BRX Do-more!
Ethernet Communications

Chapter 13

In This Chapter...

Overview

The purpose of this chapter is to help the user gain an understanding of the Ethernet communications capabilities of the BRX MPU. The BRX platform features the Do-more! DM1 technology which has a very robust communications system allowing the programmer to accomplish many tasks that can prove difficult with other PLCs. Many BRX MPUs have a built-in Ethernet port. A Pluggable Option Modules (POM) is also available that will add an additional Ethernet port.

The BRX platform provides a wide variety of Ethernet protocols to choose from and is capable of Ethernet communications to a wide variety of field devices.

The available protocols are:

- Peerlink – This is possibly the easiest of all of the protocols to implement. Consisting of multiple shared areas of memory, you place values into these memory locations and set the instruction to share them with other Do-more! PLCs.
- Do-more! Protocol (Client, Server) – The Do-more! protocol is a proprietary protocol that is used exclusively by the Do-more! family of controllers. This is a very feature rich and secure protocol for communication between Do-more! controllers.
- Modbus TCP (Client, Server) – Modbus TCP is one of the most widely used protocols for PLCs and other industrial devices. This protocol is overseen by Modbus.org. This protocol is an open standard meaning that anyone can utilize it freely. Modbus TCP can be utilized in either a client or server configuration. It supports Clients and Servers in a Peer to Peer fashion.
- HOST Ethernet (ECOM) Protocol (Client, Server) – The BRX Do-more! MPU can serve as a HOST Ethernet protocol client and server to communicate to legacy devices that utilize the HOST Ethernet protocol such as DirectLogic PLCs, C-more HMI, SCADA systems, etc.
- EtherNet/IP: Explicit Messaging (Client, Server) – EtherNet/IP is one of the fastest growing Ethernet protocols available today. It is overseen by ODVA.org. Used by a wide variety of PLCs and industrial devices, it has several variants. Explicit messaging is supported by Do-more! Controllers. Implicit messaging is currently not supported. Both client and server configurations of Explicit messaging are supported.
- SMTP (Email) – All Do-more! CPUs (H2 series, T1H series, and BRX) support unsecure SMTP email services. In addition, the BRX Do-more! CPUs with built-in Ethernet ports support secure SMTP email services that use SSL or TLS encryption.
- Ethernet Remote IO Master – All Do-more! CPUs support the use of remote expansion bases to expand the IO count of the controller. More information on this can be found in Chapter 14 – BRX Remote I/O: BX-DMIO, BX-EBC100 & GS-EDRV100.
- HTTP – BRX MPUs support Hypertext Transfer Protocol (HTTP). The following 5 HTTP commands are supported: GET, HEAD, POST, PUT and DELETE. SSL and TLS encryption are also supported.
- MQTT – MQTT was designed as a machine to machine/Interconnected Internet of Things (IIOT) protocol. It is extremely lightweight and useful where bandwidth is at a premium. BRX Do-more! MPUs support both Publish and Subscribe mechanisms to an external broker. BRX Do-more! MPUs cannot be a Broker.

Overview, continued

- SNTP (Time Server) – All Do-more! CPUs support syncing with a time server to synchronize the time. This server can be either local or an Internet based server such as time.nist.gov.
- TCP/IP (Raw packet) – The Do-more! CPUs can connect via raw commands and send and receive packets via TCP/IP. This is useful for creating or replicating a protocol that is not currently supported by the processor. To do this, you will need to have knowledge of the protocol that you wish to use.
- UDP/IP (Raw packet) - The Do-more! CPUs can connect via raw commands and send and receive packets via UDP/IP. This is useful for creating or replicating a protocol that is not currently supported by the processor. To do this, you will need to have knowledge of the protocol that you wish to use.
- DNS Lookup – Do-more! CPUs support looking up URLs via DNS if a DNS server is available. This can be useful for translating URLs to IP addresses that may change frequently such as email server addresses.
- Ping – Ping is a useful IMCP utility that can check to see if an IP address is online and available. This is frequently used with DNS lookup before sending an email to check if the current IP address of an email server is valid.

In addition to this chapter, the software help file is a valuable resource when connecting and communicating with the various protocols needed for these types of devices.

General Concepts

The BRX Do-more! MPU has tightly coupled security features incorporated into it to protect your installation. See the manual section for each protocol and also the Do-more! Designer software help file for more information on password protection and Protocol Specific Memory.

Multiple user accounts are supported and logged when accessed. When programming, session based communication with unique IDs is utilized to prevent unauthorized access.

The Do-more! Protocol can access the full memory structure and is capable of utilizing a security account to protect the data. Some HMIs and SCADA software such as C-more, C-more Micro and Point of View, can utilize the Do-more! Protocol for enhanced security while accessing the full memory area.

Protocol Specific memory areas are blocked out of the User Memory when using Modbus TCP or HOST Ethernet. These memory areas are only accessible externally by using the specific protocol that corresponds to the type of memory being accessed. They are usable by the programmer to pass data externally when using third party devices. This is discussed in more detail in each Protocol section later in this chapter.

Terminology

During the course of this chapter we will use terminology and phrases that are specific to a Protocol or Physical Medium the user should understand. By way of explanation we have included some common terms and definitions in this section. Definitions and explanations of specific parameters particular to each Protocol will be discussed later in this chapter.

Physical Medium – Wires, radios, cellular service, or satellite link. The physical method (hardware) on which the data is being transmitted or received. The physical medium contains no data information. Examples of a physical medium are: RS-232, RS-485 and Ethernet 10/100 Base T.

Protocol – A Protocol is the specification for the formatting of the data (bits, bytes and words) being transmitted through the physical medium. Examples of some common industry protocols are: Modbus TCP and EtherNet/IP.

One way to think of the Physical Medium and the Protocol is to liken them to placing a phone call. The phone call is being placed over wires, cellular service or even perhaps a satellite link. This is the physical medium. Now if the call was to China, you would say “Hello” in English. If the person on the other end understands English, they will respond with “Hello”. If the person on the other end only understands Mandarin Chinese they might respond “Ni Hao”. If you do not understand Chinese, you will be confused as to what they are saying. This is the same for a Protocol. If your PLC uses Modbus TCP and you try to talk to a PLC that only understands EtherNet/IP, then you will not be able to communicate with it. The Protocols must match in order to communicate.

Client (Master) – A Client is a Master device that requests data from a Server (Slave) device. The Client (Master) device initiates communications with a Server (Slave) device.

Server (Slave) – A Server is a Slave device that responds to a request from a Client (Master) device. The Server (Slave) device listens/replies to requests made by a Client (Master).

UDP – User Datagram Protocol (UDP) is part of the Transport Layer of the Internet Protocol Suite. The Transport Layer is Layer 4 in the Open Systems Interconnection (OSI) reference model. UDP is a simple connectionless transport mechanism for Ethernet packets. Checksums are used for data integrity, however it has no error correction or guarantee of delivery. It is considerably faster than TCP/IP due to these factors. It is widely used when data is time sensitive because dropped packets may be preferable to retries and delays.

TCP – TCP or TCP/IP is part of the Transport Layer of the Internet Protocol Suite. The Transport Layer is Layer 4 in the OSI reference model. TCP is defined as a reliable, ordered and an error checked delivery method. TCP/IP is most often used when the data integrity is more important than raw speed.

Field Device – A device external to the BRX MPU.

NOTE: *Software dialog windows and other documentation may interchangeably use the terms Client/Server or Master/Slave when referring to requesting/responding devices in Ethernet or Serial communications. In this chapter, with regards to Ethernet communications, we will use the term Client to refer to a requesting device and Server to refer to a responding device.*

IP Addressing and Subnets

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.

We suggest that users maintain their IP addressing and Subnets according to common networking practices by using private network addressing when at all possible. Examples of private addressing IP ranges can be found in the table below.

IP Addressing and Subnets

IP Address Range	Typical Subnet	Classful Description
192.168.0.0–192.168.255.255	255.255.0.0	Class C Network
172.16.0.0–172.31.255.255	255.240.0.0	Class B Network
10.0.0.0–10.255.255.255	255.0.0.0	Class A Network

***NOTE:** Notice that the IP range of 169.254.0.1 – 169.254.255.254 is not listed above. This range is reserved for APIPA addressing and is not recommended for use with most networks.*

Connecting to the Internet

We recommend that users of PLCs, HMI products and SCADA systems perform your own network security analysis to determine the proper level of security required for you application. However, the Department of Homeland Security's National Cybersecurity and Communications Integration Center (NCCIC) and Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) has provided direction related to network security and safety under an approach described as "Defense in Depth", which is published at https://www.us-cert.gov/sites/default/files/recommended_practices/NCCIC_ICS-CERT_Defense_in_Depth_2016_S508C.pdf.

This comprehensive security strategy involves physical protection methods, as well as process and policy methods. This approach creates multiple layers and levels of security for industrial automation systems. Such safeguards include the location of control system networks behind firewalls, their isolation from business networks, the use of intrusion detection systems, and the use of secure methods for remote access such as Virtual Private Networks (VPNs). Further, users should minimize network exposure for all control system devices and such control systems and these systems should not directly face the internet. Following these procedures should significantly reduce your risks both from external sources as well as internal sources, and provide a more secure system.

It is the user's responsibility to protect such systems, just as you would protect your computer and business systems. AutomationDirect recommends using one or more of these resources in putting together a secure system:

- ICS-CERT's Control Systems recommended practices at the following web address:
  https://ics-cert.us-cert.gov/Recommended-Practices

- Special Publication 800-82 of the National Institute of Standards and Technology – Guide to Industrial Control Systems (ICS) Security
  https://csrc.nist.gov/publications/detail/sp/800-82/rev-2/final

Connecting to the Internet, continued

- ISA99, Industrial Automation and Control Systems Security http://www.isa.org/MSTemplate.cfm?MicrositeID=988&CommitteeID=6821 (please note this is a summary and these standards have to be purchased from ISA)

The above set of resources provides a comprehensive approach to securing a control system network and reducing risk and exposure from security breaches.

Given the nature of any system that accesses the internet, it is incumbent upon each user to assess the needs and requirements of their application, and take steps to mitigate the particular security risks inherent in their control system.

Troubleshooting Ethernet Communications

Troubleshooting communications can seem frustrating to the novice because you cannot see what is happening in real time. However, most communications troubleshooting is straightforward if you take the right steps. Below are some recommendations for troubleshooting when a device does not communicate.

1. Communications is not instantaneous. This is a common misconception with Ethernet because Ethernet communications tends to be fast. It does take time for transactions to complete.
2. Ethernet has a capacity. This goes hand in hand with #1. Ethernet communications does have limits on capacity. It is not a "set everything at max" medium. Poll times may need to be adjusted to give everything some "breathing room".
3. IP addresses and subnets must match. Ethernet communications will fail if the IP addressing, subnets and/or gateways are not set properly.
4. Managed switches and routers will not always pass every Ethernet packet. Most block broadcast transmissions by default. This can cause issues depending on what you are trying to do.
5. Ethernet communications over the Internet can be extremely slow due to latency. This needs to be taken into account when trying to do long distance communications.
6. Cellular modems can cause issues with Ethernet communications due to overall slowness, latency and a technique called packet bursting. Packet bursting (or Frame bursting) is where the cell modem and/or the cell tower stores multiple Ethernet packets and then pushes them out all at once without regard for the proper waiting period. While this is allowed by wireless standards, wired devices connected to a wireless modem or router can have issues with this technique.
7. Ethernet is inherently good at handling noise. This does not mean that Ethernet is immune to noise issues from VFDs, welders, coils, contactors, etc. This is especially true when the Ethernet switch is powered by the same power supply as other noise producing industrial devices. For best immunity, put the Ethernet switch on its own power supply.
8. Ethernet switches, hubs, routers, etc. can have a single or sometimes multiple bad ports that may be causing issues. This is a rare occurrence, but it is possible. Try a different device and see if it helps the issue.
9. Ethernet cables can cause issues as well. Try a different cable.
10. If you are using wireless Ethernet and having issues, try using the devices in a wired configuration to see if the issue is with the wireless device or if it is a configuration issue.
11. Ethernet devices can suffer from timeouts due to latency. Try extending the timeout period to see if that is the issue. Ping is a useful command to use from your PC as well as the PLC to see the potential

Troubleshooting Ethernet Communications, continued

latency of a connection. A rule of thumb for setting timeouts is to start with the maximum ping time multiplied by at least 150%. So, if you have a ping time of 250ms, you should use at least 375ms as your timeout. Note that this is a starting point for adjusting the timeout and not a hard number that will work for every situation.

Ethernet Traffic Congestion

While several protocols can be enabled on the BRX Do-more! CPU Ethernet port, it should be noted that Ethernet can handle a finite amount of traffic. The amount of traffic load on the Ethernet port should be considered and polling times for Clients adjusted accordingly to allow ample time for the BRX Do-more! CPU to respond to and create requests for other devices.

NOTE: *Care should be taken when adding a BRX Do-more! MPU to an existing network as some protocols can create a significant traffic load.*

Communications Design

The BRX platform features the Do-more! DM1 technology which has a very robust communications system allowing the programmer to accomplish many tasks that can prove difficult with other PLCs. The DM1 technology manages the asynchronous actions of communications in the background without taking a huge hit on the PLC scan time.

Managing Communications Devices

When you make a connection with an Ethernet protocol as a Client (Master) in a Do-more! Processor, you will have created a “Device” for that connection. Each Client Device has its own timeout settings. To take full advantage of this feature, it is advisable to create a Client Device for each Server (Slave) that you will be talking to. This allows for other Ethernet communications to continue uninterrupted if one of the Servers (Slaves) falls offline. See Help topic DMD0248 in the Do-more! Designer Software for more information on creating additional Client Devices.

Modbus/TCP can benefit from using multiple Devices to manage connections because the TCP connection must be shut down and restarted for each Client (Slave) that it talks to. Think of each Device as a telephone. If I have one, I must call, talk, and hang up for each target. If I have four phones, I can call each target individually and keep all lines open. Devices work the same way. By having multiples, you can achieve faster turn around times in your communications.

Port Numbers

When doing TCP 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, while the Server device must listen for this Destination Port number. After the Server device has received the message with the Destination Port Number on which it is listening, 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 BRX Do-more! platform. Some of the Port Numbers are configurable, allowing more flexibility when going through many different router applications.

Default Port Numbers

Ethernet	Port Numbers	TCP or UDP	Configurable
Peerlink	28784	UDP	No
Do-more! Protocol (Client, Server)	28784	UDP	Yes*
Modbus TCP (Client, Server)	502	TCP	Yes
HOST Ethernet Protocol (Client, Server)	28784	UDP	No
EtherNet/IP: Explicit Messaging (Client, Server)	44818	TCP	Yes
SMTP (Email)	25	TCP	Yes
Ethernet Remote I/O	28784	UDP	No
HTTP	80, 443	TCP	Yes
MQTT	1883	TCP	No
SNTP (Time Server)	123	TCP	No
TCP Raw Packet	User defined	TCP	Yes
UDP Raw Packet	User defined	UDP	Yes

** Secondary Ethernet Connection can be enabled for the built-in Ethernet port of a BRX MPU. By default it has a UDP Port Number of 5000. It can be configured to any decimal number between 5000 and 65535, except for 28784.*

Sequencing Communications

The easy way

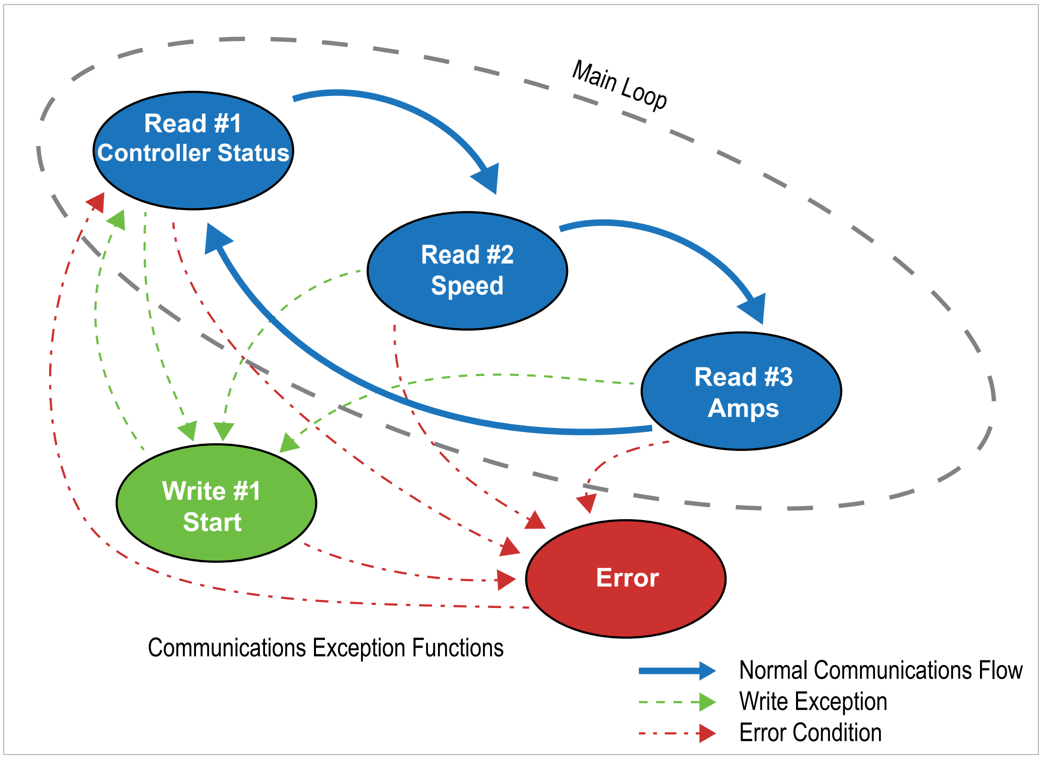
In many PLCs the programmer must manage the communications sequencing. This can be frustrating and time consuming to get the communications working correctly especially when you have multiple requests to handle. The DM1 technology eliminates this frustration and will simply manage each Comm request in the order that the program initiates a request. You no longer must write code to sequence and manage each request. For some applications this is enough to do what the programmer needs to accomplish.

More advanced sequencing – State Machines

State machines have been used to sequence communication sequences for a very long time. The reason for this is that communications by definition consists of always being in a particular state and transitioning to a different state based on some action. This is all that there is to state machine programming; do an action and wait for something to happen, then move to a different action.

Do-more! supports state machine programming. This is accomplished by the use of Stages. Stage programming has been around a long time and Do-more! PLCs take full advantage of Stage programming. For more information on Stage Programming see Do-more! Designer Help topic DMD0502.

For an example on State Machine programming using Stages, consider talking to a variable speed motor controller. What are the conditions to talk to the motor controller? By drawing out the conditions, you can see that there are definite conditions or states that the communications to the motor controller will be going through.


Main Loop
Read #1 Controller Status
Read #2 Speed
Read #3 Amps
Write #1 Start
Error
Communications Exception Functions
Normal Communications Flow
Write Exception
Error Condition

In this example we can see that we have three reads from the device and one write condition. We can also see that we need an error handling condition. The beauty of drawing the state machine conditions out in a flow chart format is that it allows you to easily see and translate the conditions to Ladder Logic code.

Sequencing Communications, continued

Main loop:

Read #1 – Read the Controller status word. This might consist of whether the drive is running, jogging, in error, etc.

Read #2 – Read the Controller speed. How fast is it running?

Read #3 – Read the amp draw. Is it within limits?

Main loop end

Things that happen occasionally:

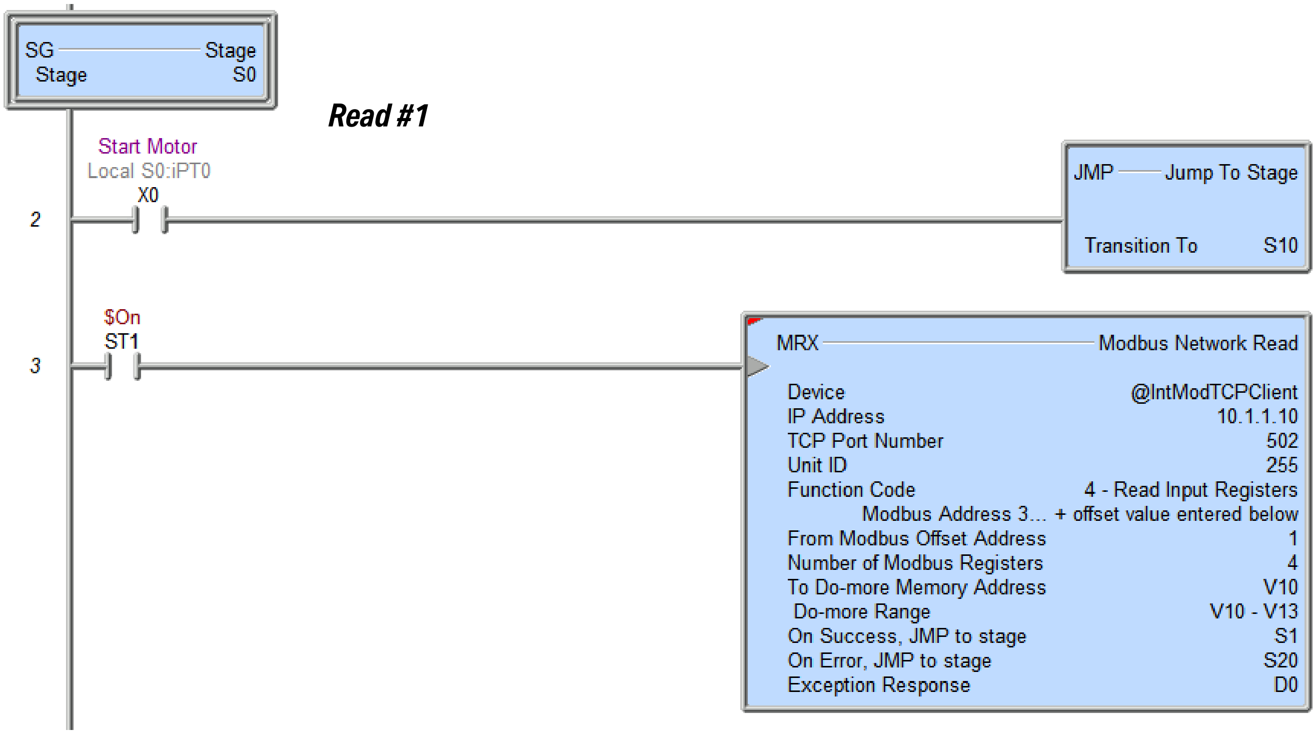
Write #1– Start the Controller running.

Error – Something bad happened to the communications. What do we need to do to flag this as a problem?

An implementation of this State Machine in ladder logic is shown below. Since the Do-more! Communication instructions have built in transitioning between Stages based on Success or Error, this eliminates a lot of programming. To accomplish the diagram the only manual transitioning that needs to occur is for the Write to start the motor controller.

While this example is somewhat incomplete in that it does not have provisions for stopping the drive or other potential needs, it does serve to show how State Programming can help to make transitioning between communications states relatively easy in the Do-more! Controller.

