that contains the data from the selected tags.

Subscribing is analogous to receiving data, and is accomplished by 'subscribing to' a publisher's global ID of any P-Series CPU on the data sharing network set up to publish its data.

The ProNET configuration works with a 1D array tag(s) that can contain up to 65535 elements, however you are limited to 32 total 32-bit elements, 64 total 16-bit elements, or 128 total 8-bit or Boolean elements of data per publisher array data type. These tags provide the local storage for the data sent and received over the data-sharing network.

When the input logic to the ProNET configuration is Enabled, it operates at a fixed rate of 10 times per second (100ms). The instruction will publish all of the elements of the array that it is configured to publish, and will process any ProNET nodes that it receives. When



Listen For Global Id: 1
Enable
Activity
Datatype Mismatch
Data Type: —
Data Array
Add a Second Datatype
Data Type: —
Data Array

the input logic is OFF, (the device is disabled), it DOES NOT publish any of its tags and DOES NOT process any ProNET nodes that it receives.

Custom Protocol over Ethernet Functionality

NOTE: *The message size for each data type is limited to 128 bytes regardless of the defined array size.*

Data Type	Number of Elements
Boolean	128
Integer 8-Bit	128
Integer 16-Bit	64
Integer 32-Bit	32
Integer 64-Bit	32

Besides Modbus RTU, EtherNet/IP, and ProNET the Productivity® 1000 system has the ability to communicate via Ethernet with other devices using the Custom Protocol over Ethernet (CPoE).

Custom Protocol over Ethernet

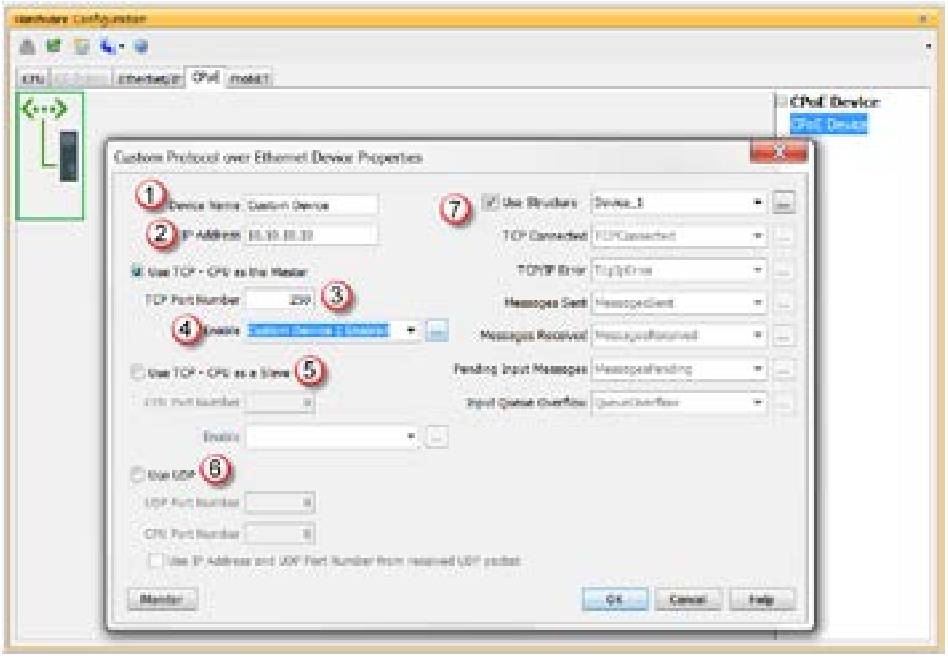
The Custom Protocol is a HEX based protocol used to communicate with devices that do not support one of the other protocols on Productivity1000. There are two steps to initiate communications via the Custom Protocol over Ethernet:

- First you must set up a device in the hardware configuration under the CPoE tab.
- Then you must use the Custom Protocol Ethernet(CPE) instruction to initiate messages.

Hardware Configuration

First you must set up a device to talk to in the CPoE tab of the hardware configuration. This will Require you to:

1. Enter a Device Name
2. Enter the IP Address of the device you wish to communicate with.



3. Enter the port number of the device.
4. Enter an Enable tag to enable the device if using TCP.
5. Choose whether you wish to Use the PLC as the master or the slave device via TCP connection
6. Choose whether you wish to use a UDP connection.
7. Enter tags for status of this device for troubleshooting (Example below shows the Structure method used).

Custom Protocol Ethernet Instruction

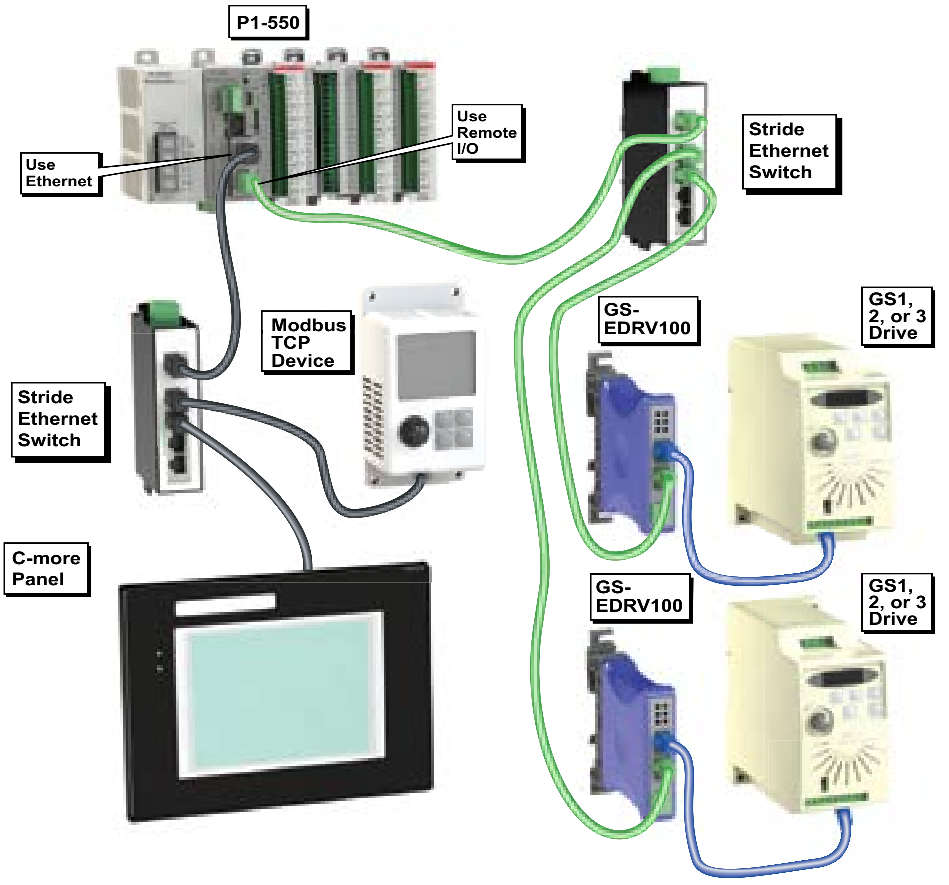
Next you must use the Custom Protocol Ethernet instruction in ladder.

- The instruction can be chosen Receive or Send messages to the Custom Device.
- The user can choose to use:
  a. A table with tags that allow the user to send a specific data.
  b. An array tag that is numerical can be used to Send/Receive from.
  c. A string tag that contains an ASCII string to be sent or string location to receive characters to.