Main loop:



Read #1

SG Stage — Stage S0

2 — Start Motor, Local S0:iPT0, X0 — JMP Jump To Stage, Transition To S10

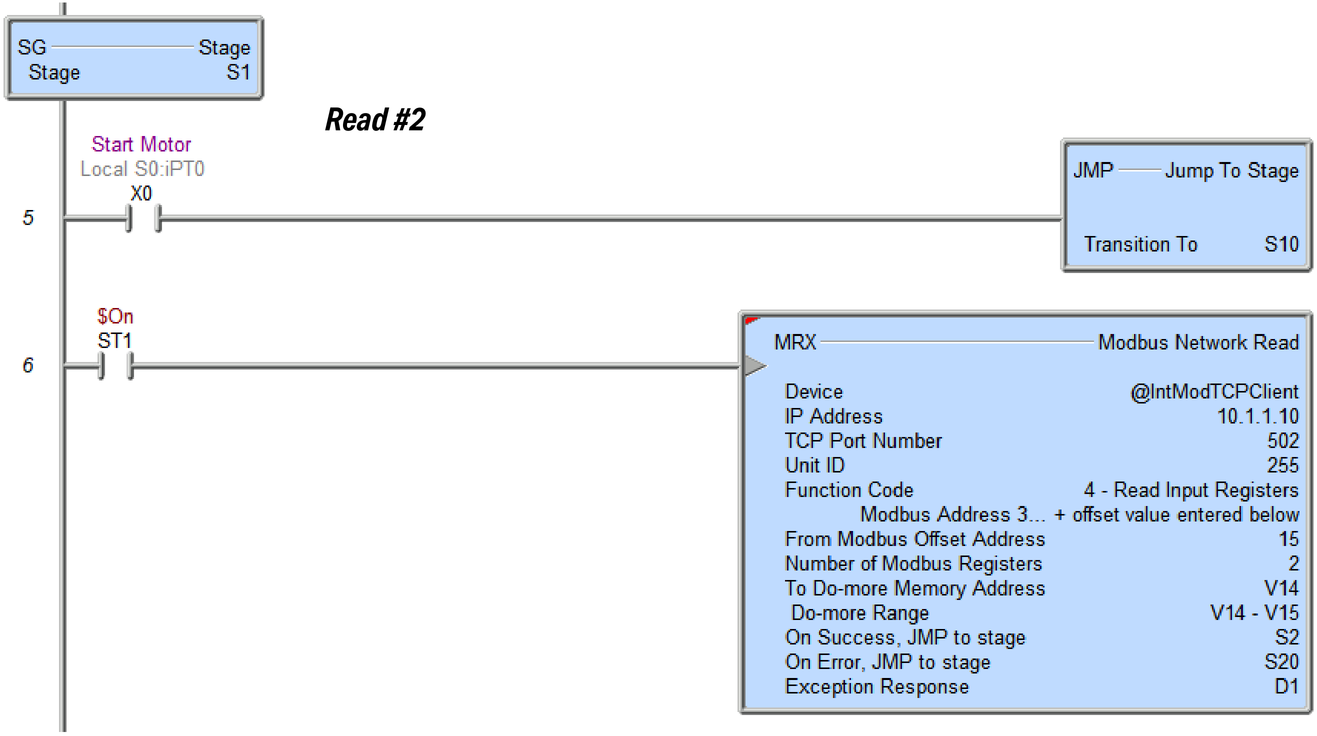
3 — $On, ST1 — MRX Modbus Network Read

Device	@IntModTCPClient
IP Address	10.1.1.10
TCP Port Number	502
Unit ID	255
Function Code	4 - Read Input Registers
Modbus Address 3...	+ offset value entered below
From Modbus Offset Address	1
Number of Modbus Registers	4
To Do-more Memory Address	V10
Do-more Range	V10 - V13
On Success, JMP to stage	S1
On Error, JMP to stage	S20
Exception Response	D0

(continued on next page)

Sequencing Communications, continued

(continued from previous page)



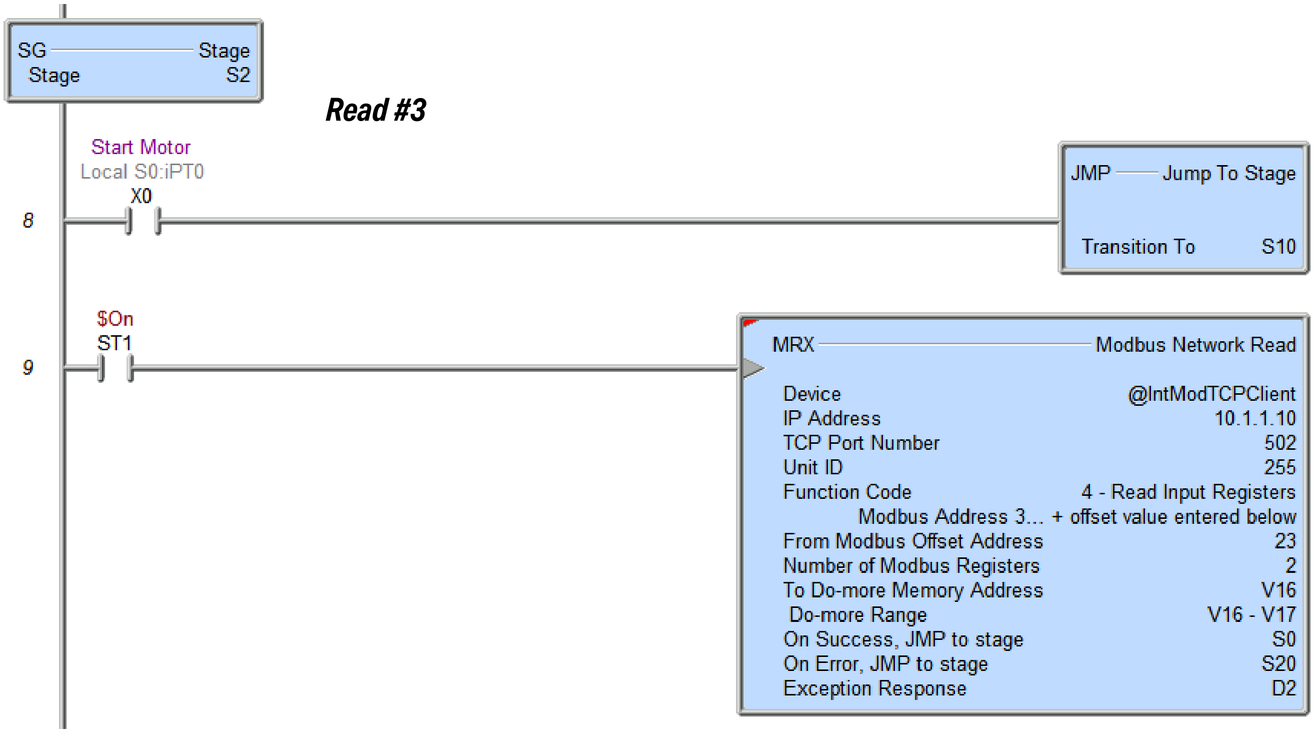
Read #2

JMP — Jump To Stage

Transition To S10

MRX — Modbus Network Read

Device	@IntModTCPClient
IP Address	10.1.1.10
TCP Port Number	502
Unit ID	255
Function Code	4 - Read Input Registers
Modbus Address 3...	+ offset value entered below
From Modbus Offset Address	15
Number of Modbus Registers	2
To Do-more Memory Address	V14
Do-more Range	V14 - V15
On Success, JMP to stage	S2
On Error, JMP to stage	S20
Exception Response	D1



Read #3

JMP — Jump To Stage

Transition To S10

MRX — Modbus Network Read

Device	@IntModTCPClient
IP Address	10.1.1.10
TCP Port Number	502
Unit ID	255
Function Code	4 - Read Input Registers
Modbus Address 3...	+ offset value entered below
From Modbus Offset Address	23
Number of Modbus Registers	2
To Do-more Memory Address	V16
Do-more Range	V16 - V17
On Success, JMP to stage	S0
On Error, JMP to stage	S20
Exception Response	D2

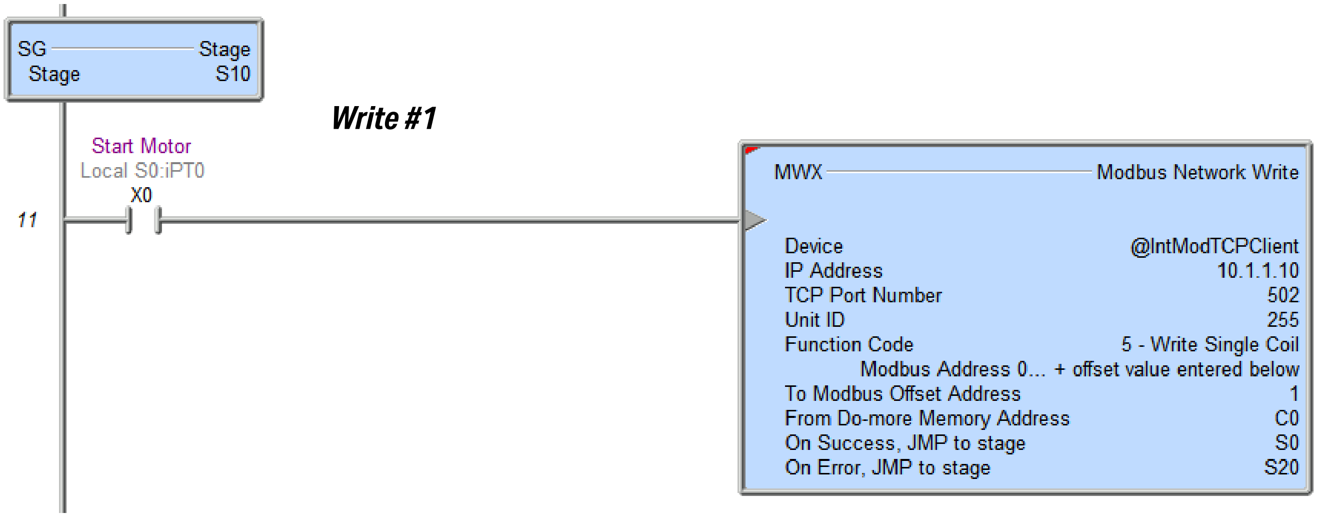
Main loop end

(continued on next page)

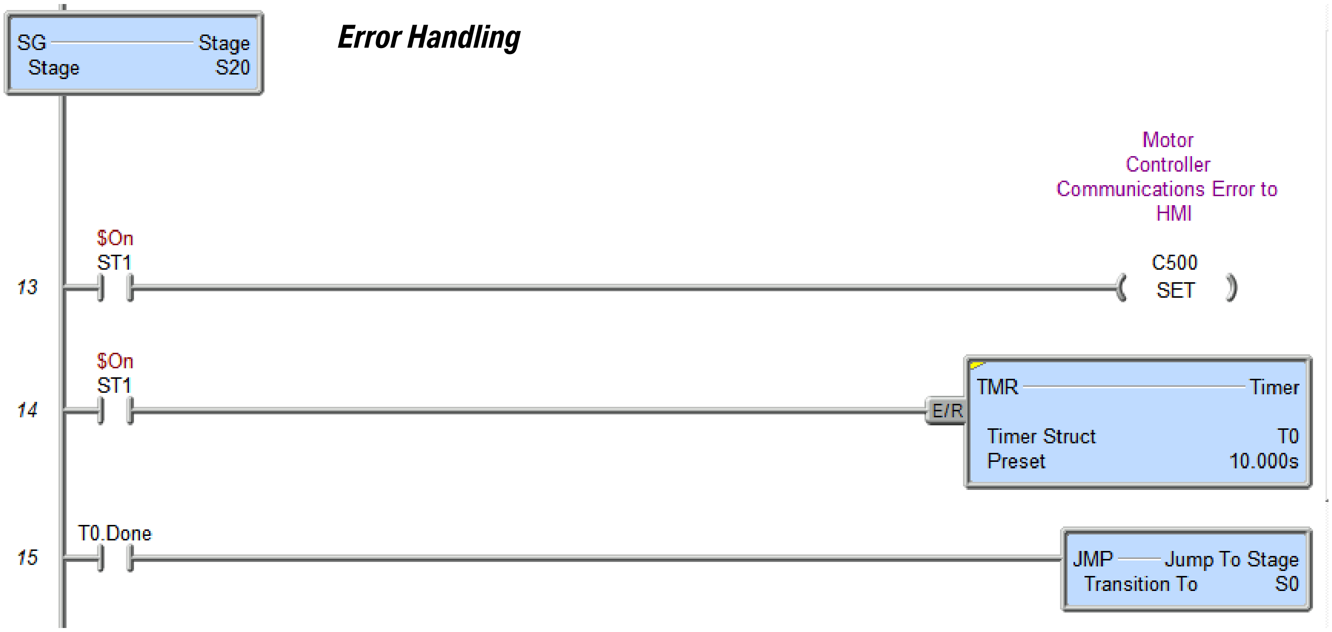
Sequencing Communications, continued

(continued from previous page)

Things that happen occasionally:



Device	@IntModTCPClient
IP Address	10.1.1.10
TCP Port Number	502
Unit ID	255
Function Code	5 - Write Single Coil
Modbus Address 0... + offset value entered below	
To Modbus Offset Address	1
From Do-more Memory Address	C0
On Success, JMP to stage	S0
On Error, JMP to stage	S20



Timer Struct T0
Preset 10.000s

Transition To S0

Ethernet

There are multiple options when doing Ethernet communications with a BRX PLC. Some MPUs have a built in Ethernet port. The BRX POM modules also have an Ethernet option

See the table below for a matrix of what protocols are supported by the different communications options.

Supported Ethernet Communications Protocols

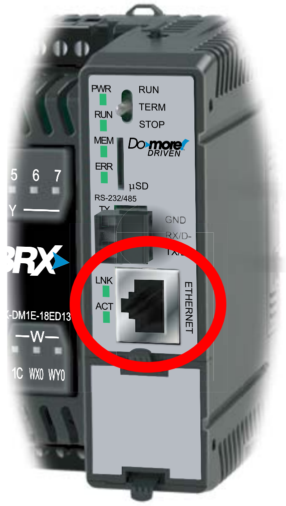
Protocol	Onboard	Ethernet POM
Peerlink	X	
Do-more Protocol (Slave)	X	X
Modbus TCP (Client)	X	
Modbus TCP (Server)	X	X
HOST Ethernet (Client)	X	
HOST Ethernet (Server)	X	X
EtherNet/IP (Client)	X	
EtherNet/IP (Server)	X	X
SMTP (Email)	X	
Ethernet Remote I/O (Master)	X	
HTTP	X	
MQTT	X	
SNTP (Time Server)	X	
TCP/IP Raw Packet	X	
UDP/IP Raw Packet	X	
DNS Lookup	X	
Ping	X	

MPU Built-in Ethernet Port

The RJ-45 Ethernet port connector is located on the CPU faceplate. Rated at 10/100 Mbps, it accepts standard CAT5e cable and has built-in auto-crossover capability.

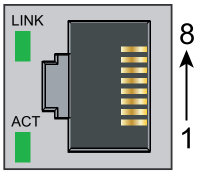
Ethernet Port Specifications

Port Name	ETHERNET
Ethernet Port Type	RJ45, CAT5e, 10/100 BASE-T, Auto Crossover
Description	Standard transformer isolated Ethernet port with built-in surge protection
Transfer Rate	10 Mbps and 100 Mbps (Green LED)
Port Status LED	LED is solid when network LINK is established. LED flashes when port is active (ACT).
Supported Protocols	Peerlink Do-more! Protocol (Client, Server) Modbus TCP (Client, Server) HOST Ethernet Protocol (Client, Server) EtherNet/IP: Explicit Messaging (Client, Server) SMTP (Email) HTTP MQTT SNTP (Time Server) TCP/IP (Raw packet), UDP/IP (Raw packet)
Cable Recommendation	C5E-STxxx-xx from AutomationDirect.com
Ethernet Port Numbers	
Modbus TCP	502 (configurable), TCP
EtherNet I/P (Explicit Messaging)	44818 (configurable), UDP
HOST Ethernet Protocol	28784, UDP
Do-more! Protocol	28784, UDP



Ethernet POM

The Ethernet LT POM can be connected to the Do-more! Designer programming software or HMI's that support the Do-more!, Modbus, or K-sequence protocol. This POM functions only as a server device.



Pin #	Signal
1	TXD+ Transmit Data
2	TXD- Transmit Data
3	RXD+ Receive Data
4	N/C
5	N/C
6	RXD- Receive Data
7	N/C
8	N/C

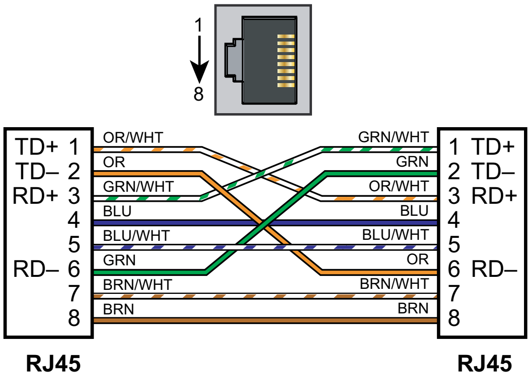
BX-P-ECOMLT Specifications	
Description	Standard transformer isolated Ethernet port with built-in surge protection.
Transfer Rate	1 Mbps throughput max
Port Status LED[1]	LINK LED is solid when network link is established. ACT LED flashes when port is active.
Supported Protocols	Do-more! Protocol (Server) Modbus (Server) K-sequence (Server) Programming/Monitoring
Cable Recommendation	C5E-STxxx-xx from AutomationDirect.com
Maximum Distance	100 meters (328 feet)
Port Type	RJ45, Category 5, Auto Crossover
Ethernet Port Numbers: Do-more! Slave Protocol Modbus TCP Slave K-sequence Slave Programming/Monitoring	 28784, UDP 502, TCP 28784, UDP 28784, UDP
Hot Swappable	Yes
Software Version Required	Do-more! Designer version 2.0 or later

1. Link LED will Blink when no cable is attached, showing that the POM has a connection to the CPU and is waiting for a working link.

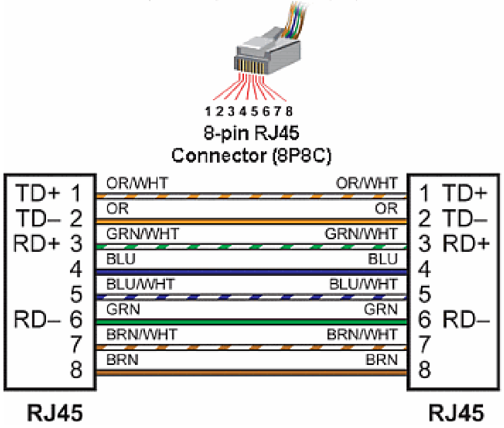
Wiring

The Ethernet port on the BRX Do-more! CPU's utilizes standard networking cables and devices. Category 5e cabling is preferred. Cables can be either patch (straight through) or crossover as the BRX Do-more! CPU Ethernet port has Auto MDI detection.

Crossover Cable
10Base-T/100Base-TX



Patch (Straight-through) Cable



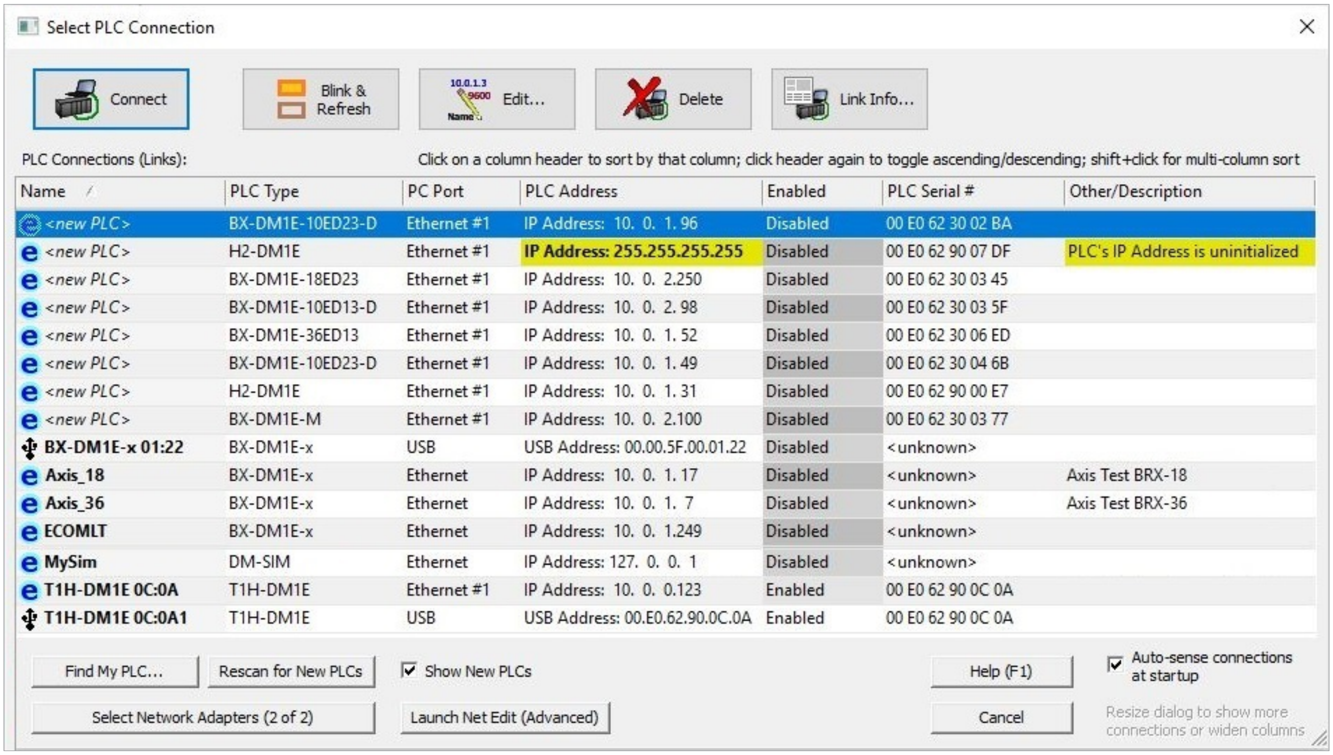
Ethernet Port Setup

The Ethernet port can be set up when you connect to the BRX MPU for the first time with the Do-more! Designer Software. Alternatively, the Ethernet port of a BRX MPU can be set up using the NetEdit utility. More information on this utility can be found in Appendix E.

Select PLC Connection

The *Select PLC Connection* dialog provides the tools that are necessary to create connections to new PLCs over the Ethernet, serial and USB ports on the PC running Do-more! Designer, and to manage those connections later as needed. The buttons along the top are operations that can be performed on the connection that is currently highlighted in the center section. The center section is a list of the currently defined connections and any PLC that can be detected that are not yet used in a connection. The selections along the bottom are used to help locate PLCs that should be in the center list but are currently not there.

When this utility is first opened, it will scan all of the network adapters, serial ports and USB ports for PLCs that are currently accessible. If the PLC is used in an existing connection its current state will be shown in the PLC Connections table. If a PLC is not currently used in a connection, it will be marked as a <new PLC> in the table.