Communications: Remote I/O and GS-Drives

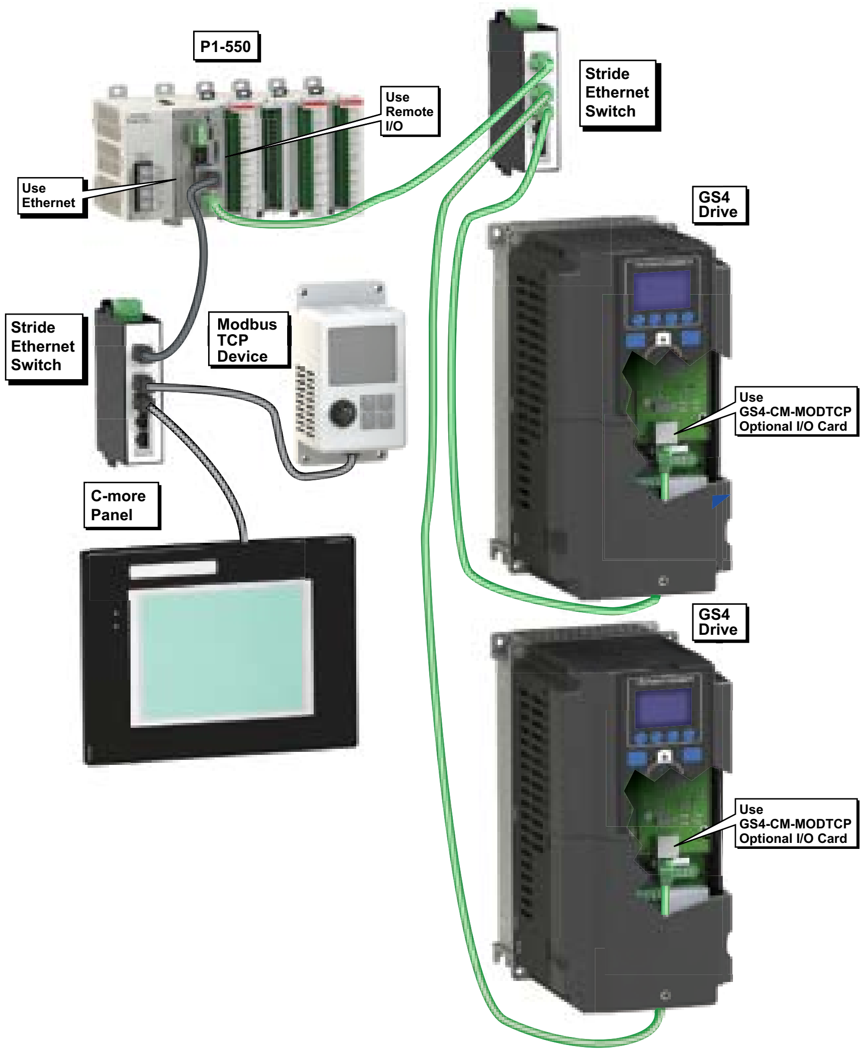
Design Considerations for Remote I/O and GS-Drives

It is important to understand that only one Remote I/O network can be on an unmanaged switch. If two or more Remote I/O networks are mixed into the same physical LAN (local area network), duplicate IP addressing will occur and the system will not function properly. Multiple Remote I/O networks can be used on a managed switch using the VLAN feature to create a virtual separation of the different networks, but multicasting messages are necessary for the network to function properly. Care must be taken when designing a system this way (using a managed switch). Even if only one Remote I/O network is being used in a facility, it is strongly recommended to keep it on a dedicated network, physically isolated from other networks. The Productivity® 1000 Remote I/O network makes use of multi-casting messages and many devices will not function properly in this situation.

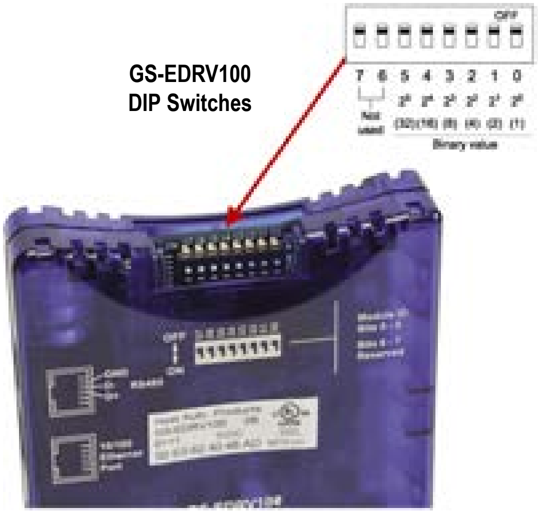


Communications: Remote I/O and GS-Drives, continued

The GS4-Drive configuration will automatically communicate through the Ethernet card installed in the drive. Some initial UDP broadcast messages occur upon discovery when initiated from the software and at power up. This should be considered if installing the GS4-Drive network with other devices.



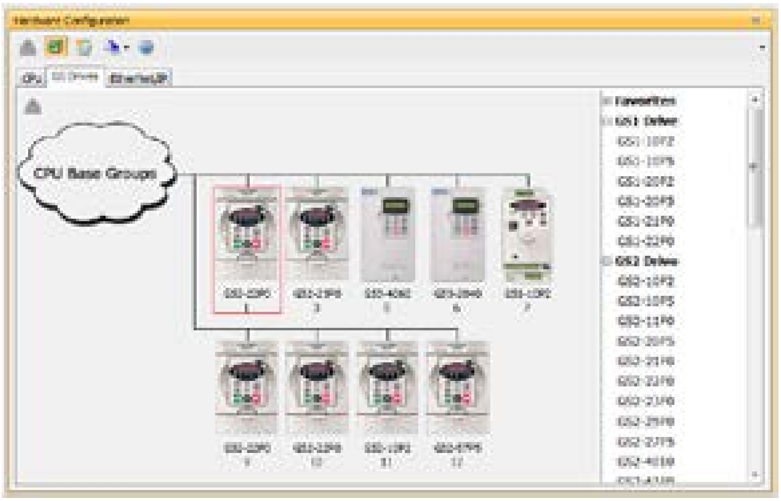
Configuration of GS-Drive Connections



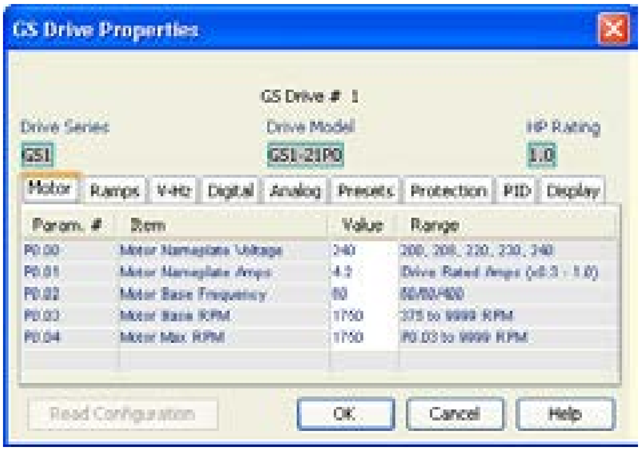
GS Drive connections are set up with a unique address for each GS drive communications module by the DIP switches on GS-EDRV100 modules, by the rotary switches on the GS4-CM-MODTCP and GS20A-CM-ENETIP modules or by using NetEdit. Since the DIP switch settings can only represent 00-63, setting an address of 64 or greater must be done using NetEdit.

For GS 1, 2 or 3 drives, after the GS-EDRV100 address has been set, be sure to connect the serial cable that comes with the GS-EDRV100 module to the GS-Drive serial port. The GS-EDRV100 will automatically configure the GS-Drive serial port to the correct settings. Once the GS-EDRV100 is properly addressed and connected to the GS-Drive, connect a straight through (patch) Ethernet cable from the Ethernet port of the GS-EDRV100 to an Ethernet switch. Connect a straight through cable from the P1 CPU Local Ethernet Port (Remote I/O) to the same switch.

Open the Productivity Suite Programming software and go online with the P1 CPU. Select Setup and then Hardware Configuration. Select the “Read Configuration” button in the upper left hand corner of this dialog, and the software will discover and display all found GS-Drives.



Configuration of GS-Drive Connections, continued

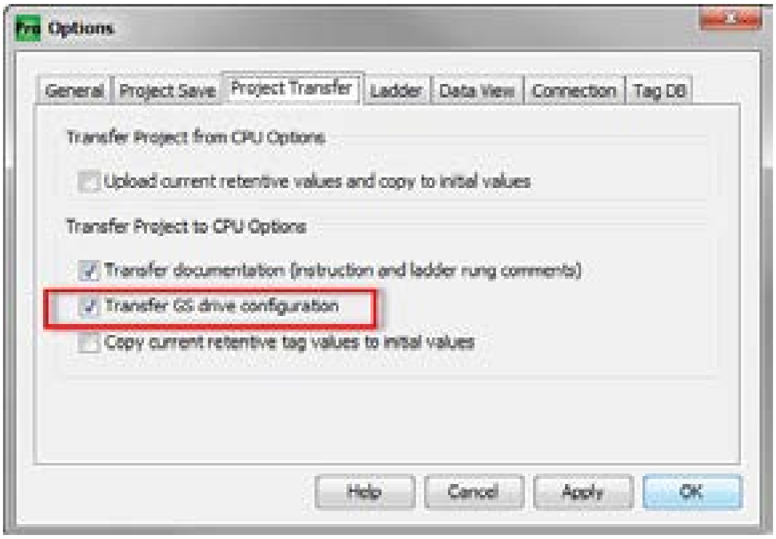

GS Drive Properties

GS Drive # 1

Drive Series: GS1
Drive Model: GS1-21P0
HP Rating: 1.0

Param. #	Item	Value	Range
P0.00	Motor Nameplate Voltage	240	200, 208, 220, 230, 240
P0.01	Motor Nameplate Amps	4.2	Drive Rated Amps (x0.3 - 1.0)
P0.02	Motor Base Frequency	60	50/60/400
P0.03	Motor Base RPM	1750	375 to 9999 RPM
P0.04	Motor Max RPM	1750	P0.03 to 9999 RPM

Once the drives have been discovered, the configuration of each drive can be read and written from the programming software.


Options

Transfer Project from CPU Options

☐ Upload current retentive values and copy to initial values

Transfer Project to CPU Options

☑ Transfer documentation (instruction and ladder rung comments)

☑ Transfer GS drive configuration

☐ Copy current retentive tag values to initial values

To allow the P1 CPU to automatically write the drive parameters on each CPU project transfer as well as when the CPU is powered up, a setting must be configured in the Productivity project. Go to Tools and Options and select the “Project Transfer” tab. Select the “Transfer GS drive configuration” as shown above. Drive parameters are ONLY transferred to the GS Drive at project transfer or at boot-up of the CPU.

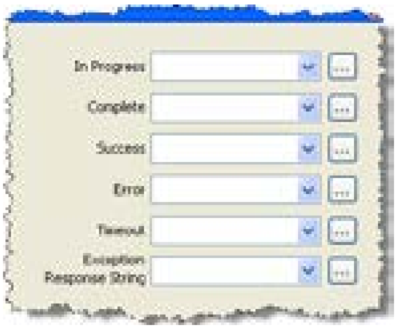
Configuration of GS-Drive Connections, (continued)

To monitor the status of the connection between the P1 CPU and the GS Drives, there is a Communications Heartbeat function that can be configured for the GS Drives. Use the status bits of the GS Read and GS Write instructions as shown below.

There are two possible communication paths that could be lost:

- P1 CPU to the GS drive communication module
- GS drive communication module to the GS drive

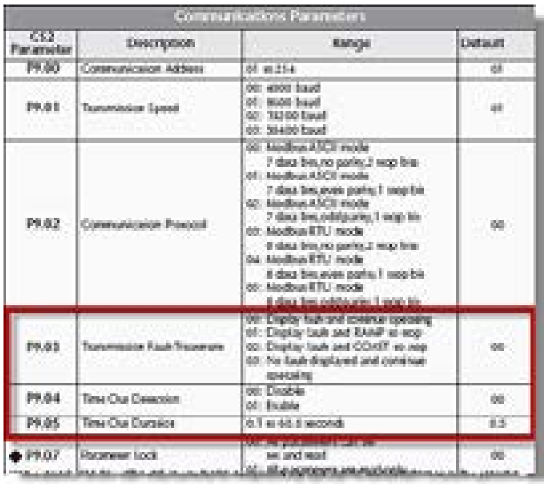
If a Timeout occurs or an error is received, this can be monitored in the ladder code and appropriate action can be taken.



In Progress
Complete
Success
Error
Timeout
Exception Response String

To configure the GS Drives to detect and react to loss of communications, there is a set of parameters that should be configured in the drive. The example below shows the parameters for a GS1, 2, or 3 series drive.

- Parameter P9.03 determines what the drive will do when it detects loss of communications.
- Parameter P9.04 enables the transmission loss detection feature.
- Parameter P9.05 determines the amount of time the drive will wait for a transmission before assuming that the link is lost and react according to how parameter P9.03 is configured.

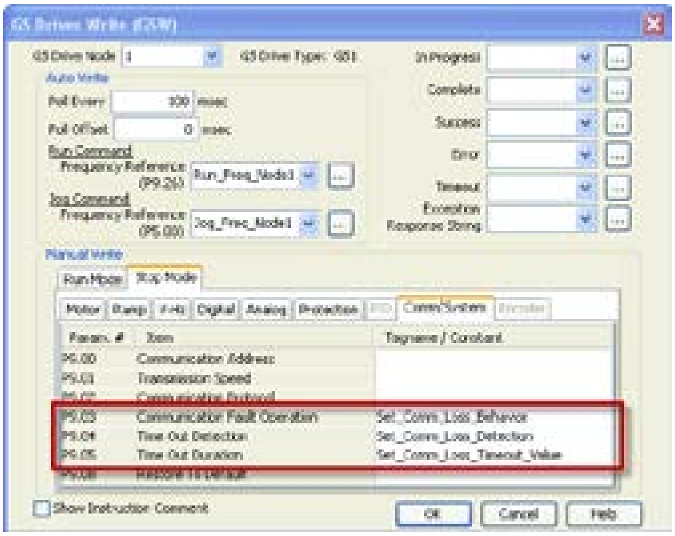


Communications Parameters

GS2 Parameter	Description	Range	Default
P9.00	Communication Address	01 to 254	01
P9.01	Transmission Speed	00: 4800 baud 01: 9600 baud 02: 19200 baud 03: 38400 baud	01
P9.02	Communication Protocol	00: Modbus ASCII mode 7 data bits,no parity,2 stop bits 01: Modbus ASCII mode 7 data bits,even parity,1 stop bit 02: Modbus ASCII mode 7 data bits,odd parity,1 stop bit 03: Modbus RTU mode 8 data bits,no parity,2 stop bits 04: Modbus RTU mode 8 data bits,even parity,1 stop bit 05: Modbus RTU mode 8 data bits,odd parity,1 stop bit	00
P9.03	Transmission Fault Treatment	00: Display fault and continue operating 01: Display fault and RAMP to stop 02: Display fault and COAST to stop 03: No fault displayed and continue operating	00
P9.04	Time Out Detection	00: Disable 01: Enable	00
P9.05	Time Out Duration	0.1 to 60.0 seconds	0.5
◆ P9.07	Parameter Lock	00: All parameters can be set and read 01: [illegible]	00

The GS-EDRV100 reads these configured parameters, and if they are configured for detecting communications loss, it will also monitor for loss of communications on the Ethernet side. If communications are lost on the Ethernet side, the GS-EDRV100 will shut down the GS Drive.