PLC Connections (Links):

Click on a column header to sort by that column; click header again to toggle ascending/descending; shift+click for multi-column sort

Name	PLC Type	PC Port	PLC Address	Enabled	PLC Serial #	Other/Description
<new PLC>	BX-DM1E-10ED23-D	Ethernet #1	IP Address: 10. 0. 1. 96	Disabled	00 E0 62 30 02 BA	
<new PLC>	H2-DM1E	Ethernet #1	IP Address: 255.255.255.255	Disabled	00 E0 62 90 07 DF	PLC's IP Address is uninitialized
<new PLC>	BX-DM1E-18ED23	Ethernet #1	IP Address: 10. 0. 2.250	Disabled	00 E0 62 30 03 45	
<new PLC>	BX-DM1E-10ED13-D	Ethernet #1	IP Address: 10. 0. 2. 98	Disabled	00 E0 62 30 03 5F	
<new PLC>	BX-DM1E-36ED13	Ethernet #1	IP Address: 10. 0. 1. 52	Disabled	00 E0 62 30 06 ED	
<new PLC>	BX-DM1E-10ED23-D	Ethernet #1	IP Address: 10. 0. 1. 49	Disabled	00 E0 62 30 04 6B	
<new PLC>	H2-DM1E	Ethernet #1	IP Address: 10. 0. 1. 31	Disabled	00 E0 62 90 00 E7	
<new PLC>	BX-DM1E-M	Ethernet #1	IP Address: 10. 0. 2.100	Disabled	00 E0 62 30 03 77	
BX-DM1E-x 01:22	BX-DM1E-x	USB	USB Address: 00.00.5F.00.01.22	Disabled	<unknown>	
Axis_18	BX-DM1E-x	Ethernet	IP Address: 10. 0. 1. 17	Disabled	<unknown>	Axis Test BRX-18
Axis_36	BX-DM1E-x	Ethernet	IP Address: 10. 0. 1. 7	Disabled	<unknown>	Axis Test BRX-36
ECOMLT	BX-DM1E-x	Ethernet	IP Address: 10. 0. 1.249	Disabled	<unknown>	
MySim	DM-SIM	Ethernet	IP Address: 127. 0. 0. 1	Disabled	<unknown>	
T1H-DM1E 0C:0A	T1H-DM1E	Ethernet #1	IP Address: 10. 0. 0.123	Enabled	00 E0 62 90 0C 0A	
T1H-DM1E 0C:0A1	T1H-DM1E	USB	USB Address: 00.E0.62.90.0C.0A	Enabled	00 E0 62 90 0C 0A	

☑ Show New PLCs

☑ Auto-sense connections at startup

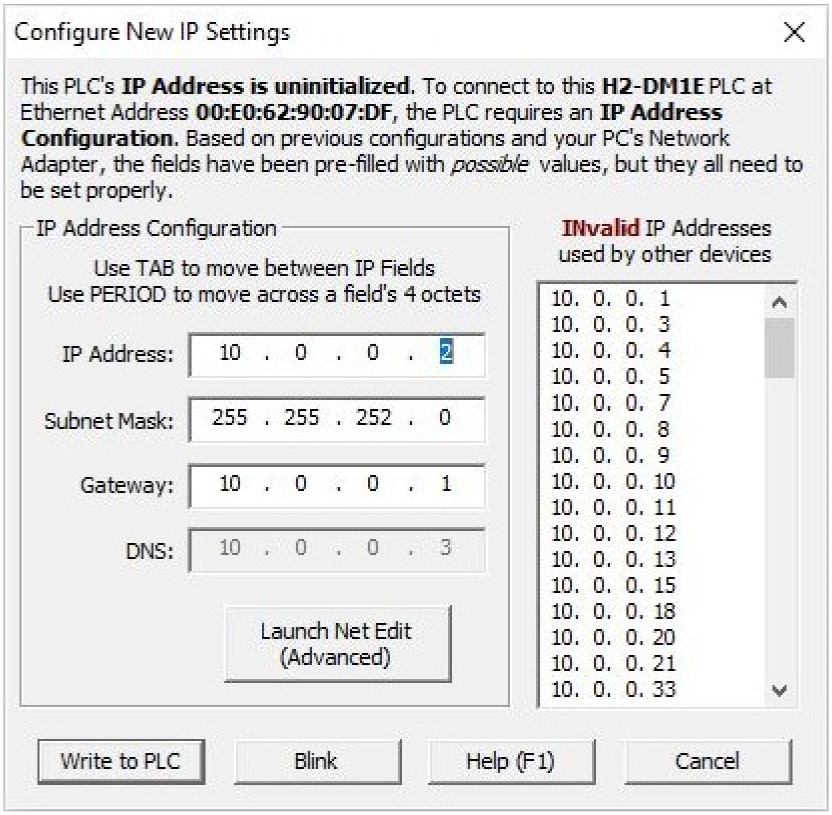
Resize dialog to show more connections or widen columns

Clicking the **Connect** button will cause Do-more! Designer to attempt to connect with the currently highlighted PLC from the list. Depending on the current state of the PLC there could be one of several things happen:

If the PLC's IP Address has not been configured the *Configure New IP Settings* dialog will be displayed. Before an Ethernet connection can be made, the PLC's Ethernet port must have its IP Address information configured with values that allow it to operate on the same network and the PC attempting to connect to it. To that end, when a PLC is selected that does not have its IP Address configuration setup the *Configure New IP Settings* dialog will be displayed.

Ethernet Port Setup, continued

When this dialog is first opened, it will examine the Ethernet settings of the NIC where the PLC was found and prefill the Subnet Mask, Gateway, and DNS fields with the same values as the NIC where it was found. It will then search for an unused IP Address on the same network as the NIC and prefill that field with a unique address. It will also generate a list of the IP Addresses on that network that are already in use by another device.



Configure New IP Settings

This PLC's **IP Address is uninitialized.** To connect to this **H2-DM1E** PLC at Ethernet Address **00:E0:62:90:07:DF**, the PLC requires an **IP Address Configuration.** Based on previous configurations and your PC's Network Adapter, the fields have been pre-filled with *possible* values, but they all need to be set properly.

IP Address Configuration

Use TAB to move between IP Fields
Use PERIOD to move across a field's 4 octets

IP Address: 10 . 0 . 0 . 2

Subnet Mask: 255 . 255 . 252 . 0

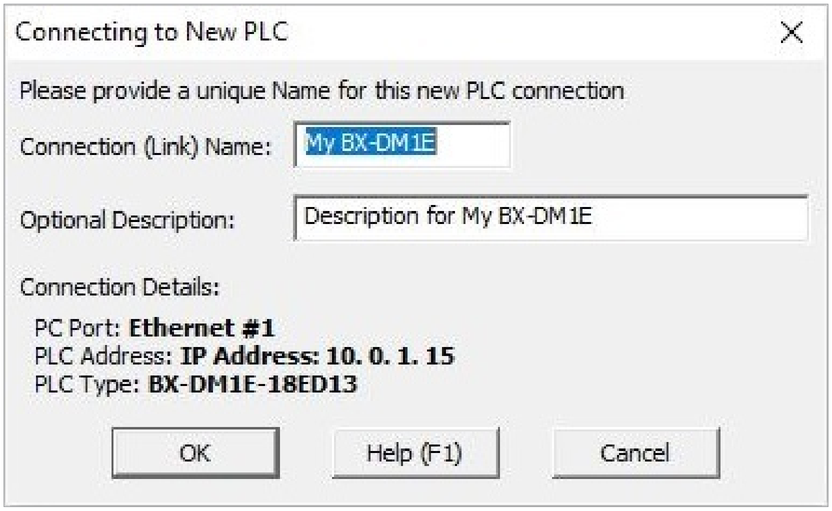
Gateway: 10 . 0 . 0 . 1

DNS: 10 . 0 . 0 . 3

INvalid IP Addresses used by other devices

```
10. 0. 0. 1
10. 0. 0. 3
10. 0. 0. 4
10. 0. 0. 5
10. 0. 0. 7
10. 0. 0. 8
10. 0. 0. 9
10. 0. 0. 10
10. 0. 0. 11
10. 0. 0. 12
10. 0. 0. 13
10. 0. 0. 15
10. 0. 0. 18
10. 0. 0. 20
10. 0. 0. 21
10. 0. 0. 33
```

If the PLC’s IP Address has been configured but the PLC is not already used in a connection the ***Connect to New PLC*** dialog will be displayed. Each connection must have a unique name, and each can have an optional description.



Connecting to New PLC

Please provide a unique Name for this new PLC connection

Connection (Link) Name: My BX-DM1E

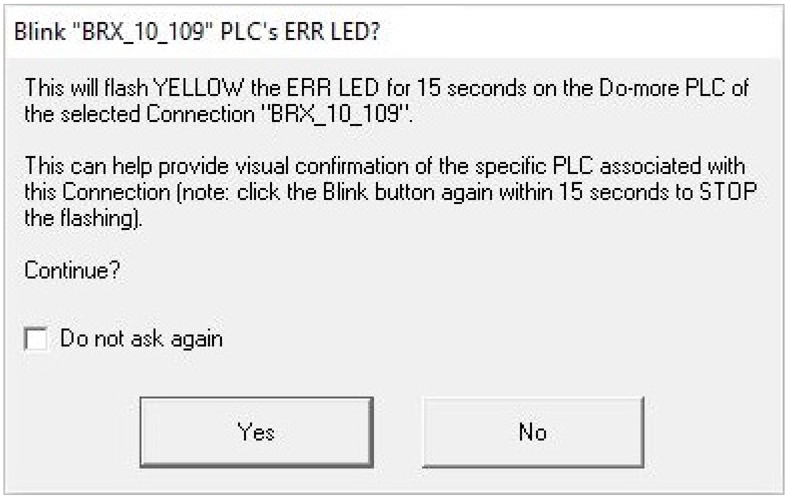
Optional Description: Description for My BX-DM1E

Connection Details:

PC Port: **Ethernet #1**
PLC Address: **IP Address: 10. 0. 1. 15**
PLC Type: **BX-DM1E-18ED13**

Ethernet Port Setup, continued

Blink and **Refresh** cause the selected PLC to blink its ERR led for 15 seconds. This is useful for visually confirming that the connection you have selected from the list is the actual PLC you think it is. Enabling the **Do Not Ask Again** option will bypass this confirmation dialog in the future. It will also reread the information that was read from the PLC when the Select PLC Connection dialog was first launched.



Blink "BRX_10_109" PLC's ERR LED?

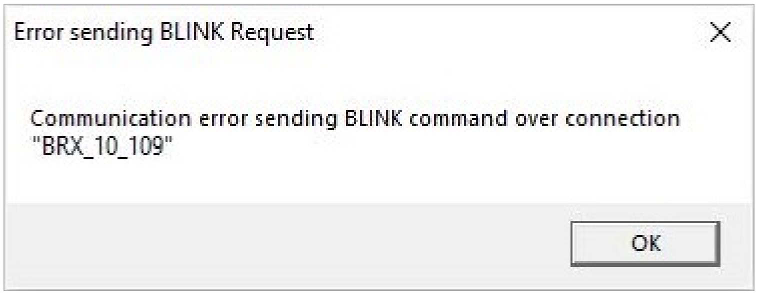
This will flash YELLOW the ERR LED for 15 seconds on the Do-more PLC of the selected Connection "BRX_10_109".

This can help provide visual confirmation of the specific PLC associated with this Connection (note: click the Blink button again within 15 seconds to STOP the flashing).

Continue?

□ Do not ask again

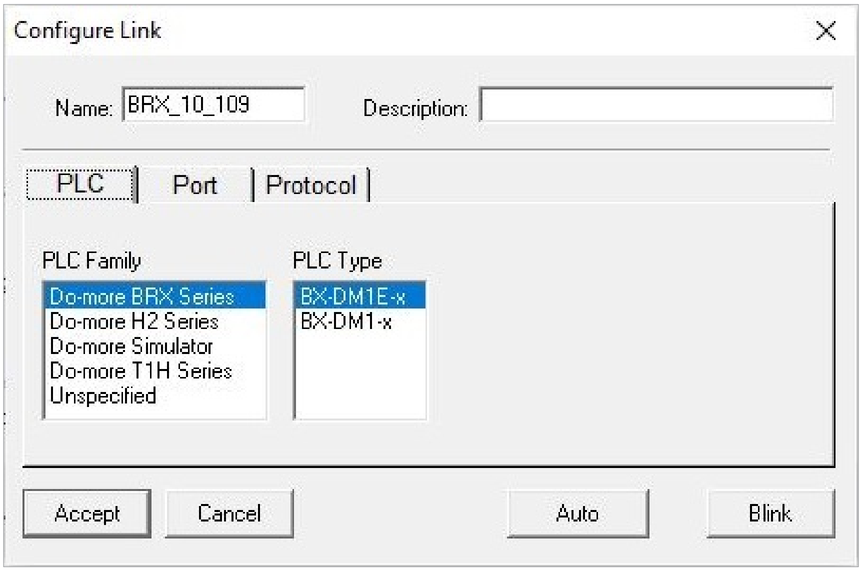
If the attempt to get the PLC to blink is not successful, you will see a message similar to the following and the Enabled state will be set to “Bad Comm”.



Error sending BLINK Request

Communication error sending BLINK command over connection "BRX_10_109"

Clicking the **Edit** button will open the highlighted connection in the ***Configure Link*** dialog where the communication parameters of the link used in the connection can be changed.



Configure Link

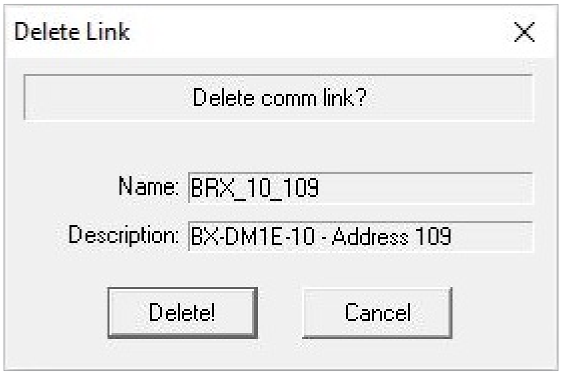
Name: BRX_10_109 Description:

PLC Family: Do-more BRX Series (selected), Do-more H2 Series, Do-more Simulator, Do-more T1H Series, Unspecified

PLC Type: BX-DM1E-x (selected), BX-DM1-x

Ethernet Port Setup, continued

Clicking the **Delete** button will remove the highlighted connection from the list.



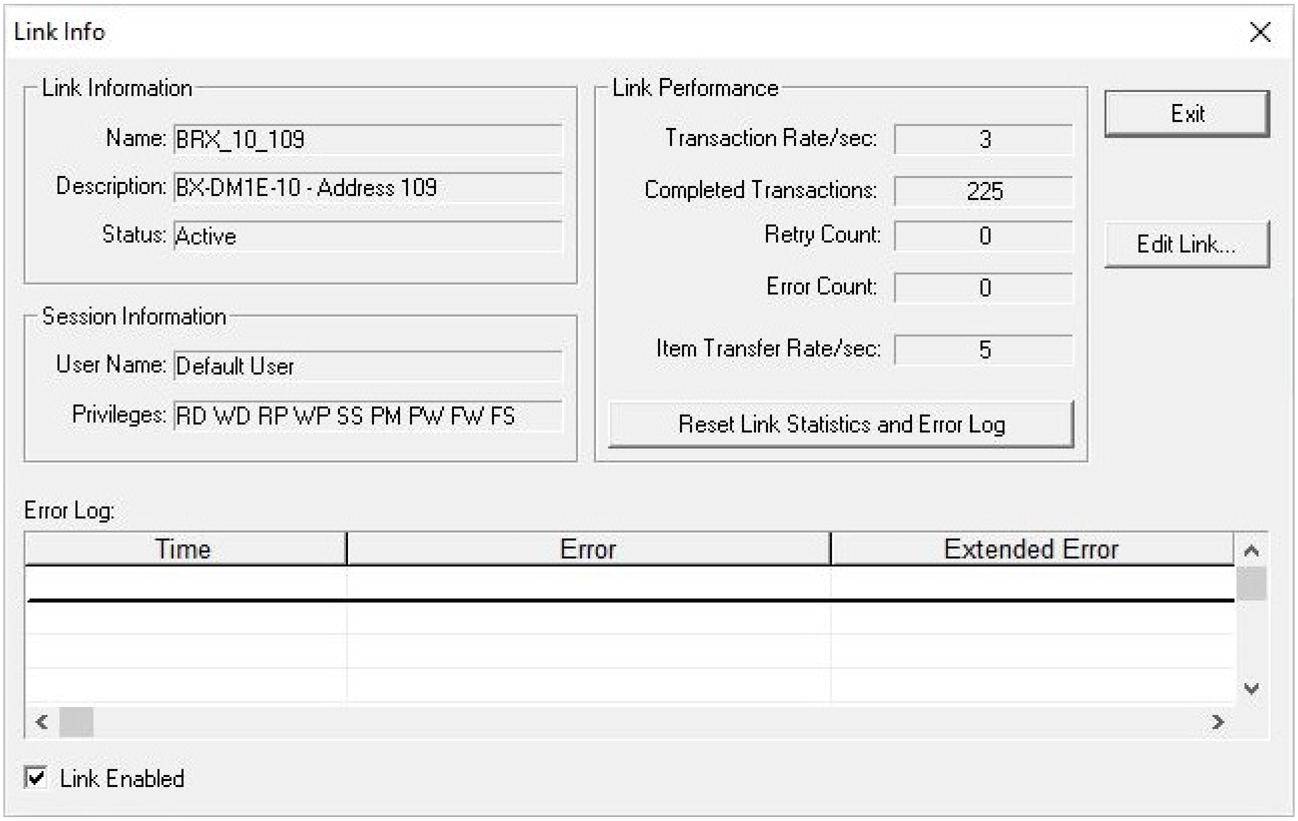
Delete Link

Delete comm link?

Name: BRX_10_109

Description: BX-DM1E-10 - Address 109

Clicking **Link Info** will open the Link Info dialog for the highlighted connection where the operational statistics of the connection can be seen, and any errors that have occurred while that connection has been in use are logged.



Link Info

Link Information

Name: BRX_10_109

Description: BX-DM1E-10 - Address 109

Status: Active

Session Information

User Name: Default User

Privileges: RD WD RP WP SS PM PW FW FS

Link Performance

Transaction Rate/sec: 3

Completed Transactions: 225

Retry Count: 0

Error Count: 0

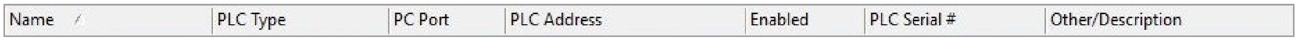
Item Transfer Rate/sec: 5

Error Log:

Time	Error	Extended Error

☑ Link Enabled

The *PLC Connections* list in the center of the dialog contains entries for all of the existing connections, and entries for any Do-more! compatible devices found on all of the Ethernet networks connected to the PC. The list can be sorted by any column in the table; click on the column header to sort by that column.



Name	PLC Type	PC Port	PLC Address	Enabled	PLC Serial #	Other/Description

The **Name** column will have the either the Connection Name for existing connections, or <new PLC> for PLCs that have detected on the Ethernet, Serial, and USB ports of the PC, but are not used in an existing connection.

Ethernet Port Setup, continued

The **PLC Type** column will contain the Full PLC Part Number if communication to this PLC was successful, or the base model number stored in the connection if PLC is offline.

The **PC Port** column shows which port on the PC the connection is using, either Ethernet #1, Ethernet #2, etc., (if the PLC is online), Ethernet (if the PLC is offline), COM1, COM2, etc. (if serial), or USB.

The **PLC Address** column contains the address of the PLC that is used in the connection:

> If the PC Port is Ethernet then the address is the user-assigned IP Address of the PLC. Note: if the IP Address is the factory-assigned 255.255.255.255.255, this means the PLC's IP Address is uninitialized. The IP Address configuration will have to be assigned by the user before this PLC can be used in a connection.
>
> If the PC Port is USB the address is the factory-assigned address of the USB port of the CPU.
>
> If the PC Port is a Serial port the address is simply the name of the serial port, e.g. COM1.

The **Enabled** column shows the state of the connection:

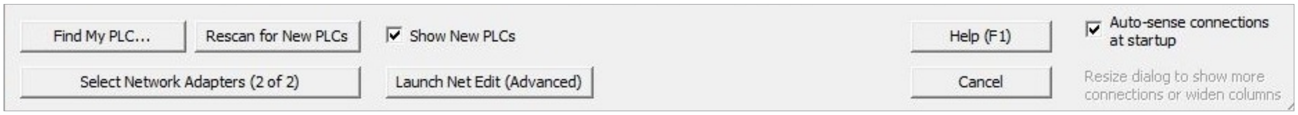
> **Enabled** means the PLC used in the connection is online.
>
> **Disabled** means the PLC used in the connection is offline.
>
> **Bad Comm** means an attempt to contact the PLC using the current settings in the connection failed.

The **PLC Serial Number** column shows the factory-configured serial number of the CPU:

> A 12 digit MAC address is shown if the connection is through an Ethernet port on the CPU, through an Ethernet POM (BX-P-ECOMLT), or through an ECOM100 module (H2-ECOM100).
>
> A 12 digit serial number is shown if the connection is through a USB port on the CPU, or a USB POM (BX-P-USB-B).
>
> <unknown > means the serial number couldn't be read using the current settings in the connection.

The **Other / Description** column shows the description field from the link that's used by the connection and any additional information about the connection status.

The selections along the bottom are used to help locate PLCs that should be in the center list but are currently not there.

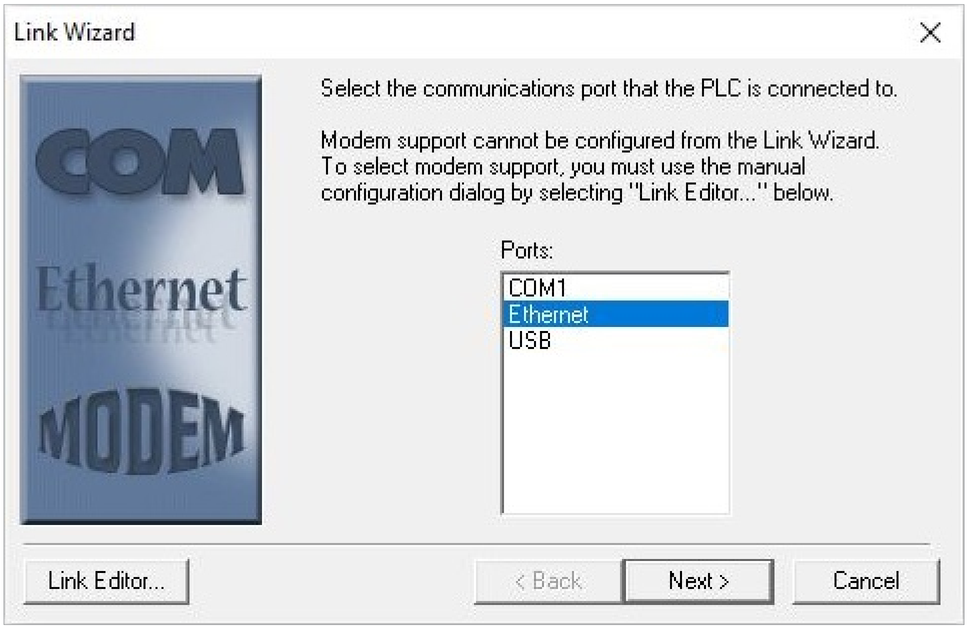

Find My PLC...
Rescan for New PLCs
Help (F1)
Select Network Adapters (2 of 2)
Launch Net Edit (Advanced)
Cancel
Resize dialog to show more connections or widen columns

☑ Show New PLCs

☑ Auto-sense connections at startup

Clicking the **Find My PLC...** button will open the initial dialog of the Link Wizard which will step you through the process of manually locating a new PLC on the Ethernet port, the serial port, or the USB port and creating a link for that PLC that can be used in a new connection.

Ethernet Port Setup, continued



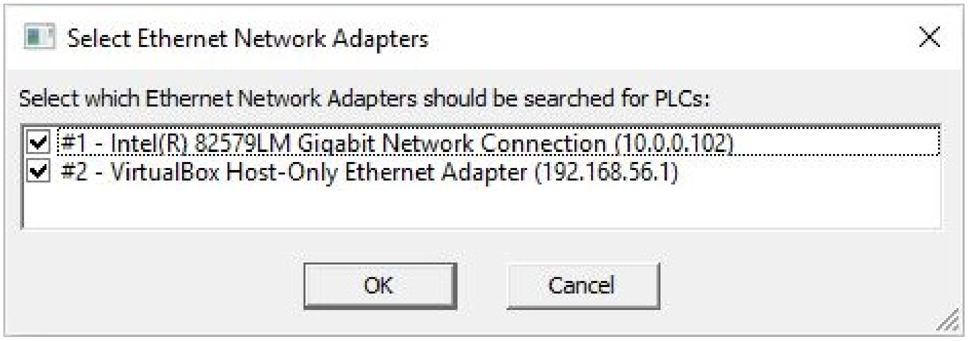
Select the communications port that the PLC is connected to.

Modem support cannot be configured from the Link Wizard. To select modem support, you must use the manual configuration dialog by selecting "Link Editor..." below.

Clicking the **Rescan For New PLCs** button will rerun the network scan that was run when you initially ran the Select PLC Connection utility.

If the **Show New PLCs** option is enabled, PLCs that are not used in an existing connection will be shown in the table as <new PLCs>. If this option is disabled, only PLCs that are already used in a connection will be shown.

Clicking the **Select Network Adapters** button will open a dialog where all of the network adapters (NICs) in the PC will be listed. By default, the Select PLC Connection dialog will search for accessible PLCs on all of the NICs in the PC. You can disable any (or all) of the network adapters in the list which will prevent them from being used in future network operations.



Select which Ethernet Network Adapters should be searched for PLCs:

☑ #1 - Intel(R) 82579LM Gigabit Network Connection (10.0.0.102)
☑ #2 - VirtualBox Host-Only Ethernet Adapter (192.168.56.1)

Clicking the **Launch NetEdit (Advanced)** button will open NetEdit as a separate utility.

Having the **Auto-sense Connections at Startup** option enabled will cause this utility to step through each of the existing connections in an attempt to communicate with the PLC defined in the connection. Doing this work when the utility is first opened is what fills in the information about the PLC’s state in the table. Stepping through all of the existing connection does take some time, so disabling this option will make the utility start up faster, but at the expense of having up-to-date state information for the connections.

Ethernet Protocols

The BRX Do-more! MPU has several Ethernet Protocol choices for communicating to external devices. In this section we will discuss each choice.

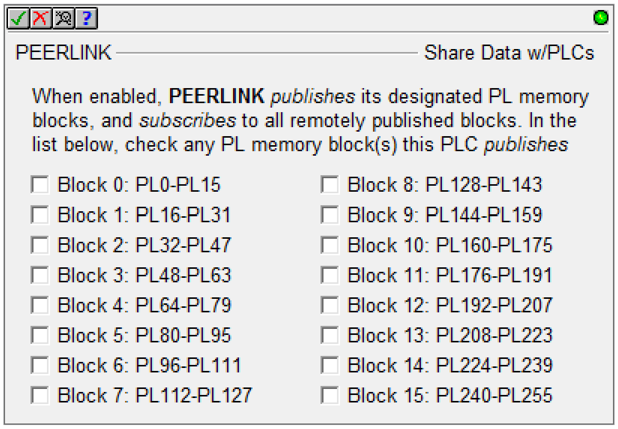
Supported Ethernet Communications Protocols

Protocol	Onboard	Ethernet POM
Peerlink	X	
Do-more Protocol (Slave)	X	X
Modbus TCP (Client)	X	
Modbus TCP (Server)	X	X
HOST Ethernet (ECOM) (Client)	X	
HOST Ethernet (ECOM) (Server)	X	X
EtherNet/IP (Client)	X	
EtherNet/IP (Server)	X	X
SMTP (Email)	X	
Ethernet Remote I/O (Master)	X	
HTTP	X	
MQTT	X	
SNTP (Time Server)	X	
TCP/IP Raw Packet	X	
UDP/IP Raw Packet	X	
DNS Lookup	X	
Ping	X	

PEERLINK Instruction

The PEERLINK instruction allows easy data sharing across any PLC that is running Do-more! technology.

To set this instruction up, place your data into the appropriate PL (PEERLINK) registers that you wish this PLC to share to other Do-more! PLCs. Then check the box that corresponds to that memory area.



PEERLINK — Share Data w/PLCs

When enabled, **PEERLINK** *publishes* its designated PL memory blocks, and *subscribes* to all remotely published blocks. In the list below, check any PL memory block(s) this PLC *publishes*

☐ Block 0: PL0-PL15
☐ Block 1: PL16-PL31
☐ Block 2: PL32-PL47
☐ Block 3: PL48-PL63
☐ Block 4: PL64-PL79
☐ Block 5: PL80-PL95
☐ Block 6: PL96-PL111
☐ Block 7: PL112-PL127
☐ Block 8: PL128-PL143
☐ Block 9: PL144-PL159
☐ Block 10: PL160-PL175
☐ Block 11: PL176-PL191
☐ Block 12: PL192-PL207
☐ Block 13: PL208-PL223
☐ Block 14: PL224-PL239
☐ Block 15: PL240-PL255

When this instruction is enabled, it will then automatically broadcast this information to all other Do-more! PLCs that are listening with a PEERLINK instruction.

To listen for broadcasts from other Do-more! PLCs, all you have to do is place the instruction in your ladder code. If this PLC is not sharing data, do not check any boxes. Incoming data will automatically be placed into the same register area that the broadcasting PLC has checked.

PEERLINK Instruction, continued

Using PUBLISH and SUBSCRIBE instructions can help to get data into and out of the PL memory area as the PL memory area is untyped data. See the Do-more! Designer help file for more information on PEERLINK, PUBLISH and SUBSCRIBE instructions.

***NOTE:** Keep in mind that multiple Do-more! PLCs cannot broadcast to the same memory area. This will cause an error.*

Do-more! Protocol

The Do-more! protocol is a proprietary protocol that is used exclusively by the Do-more! family of controllers. This is a very feature rich and secure protocol for communication with the Do-more! Designer software and between Do-more! controllers. It can also be used to communicate between multiple Do-more! controllers or between some brands of HMI's, such as C-more, to a Do-more! controller. Some SCADA systems such as Point of View also support the Do-more! Protocol.

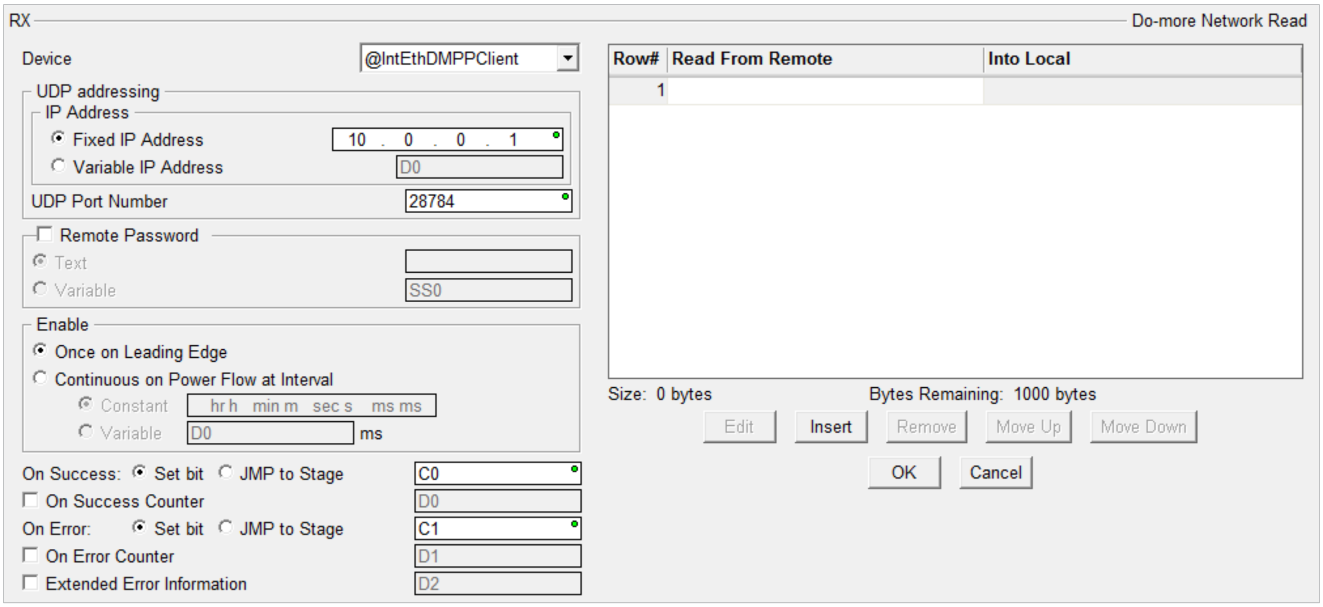
RX Instruction

The Do-more! Network Read (RX) instruction uses the Do-more! proprietary protocol to read incoming data on the on-board Ethernet port from another Ethernet-equipped Do-more! CPU. RX uses UDP (not TCP) protocol to communicate with the remote Do-more! CPU.

Each RX instruction can contain up to 50 individual read requests for a total of up to 1000 bytes of data. The RX instruction can read from all of the built-in memory blocks, all of the built-in structures, and any user-created memory blocks from the remote PLC. RX does NOT support reading a Heap Item from a remote Do-more! CPU.

In contrast to DirectLOGIC (DLRX / DLWX) and Modbus TCP (MRX / MWX) network communication which only has access to the protocol-specific memory blocks in the remote CPUs, Do-more! RX has direct access to nearly all the memory in the remote Do-more! CPU, including direct access to the CPU I/O memory.

The RX instruction establishes a session with the remote Do-more! CPU. The session is established using one of the User Accounts on the remote Do-more! CPU. It is through that User Account (System Security) that any access restrictions on what can be accessed can be enforced. By default it will use the Default User account (no password required).



RX — Do-more Network Read

Device: @IntEthDMPPClient

UDP addressing
IP Address
● Fixed IP Address: 10 . 0 . 0 . 1
○ Variable IP Address: D0
UDP Port Number: 28784

□ Remote Password
● Text
○ Variable: SS0

Enable
● Once on Leading Edge
○ Continuous on Power Flow at Interval
● Constant: hr h min m sec s ms ms
○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage: C0
□ On Success Counter: D0
On Error: ● Set bit ○ JMP to Stage: C1
□ On Error Counter: D1
□ Extended Error Information: D2

Row#	Read From Remote	Into Local
1		

Size: 0 bytes Bytes Remaining: 1000 bytes

Do-more! Protocol, continued

IP Address – The IP Address of the Do-more! CPU to read the data from. This can be either a Fixed (static) IP Address or a Variable (dynamic) value.

Fixed IP Address – The TCP Address assigned to the Do-more! CPU. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address – The IP Address resides in the specified memory location. This can be any readable DWord numeric location.

UDP Port Number – The port number of the Do-more! CPU to read the data from. The default value of 28784 (0x7070) is typically the correct number for Do-more! protocol. It is possible to use a different UDP Port Number if the Enable Secondary Ethernet Connection is turned on under the CPU Configuration settings. This secondary UDP Port Number defaults to 5000. This UDP Port Number can be any decimal value between 5000 and 65535, except for 28784 (the port number used by Do-more! Designer)..

Remote Password – The RX instruction requires that a communication session be established with the remote Do-more! CPU before the data read operations can be processed. Depending on how the remote system is configured, this may require the user to enter the password for the User Account to allow the session to be established. If no remote password is selected, the connection will be established using the Default User account.

Enable – Designates how this instruction is enabled. Select from one of the following:

Once on Leading Edge – Select this option to have this instruction run to completion exactly one time. Typically, this will take more than one controller scan. Configured this way the Do-more! Network Read (RX) instruction is Edge Triggered.

Continuous on Power Flow at Interval – Select this option to have this instruction run as long as the instruction has power flow. After the Do-more! Network Read (RX) has initially run, if the instruction still has power flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select how much time (in milliseconds) to wait between successive runs.

- **Constant** – specifies the interval time in Hours / Minutes / Seconds / Milliseconds.
- **Variable** – This can be any readable numeric location that contains a value between 0 and 2,147,483,647 which represents the number of milliseconds to wait before running again. A value of 0 ms means this instruction will be set to run on the next scan.

On Success – Selects which of the following actions to perform if the Network Read operation is successful:

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Success Counter – Enable this option and select a DWord location to store the total number of times the RX completed successfully. This can be any DWord location.

Do-more! Protocol, continued

On Error – Selects which of the following actions to perform if the Network Read operation is unsuccessful:

Set Bit – Enable this selection then specify writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error Counter – Enable this option and select a DWord location to store the total number of times the RX failed to complete. This can be any DWord location.

Extended Error Information – Enable this selection then enter a memory location to store any error codes returned for this instruction. This can be any writable numeric location. The list following is of the Extended error responses and their meaning:

-1 = Protocol Error occurred. The value in LastProtoError (DST38) contains the protocol error code as follows:

2 (0x02) = Out Of Sessions: the remote PLC currently has 32 concurrent connections and cannot accept any more.

3 (0x03) = Illegal Operation: the User Account for the password in the instruction does not have Write Data privilege.

4 (0x04) = Invalid Session: an error occurred with the session to the remote PLC, for example the remote PLC lost power during the session.

6 (0x06) = Invalid Argument: the write requests are not formed properly (you should never see this).

14 (0x0E) = Invalid Password: the password in the instruction does not match any User Account in the remote PLC,

20 (0x14) = Bad DMPP Request: one or more of the write requests cannot be processed, most likely the request is for a location that is out of range on the remote PLC or the memory block doesn’t exist on the remote PLC. The Extended Error location will contain the entry number of the write request that is causing the error.

0 = No Extended Error

1 – 50 = row number of the bad read request if the Last Protocol Error Code was 20 (0x14).

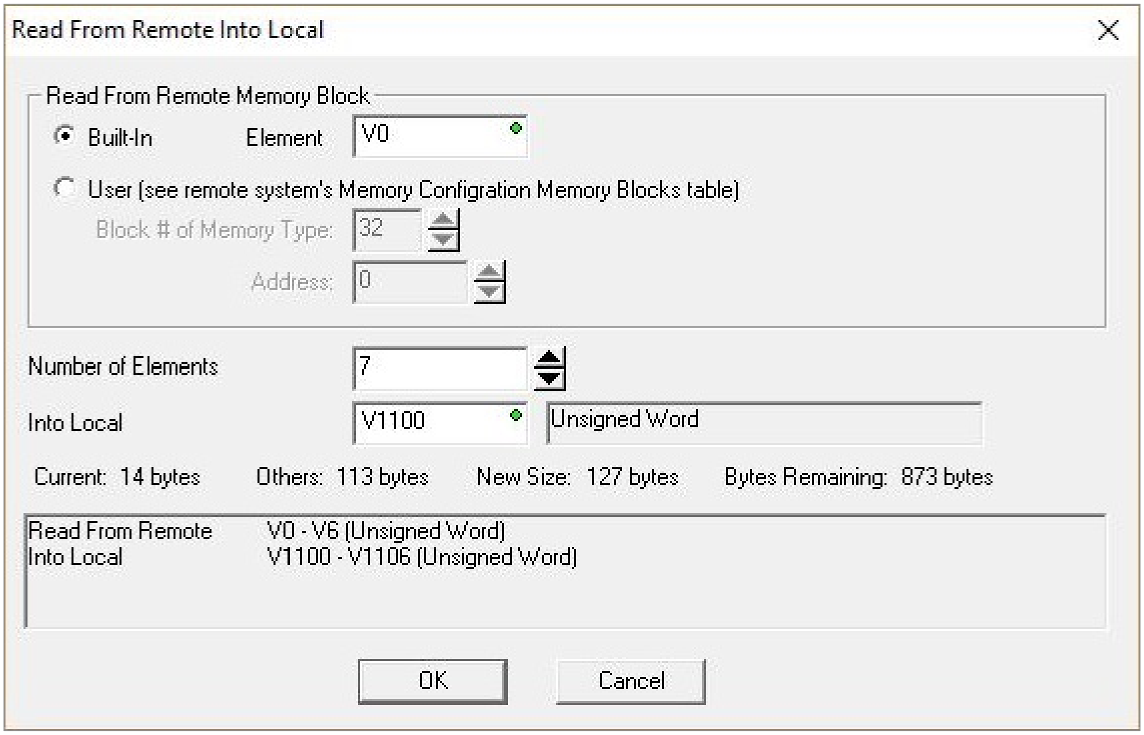
Insert – Select Insert to open Read From Remote Into Local setup dialog box (See page following).

Read From Remote Into Local

Be aware that because the local PLC does not have access to the memory configuration of the remote system, there is no guarantee that a read request that is correctly formed on the local PLC will work when the remote system tries to execute it. For example, you could request data from a memory location that is valid locally

Do-more! Protocol, continued

(D4000–D4095) but the range of D memory locations may have been adjusted in the remote PLC to stop at D3000.



Read From Remote Into Local

Read From Remote Memory Block

● Built-In Element: V0

○ User (see remote system's Memory Configration Memory Blocks table)

Block # of Memory Type: 32

Address: 0

Number of Elements: 7

Into Local: V1100 Unsigned Word

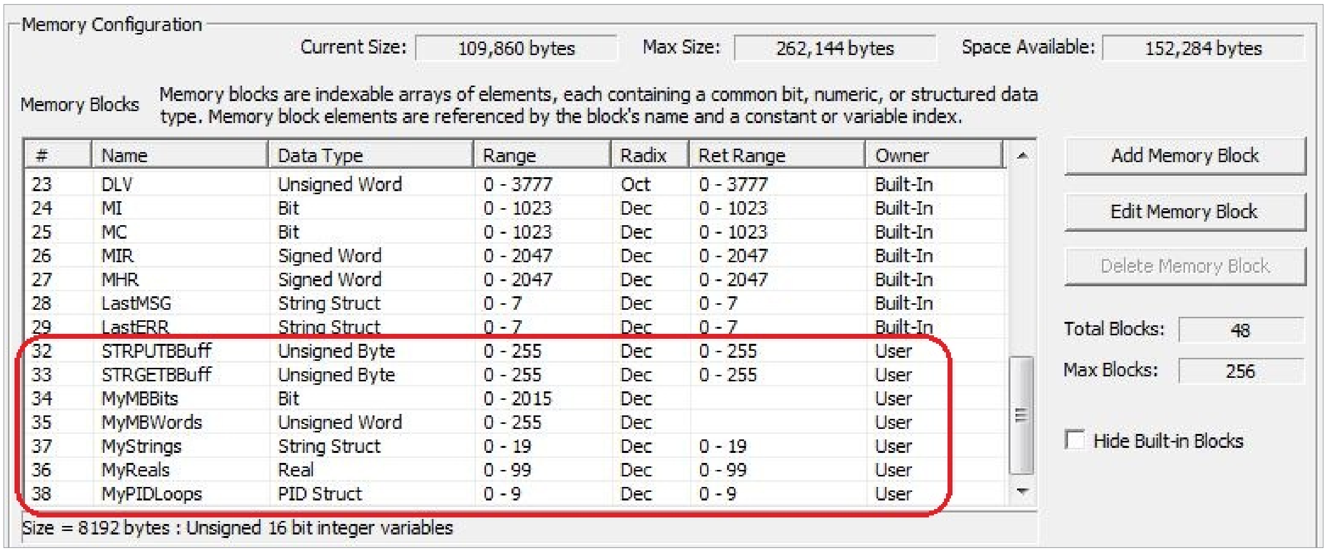
Current: 14 bytes Others: 113 bytes New Size: 127 bytes Bytes Remaining: 873 bytes

Read From Remote V0 - V6 (Unsigned Word)
Into Local V1100 - V1106 (Unsigned Word)

Read From Remote into Local selects the first location in the remote PLC to begin reading data.

Select **Built-In** to read from one of the built-in memory blocks in the remote PLC. You will need to enter the first element in the memory block in the Element field.

Select **User** to read from a memory block that was manually created (not built-in). You will need to enter the Block Number assigned to the user memory block in the remote PLC and the offset of the first element in the memory block in the Address field. When User Blocks are created they are assigned a number starting at 32. You can see the number that is assigned to a user memory block in the Memory Configuration page of the System Configuration as shown below:



Memory Configuration

Current Size: 109,860 bytes Max Size: 262,144 bytes Space Available: 152,284 bytes

Memory Blocks Memory blocks are indexable arrays of elements, each containing a common bit, numeric, or structured data type. Memory block elements are referenced by the block's name and a constant or variable index.

#	Name	Data Type	Range	Radix	Ret Range	Owner
23	DLV	Unsigned Word	0 - 3777	Oct	0 - 3777	Built-In
24	MI	Bit	0 - 1023	Dec	0 - 1023	Built-In
25	MC	Bit	0 - 1023	Dec	0 - 1023	Built-In
26	MIR	Signed Word	0 - 2047	Dec	0 - 2047	Built-In
27	MHR	Signed Word	0 - 2047	Dec	0 - 2047	Built-In
28	LastMSG	String Struct	0 - 7	Dec	0 - 7	Built-In
29	LastERR	String Struct	0 - 7	Dec	0 - 7	Built-In
32	STRPUTBBuff	Unsigned Byte	0 - 255	Dec	0 - 255	User
33	STRGETBBuff	Unsigned Byte	0 - 255	Dec	0 - 255	User
34	MyMBBits	Bit	0 - 2015	Dec		User
35	MyMBWords	Unsigned Word	0 - 255	Dec		User
37	MyStrings	String Struct	0 - 19	Dec	0 - 19	User
36	MyReals	Real	0 - 99	Dec	0 - 99	User
38	MyPIDLoops	PID Struct	0 - 9	Dec	0 - 9	User

Size = 8192 bytes : Unsigned 16 bit integer variables

Total Blocks: 48

Max Blocks: 256

□ Hide Built-in Blocks

Do-more! Protocol, continued

Number of Elements is the number of consecutive elements in the memory block to read.

Into Local selects the beginning location in the local PLC to store the data from the read operation. This location must be the same data type and have the same element size as the specified Read From Remote into Local data location. For example you can read from WY (Signed Word) into N (Signed Word) but you cannot read from WY (Signed Word) into V (Unsigned Word).

The remaining fields are to help you maintain the size limitations of the instruction. These fields validate immediately as you make changes in the editor. Current is the number of bytes in the read request currently being edited, Others is the number of bytes in the read requests already in this instruction, **New Size** is the total number of bytes of all the read requests in this instruction (including the one currently being edited), and **Bytes Remaining** is the number of bytes of the 1000 byte limit remaining for read requests in this instruction.