Configuration of GS-Drive Connections, (continued)



GS Drives Write (GSW)

GS Drive Node: 1 GS Drive Type: GS1

Auto Write

Poll Every: 100 msec

Poll Offset: 0 msec

Run Command — Frequency Reference (P9.26): Run_Freq_Node1

Jog Command — Frequency Reference (P5.00): Jog_Freq_Node1

In Progress / Complete / Success / Error / Timeout / Exception Response String

Manual Write — Run Mode | Stop Mode

Motor | Ramp | V-Hz | Digital | Analog | Protection | PID | Comm/System | Encoder

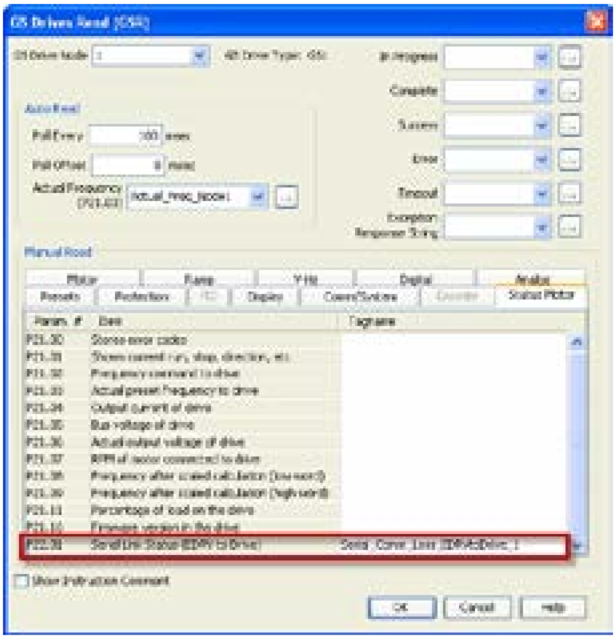
Param. #	Item	Tagname / Constant
P9.00	Communication Address	
P9.01	Transmission Speed	
P9.02	Communication Protocol	
P9.03	Communication Fault Operation	Set_Comm_Loss_Behavior
P9.04	Time Out Detection	Set_Comm_Loss_Detection
P9.05	Time Out Duration	Set_Comm_Loss_Timeout_Value
P9.08	Restore to Default	

Show Instruction Comment

OK Cancel Help

It is very important to note that if the communications loss feature is enabled: either a GS Drive Read or GS Drive Write instruction needs to be configured to communicate to the GS-EDRV100 and GS Drive at a poll rate that will prevent the GS-EDRV100 and GS Drive from detecting a loss of communication.

There is also a parameter (P22.01) that can be monitored to check the health of the serial connection between the GS-EDRV100 and the GS Drive. This parameter can be monitored in the ladder code and appropriate action taken if serial communications loss is detected.



GS Drives Read (GSR)

GS Drive Node: 1 GS Drive Type: GS1

Auto Read

Poll Every: 100 msec

Poll Offset: 0 msec

Actual Frequency (P21.03): Actual_Freq_Node1

In Progress / Complete / Success / Error / Timeout / Exception Response String

Manual Read

Motor | Ramp | V-Hz | Digital | Analog | Presets | Protection | PID | Display | Comm/System | Encoder | Status Motor

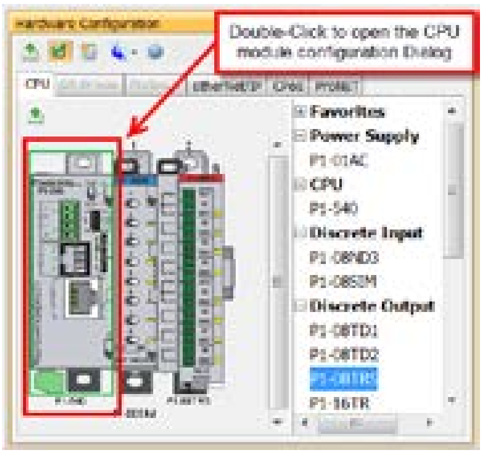
Param. #	Item	Tagname
P21.00	Stores error codes	
P21.01	Shows current run, stop, direction, etc.	
P21.02	Frequency command to drive	
P21.03	Actual present frequency to drive	
P21.04	Output current of drive	
P21.05	Bus voltage of drive	
P21.06	Actual output voltage of drive	
P21.07	RPM of motor connected to drive	
P21.08	Frequency after scaled calculation (low word)	
P21.09	Frequency after scaled calculation (high word)	
P21.11	Percentage of load on the drive	
P21.12	Firmware version in the drive	
P22.01	Serial Link Status (EDRV to Drive)	Serial_Conn_Loss_EDRVtoDrive_1

Show Instruction Comment

OK Cancel Help

Communications: Port Configuration

The Communications Port Configuration for any module containing comm ports is accessed from the Hardware Configuration window. For example, to access the P1-540/550 communications port configuration, first select the CPU from the Hardware Configuration window by double left-clicking or by right-clicking the CPU and selecting Open from the drop down menu. This will display the P1-540/550 configuration window seen here.

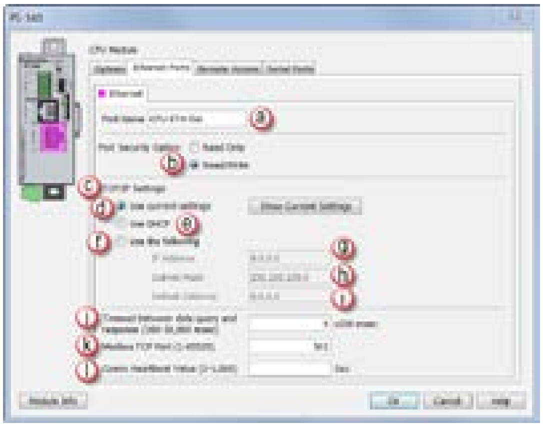


The following descriptions will focus on the P1-540/550 communications ports.

Ethernet Configuration

Ethernet Ports: The 10/100Base-T Ethernet port on the P1-540/550 CPU.

Ethernet: The Ethernet port can connect to Modbus TCP Client devices, Modbus TCP Server devices and PCs running the Productivity1000 programming software. The Ethernet Port is configured with an IP Address, Subnet Mask and Default Gateway, allowing it to function seamlessly on a typical LAN network.



External Ethernet Port Settings

***NOTE:** Two CPU Remote I/O networks cannot co-exist on the same LAN.*

a) Port Name: Allows the entry of a unique Name for the Ethernet Port. This Name is referenced in the Communications instructions (MRX, MWX, RX, WX) to select the Port to send the request from.

b) Port Security Option: This Option can be used as a simple Security measure to prevent Modbus TCP write requests from being accepted by the CPU. To allow Reads and Writes, select Read/Write.

c) TCP/IP Settings: The IP Setting of this Port may be changed in several ways:

a. The settings may be entered manually in the Choose CPU tool in the Productivity Suite programming software. This allows the user to make changes to the IP to allow connection by the computer running the Productivity Suite programming software. Changes are sent using Multicast Messages.

b. The TCP/IP Settings can be saved as part of the project. This must be Enabled in the P1-540 Hardware Configuration Settings by selecting Use the Following (Item f below). If handled this way, the Settings stored in the project will take effect at Project Transfer and at boot up only. The Settings may be changed after boot up.

d) Use Current Settings: When selected, Project Transfer or boot up will not make changes to the
TCP/IP Settings of the CPU.

e) Use DHCP: This specifies that the CPU should request its IP Settings from a DHCP Server on the network.

***NOTE:** If the CPU is set to use DHCP for it's IP Settings it cannot, in all likelihood, be used as a Modbus TCP Server.*

f) Use The Following: If this Option is selected, the CPU will set itself to the specified project Settings upon Project Transfer or at boot up.

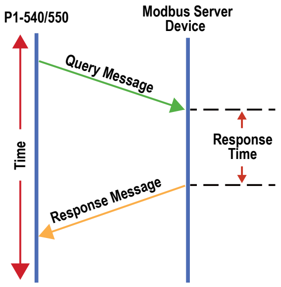
g) IP Address: This field is where the IP Address is specified in Four Octets.

For Example: 192.168.1.5

h) Subnet Mask: This field is where the Subnet Mask is specified in Four Octets (i.e., 255.255.255.0). The Subnet Mask is used in conjunction with the IP Address to configure a Logical Network.

i) Default Gateway: This field is where the Default Gateway Address is specified in Four Octets (i.e., 192.168.1.1). This is typically the IP Address of the router on the network. If a target IP Address is specified in an outgoing message from the CPU that is not in the Local Subnet, the Default Gateway Address is where this message will be sent.

j) Timeout Between Data Query and Response: The Time period specified in this field is the Time between the queries sent from the CPU (via a Communication instruction, such as a MRX, MWX, RX or WX) and the Time a response from that device is received. If the Response takes longer to receive or is not received within the specified Time period, a Timeout Error will occur for the given instruction.
Each instruction has a Timeout Status bit that can be assigned to it.



(k) Modbus TCP Port: This is the listening TCP Port Number for Modbus TCP connections. If necessary, this value can be adjusted for advanced router access. In most situations, this Port Number should be left at 502.

(l) Comm Heartbeat Value: This feature allows the ladder logic in the CPU to know if a device has stopped communicating to the CPU. If a value is placed in this field, the CPU will start a timer between each communication packet coming in to the CPU. If a communication packet fails to be received by the CPU within the specified time period, the System Bit Ethernet Heartbeat Timeout Bit will become true.