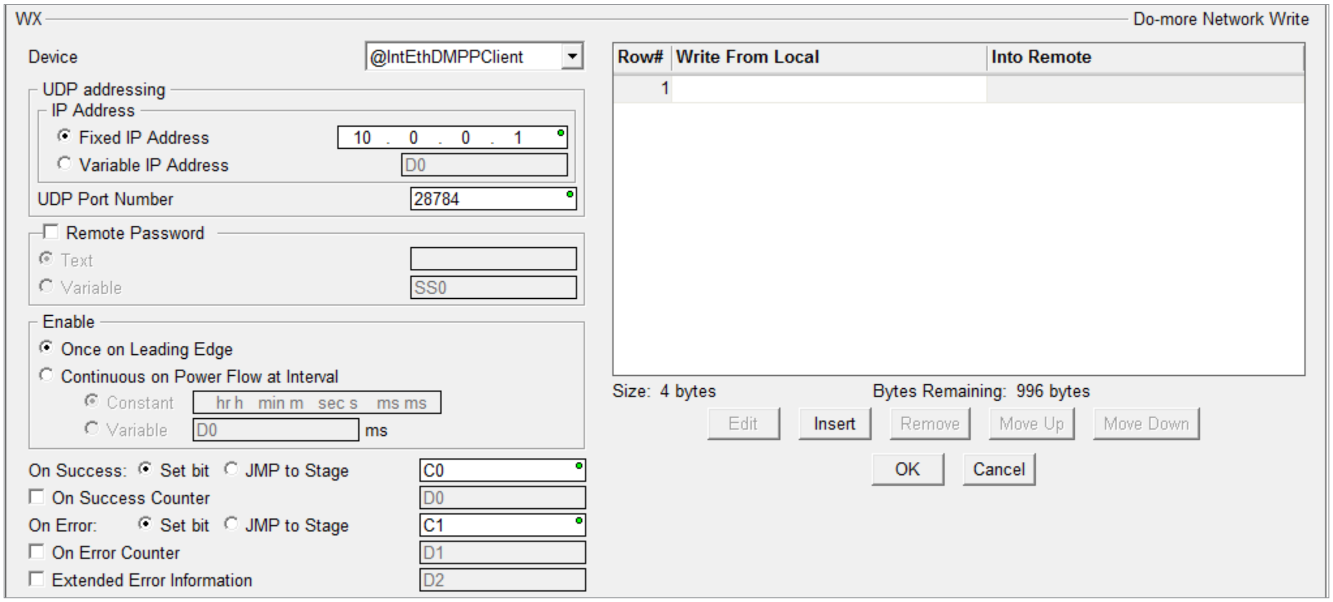
WX Instruction

The Do-more! Network Write (WX) instruction uses the Do-more! proprietary protocol to write data over the on-board Ethernet port to another Ethernet-equipped Do-more! CPU. WX uses UDP (not TCP) protocol to communicate with the remote Do-more! CPU.

Each WX instruction can contain up to 50 individual write requests for a total of up to 1000 bytes of data. The WX instruction can write to all of the built-in memory blocks, all of the built-in structures, and any user-created memory blocks in the remote PLC. WX does NOT support writing to a Heap Item in the remote Do-more! CPU.

In contrast to DirectLOGIC (DLRX / DLWX) and Modbus/TCP (MRX / MWX) network communication which only has access to the protocol-specific memory blocks in the remote CPUs, Do-more! WX has direct access to nearly all of the memory in the remote Do-more! CPU – including direct access to the PLC's I/O memory.

The WX instruction establishes a session with the remote Do-more! CPU. The session is established using one of the User Accounts on the remote Do-more! CPU. It is through that User Account (System Security) that any access restrictions on what can be accessed can be enforced. By default it will use the Default User account (no password required).



WX — Do-more Network Write

Device: @IntEthDMPPClient

UDP addressing
IP Address
● Fixed IP Address: 10 . 0 . 0 . 1
○ Variable IP Address: D0
UDP Port Number: 28784

□ Remote Password
● Text
○ Variable: SS0

Enable
● Once on Leading Edge
○ Continuous on Power Flow at Interval
● Constant: hr h min m sec s ms ms
○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage: C0
□ On Success Counter: D0
On Error: ● Set bit ○ JMP to Stage: C1
□ On Error Counter: D1
□ Extended Error Information: D2

Row#	Write From Local	Into Remote
1		

Size: 4 bytes
Bytes Remaining: 996 bytes

Do-more! Protocol, continued

IP Address – The IP Address of the Do-more! CPU to write the data to. This can be either a Fixed (static) IP Address or a Variable (dynamic) value.

Fixed IP Address – The TCP Address assigned to the Do-more! CPU. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address – The IP Address resides in the specified memory location. This can be any readable DWord numeric location.

UDP Port Number – The port number of the Do-more! CPU to write the data to. The default value of 28784 (0x7070) is typically the correct number for Do-more! protocol. It is possible to use a different UDP Port Number if the Enable Secondary Ethernet Connection is turned on under the CPU Configuration settings. This secondary UDP Port Number defaults to 5000. This UDP Port Number can be any decimal value between 5000 and 65535, except for 28784 (the port number used by Do-more! Designer).

Remote Password – The WX instruction requires that a communication session be established with the remote Do-more! CPU before the data write operations can be processed. Depending on how the remote system is configured, this may require the user to enter the password for the User Account to allow the session to be established. If no remote password is selected, the connection will be established using the Default User account.

Enable – Designates how this instruction is enabled. Select from one of the following:

Once on Leading Edge – Select this option to have this instruction run to completion exactly one time. Typically, this will take more than one controller scan. Configured this way the Do-more! Network Write (WX) instruction is Edge Triggered.

Continuous on Power Flow at Interval – select this option to have this instruction run as long as the instruction has power flow. After the Do-more! Network Write (WX) has initially run, if the instruction still has power flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select the time (in milliseconds) delay between successive runs.

Constant - specifies the interval time in Hours / Minutes / Seconds / Milliseconds.

Variable - This can be any readable numeric location that contains a value between 0 and 2,147,483,647 which represents the number of milliseconds to wait before running again. A value of 0 ms means this instruction will be set to run on the next scan.

On Success – Selects which of the following actions to perform if the Network Write operation is successful:

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Success Counter – Enable this option and select a DWord location to store the total number of times the WX completed successfully. This can be any DWord location.

On Error – Selects which of the following actions to perform if the Network Write operation is unsuccessful:

Set Bit – Enable this selection then specify writable bit location.

Do-more! Protocol, continued

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error Counter – Enable this option and select a DWord location to store the total number of times the RX failed to complete. This can be and DWord location.

Extended Error Information – Enable this selection then enter a memory location to store any error codes returned for this instruction. This can be any writable numeric location. The following lists the Extended error responses and their meaning:

-1 = Protocol Error occurred. The value in LastProtoError (DST38) contains the protocol error code as follows:

2 (0x02) = Out Of Sessions: the remote PLC currently has 32 concurrent connections and cannot accept any more.

3 (0x03) = Illegal Operation: the User Account for the password in the instruction does not have Write Data privilege.

4 (0x04) = Invalid Session: an error occurred with the session to the remote PLC, for example the remote PLC lost power during the session.

6 (0x06) = Invalid Argument: the write requests are not formed properly (you should never see this).

14 (0x0E) = Invalid Password: the password in the instruction does not match any User Account in the remote PLC.

20 (0x14) = Bad DMPP Request: one or more of the write requests cannot be processed, most likely the request is for a location that is out of range on the remote PLC or the memory block doesn't exist on the remote PLC. The Extended Error location will contain the entry number of the write request that is causing the error.

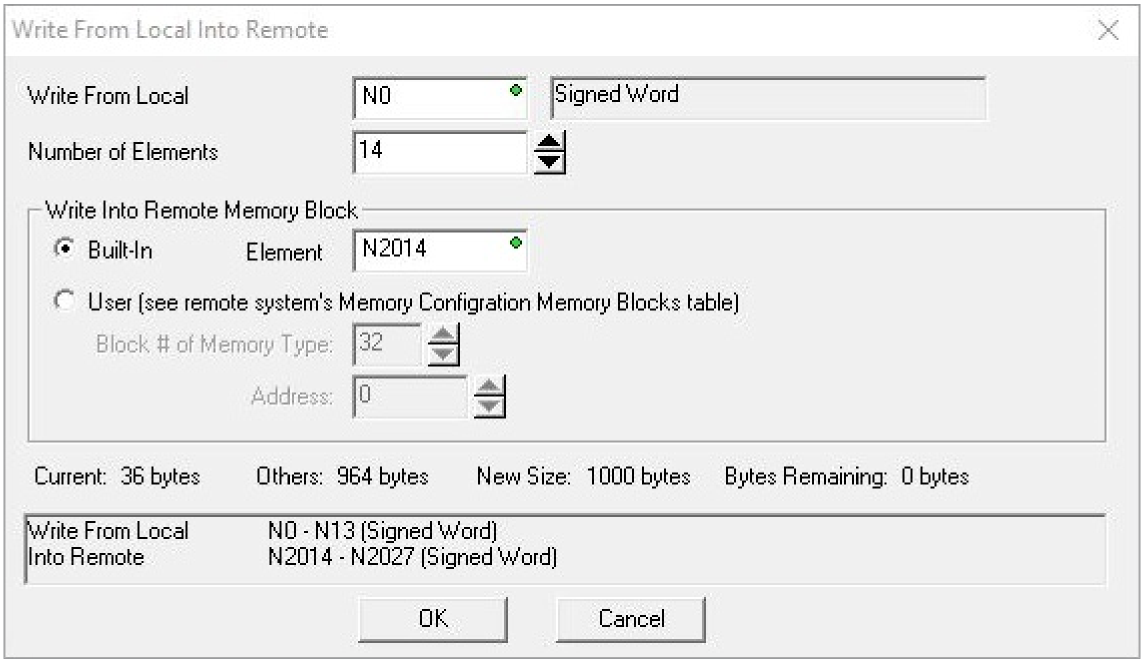
0 = No Extended Error

1 – 50 = Row number of the bad read request if the Last Protocol Error Code was 20 (0x14).

Insert – Select Insert to open Write From Local Into Remote dialog box.

Write From Local Into Remote

Write From Local into Remote selects the first location of the source data in the Local PLC to write to the remote PLC.



Write From Local Into Remote

Write From Local: N0 Signed Word

Number of Elements: 14

Write Into Remote Memory Block

● Built-In Element: N2014

○ User (see remote system's Memory Configration Memory Blocks table)

Block # of Memory Type: 32

Address: 0

Current: 36 bytes Others: 964 bytes New Size: 1000 bytes Bytes Remaining: 0 bytes

Write From Local N0 - N13 (Signed Word)
Into Remote N2014 - N2027 (Signed Word)

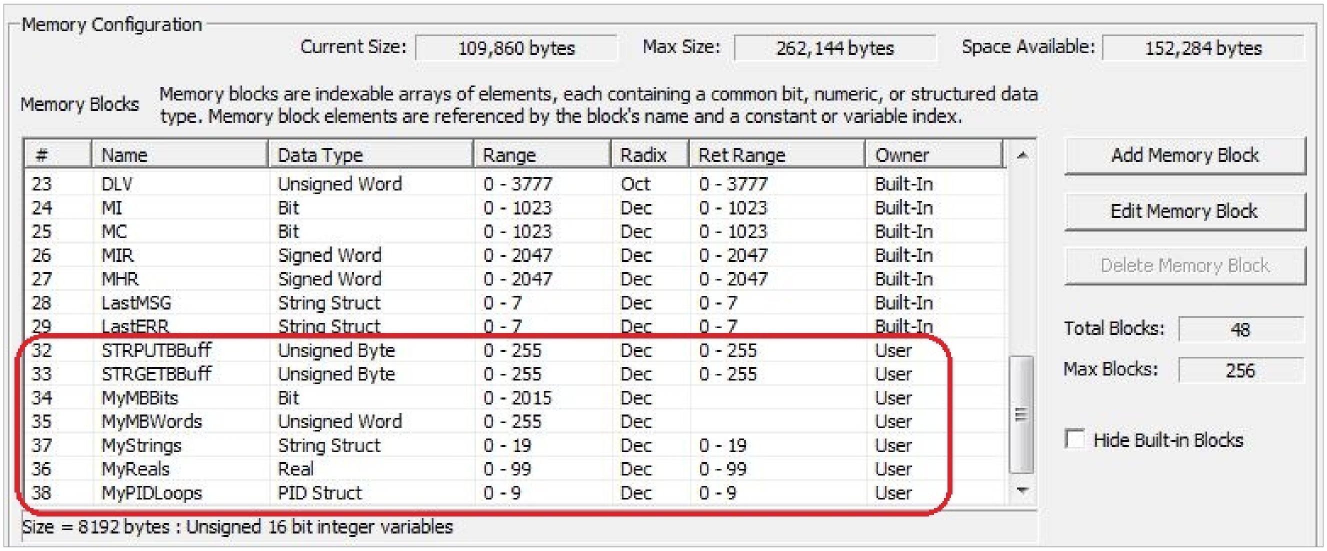
Do-more! Protocol, continued

Number of Elements is the number of consecutive elements in the memory block to write.

Write Into Remote Memory Block selects the beginning location in the remote PLC to store the data from the write operation. This location must be the same data type and have the same element size as the specified Write From Local data location. For example you can write from WY (Signed Word) into N (Signed Word) but you cannot write from WY (Signed Word) into V (Unsigned Word).

Select **Built-In** to write into one of the built-in memory blocks in the remote PLC. You will need to enter the first element in the memory block in the Element field.

Select **User** to write into a memory block that was manually created (not built-in). You will need to enter the Block Number assigned to the user memory block in the remote PLC and the offset of the first element in the memory block in the Address field. When User Blocks are created they are assigned a number starting at 32. You can see the number that is assigned to a user memory block in the Memory Configuration page of the System Configuration as shown below:



Memory Configuration

Current Size: 109,860 bytes Max Size: 262,144 bytes Space Available: 152,284 bytes

Memory Blocks: Memory blocks are indexable arrays of elements, each containing a common bit, numeric, or structured data type. Memory block elements are referenced by the block's name and a constant or variable index.

#	Name	Data Type	Range	Radix	Ret Range	Owner
23	DLV	Unsigned Word	0 - 3777	Oct	0 - 3777	Built-In
24	MI	Bit	0 - 1023	Dec	0 - 1023	Built-In
25	MC	Bit	0 - 1023	Dec	0 - 1023	Built-In
26	MIR	Signed Word	0 - 2047	Dec	0 - 2047	Built-In
27	MHR	Signed Word	0 - 2047	Dec	0 - 2047	Built-In
28	LastMSG	String Struct	0 - 7	Dec	0 - 7	Built-In
29	LastERR	String Struct	0 - 7	Dec	0 - 7	Built-In
32	STRPUTBBuff	Unsigned Byte	0 - 255	Dec	0 - 255	User
33	STRGETBBuff	Unsigned Byte	0 - 255	Dec	0 - 255	User
34	MyMBBits	Bit	0 - 2015	Dec		User
35	MyMBWords	Unsigned Word	0 - 255	Dec		User
37	MyStrings	String Struct	0 - 19	Dec	0 - 19	User
36	MyReals	Real	0 - 99	Dec	0 - 99	User
38	MyPIDLoops	PID Struct	0 - 9	Dec	0 - 9	User

Size = 8192 bytes : Unsigned 16 bit integer variables

Total Blocks: 48

Max Blocks: 256

The remaining fields are to help you maintain the size limitations of the instruction. These fields validate immediately as you make changes in the editor. **Current** is the number of bytes in the write request currently being edited, **Others** is the number of bytes in the write requests already in this instruction, **New Size** is the total number of bytes of all the write requests in this instruction (including the one currently being edited), and **Bytes Remaining** is the number of bytes of the 1000 byte limit remaining for write requests in this instruction.

Modbus TCP

Modbus TCP is a protocol overseen by Modbus.org. This standard is an open standard meaning that anyone can utilize it freely.

Modbus TCP can be utilized as either a client or server configuration. It supports Clients and Servers in a Peer to Peer fashion.

Modbus TCP Server (Slave)

As a Modbus TCP Server, the BRX Do-more! MPU is functioning as a listening/replying device. The external Client device will request data registers from the BRX Do-more! and the BRX Do-more! will reply with the appropriate data.

All Modbus Client data is stored in four sets of registers in the BRX Do-more!. This memory area is blocked off specifically for Modbus communications. You must place data in these registers in order for a Modbus Client device to be able to access it.

Modbus Data Registers

Register Type	Register Name	Range
Coil	MC	00000–01023
Discrete Input	MI	10000–11023
Holding Register	MHR	40000–42047
Input Register	MIR	30000–32047

The Modbus data area is loosely data typed and casting or other instructions such as PUBLISH and SUBSCRIBE can be utilized to convert data in this area to the proper data type needed as well. Please see the help file for more information on casting, PUBLISH and SUBSCRIBE.

***NOTE:** Ranges can be expanded in the Memory Configuration section of the Do-more! Designer software as needed.*

The Modbus TCP Server (Slave) supports the following function codes:

Modbus

Function Code	Description
1	Read coil
2	Read discrete inputs
3	Read holding registers
4	Read input registers
5	Write single coil
6	Write single registers
7	Read exception status
15	Write multiple coils
16	Write multiple registers
22	Mask write registers

Modbus TCP, continued

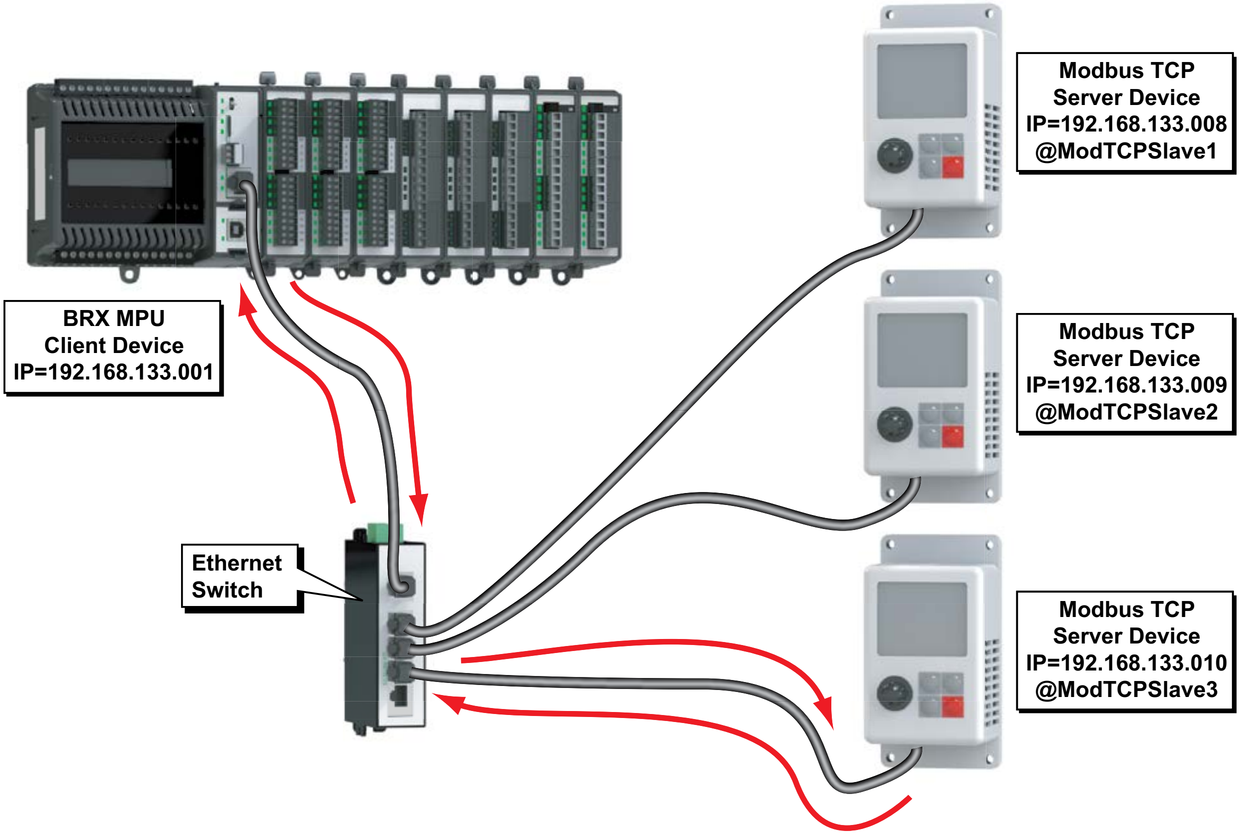
Modbus TCP Client (Master)

As a Modbus TCP Client, the BRX Do-more! MPU is requesting data from a Modbus TCP Server device.

In order for this to work, you need to know quite a few things about your Server device such as the function codes that it supports, the data registers that are accessible and possibly the Unit ID or Slave Address.

In order to utilize the BRX Do-more! as a Modbus TCP Client you will need to create a Modbus TCP Client Device. This device will handle all of the communications with the external Modbus TCP servers by using the Ethernet port on the front of the BRX MPU.

***NOTE:** The default @IntMODTCPClient device is created automatically. You can choose to use this device, however we strongly recommend that you make a new Device for each Modbus TCP server that you will be communicating with. This will allow simultaneous communications to flow uninterrupted if one of the Modbus TCP servers goes offline.*



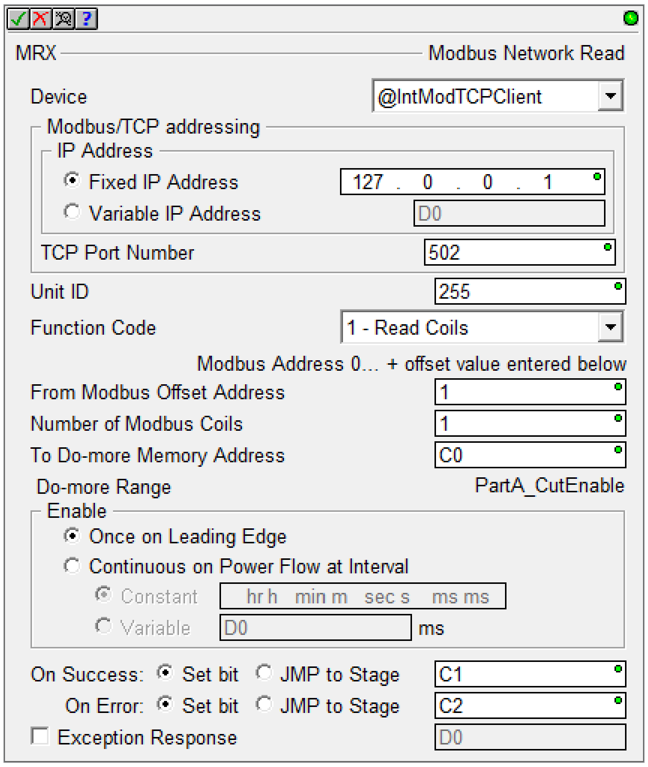
As a Modbus TCP Client, you do not have to sequence the MRX read and MRX write instructions. If you are so inclined, you can just drop them into your program and they will work in a round robin manner. However, sequencing the instructions will give you better control and allow you to build complex communication patterns to optimally communicate with your devices.

The communications instructions in Do-more! Designer have Success and Error built into the instructions so that you can either set a bit or move to a Stage if you are doing state style programming. There are examples of using Stage (state) programming in the software help file to show how you could use this to implement a complex communications routine.

Modbus TCP, continued

MRX Instruction

The MRX instruction is used to read from a Modbus TCP Server. Following is a brief description of the MRX instruction parameters. For more detailed information please refer to the Do-more! Designer help files.



MRX — Modbus Network Read

Device: @IntModTCPClient

Modbus/TCP addressing

IP Address

● Fixed IP Address: 127 . 0 . 0 . 1

○ Variable IP Address: D0

TCP Port Number: 502

Unit ID: 255

Function Code: 1 - Read Coils

Modbus Address 0... + offset value entered below

From Modbus Offset Address: 1

Number of Modbus Coils: 1

To Do-more Memory Address: C0

Do-more Range: PartA_CutEnable

Enable

● Once on Leading Edge

○ Continuous on Power Flow at Interval

● Constant: hr h min m sec s ms ms

○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage: C1

On Error: ● Set bit ○ JMP to Stage: C2

□ Exception Response: D0

Device – The device associated with the physical port that you want to communicate from. @IntModTCPClient is the name of the device associated with the built in Ethernet port when set as a Modbus TCP Client. We strongly recommend that you make a new Device for each Modbus TCP server that you will be communicating with. This will allow simultaneous communications to flow uninterrupted if one of the Modbus TCP servers goes offline.

IP Address – The IP Address of the Modbus TCP Server to read the data from. This can be either a Fixed (static) IP Address or a Variable (dynamic) value.

Fixed IP Address – The TCP Address assigned to the Modbus TCP Server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address – The IP Address resides in the specified memory location. This can be any readable DWord numeric location.

TCP Port Number – This should normally be set to 502 unless the end device has had the default port number changed.

Unit ID – The ID number of the Server device. Typically this is 255 unless you are talking to a Modbus Serial Gateway style of device.

Modbus TCP, continued

Function Code – selects which of the following Modbus function codes to use:

1 - Read Coils

2 - Read Discrete Inputs

3 - Read Holding Registers

4 - Read Input Registers

7 - Read Exception Status

From Modbus Offset Address – The address in the Modbus Server that you will be reading from. This address may be offset by a value of +1 depending on how the manufacturer followed the Modbus standard.

Number of Modbus Coils/ Registers – Based on the Function Code selected, this selection specifies how many consecutive elements to read.

To Do-more Memory Address – Specifies the beginning address of a range of bits or numeric locations in the CPU where the data that is read will be stored. This data type (bit or register) must match the type expected by the Function Code.

Do-more Range – This is the ending register where the data will be stored at, calculated by taking the To Do-more Memory address and adding the Number of Modbus Coils/Registers value to it.

Enable – Designates how this instruction will operate. Select from one of the following:

Once on Leading Edge - Select this option to have this instruction run to completion exactly one time. Typically, this instruction will take more than one controller scan to complete. Configured this way the instruction is Edge Triggered.

Continuous on Power Flow at Interval - Select this option to have this instruction run as long as the instruction has power flow. After the instruction has initially run, if the instruction still has power flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select how much time (in milliseconds) to wait between successive runs. A value of 0ms means the instruction will re-run immediately.

On Success – When the instruction completes successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error – When the instruction does not complete successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

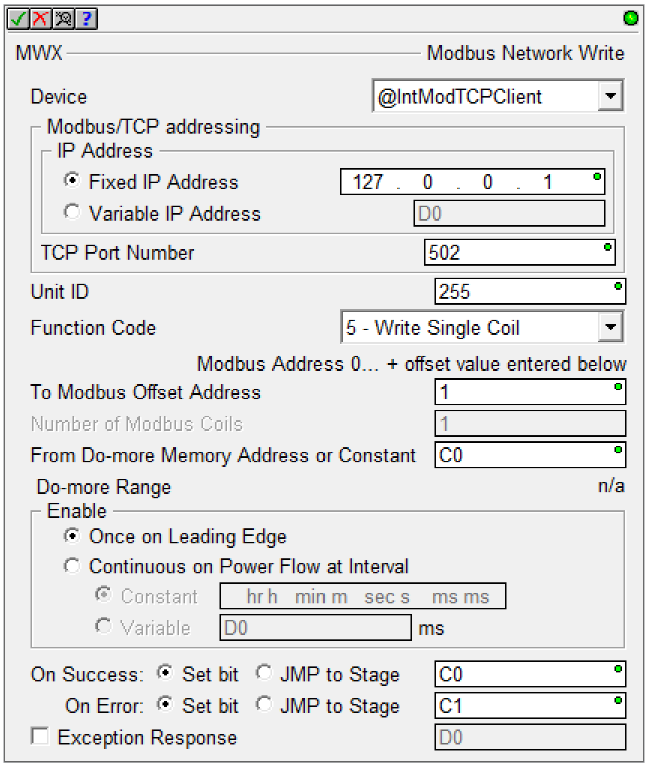
JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Exception Response – For errors where the message was received properly by the Server device, this will contain a value to indicate why the Server rejected the message. This can aid in troubleshooting the issue with the message.

Modbus TCP, continued

MWX Instruction

The MWX instruction is used to write to a Modbus TCP Server (Slave). For specific information please refer to the Do-more! Designer help files.



MWX — Modbus Network Write

Device: @IntModTCPClient

Modbus/TCP addressing

IP Address

● Fixed IP Address: 127 . 0 . 0 . 1

○ Variable IP Address: D0

TCP Port Number: 502

Unit ID: 255

Function Code: 5 - Write Single Coil

Modbus Address 0... + offset value entered below

To Modbus Offset Address: 1

Number of Modbus Coils: 1

From Do-more Memory Address or Constant: C0

Do-more Range: n/a

Enable

● Once on Leading Edge

○ Continuous on Power Flow at Interval

● Constant: hr h min m sec s ms ms

○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage: C0

On Error: ● Set bit ○ JMP to Stage: C1

□ Exception Response: D0

Device – The device associated with the physical port that you want to communicate from. @IntModTCPClient is the name of the device associated with the built in Ethernet port when set as a Modbus TCP Client. We strongly recommend that you make a new Device for each Modbus TCP Server that you will be communicating with. This will allow simultaneous communications to flow uninterrupted if one of the Modbus TCP servers goes offline.

IP Address – The IP Address of the Modbus TCP Server to read the data from. This can be either a Fixed (static) IP Address or a Variable (dynamic) value.

Fixed IP Address – The TCP Address assigned to the Modbus TCP Server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address – The IP Address resides in the specified memory location. This can be any readable DWord numeric location.

TCP Port Number – This should normally be set to 502 unless the end device has had the default port number changed.

Unit ID – The ID number of the Modbus TCP Server device. Typically this is 255 unless you are talking to a Modbus Serial Gateway style of device.

Modbus TCP, continued

Function Code – Selects which of the following Modbus function codes to use:

5 - Write Single Coil

6 - Write Single Register

15 - Write Multiple Coils

16 - Write Multiple Registers

To Modbus Offset Address – The starting register that you will be writing to. This address may be offset by a value of +1 depending on how the manufacturer followed the Modbus standard.

Number of Modbus Coils/ Registers – This selection specifies how many consecutive elements to write from the Modbus Offset Address.

From Do-more Memory Address – Specifies the beginning address of a range of bits or numeric locations in the CPU where the data that will be written from. This data type (bit or register) must match the type expected by the Function Code.

Do-more Range – This is the ending register where the data will be written from, calculated by taking the To Do-more Memory address and adding the Number of Modbus Coils/Registers value to it.

Enable – Designates how this instruction will operate. Select from one of the following:

Once on Leading Edge – Select this option to have this instruction run to completion exactly one time. Typically, this instruction will take more than one controller scan to complete. Configured this way the instruction is Edge Triggered.

Continuous on Power Flow at Interval – Select this option to have this instruction run as long as the instruction has power flow. After the instruction has initially run, if the instruction still has Power Flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select how much time (milliseconds) to wait between successive runs. A value of 0ms means the instruction will re-run immediately.

On Success – When the instruction completes successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error – When the instruction does not complete successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Exception Response – For errors where the message was received properly by the Server device, this will contain a value to indicate why the Server rejected the message. This can aid in troubleshooting the issue with the message.

HOST Ethernet Protocol (Client, Server)

The BRX Do-more! MPU can serve as a HOST Ethernet Protocol Client and Server to communicate to legacy devices that utilize the HOST Ethernet protocol such as DirectLogic PLCs, C-more HMI, SCADA systems, etc.

All data is stored in four sets of registers in the BRX Do-more!. This memory area is blocked off specifically for HOST Ethernet Protocol communications. You must place data in these registers so that a HOST Ethernet Protocol Client device will be able to access it.

HOST Data Registers

DirectLogic Type	Do-more! Block Name	Default Octal Range
Discrete Input (X)	DLX	0–777
Discrete Output (Y)	DLY	0–777
Control Relay (C)	DLC	0–777
16-Bit Data (V)	DLV	0–3777

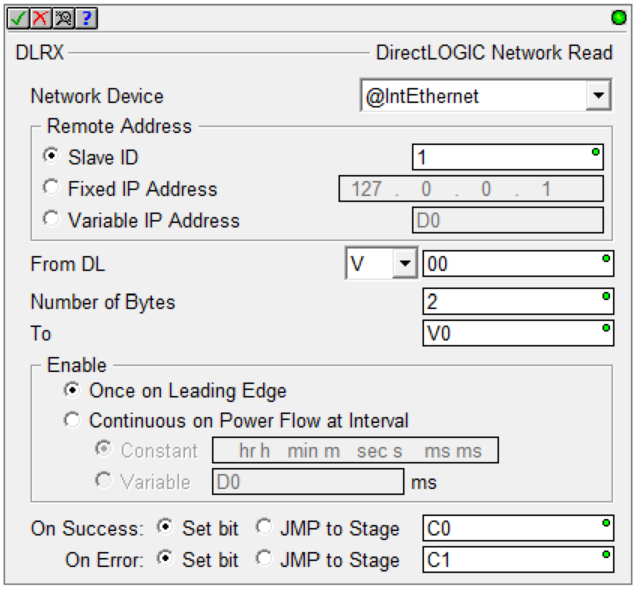
The HOST Ethernet Protocol data area is loosely data typed and casting or other instructions such as PUBLISH and SUBSCRIBE can be utilized to convert data in this area to the proper data type needed as well. Please see the help file for more information on casting, PUBLISH and SUBSCRIBE.

NOTE: *Ranges can be expanded in the Memory Configuration section of the Do-more! Designer software as needed.*

DirectLogic Client (Master)

The BRX Do-more! MPU can be a HOST Ethernet Protocol Client to communicate to legacy devices that utilize the HOST Ethernet protocol such as DirectLogic PLCs.

__DLRX__



DLRX DirectLOGIC Network Read

Network Device: @IntEthernet

Remote Address
- ● Slave ID: 1
- ○ Fixed IP Address: 127 . 0 . 0 . 1
- ○ Variable IP Address: D0

From DL: V 00

Number of Bytes: 2

To: V0

Enable
- ● Once on Leading Edge
- ○ Continuous on Power Flow at Interval
  - ● Constant: hr h min m sec s ms ms
  - ○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage: C0

On Error: ● Set bit ○ JMP to Stage: C1

HOST Ethernet Protocol (Client, Server), continued

Network Device – The device associated with the physical port that you want to communicate from. @IntEthernet is the name of the device associated with the built in Ethernet port.

Remote Address – specifies which of the following addressing modes to use:

Slave ID - Selects the Slave ID (Module ID) of the remote DirectLOGIC server. The Slave ID can be any constant value in the range of 1 to 90, or any readable numeric location that contains a value in that range. If Slave ID is selected, a TCP/IP broadcast is used to perform the network read operation. This means that both the Do-more! controller and the remote ECOM module must be in the same Broadcast Domain.

Fixed IP Address – Specifies the IP Address assigned to the remote server ECOM module. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address – The IP Address resides in a memory location in the PLC, allowing the IP Address to be changed at runtime. This can be any readable DWord numeric location. Each octet of the IP Address is stored in one byte of the Variable Address location.

From DL - Designates the data type and the address of the data to read from the DirectLOGIC controller.

V-Memory – Locations in a DirectLOGIC controller are unsigned 16-bit values. Each V-Memory location is 2 bytes in length, so reading V-Memory requires the length be in 2-byte increments. The memory address value must begin on a byte boundary. Single Bit locations in the DirectLOGIC controllers cannot be read individually, you must read the byte that contains the desired bit.

Number of Bytes – Designates the number of elements of the selected type to read.
X, Y, C, S, T, CT, GX, GY, SP – Bit locations must be read in 1-byte increments.

Enable – designates how this instruction will operate. Select from one of the following:

Once on Leading Edge - select this option to have this instruction run to completion exactly one time. Typically, this instruction will take more than one controller scan to complete. Configured this way the instruction is Edge Triggered.

Continuous on Power Flow at Interval - Select this option to have this instruction run as long as the instruction has power flow. After the instruction has initially run, if the instruction still has power flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select how much time (in milliseconds) to wait between successive runs. A value of 0ms means the instruction will re-run immediately.

On Success – When the instruction completes successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

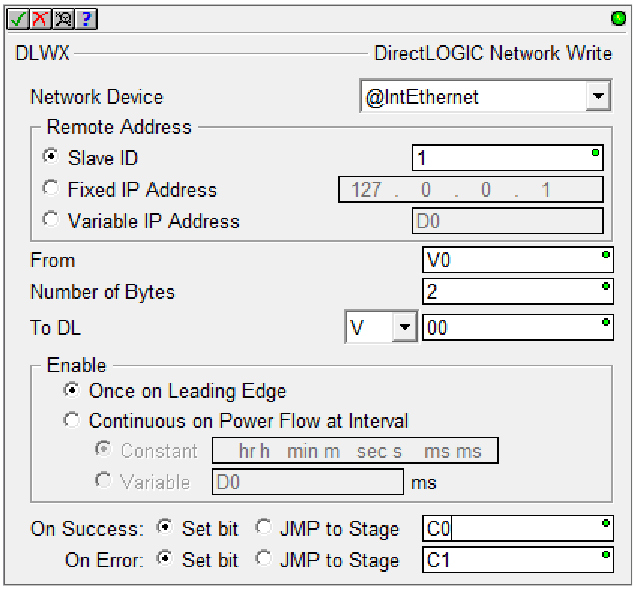
On Error – When the instruction does not complete successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

HOST Ethernet Protocol (Client, Server), continued

DLWX



DLWX — DirectLOGIC Network Write

Network Device: @IntEthernet

Remote Address
- ● Slave ID: 1
- ○ Fixed IP Address: 127 . 0 . 0 . 1
- ○ Variable IP Address: D0

From: V0

Number of Bytes: 2

To DL: V 00

Enable
- ● Once on Leading Edge
- ○ Continuous on Power Flow at Interval
  - ● Constant: hr h min m sec s ms ms
  - ○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage C0

On Error: ● Set bit ○ JMP to Stage C1

Network Device – The device associated with the physical port that you want to communicate from. @IntEthernet is the name of the device associated with the built in Ethernet port.

Remote Address - Specifies which of the following addressing modes to use:

Slave ID – Selects the Slave ID (Module ID) of the remote DirectLOGIC server. This can be any constant value in the range of 1 to 90, or any readable numeric location that contains a value in that range. If Slave ID is selected, the a TCP/IP broadcast is used to perform the network write operation, this means that both the Do-more! controller and the remote ECOM module must be in the same Broadcast Domain.

Fixed IP Address – Specifies the IP Address assigned to the remote server ECOM module. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address – The IP Address resides in a memory location in the PLC. This allows the IP Address to be changed at runtime. This can be any readable DWord numeric location. Each octet of the IP Address is stored in one byte of the Variable Address location.

From – Specifies the beginning memory address in the Do-more! controller of the data to send to the remote server. This value can be any readable numeric location.

To DL – Designates the data type and the address of the data to Write to the DirectLOGIC controller.

V-Memory locations in a DirectLOGIC controller are unsigned 16-bit values. Each V-Memory location is 2 bytes in length, so reading V-Memory requires the length be in 2-Byte increments. The memory address value must begin on a Byte boundary. Single Bit locations in the DirectLOGIC controllers cannot be written individually, you must write the Byte that contains the desired Bit.

Number of Bytes – Designates the number of elements of the selected type to read.
X, Y, C, S, T, CT, GX, GY, SP – Bit locations must be read in 1-Byte increments.

HOST Ethernet Protocol (Client, Server), continued

Enable – designates how this instruction will operate. Select from one of the following:

Once on Leading Edge – Select this option to have this instruction run to completion exactly one time. Typically, this instruction will take more than one controller scan to complete. Configured this way the instruction is Edge Triggered.

Continuous on Power Flow at Interval – Select this option to have this instruction run as long as the instruction has power flow. After the instruction has initially run, if the instruction still has Power Flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select how much time (in milliseconds) to wait between successive runs. A value of 0ms means the instruction will re-run immediately.

On Success – When the instruction completes successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

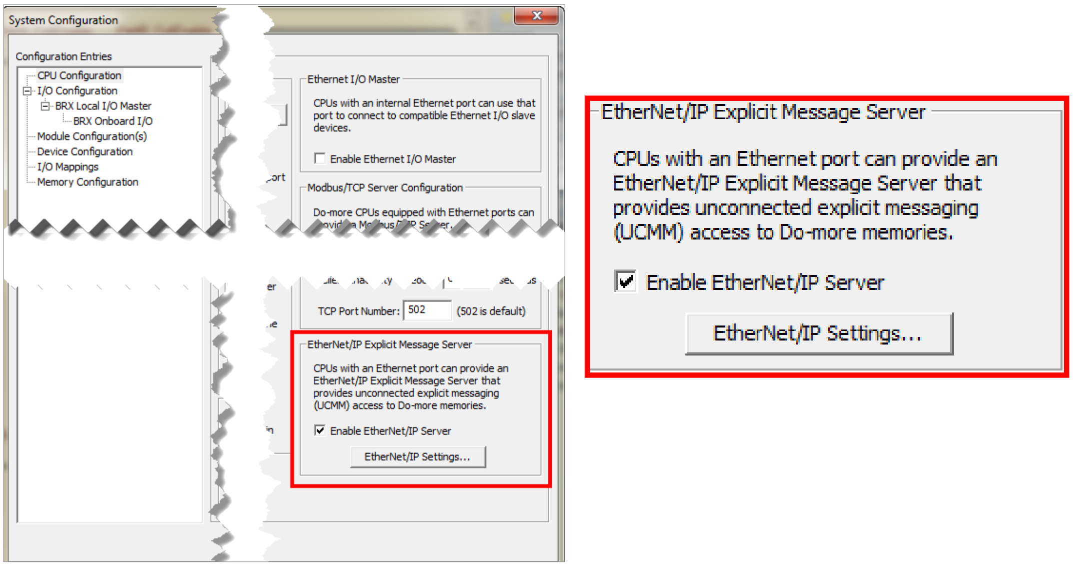
On Error – When the instruction does not complete successfully this action will be performed.

Set Bit – Enable this selection then specify any writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

EtherNet/IP: Explicit Messaging (Server)

The EtherNet/IP Explicit Message Server must be enabled in the System Configuration dialog box.



CPUs with an internal Ethernet port can use that port to connect to compatible Ethernet I/O slave devices.

□ Enable Ethernet I/O Master

Do-more CPUs equipped with Ethernet ports can provide a Modbus/TCP Server.

TCP Port Number: 502 (502 is default)

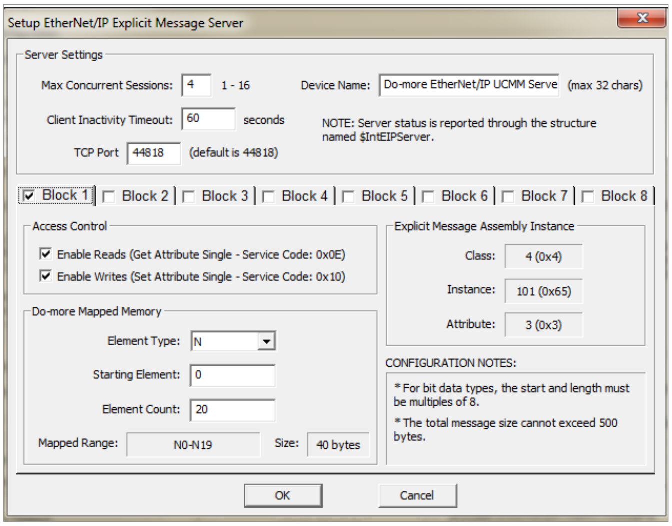
EtherNet/IP Explicit Message Server

CPUs with an Ethernet port can provide an EtherNet/IP Explicit Message Server that provides unconnected explicit messaging (UCMM) access to Do-more memories.

☑ Enable EtherNet/IP Server

EtherNet/IP Settings...

Once this is enabled, you must click the EtherNet/IP Settings button to set the data blocks that will be served to outside EtherNet/IP Clients.



Setup EtherNet/IP Explicit Message Server

Server Settings

Max Concurrent Sessions: 4 1 - 16

Device Name: Do-more EtherNet/IP UCMM Serve (max 32 chars)

Client Inactivity Timeout: 60 seconds

NOTE: Server status is reported through the structure named $IntEIPServer.

TCP Port 44818 (default is 44818)

☑ Block 1 | □ Block 2 | □ Block 3 | □ Block 4 | □ Block 5 | □ Block 6 | □ Block 7 | □ Block 8

Access Control

☑ Enable Reads (Get Attribute Single - Service Code: 0x0E)

☑ Enable Writes (Set Attribute Single - Service Code: 0x10)

Explicit Message Assembly Instance

Class: 4 (0x4)

Instance: 101 (0x65)

Attribute: 3 (0x3)

Do-more Mapped Memory

Element Type: N

Starting Element: 0

Element Count: 20

Mapped Range: N0-N19 Size: 40 bytes

CONFIGURATION NOTES:

* For bit data types, the start and length must be multiples of 8.

* The total message size cannot exceed 500 bytes.

OK Cancel

TCP Port Number (44818 is default) – Designates the TCP port number on which the EtherNet/IP Explicit Message Server will accept connections. The default value of 44818 is the industry standard and will rarely need to be changed. However, this can be any constant value between 0 and 65535.

Device Name – Up to 32 characters that will be returned in requests for the Identity Class.

NOTE: *Runtime status of the EtherNet/IP Server is accessed through the built-in structure $IntEIPServer which has the following members:*

.ActiveSessions (read-only) – The number of concurrent open connections to EtherNet/IP clients.
.LastError – Last error reported to an EtherNet/IP client.
.Errors – Total number of errors returned to all EtherNet/IP clients.
.Transactions – Total number or completed client requests to EtherNet/IP clients.

EtherNet/IP: Explicit Messaging (Server), continued

Select the quantity of Data Blocks (up to 8 data blocks - Block 1 to Block 8) that will be made available to EtherNet/IP Clients. You can specify how many data blocks will be used and then configure each of the blocks in the sections below.

Access Control – Specify external access ability.

Enable Reads (Get Single Attribute – Service Code: 0x0E) – Allows EtherNet/IP Clients to read from this data block using Get Single Attribute.

Enable Writes (Set Single Attribute – Service Code: 0x10) – Allows EtherNet/IP Clients to write to this data block using Set Single Attribute.

Do-more Mapped Memory – Specifies the first location in a data block in the Do-more! PLC memory that will be accessed when requests with this Class/Instance/Attribute are received.

Element Type – Select the memory block to use from the drop-down list of available numeric memory blocks.

Starting Element – Specify the first element in the memory block to use.

Element Count – The number of successive Elements in the Do-more! memory block to use. The maximum total size on an EtherNet/IP request is 500 bytes, so the maximum number of elements per block will depend on the size of the individual elements. Refer to the chart below:

Element Count

Element Type	Maximum Number of This Type per Data Block
Bit	4000
Byte	500
Word	250
DWord	125
Real	125

Mapped Range – Displays the currently selected range of Elements.