Local Ethernet Port Settings

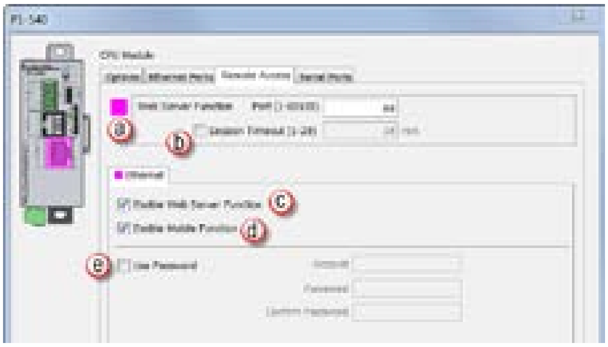
(m) Timeout Between Data Query and Response: The Time period specified in this field is the Time between the queries sent from the CPU (for Remote I/O Nodes and GS Drive Nodes) and the Time a Response from that device is Received. If the Response takes longer to receive (or is not received) than the specified Time period, a Timeout Error will occur for the given device and an Error will be generated in the Error Log. See Modbus Server diagram shown on previous page.

(n) Comm Heartbeat Value: This value specifies how long the Remote I/O Slaves should wait for a communication packet from the CPU. If a communication packet is not received from the CPU within the specified time period, all outputs on the Remote Slave will be turned OFF.

Remote Access Configuration

(a) Web Server Function: Provides the ability to make a non secure web connection to the P1-540/550 in order to access the USB pen drive and view read-only system tags. When enabled, a port number selection is required.

◊ Port: (Default 80) Allows user to set a port number ranging from 1-65535.



(b) Session Timeout: Allows the user to set a specific time limit (1–20 min.) on inactivity that will close the Web Server connection. If there is no activity between the PC and the Web Server for the specified time limit, the connection will close.

(c) Enable Web Server Function: Select this box to enable/disable Web Server Function > See bullet above.

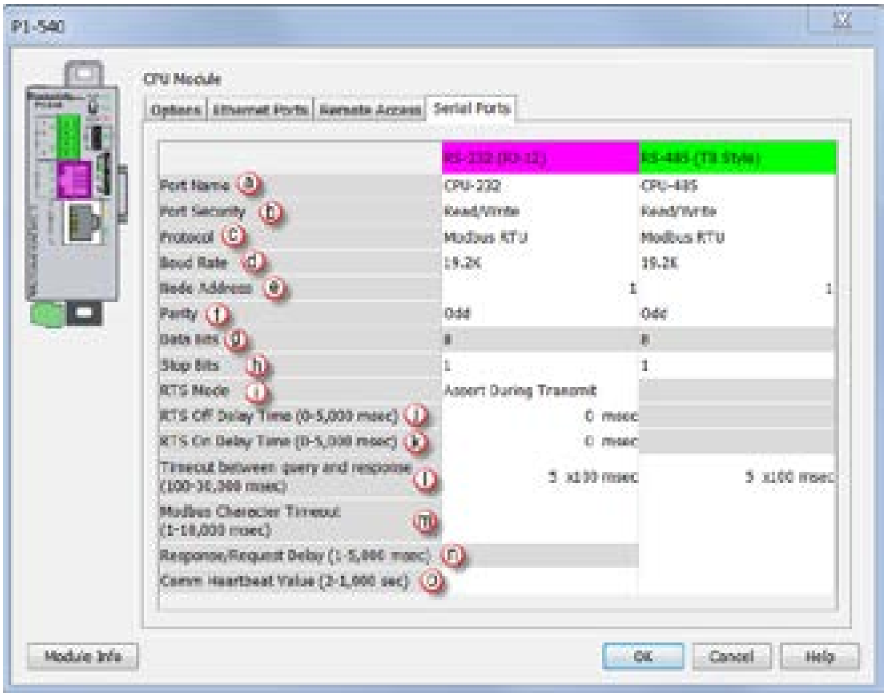
(d) Mobile Function: Enables Remote Access which allows the CPU Data Remote Monitor App to monitor the selected tags.

(e) Password Option: Allows the user to set a password for access to the Web Server.

◊ Enter an account name and password of up to a combination of 16 numbers and characters (can include special characters).

Serial Configuration

When the Serial Ports Tab is selected, the Serial Ports settings are displayed as shown below.



	RS-232 (RJ-12)	RS-485 (TB Style)
Port Name (a)	CPU-232	CPU-485
Port Security (b)	Read/Write	Read/Write
Protocol (c)	Modbus RTU	Modbus RTU
Baud Rate (d)	19.2K	19.2K
Node Address (e)	1	1
Parity (f)	Odd	Odd
Data Bits (g)	8	8
Stop Bits (h)	1	1
RTS Mode (i)	Assert During Transmit	
RTS Off Delay Time (0-5,000 msec) (j)	0 msec	
RTS On Delay Time (0-5,000 msec) (k)	0 msec	
Timeout between query and response (100-30,000 msec) (l)	5 x100 msec	5 x100 msec
Modbus Character Timeout (1-10,000 msec) (m)		
Response/Request Delay (1-5,000 msec) (n)		
Comm Heartbeat Value (2-1,000 sec) (o)		

There are two Serial Ports on the P1-540 and P1-550 CPUs; an RS-232 Port with an RJ-12 connector and a 2-wire RS-485 Port with a removable three point terminal block. Both Ports are capable of Modbus RTU Client (device that initiates communications requests) and Server (device that responds to communications requests) communications. They are also capable of ASCII outgoing strings and incoming strings.

RS-232 and RS-485 Port Settings

(a) Port Name: Allows the entry of a unique name for the RS-232 and RS-485 Ports. This name is referenced inside of the Communications instructions (MRX, MWX, RX, WX) and ASCII instructions (AIN, AOUT, CPO, CPI) to select the Port to send or receive the request.

(b) Port Security: This Option can be used as a simple Security measure to prevent Modbus TCP write requests from being accepted by the CPU. To allow Reads and Writes, select Read/Write.

(c) Protocol: This field determines whether the Port is used for Modbus RTU communications, sending or receiving ASCII Strings or performing the Custom Protocol function.

(d) Baud Rate: Choose the Baud Rate that your device and the CPU should communicate in this field. The appropriate choice will vary greatly with device, application and environment. The important point is that all devices communicating on the network need to be set to the same Baud Rate. The available Baud Rates are 1200, 2400, 9600, 19200, 33600, 38400, 57600 and 115200 bps.

RS-232 and RS-485 Port Settings, continued

ⓔ Node Address: This field is used only when the CPU is a Modbus RTU Server device. This field is used to uniquely identify the CPU on the network. This setting is also sometimes referred to as a Station Address. This field can be set from 1 to 247.

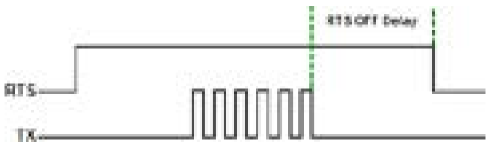
ⓕ Parity: The Parity Bit is used as a simple, low-level form of Error Detection. All devices on the network need to be at the same Parity setting. The appropriate choice will vary with devices. Valid selections are None, Even and Odd.

ⓖ Data Bits: This field determines whether the communications packet uses Seven Data Bits or Eight Data Bits. Eight Data Bits is the only valid selection for Modbus RTU. Either Seven or Eight Data Bits can be selected when using ASCII communications. Set this field to match the device that is connected to the CPU.

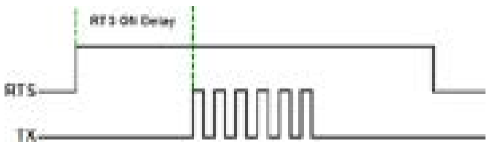
ⓗ Stop Bits: This field determines whether the communications packet uses One or Two Stop Bits. Set this field to match the device that is connected to the CPU.

ⓘ RTS Mode: This field allows selection of whether or not RTS is asserted during data transmission. Used for hardware handshaking in the standard way. You may need to manually configure RTS. Refer to your instrument documentation to determine its specific behavior.