Size – Displays the size of the selected Element range in Bytes.

Explicit Messaging Assembly Instance – Displays the Path (class / instance / attribute) of the data block being configured.

Class – The Class of all the data blocks is fixed at 0x04 (assembly class).

Instance – Each of the 8 blocks is assigned a unique Instance as shown in this table.

Instance Blocks

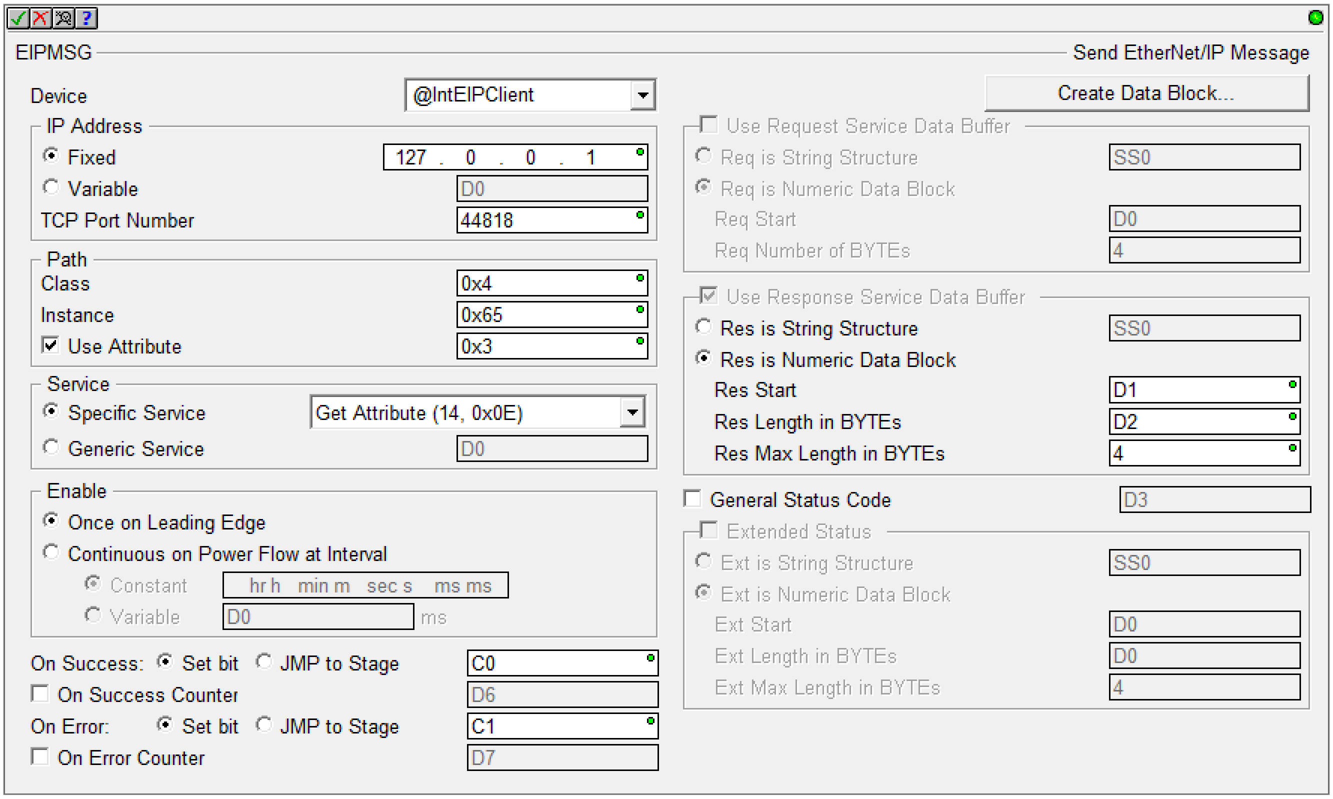
Block Number	Instance ID
0	101 (0x65)
1	102 (0x66)
2	103 (0x67)
3	104 (0x68)
4	105 (0x69)
5	106 (0x6A)
6	107 (0x6B)
7	108 (0x6C)

Attribute – The Attribute of each of these data blocks is fixed at 0x03.

CONFIGURATION NOTES: – This area contains information pertinent to the configuration selections being made.

EtherNet/IP: Explicit Messaging (Client)

EIPMSG Instruction – The Send EtherNet/IP Message instruction implements an Explicit Unconnected EtherNet/IP Client using the on-board Ethernet port of a Do-more! CPU. An explicit message client initiates request/response oriented communications with EtherNet/IP servers. Message rates and latency requirements should not be too demanding. Examples of other explicit message servers you can talk to are barcode scanners, scales, drives, or other intelligent devices.



EIPMSG — Send EtherNet/IP Message

Device: @IntEIPClient

IP Address
- ● Fixed: 127 . 0 . 0 . 1
- ○ Variable: D0
- TCP Port Number: 44818

Path
- Class: 0x4
- Instance: 0x65
- ☑ Use Attribute: 0x3

Service
- ● Specific Service: Get Attribute (14, 0x0E)
- ○ Generic Service: D0

Enable
- ● Once on Leading Edge
- ○ Continuous on Power Flow at Interval
  - ● Constant: hr h min m sec s ms ms
  - ○ Variable: D0 ms

On Success: ● Set bit ○ JMP to Stage: C0
☐ On Success Counter: D6
On Error: ● Set bit ○ JMP to Stage: C1
☐ On Error Counter: D7

Create Data Block...

☐ Use Request Service Data Buffer
- ○ Req is String Structure: SS0
- ● Req is Numeric Data Block
- Req Start: D0
- Req Number of BYTEs: 4

☑ Use Response Service Data Buffer
- ○ Res is String Structure: SS0
- ● Res is Numeric Data Block
- Res Start: D1
- Res Length in BYTEs: D2
- Res Max Length in BYTEs: 4

☐ General Status Code: D3

☐ Extended Status
- ○ Ext is String Structure: SS0
- ● Ext is Numeric Data Block
- Ext Start: D0
- Ext Length in BYTEs: D0
- Ext Max Length in BYTEs: 4

Device – Designates which of the pre-configured EtherNet/IP Client devices to use when sending the message. Part of the configuration for a device is assigning a name to the device. It is that name which will show up in the Device selection drop-down menu. For more information on configuring devices go to the Help file.

IP Address – The IP Address of the EtherNet/IP Server to send the message to. This can be either a Fixed (static) IP Address or a Variable (dynamic) value as described below:

Fixed Address – The TCP/IP Address assigned to the EtherNet/IP Server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

NOTE: *Invoking the Element Browser (F9) for this field will bring up the IP Address Lookup utility that can find the IP Address for a given name.*

Variable IP Address – The IP Address resides in the specified memory location. This can be any readable DWord numeric location.

TCP Port Number – The port number of the EtherNet/IP Server to send the message to. The default value of 44818 is typically the correct port number for EtherNet/IP protocol. This can be any constant value between 0 and 65535, or any readable numeric location containing a value in that range.

EtherNet/IP: Explicit Messaging (Client), continued

Path – Specifies the parameters for the request. The specific values needed for the fields will be provided by the manufacturer of the EtherNet/IP server that you are talking to.

Class – The Class ID value (defined by the EtherNet/IP Server). This can be any positive integer value or any readable numeric location.

Instance – The Instance ID value (defined by the EtherNet/IP Server). This can be any positive integer value or any readable numeric location.

Use Attribute – Enable this option to specify the Attribute value (defined by the EtherNet/IP Server). This can be any positive integer value or any readable numeric location.

Service - Specifies the operation to perform on the set of objects. Choose from the following list of predefined Services, or select Generic and enter the Service number.

Specific Service – Select one of the predefined Service Requests below:

Get Single Attribute (14, 0x0E) - request a single attribute

Set Single Attribute (16, 0x10) - write a single attribute

Get All Attributes (1, 0x01) - request all of the attributes

Set All Attributes (2, 0x02) - write to all of the attributes

Generic Service – Specify a Service that is NOT one of the predefined Service Requests. This can be any constant integer value or readable memory location.

Create Data Block – If an appropriate data block does not already exist, or if you want to create an additional data block for use in this instruction, then click this button to open a dialog where you can create a new data block of the required type.

Use Request Service Data Buffer – This selection will be automatically enabled when any Set Attribute service is selected and automatically disabled when any Get Attribute service is selected. This buffer can be enabled any time the Generic Service is selected.

Req is String Structure – Select this option if the data for the Set Attribute service or any Generic service is contained in a String, then enter the String element to use. This can be any of the system-defined Short Strings, or system-defined Long Strings, or any of the user-defined Strings. The maximum length of the String to send is 500 bytes.

Req is Numeric Data Block – Select this option if the data for the Set Attribute service or any Generic service is contained in a numeric memory block. The maximum size of the data block that can be sent in a single service request 500 bytes (250 Words, 125 DWords, 125 Reals).

Req Start – Specify the first element of the memory block that is the data for the Set Attribute or Generic service.

Req Number of Bytes - The number of consecutive BYTEs of data for the Set Attribute or Generic service (Words = 2 Bytes, DWord = 4 Bytes, Real = 4 Bytes).

Use Response Service Data Buffer – This selection will be automatically enabled when any Get Attribute service or any Generic service is selected and automatically disabled when any Set Attribute service is selected. This buffer can be enabled any time the Generic Service is selected.

Res is String Structure – Select this option to store the data from the Get Attribute service or any Generic service in a String, then enter the String element to use. This can be any of the system-defined Short Strings, or system-defined Long Strings, or any of the user-defined Strings. The maximum length of the String that could potentially be received is 500 bytes, so make sure the String can handle the maximum response for the desired service.

EtherNet/IP: Explicit Messaging (Client), continued

Res is Numeric Data Block – Select this option to store the data from the Get service or any Generic service in a numeric memory block. The maximum size of the data block that can be read in a single service request 500 bytes (250 Words, 125 DWords, 125 Reals).

Res Start – Specify the first element in the numeric data block to store the data that was returned by the Get Attribute or Generic service. This can be any writable numeric location.

Res Length in BYTEs – Specify a memory location to store the actual number of Bytes of data that was returned by the Get Attribute or Generic service. This can be any writable numeric location.

Res Max Length in BYTEs – Specify the maximum number of BYTEs of the returned data to retain to store in the data block. This can be any positive integer constant between 1 and 500 or any readable numeric location.

***NOTE:** The byte length value should be a multiple of the number of BYTEs in a single element in the numeric memory block. For example, if the memory block consists of DWords or Reals, this value should be a multiple of 4, as there are 4 BYTEs per DWord or Real.*

General Status Code – enable this option to store the value returned from the EtherNet/IP Server in response to processing the Service Request; enter the numeric location to store the value. This can be any writable numeric location. This value could indicate success or be an error code. Consult the documentation for the EtherNet/IP Server for information on how to interpret General Status Code values.

Extended Status – Enable this option to store any extended status value returned from the EtherNet/IP Server in response to processing the Service Request.

Ext is String Structure – Select this option to store the extended status information in a String, then enter the destination String element. This can be any of the system-defined Short Strings, or system-defined Long Strings, or any of the user-defined Strings.

Ext is Numeric Data Block – Select this option to store the Extended Status data in a numeric data block.

Ext Start – Specify the first element in the numeric data block to store the Extended Status data. This can be any writable numeric location.

Ext Length in BYTEs – Specify a memory location to store the actual number of Bytes of Extended Status data. This can be any writable numeric location.

Ext Max Length in BYTEs – Specify the maximum number of BYTEs of Extended Status data to store in the data block. This can be any positive integer constant between 1 and 500, or any readable numeric location.

***NOTE:** The byte length value should be a multiple of the number of BYTEs in a single element in the numeric memory block. For example, if the memory block consists of DWords or Reals, this value should be a multiple of 4, as there are 4 BYTEs per DWord or Real.*

Enable – Designates how this instruction will operate. Select from one of the following:

Once on Leading Edge - Select this option to have this instruction run to completion exactly one time. Typically, this instruction will take more than one controller scan to complete. Configured this way the instruction is Edge Triggered.

EtherNet/IP: Explicit Messaging (Client), continued

Continuous on Power Flow at Interval – Select this option to have this instruction run as long as the instruction has power flow. After the instruction has initially run, if the instruction still has power flow, the instruction will remain enabled and will wait the specified amount of time before running again. The following options select how much time (in milliseconds) to wait between successive runs. A value of zero milliseconds (0ms) means the instruction will re-run immediately.

On Success – When the instruction completes successfully this action will be performed.

Set Bit – Enable this selection then specify writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Success Counter – Enable this option and select a DWord location to store the total number of times the RX completed successfully. This can be any DWord location.

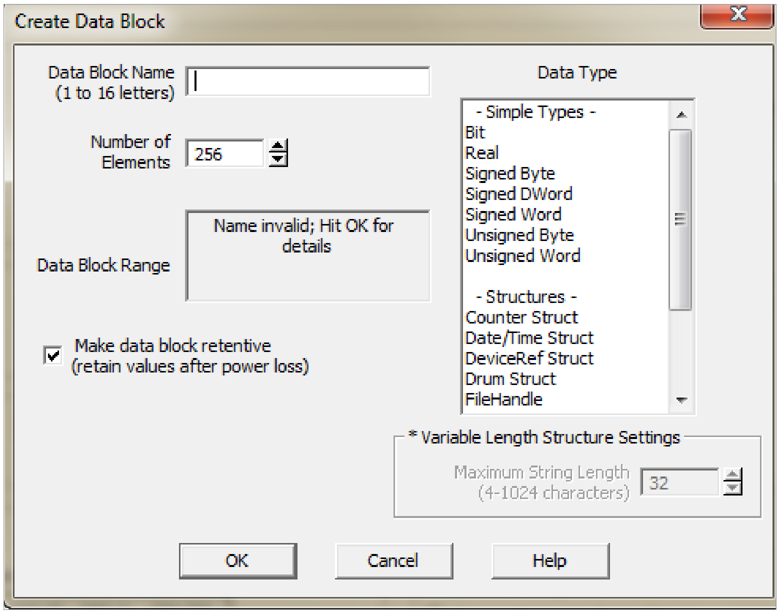
On Error – When the instruction does not complete successfully this action will be performed.

Set Bit – Enable this selection then specify writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error Counter – Enable this option and select a DWord location to store the total number of times the RX failed to complete. This can be any DWord location.

Create Data Block – If an appropriate data block does not already exist, or if you want to create an additional data block for use in this instruction, then click this button to open a dialog where you can create a new data block of the required type.



Create Data Block

Data Block Name (1 to 16 letters):

Number of Elements: 256

Data Block Range: Name invalid; Hit OK for details

☑ Make data block retentive (retain values after power loss)

Data Type

- Simple Types -
Bit
Real
Signed Byte
Signed DWord
Signed Word
Unsigned Byte
Unsigned Word

- Structures -
Counter Struct
Date/Time Struct
DeviceRef Struct
Drum Struct
FileHandle

* Variable Length Structure Settings

Maximum String Length (4-1024 characters): 32

Data Block Name (1 to 16 letters) – Block names must be unique, and consist of 1 to 16 characters (A-Z, a-z; no numbers, no spaces).

Number of Elements – Specifies the number of bytes in the data block. The data blocks must be created on a DWord (4-byte) boundary. The maximum number of Bytes that can be received from a single packet is 1024.

EtherNet/IP: Explicit Messaging (Client), continued

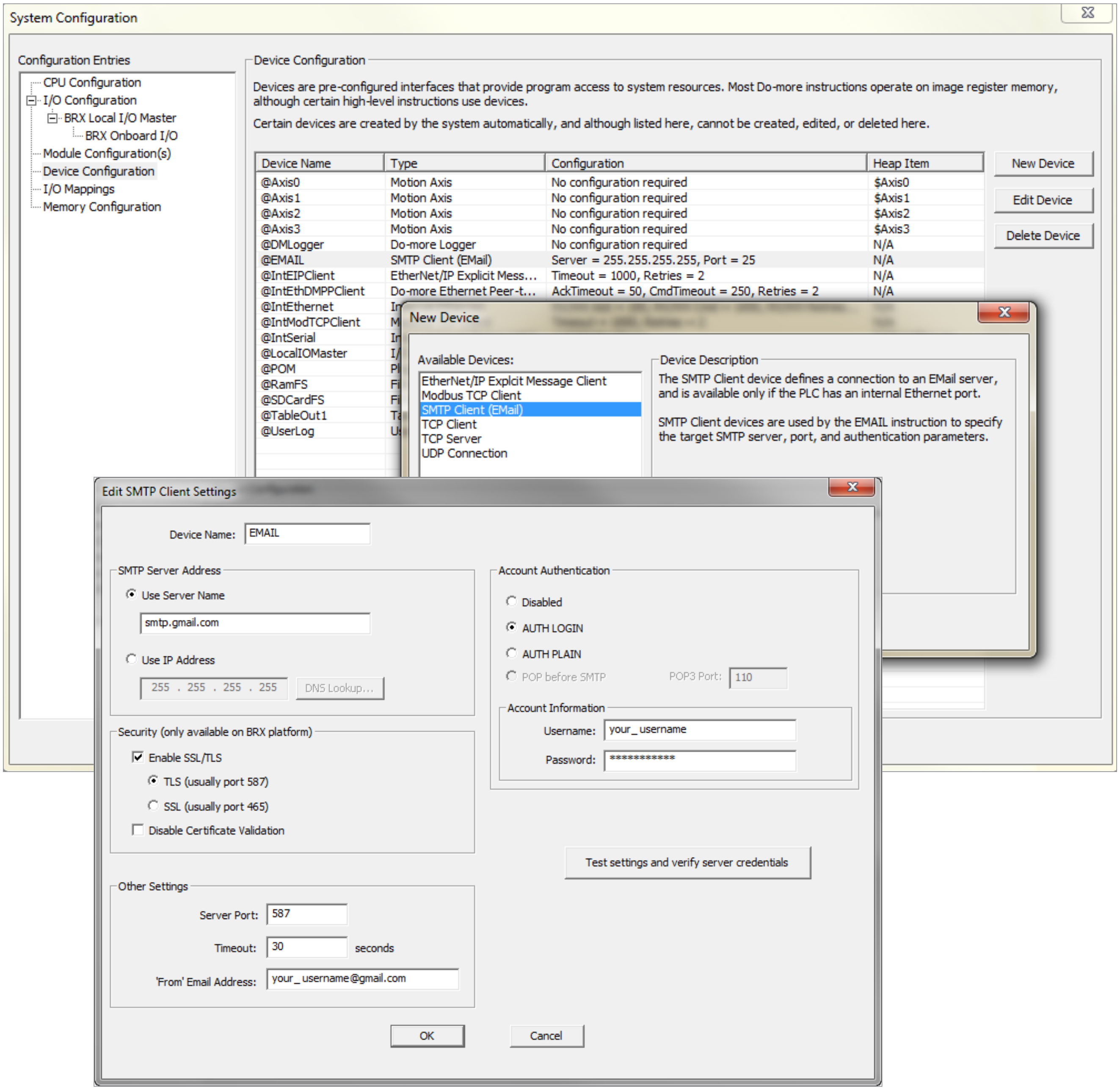
Data Block Range – Displays the first and last element of the block that will be created based on the current entries for Data Block Name and Number of Elements.

Data Type – The data block will consist of Unsigned Bytes.

Make Data Block Retentive (retain values after power loss) – A data block marked as retentive will hold its state through a power cycle or a Program-to-Run mode transition. The status of memory NOT marked as retentive will be cleared at power up and during a Program-to-Run mode transition.

SMTP – EMAIL

The Edit SMTP Client Settings dialog is used to configure an SMTP connection from the Do-more! controller to an SMTP server in order for the controller to send Email. The information that is required to configure an SMTP connection is always going to be specific to the installation. It is up to the programmer to locate the required information.



Devices are pre-configured interfaces that provide program access to system resources. Most Do-more instructions operate on image register memory, although certain high-level instructions use devices.

Certain devices are created by the system automatically, and although listed here, cannot be created, edited, or deleted here.

Device Name	Type	Configuration	Heap Item
@Axis0	Motion Axis	No configuration required	$Axis0
@Axis1	Motion Axis	No configuration required	$Axis1
@Axis2	Motion Axis	No configuration required	$Axis2
@Axis3	Motion Axis	No configuration required	$Axis3
@DMLogger	Do-more Logger	No configuration required	N/A
@EMAIL	SMTP Client (EMail)	Server = 255.255.255.255, Port = 25	N/A
@IntEIPClient	EtherNet/IP Explicit Mess...	Timeout = 1000, Retries = 2	N/A
@IntEthDMPPClient	Do-more Ethernet Peer-t...	AckTimeout = 50, CmdTimeout = 250, Retries = 2	N/A
@IntEthernet	In		
@IntModTCPClient	M		
@IntSerial	In		
@LocalIOMaster	I/		
@POM	Pl		
@RamFS	Fi		
@SDCardFS	Fi		
@TableOut1	Ta		
@UserLog	U		

New Device

Available Devices:
EtherNet/IP Explcit Message Client
Modbus TCP Client
SMTP Client (EMail)
TCP Client
TCP Server
UDP Connection

Device Description
The SMTP Client device defines a connection to an EMail server, and is available only if the PLC has an internal Ethernet port.

SMTP Client devices are used by the EMAIL instruction to specify the target SMTP server, port, and authentication parameters.

Edit SMTP Client Settings

Device Name: EMAIL

SMTP Server Address
● Use Server Name
smtp.gmail.com
○ Use IP Address
255 . 255 . 255 . 255 DNS Lookup...

Security (only available on BRX platform)
☑ Enable SSL/TLS
● TLS (usually port 587)
○ SSL (usually port 465)
□ Disable Certificate Validation

Other Settings
Server Port: 587
Timeout: 30 seconds
'From' Email Address: your_username@gmail.com

Account Authentication
○ Disabled
● AUTH LOGIN
○ AUTH PLAIN
○ POP before SMTP POP3 Port: 110

Account Information
Username: your_username
Password: ***********

Test settings and verify server credentials

OK Cancel

SMTP, continued

Device Name is the name given to the SMTP Client that will be referenced in Send Email (EMAIL) instructions. Device Names can consists of 1 to 16 alphanumeric characters and must follow Nickname.

***NOTE:** The Email instruction in Do-more! Technology v2.2 and later can perform a DNS name resolution internally. That is, they can use the DNS protocol to resolve a name of an SMTP server to its IP address when the Email instruction is executed. This also means the SMTP Client can be configured to establish encrypted connections to Email servers and optionally require trust certificate validation. The Email instruction in Do-more! Technology versions previous to v2.2 do not have this internal DNS capability. This means the SMTP Client in those previous versions must contain the IP address of the SMTP server, not the name of the SMTP server.*

SMTP Server Address selects how the connection to the SMTP server will be established.

Select **Use Server Name** to enter a text string that will be resolved when the EMail instruction is executed. This selection is required if a TLS or SSL encryption to a secure Email server is selected. The CPU will use the DNS server in the Ethernet Settings configuration to resolve the server name entered.

Select **Use IP Address** to enter the static IP address of the SMTP server. Clicking the **DNS Loopkup** button will open the IP Address Lookup utility in Do-more Designer that can search for the IP Address assigned to a given SMTP Server name. This requires a functional connection to a DNS Server. Enter the URL for the SMTP server then click the **Lookup** button. If the operation is successful click the **Select** button to use the IP address that was found.