ⓙ RTS Off Delay Time (RS-232 Only): This Time period is the amount of Time between the end of the data transmission to when the RTS signal is turned off. The diagram below (left) illustrates this. This setting may be needed when using media converters (RS-232 to RS-422/485 converters) and/or radio modems. A delay may be needed at the end of the data transmission for processing time in the devices.



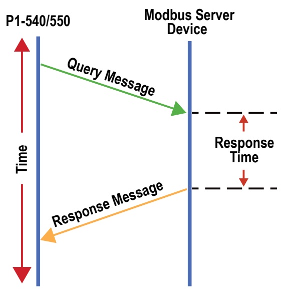
j. RTS Off Delay



k. RTS On Delay

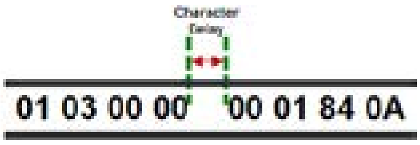
ⓚ RTS On Delay Time (RS-232 Only): This Time period is the amount of Time between when the RTS Signal is turned ON and the data transmission begins. The diagram above (right) illustrates this. This setting may be needed when using media converters (RS-232 to RS-485 converters) and/or radio modems. A delay may be needed after the assertion of the RTS Signal and when the data transmission begins for processing time in the device.

ⓛ Timeout Between Query and Response: The Time period specified in this field is the Time between the queries sent from the CPU (via a Communication instruction, such as an MRX, MWX, RX, or WX) and the Time a Response from that device is Received. If the Response takes longer to receive (or is not received) than the specified Time period, a Timeout Error will occur for the given instruction. Each instruction has a Timeout Status bit that can be assigned to it.



RS-232 and RS-485 Port Settings, continued

m Modbus Character Timeout: The Modbus Character Delay Time is specified as the Time between two bytes (or characters) within a given Modbus Message. The Modbus RTU specification states that this time must be no more than 1.5 Character Times (real time based on Baud Rate). Sometimes delays do occur between bytes when using radio modems, media converters, etc. This setting allows some tolerance in these situations for the incoming Modbus Messages in the CPU. The CPU will wait for the amount of time specified in this field before discarding the incomplete packet. If the CPU does not receive the remainder of the Message within the specified Time Frame, it will discard the first portion of the Message and wait for a new Message.

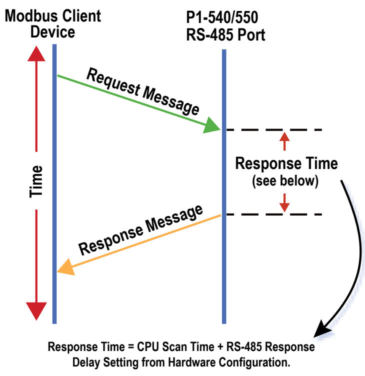


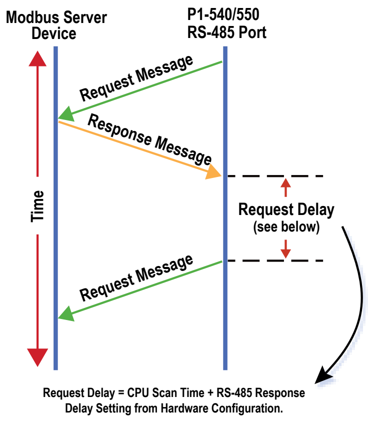
n Response/Request Delay (RS-485 Only): This setting is used when the CPU is a Modbus RTU Server or Client on the RS-485 Port.

The total Response Time can be up to the Total CPU Scan Time + the Value specified in this field. When using 2-wire RS-485 communications, sometimes Echoes can occur since both devices use the same differential signal pair to send and receive.

a. If acting as a Server (on left below), upon receiving a Modbus Request, the CPU will wait for the time period specified in this field before sending a Response. This can be used with slow clients that need extra time to change from sending to receiving.

b. If acting as a Client (on right below), after receiving a Modbus Response, the CPU will wait for the time period specified in this field before sending another Request. This can be used to delay request messages in order to give extra time for slow server devices.


Modbus Client Device
P1-540/550 RS-485 Port
Request Message
Response Time (see below)
Time
Response Message
Response Time = CPU Scan Time + RS-485 Response Delay Setting from Hardware Configuration.

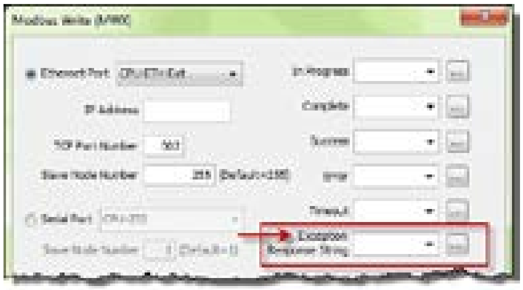

Modbus Server Device
P1-540/550 RS-485 Port
Request Message
Response Message
Request Delay (see below)
Time
Request Message
Request Delay = CPU Scan Time + RS-485 Response Delay Setting from Hardware Configuration.

o Comm Heartbeat Value: This feature allows the ladder logic in the CPU to know if a device has stopped communicating to the CPU. If a value is placed in this field, the CPU will start a timer between each communication packet coming in to the CPU. If a communication packet fails to be received by the CPU within the specified Time period, the System Bit RS-232 Heartbeat Timeout Bit or RS-485 Heartbeat Timeout Bit will become true.

Communications: Error Codes

NOTE: *The only time you will see Communications Error Codes is when the CPU is the Master of a Communications Network.*

To simplify the process of identifying a possible Error, the Productivity® 1000 CPU will automatically report to a specific memory location an Error Code that helps identify the existing issue. The Error Codes are reported in the Exception Response String Tag specified in the instruction as shown below.



The Exception Response String field is available on the following instructions:

The Table shown below provides a list of Productivity1000 Communication Error Codes that may be reported by the Productivity CPU.

- GS Drives Read
- GS Drives Write
- Modbus Read
- Modbus Write
- Network Read
- Network Write
- Dataworx Request

Productivity1000 Communication Error Codes

Error Code	Description	Suggested Fix
01	Function Code not supported.	Check instruction or connected device and correct Function code or address range selected.
02	Address out of range. This error is typically generated when a Modbus address has been requested that does not exist in the CPU.	Check instruction or connected device and correct Function code or address range selected.
03	Illegal Data Value. This error is typically generated when the Modbus request sent to the CPU is formed incorrectly.	Check the Modbus request against the Modbus protocol specification (www.modbus.org) to verify that it was formed correctly.
04	Device Failure.	Check connected device.
06	Slave Device is Busy. This error is typically due to excess communications to the EDRV.	Slow down the poll rate in the GS instruction.

P1000 EtherNet/IP Error Codes

✓ — CPU server currently supported errors ✗ — CPU server (will not generate error)
Note: Other adapters may generate this error

P1000 EtherNet/IP Error Codes				
General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x0100	Connection In Use/ Duplicate Forward Open	A connection is already established from the target device sending a Forward Open request or the target device has sent multiple forward open request. This could be caused by poor network traffic. Check the cabling, switches and connections.	✓
0x01	0x0103	Transport Class/ Trigger Combination not supported	The Transport class and trigger combination is not supported. The Productivity1000 CPU only supports Class 1 and Class 3 transports and triggers: Change of State and Cyclic.	✓
0x01	0x0106	Owner Conflict	An existing exclusive owner has already configured a connection to this Connection Point. Check to see if other Scanner devices are connected to this adapter or verify that Multicast is supported by adapter device if Multicast is selected for Forward Open. This could be caused by poor network traffic. Check the cabling, switches and connections.	✓
0x01	0x0107	Target Connection Not Found	This occurs if a device sends a Forward Close on a connection and the device can't find this connection. This could occur if one of these devices has powered down or if the connection timed out on a bad connection. This could be caused by poor network traffic. Check the cabling, switches and connections.	✓
0x01	0x0108	Invalid Network Connection Parameter	This error occurs when one of the parameters specified in the Forward Open message is not supported such as Connection Point, Connection type, Connection priority, redundant owner or exclusive owner. The Productivity1000 CPU does not return this error and will instead use errors 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125 or 0x0132 instead.	✓
0x01	0x0109	Invalid Connection Size	This error occurs when the target device doesn't support the requested connection size. Check the documentation of the manufacturer's device to verify the correct Connection size required by the device. Note that most devices specify this value in terms of bytes. The Productivity1000 CPU does not return this error and will instead use errors 0x0126, 0x0127 and 0x0128.	✗
0x01	0x0110	Target for Connection Not Configured	This error occurs when a message is received with a connection number that does not exist in the target device. This could occur if the target device has powered down or if the connection timed out. This could be caused by poor network traffic. Check the cabling, switches and connections.	✗
0x01	0x0111	RPI Not Supported	This error occurs if the Originator is specifying an RPI that is not supported. The Productivity1000 CPU will accept a minimum value of 10ms on a CIP Forward Open request. However, the CPU will produce at the specified rate up to the scan time of the installed project. The CPU cannot product any faster than the scan time of the running project.	✓

P1000 EtherNet/IP Error Codes

General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x0112	RPI Value not acceptable	This error can be returned if the Originator is specifying an RPI value that is not acceptable. There may be six additional values following the extended error code with the acceptable values. An array can be defined for this field in order to view the extended error code attributes. If the Target device supports extended status, the format of the values will be as shown below: • Unsigned Integer 16, Value = 0x0112, Explanation: Extended Status code • |Unsigned Integer 8, Value = variable, Explanation: Acceptable Originator to Target RPI type, values: 0 = The RPI specified in the forward open was acceptable (O -> T value is ignored), 1 = unspecified (use a different RPI), 2 = minimum acceptable RPI (too fast), 3 = maximum acceptable RPI (too slow), 4 = required RPI to corrected mismatch (data is already being consumed at a different RPI), 5 to 255 = reserved. • Unsigned Integer 32, Value = variable, Explanation: Value of O -> T RPI that is within the acceptable range for the application. • Unsigned Integer 32, Value = variable, Explanation: Value of T -> O RPI that is within the acceptable range for the application.	✗
0x01	0x0113	Out of Connections	The Productivity1000 EtherNet/IP Adapter connection limit of 4 when doing Class 3 connections has been reached. An existing connection must be dropped in order for a new one to be generated.	✓
0x01	0x0114	Vendor ID or Product Code Mismatch	The compatibility bit was set in the Forward Open message but the Vendor ID or Product Code did not match.	✓
0x01	0x0115	Device Type Mismatch	The compatibility bit was set in the Forward Open message but the Device Type did not match.	✓
0x01	0x0116	Revision Mismatch	The compatibility bit was set in the Forward Open message but the major and minor revision numbers were not a valid revision.	✓
0x01	0x0117	Invalid Produced or Consumed Application Path	This error is returned from the Target device when the Connection Point parameters specified for the O -> T (Output) or T -> O (Input) connection is incorrect or not supported. The Productivity1000 CPU does not return this error and uses the following error codes instead: 0x012A, 0x012B or 0x012F.	✗
0x01	0x0118	Invalid or Inconsistent Configuration Application Path	This error is returned from the Target device when the Connection Point parameter specified for the Configuration data is incorrect or not supported. The Productivity1000 CPU does not return this error and uses the following error codes instead: 0x0129 or 0x012F.	✗
0x01	0x0119	Non-listen Only Connection Not Opened	This error code is returned when an Originator device attempts to establish a listen only connection and there is no non-listen only connection established. The Productivity1000 CPU does not support listen only connections as Scanner or Adapter.	✓

P1000 EtherNet/IP Error Codes

General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x011A	Target Object Out of Connections	The maximum number of connections supported by this instance of the object has been exceeded.	✗
0x01	0x011B	RPI is smaller than the Production Inhibit Time	The Target to Originator RPI is smaller than the Target to Originator Production Inhibit Time. Consult the manufacturer's documentation as to the minimum rate that data can be produced and adjust the RPI to greater than this value.	✗
0x01	0x011C	Transport Class Not Supported	The Transport Class requested in the Forward Open is not supported. Only Class 1 and Class 3 classes are supported in the Productivity1000 CPU.	✗
0x01	0x011D	Production Trigger Not Supported	The Production Trigger requested in the Forward Open is not supported. In Class 1, only Cyclic and Change of state are supported in the Productivity1000 CPU. In Class 3, Application object is supported.	✗
0x01	0x011E	Direction Not Supported	The Direction requested in the Forward Open is not supported.	✗
0x01	0x011F	Invalid Originator to Target Network Connection Fixed/ Variable Flag	The Originator to Target fixed/variable flag specified in the Forward Open is not supported. Only Fixed is supported in the Productivity1000 CPU.	✗
0x01	0x0120	Invalid Target to Originator Network Connection Fixed/ Variable Flag	The Target to Originator fixed/variable flag specified in the Forward Open is not supported. Only Fixed is supported in the Productivity1000 CPU.	✗
0x01	0x0121	Invalid Originator to Target Network Connection Priority	The Originator to Target Network Connection Priority specified in the Forward Open is not supported. Low, High, Scheduled and Urgent are supported in the Productivity1000 CPU.	✗
0x01	0x0122	Invalid Target to Originator Network Connection Priority	The Target to Originator Network Connection Priority specified in the Forward Open is not supported. Low, High, Scheduled and Urgent are supported in the Productivity1000 CPU.	✗
0x01	0x0123	Invalid Originator to Target Network Connection Type	The Originator to Target Network Connection Type specified in the Forward Open is not supported. Only Unicast is supported for O -> T (Output) data in the Productivity1000 CPU.	✓
0x01	0x0124	Invalid Target to Originator Network Connection Type	The Target to Originator Network Connection Type specified in the Forward Open is not supported. Multicast and Unicast is supported in the Productivity1000 CPU. Some devices may not support one or the other so if this error is encountered try the other method.	✓
0x01	0x0125	Invalid Originator to Target Network Connection Redundant_Owner	The Originator to Target Network Connection Redundant_ Owner flag specified in the Forward Open is not supported. Only Exclusive owner connections are supported in the Productivity1000 CPU.	✓
0x01	0x0126	Invalid Configuration Size	This error is returned when the Configuration data sent in the Forward Open does not match the size specified or is not supported by the Adapter. The Target device may return an additional Unsigned Integer 16 value that specifies the maximum size allowed for this data. An array can be defined for this field in order to view the extended error code attributes.	✗

P1000 EtherNet/IP Error Codes

General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x0127	Invalid Originator to Target Size	This error is returned when the Originator to Target (Output data) size specified in the Forward Open does not match what is in the Target. Consult the documentation of the Adapter device to verify the required size. Note that if the Run/Idle header is requested, it will add 4 additional bytes and must be accounted for in the Forward Open calculation. The Productivity1000 CPU always requires the Run/Idle header so if the option doesn't exist in the Scanner device, you must add an additional 4 bytes to the O -> T (Output) setup. Some devices may publish the size that they are looking for as an additional attribute (Unsigned Integer 16 value) of the Extended Error Code. An array can be defined for this field in order to view the extended error code attributes. Note: This error may also be generated when a Connection Point value that is invalid for IO Messaging (but valid for other cases such as Explicit Messaging) is specified, such as 0. Please verify if the Connection Point value is valid for IO Messaging in the target device.	✓
0x01	0x0128	Invalid Target to Originator Size	This error is returned when the Target to Originator (Input data) size specified in the Forward Open does not match what is in Target. Consult the documentation of the Adapter device to verify the required size. Note that if the Run/Idle header is requested, it will add 4 additional bytes and must be accounted for in the Forward Open calculation. The Productivity1000 CPU does not support a Run/Idle header for the T -> O (Input) data. Some devices may publish the size that they are looking for as an additional attribute (Unsigned Integer 16 value) of the Extended Error Code. An array can be defined for this field in order to view the extended error code attributes. Note: This error may also be generated when a Connection Point value that is invalid for IO Messaging (but valid for other cases such as Explicit Messaging) is specified, such as 0. Please verify if the Connection Point value is valid for IO Messaging in the target device.	✓
0x01	0x0129	Invalid Configuration Application Path	This error will be returned by the Productivity1000 CPU if a Configuration Connection with a size other than 0 is sent to the CPU. The Configuration Connection size must always be zero if it this path is present in the Forward Open message coming from the Scanner device.	✓
0x01	0x012A	Invalid Consuming Application Path	This error will be returned by the Productivity1000 CPU if the Consuming (O -> T) Application Path is not present in the Forward Open message coming from the Scanner device or if the specified Connection Point is incorrect.	✓
0x01	0x012B	Invalid Producing Application Path	This error will be returned by the Productivity1000 CPU if the Producing (T -> O) Application Path is not present in the Forward Open message coming from the Scanner device or if the specified Connection Point is incorrect.	✓
0x01	0x012C	Config. Symbol Does not Exist	The Originator attempted to connect to a configuration tag name that is not supported in the Target.	✗
0x01	0x012D	Consuming Symbol Does not Exist	The Originator attempted to connect to a consuming tag name that is not supported in the Target.	✗
0x01	0x012E	Producing Symbol Does not Exist	The Originator attempted to connect to a producing tag name that is not supported in the Target.	✗
0x01	0x012F	Inconsistent Application Path Combination	The combination of Configuration, Consuming and Producing application paths specified are inconsistent.	✗

P1000 EtherNet/IP Error Codes

General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x0130	Inconsistent Consume data format	Information in the data segment not consistent with the format of the data in the consumed data.	✗
0x01	0x0131	Inconsistent Product data format	Information in the data segment not consistent with the format of the data in the produced data.	✗
0x01	0x0132	Null Forward Open function not supported	The target device does not support the function requested in the NULL Forward Open request. The request could be such items as “Ping device”, “Configure device application”, etc.	✗
0x01	0x0133	Connection Timeout Multiplier not acceptable	The Connection Multiplier specified in the Forward Open request not acceptable by the Target device (once multiplied in conjunction with the specified timeout value). Consult the manufacturer device’s documentation on what the acceptable timeout and multiplier are for this device.	✗
0x01	0x0203	Connection Timed Out	This error will be returned by the Productivity1000 CPU if a message is sent to the CPU on a connection that has already timed out. Connections time out if no message is sent to the CPU in the time period specified by the RPI rate X Connection multiplier specified in the Forward Open message.	✗
0x01	0x0204	Unconnected Request Timed Out	This time out occurs when the device sends an Unconnected Request and no response is received within the specified time out period. In the Productivity1000 CPU, this value may be found in the hardware configuration under the Ethernet port settings for the P1-540/550.	✓
0x01	0x0205	Parameter Error in Unconnected Request Service	This error occurs when Connection Tick Time/Connection time-out combination is specified in the Forward Open or Forward Close message this is not supported by the device.	✗
0x01	0x0206	Message Too Large for Unconnected Send Service	Occurs when Unconnected_Send message is too large to be sent to the network.	✗
0x01	0x0207	Unconnected Acknowledge without Reply	This error occurs if an Acknowledge was received but no data response occurred. Verify that the message that was sent is supported by the Target device using the device manufacturer’s documentation.	✗
0x01	0x0301	No Buffer Memory Available	This error occurs if the Connection memory buffer in the target device is full. Correct this by reducing the frequency of the messages being sent to the device and/or reducing the number of connections to the device. Consult the manufacturer’s documentation for other means of correcting this.	✗
0x01	0x0302	Network Bandwidth not Available for Data	This error occurs if the Producer device cannot support the specified RPI rate when the connection has been configured with schedule priority. Reduce the RPI rate or consult the manufacturer’s documentation for other means to correct this.	✗
0x01	0x0303	No Consumed Connection ID Filter Available	This error occurs if a Consumer device doesn’t have an available consumed_connection_id filter.	✗
0x01	0x0304	Not Configured to Send Scheduled Priority Data	This error occurs if a device has been configured for a scheduled priority message and it cannot send the data at the scheduled time slot.	✗

P1000 EtherNet/IP Error Codes

General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x0305	Schedule Signature Mismatch	This error occurs if the schedule priority information does not match between the Target and the Originator.	✗
0x01	0x0306	Schedule Signature Validation not Possible	This error occurs when the schedule priority information sent to the device is not validated.	✗
0x01	0x0311	Port Not Available	This error occurs when a port number specified in a port segment is not available. Consult the documentation of the device to verify the correct port number.	✗
0x01	0x0312	Link Address Not Valid	The Link address specified in the port segment is not correct. Consult the documentation of the device to verify the correct port number.	✗
0x01	0x0315	Invalid Segment in Connection Path	This error occurs when the target device cannot understand the segment type or segment value in the Connection Path. Consult the documentation of the device to verify the correct segment type and value. If a Connection Point greater than 255 is specified this error could occur.	✓
0x01	0x0316	Forward Close Service Connection Path Mismatch	This error occurs when the Connection path in the Forward Close message does not match the Connection Path configured in the connection. Contact Tech Support if this error persists.	✗
0x01	0x0317	Scheduling Not Specified	This error can occur if the Schedule network segment or value is invalid.	✗
0x01	0x0318	Link Address to Self Invalid	If the Link address points back to the originator device, this error will occur.	✗
0x01	0x0319	Secondary Resource Unavailable	This occurs in a redundant system when the secondary connection request is unable to duplicate the primary connection request.	✗
0x01	0x031A	Rack Connection Already established	The connection to a module is refused because part or all of the data requested is already part of an existing rack connection.	✗
0x01	0x031B	Module Connection Already established	The connection to a rack is refused because part or all of the data requested is already part of an existing module connection.	✗
0x01	0x031C	Miscellaneous	This error is returned when there is no other applicable code for the error condition. Consult the manufacturer's documentation or contact Tech support if this error persist.	✗
0x01	0x031D	Redundant Connection Mismatch	This error occurs when these parameters don't match when establishing a redundant owner connection: O -> T RPI, O -> T Connection Parameters, T -> O RPI, T -> O Connection Parameters and Transport Type and Trigger.	✗
0x01	0x031E	No more User Configurable Link Resources Available in the Producing Module	This error is returned from the Target device when no more available Consumer connections available for a Producer.	✗

P1000 EtherNet/IP Error Codes

General Status Error	Extended Status Error	Name	Description	P1000 Supported
0x01	0x031F	No User Configurable Link Consumer Resources Configured in the Producing Module	This error is returned from the Target device when no Consumer connections have been configured for a Producer connection.	✗
0x01	0x0800	Network Link Offline	The Link path is invalid or not available.	✗
0x01	0x0810	No Target Application Data Available	This error is returned from the Target device when the application has no valid data to produce.	✗
0x01	0x0811	No Originator Application Data Available	This error is returned from the Originator device when the application has no valid data to produce.	✗
0x01	0x0812	Node Address has changed since the Network was scheduled	This specifies that the router has changed node addresses since the value configured in the original connection.	✗
0x01	0x0813	Not Configured for Off-subnet Multicast	The producer has been requested to support a Multicast connection for a consumer on a different subnet and does not support this functionality.	✗
0x01	0x0814	Invalid Produce/ Consume Data format	Information in the data segment not consistent with the format of the data in the consumed or produced data. Errors 0x0130 and 0x0131 are typically used for this situation in most devices now.	✗
0x02	N/A	Resource Unavailable for Unconnected Send	The Target device does not have the resources to process the Unconnected Send request.	✗
0x04	N/A	Path Segment Error in Unconnected Send	The Class, Instance or Attribute value specified in the Unconnected Explicit Message request is incorrect or not supported in the Target device. Check the manufacturer's documentation for the correct codes to use.	✗
0x09	Index to error	Error in Data Segment	This error code is returned when an error is encountered in the Data segment portion of a Forward Open message. The Extended Status value is the offset in the Data segment where the error was encountered.	✗
0x0C	Optional	Object State Error	This error is returned from the Target device when the current state of the Object requested does not allow it to be returned. The current state can be specified in the Optional Extended Error status field.	✗
0x10	Optional	Device State Error	This error is returned from the Target device when the current state of the Device requested does not allow it to be returned. The current state can be specified in the Optional Extended Error status field.	✗
0x13	N/A	Not Enough Data	Not enough data was supplied in the service request specified.	✗
0x15	N/A	Too Much Data	Too much data was supplied in the service request specified.	✗