***NOTE:** In Do-more! Technology versions previous to v2.2, the IP Address of the SMTP Server can be resolved at runtime by using additional ladder logic instructions, specifically the Name to IP Address (DNSLOOKUP) instruction can be used to find the IP Address associated with a Server's name, then the Write Device Register (DEVWRITE) instruction can be used to set the SMTP Client Device to use the IP Address that was found. Refer to the example program in the Send Email (EMAIL) instruction to see how this is done.*

Security (only available on BRX platform) options are used when establishing an encrypted connection to a secure SMTP server instead of a plain-text connection to an unsecured port 25 server.

Enable SSL / TLS uses an encrypted connection to the SMTP Server. Disabling this option will set the Server Port number to 25.

- **TLS (usually port 587)** will establish an encrypted connection to the SMTP Server Address using STARTTLS protocol.
- **SSL (usually port 465)** will establish an encrypted connection to the SMTP Server Address using SSL protocol.

Disable Certificate Validation – If this option is NOT selected, when Do-more! Designer creates the SMTP Client, it will retrieve the security certificate for the specified SMTP server from the PC running Do-more! Designer, and will attach that certificate to the SMTP Client device when it is downloaded to the CPU. Each time an EMail instruction uses the SMTP Client device, the certificate is validated as part of establishing the encrypted connection. Checking the **Disable Certificate Validation** option will prevent the SMTP Client device from attempting to validate the certificate as part of establishing the encrypted connection.

Server Port – the IP Port number to use when creating the connection to the SMTP server:

- 25 is the standard Server Port number used by plain-text (unencrypted) SMTP servers.
- 587 is the standard Server Port number for encrypted connections using STARTTLS protocol.
- 465 is the standard Server Port number used for encrypted connections using SSL protocol.

SMTP, continued

The Server Port number can be read at runtime with the Read Device Register (DEVREAD) instruction, and changed at runtime through the Write Device Register (DEVWRITE) instruction.

Timeout – the amount of time (in seconds) the SMTP Client will attempt to connect to the specified SMTP server before reporting an error. The default value of 30 seconds should be sufficient in most instances. Be aware that it is quite normal for communications with SMTP servers to take several seconds of time - especially so when using SLS / TLS encryption. Setting this value too low will only cause needless problems.

From Email Address – specifies the Email address that all Emails using this SMTP Client will use in the 'From' field. Email addresses must be in the form of X@Y.Z. SMTP servers typically require that the 'From' address be configured as a recipient address on that SMTP server before they will accept Emails from that address.

Authentication – used if the SMTP server requires authentication before it will accept an Email from the Do-more! CPU. Select one of the following methods:

- **Disabled**: if the SMTP server does not require authentication.
- **AUTH LOGIN**: authenticate by logging into the SMTP Server with the Username and Password specified below.
- **AUTH PLAIN**: authenticate by logging into the SMTP Server with the Username and Password specified below.
- **POP before SMTP**: an older authentication method that attempts to get Email before attempting to send an Email, the premise being that if an Email client can log in and read Email the Client must be legitimate.
- **Pop3 Port**: the default port number of 110 is the standard IP port number for POP3 requests and should not need to be change. The Pop3 Port number can be examined at runtime with the Read Device Register (DEVREAD) instruction, and changed at runtime through the Write Device Register (DEVWRITE) instruction.

Account Information – These authentication methods require a UserID and Password. For 'AUTH LOGIN' and 'AUTH PLAIN', the account information is for the SMTP account. For the 'POP before SMTP' method, the account information is for the POP3 account.

Username – 1 to 64 characters

Password – 1 to 19 characters

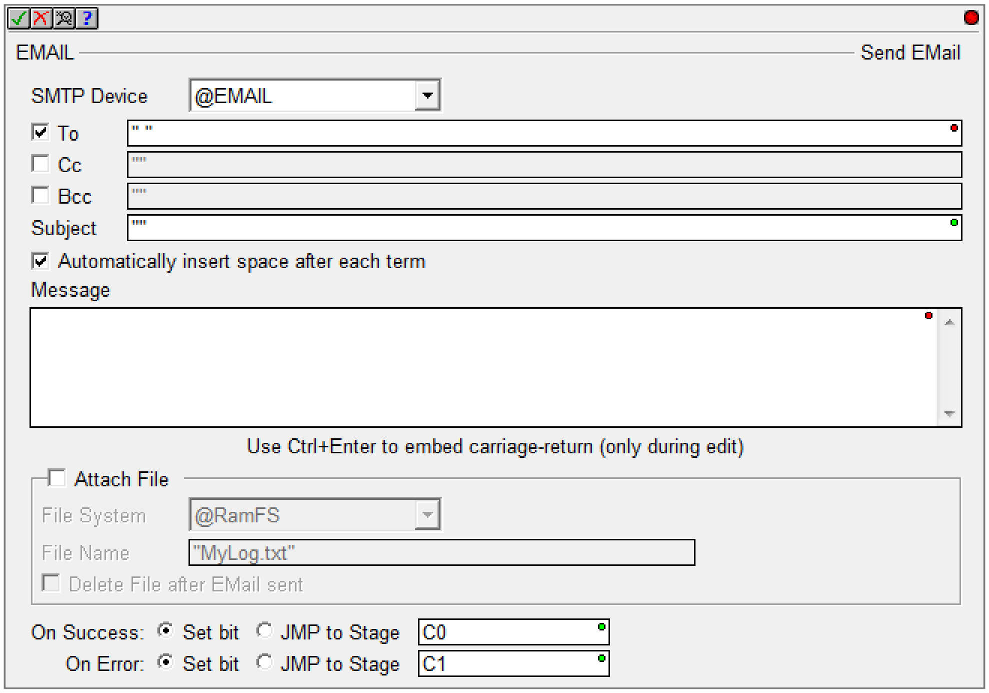
Clicking the **Test settings and verify server credentials** button will attempt to establish a connection to the specified SMTP Server using the options and parameters that have been selected. This step is required if the SMTP Client will use Certificate Validation.

***NOTE:** This verification work is done by Do-more! Designer - not the PLC - but it will mimic the work the PLC will do when it attempts to send Email using this SMTP Client. The Help text for the Send Email (EMAIL) instruction has a section detailing how to use Do-More! Logger to debug SMTP connection issues from the PLC's perspective.*

SMTP, continued

EMAIL (Send Email instruction)

The Send Email (EMAIL) instruction is used to send an Email message. This instruction is only valid for Do-more! controllers that have an on-board Ethernet port. The message portion of the Email can be any combination of text and data elements from the controller. An Email can also send an attachment which can be any file on any of the built-in file systems.



EMAIL — Send EMail

SMTP Device: @EMAIL

☑ To: " "

☐ Cc: ""

☐ Bcc: ""

Subject: ""

☑ Automatically insert space after each term

Message:

Use Ctrl+Enter to embed carriage-return (only during edit)

☐ Attach File

File System: @RamFS

File Name: "MyLog.txt"

☐ Delete File after EMail sent

On Success: ● Set bit ○ JMP to Stage C0

On Error: ● Set bit ○ JMP to Stage C1

SMTP Device – Designates the SMTP Client device to use to send the Email. An SMTP Client device must be configured before the Send Email instruction can be added to a program.

To – One or more primary audience Email addresses separated by semicolons. This can be a string literal (text in double quotes), or any String element.

Cc (optional) – One or more Courtesy Copy (or Carbon Copy) Email addresses separated by semicolons. This can be a string literal (text in double quotes), or any String element.

Bcc (optional) – One or more Blind Carbon Copy Email addresses separated by semicolons. This can be a string literal (text in double quotes), or any String element.

Subject – The Total number of characters in Subject line is limited to 1024 characters. This can be a string literal (text in double quotes), or any readable String element.

Automatically insert space after each term – Will insert a space between the terms when the instruction is processed. This is most useful when the Message Field contains only a list of elements that would otherwise require a manually entered space character to separate the items.

Message – Text box in which to place the body of the Email message. There can be up to 1023 characters of data in a Message field. This data can consist of any combination of the following:

- String Literal (text in double quotes).
- Control characters.
- Any readable String element.
- Controller data elements (V0, D0, T0.Acc, etc.).
- Data formatting functions (FmtInt, FmtReal, etc.).
- String selection function (Lookup).

SMTP, continued

For a complete description of the available data options provided by the scripting language for use in the Text Field, see the Help file.

Attach File – Enable this option to send an existing file from one of the file Systems in the Do-more! CPU along with the Message text.

File System – Specifies which of the available file systems contains the file.

@RamFS – The 1 MB file system in the Do-more! CPU's system RAM. All Do-more! CPUs will have this file system available.

@SDCardFS – On PLC systems that have the microSD card slot, this selection is the file system on the removable media in that slot.

File Name – The full path (including any directories) of the file on the specified file system. This can be text enclosed in double quotes, or any system or user-defined string. The File Name allows a maximum length of 255 characters including spaces and non-alphanumeric characters, excluding the following characters which have special meaning to the file system * ? " : < >. The File Name is not case sensitive.

Delete File After Email Sent – Enable this option to delete the specified file AFTER the Email has been successfully sent to the specified SMTP Server.

On Success – When the instruction completes successfully this action will be performed.

Set Bit – Enable this selection then specify writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error – When the instruction does not complete successfully this action will be performed

Set Bit – Enable this selection then specify writable bit location.

JMP to Stage – Enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Testing the Send Email Instruction:

When the SMTP Client device was created, the Verify SMTP Server Configuration utility was available to test the SMTP Client configuration from within Do-more! Designer. But that doesn't guarantee the configuration will work once it's downloaded to the PLC if the PLC will be deployed on a different network, or if at some point in the future the stored configuration no longer matches the SMTP Server being targeted. The CPU has two System Strings that automatically receive error and status information while the CPU is in RUN mode. The contents of these two system strings can be viewed on the System Status tab of the System Info dialog, or by adding MSG and ERR to a Data View:

The system string MSG receives status information any time an instruction sets an error code.

The system string ERR will display an associated text message that explains what the error code means.

SMTP, continued

Do-more! Logger

By far, the most useful tool for debugging problems with getting Email to function properly from the PLC's perspective is Do-more! Logger. Setting the system status bit $EnableMsgDump (ST36) ON will cause the Send Email instruction to automatically echo the SMTP traffic between the Do-more! CPU and the SMTP Server to the Do-more! Logger. Do-more! Logger will capture and display this SMTP conversation which allows to programmer to see what changes need to be made to correct any errors.

Send a Text Message via Email

Another option to test the SMTP Client is to send an Email to one of the cellphone carriers email-to-text services so that a text message is generated and sent to the specified cell phone number instead of an email. The web site http://www.emailtextmessages.com/ contains a list of email addresses that can be used to send text messages to phones. This is the most simple way of sending text messages from the Do-more! CPU to a cell phone. These email addresses essentially act as a direct link to a carrier's SMS gateway. For example, for Verizon customers use TenDigitPhoneNumber@vtext.com as the Send To address.

The SMTP Protocol Primer

A typical example of sending a message via SMTP to two mailboxes (alice and theboss) located in the same mail domain (example.com or localhost.com) is reproduced in the following session exchange. In this example, the conversation parts are prefixed with S: and C:, for server and client, respectively; these labels are not part of the exchange.

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: “Bob Example” <bob@example.org>
C: To: “Alice Example” <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 January 2008 16:02:43 -0500
C: Subject: Test message
C:
```

SMTP, continued

```
C: Hello Alice.
C: This is a test message with 5 header fields and 4 lines in the message body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
{The server closes the connection}
```

After the SMTP client establishes a reliable communications channel to the SMTP server, the session is opened with a greeting by the server, usually containing its fully qualified domain name (FQDN).

The client initiates its conversation by responding with a HELO command identifying itself in the command's parameter with its FQDN (or an address literal if none is available).

The client notifies the receiver of the originating email address of the message in a MAIL FROM command. In this example, the email message is sent to two mailboxes on the same SMTP server: one for each recipient listed in the To and Cc header fields. The corresponding SMTP command is RCPT TO. Each successful reception and execution of a command is acknowledged by the server with a result code and response message 250 Ok.

The transmission of the body of the mail message is initiated with a DATA command after which it is transmitted verbatim line by line and is terminated with an end-of-data sequence. This sequence consists of a new-line (<CR><LF>), a single full stop (period), followed by another new-line. Since a message body can contain a line with just a period as part of the text, the client sends two periods every time a line starts with a period; correspondingly, the server replaces every sequence of two periods at the beginning of a line with a single one. Such escaping method is called dot-stuffing.

The server's positive reply to the end-of-data, as exemplified, implies that the server has taken the responsibility of delivering the message. A message can be doubled if there is a communication failure at this time, e.g. due to a power shortage: Until the sender has received that 250 reply, it must assume the message was not delivered. On the other hand, after the receiver has decided to accept the message, it must assume the message has been delivered to it. Thus, during this time span, both agents have active copies of the message that they will try to deliver. The probability that a communication failure occurs exactly at this step is directly proportional to the amount of filtering that the server performs on the message body, most often for anti-spam purposes.

The QUIT command ends the session. If the email has other recipients located elsewhere, the client would QUIT and connect to an appropriate SMTP server for subsequent recipients after the current destination had been queued. The information that the client sends in the HELO and MAIL FROM commands are added as additional header fields to the message by the receiving server. It adds a Received and Return-Path header field, respectively.

The server responds with a 221 reply then closes the communication channel.

SMTP, continued

SMTP Server Response Codes

SMTP Server Response Codes		
Response Code	Meaning	How to Solve / What To Do
101	The SMTP server is unable to connect.	Try to change the server's name (maybe it was spelled incorrectly) or the connection port.
111	Connection refused or inability to open an SMTP stream.	This error normally refers to a connection issue with the remote SMTP server, depending on firewalls or misspelled domains. Double-check all the configurations and in case ask your provider.
211	System status message or help reply.	It comes with more information about the server.
214	A response to the HELP command.	It contains information about your particular server, normally pointing to a FAQ page.
220	The server is ready.	It's just a welcome message. Just read it and be happy that everything is working (so far).
221	The server is closing its transmission channel. It can come with side messages like "Goodbye" or "Closing connection".	The mailing session is going to end, which simply means that all messages have been processed.
250	Its typical side message is "Requested mail action okay completed": meaning that the server has transmitted a message.	No error; everything has worked and your email has been delivered.
251	"User not local will forward": the recipient's account is not on the present server, so it will be relayed to another.	It's a normal transfer action.
252	The server cannot verify the user, but it will try to deliver the message anyway.	The recipient's email account is valid, but not verifiable. Normally the server relays the message to another one that will be able to check it.
354	The side message can be very cryptic ("Start mail input end <CR><LF>.<CR><LF>"). It's the typical response to the DATA command.	The server has received the "From" and "To" details of the email, and is ready to get the body message.
420	"Timeout connection problem": there have been issues during the message transfer.	This error message is produced only by GroupWise servers. Either your email has been blocked by the recipient's firewall, or there's a hardware problem.
421	The service is unavailable due to a connection problem: it may refer to an exceeded limit of simultaneous connections, or a more general temporary problem.	The server (yours or the recipient's) is not available at the moment, so the dispatch will be tried again later.
422	The recipient's mailbox has exceeded its storage limit.	Create some free room in the destination mailbox.
431	Not enough space on the disk, or an "out of memory" condition due to a file overload.	This error may depend on too many messages sent to a particular domain. You should try again sending smaller sets of emails instead of one big mail-out.
432	Typical side-message: "The recipient's Exchange Server incoming mail queue has been stopped".	A Microsoft Exchange Server's SMTP error code
441	The recipient's server is not responding.	There's an issue with the user's incoming server.
442	The connection was dropped during the transmission.	A typical network connection problem.
446	The maximum hop count was exceeded for the message: an internal loop has occurred.	Ask your SMTP provider to verify what has happened.
447	Your outgoing message timed out because of issues concerning the incoming server.	This happens generally when you exceeded your server's limit of number of recipients for a message. Try to send it again segmenting the list in different parts.
449	A routing error.	Like error 432, it's related only to Microsoft Exchange.

Table continued on next page

SMTP, continued

SMTP Server Response Codes (continued)

Response Code	Meaning	How to Solve / What To Do
450	"Requested action not taken – The user's mailbox is unavailable". The mailbox has been corrupted or placed on an offline server, or your email hasn't been accepted for IP problems or blacklisting.	The server will retry to mail the message again, after some time.
451	"Requested action aborted – Local error in processing". Your ISP's server or the server that got a first relay from yours has encountered a connection problem.	It's normally a transient error due to a message overload, but it can refer also to a rejection due to a remote anti-spam filter.
452	Too many emails sent or too many recipients: more in general, a server storage limit exceeded.	Again, the typical cause is a message overload. Usually the next try will succeed: in case of problems on your server it will come with a side-message like "Out of memory".
471	An error of your mail server, often due to an issue of the local anti-spam filter.	Contact your SMTP service provider to fix the situation.
500	A syntax error: the server couldn't recognize the command.	It may be caused by a bad interaction of the server with your firewall or anti-virus. Read carefully their instructions to solve it.
501	Another syntax error, not in the command but in its parameters or arguments.	In the majority of the times it's due to an invalid email address, but it can also be associated with connection problems (and again, an issue concerning your anti-virus settings).
502	The command is not implemented.	The command has not been activated yet on your own server.
503	The server has encountered a bad sequence of commands, or it requires an authentication.	In case of "bad sequence", the server has pulled off its commands in a wrong order, usually because of a broken connection. If an authentication is needed, you should enter your username and password.
504	A command parameter is not implemented.	Like error 501, is a syntax problem.
510	Bad email address.	One of the addresses in your To, Cc or Bcc line doesn't exist. Check again your recipients' accounts and correct any possible misspelling.
511	Bad email address.	One of the addresses in your To, Cc or Bcc line doesn't exist. Check again your recipients' accounts and correct any possible misspelling.
512	A DNS error: the host server for the recipient's domain name cannot be found.	Check again all your recipients' addresses: there will likely be an error in a domain name (like mail@domain.coom instead of mail@domain.com).

Ethernet Remote IO Master

Do-more! PLCs can utilize Remote I/O in several ways. This is covered in Chapter 14 of this manual.

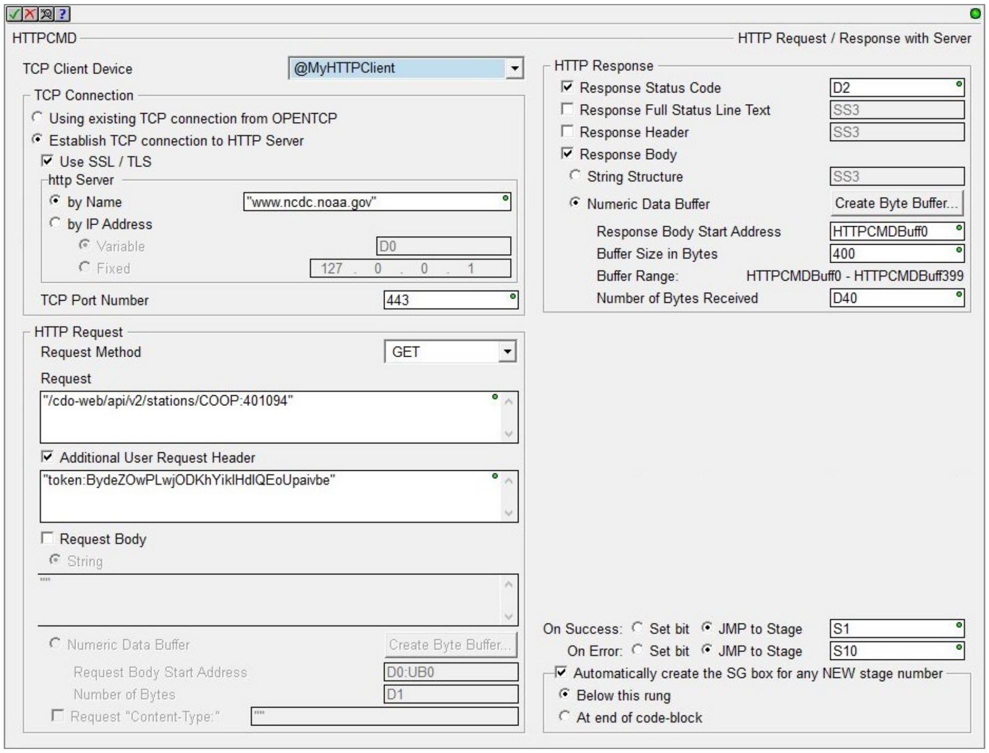
HTTP

The Hypertext Transfer Protocol (HTTP) works as a request-response protocol between a network client and server. A client submits an HTTP request to the server; then the server returns a response to the client. The response contains both status information about the request itself, and the requested content if applicable.

HTTPCMD – Execute HTTP Command

***NOTE:** This instruction can only be used with a BRX MPU!*

The Execute HTTP Command (HTTPCMD) instruction implements the following 5 HTTP commands: GET which is used to request data from a specified resource. HEAD which is almost identical to GET, but without returning data in the Response Body, POST and PUT which are used to send data to a server to create or update a resource, and DELETE which removes the specified resource.



HTTPCMD — HTTP Request / Response with Server

TCP Client Device: @MyHTTPClient

TCP Connection
- ○ Using existing TCP connection from OPENTCP
- ● Establish TCP connection to HTTP Server
- ☑ Use SSL / TLS

http Server
- ● by Name: "www.ncdc.noaa.gov"
- ○ by IP Address
  - ● Variable: D0
  - ○ Fixed: 127 . 0 . 0 . 1

TCP Port Number: 443

HTTP Request
- Request Method: GET
- Request: "/cdo-web/api/v2/stations/COOP:401094"
- ☑ Additional User Request Header: "token:BydeZOwPLwjODKhYiklHdlQEoUpaivbe"
- ☐ Request Body
  - ● String: ""
  - ○ Numeric Data Buffer — Create Byte Buffer...
  - Request Body Start Address: D0:UB0
  - Number of Bytes: D1
  - ☐ Request "Content-Type:" ""

HTTP Response
- ☑ Response Status Code: D2
- ☐ Response Full Status Line Text: SS3
- ☐ Response Header: SS3
- ☑ Response Body
  - ○ String Structure: SS3
  - ● Numeric Data Buffer — Create Byte Buffer...
  - Response Body Start Address: HTTPCMDBuff0
  - Buffer Size in Bytes: 400
  - Buffer Range: HTTPCMDBuff0 - HTTPCMDBuff399
  - Number of Bytes Received: D40

On Success: ○ Set bit ● JMP to Stage: S1

On Error: ○ Set bit ● JMP to Stage: S10

☑ Automatically create the SG box for any NEW stage number
- ● Below this rung
- ○ At end of code-block

The *TCP Connection* group contains selections that establish the required network connection to an HTTP server:

Use Existing TCP Connection from OPENTCP allows this instruction to use a TCP connection that has already been established by an Open TCP Connection (OPENTCP) instruction. If this option is selected the TCP connection will NOT be closed when this instruction is completed.

Establish TCP Connection to HTTP Server will establish a new TCP connection to an HTTP server. Once the instruction has completed execution it will close the TCP connection. The connection will be established using the following credentials.

Enable the **Use SSL / TLS** option if the HTTP server requires an encrypted connection.

HTTP, continued

http / https Server selects how the connection to the HTTP Server will be established:

By Name is the name of the HTTP server that will receive the request. This can be text that is already stored in a system-defined String, or a user-defined String, or enter the text within double quotes in the space below. The Name is not case-sensitive.

By IP Address will use the IP Address of the server. This can be either the traditional dotted-decimal form for a static IP address or a DWord numeric location containing the IP Address if it needs to be set at runtime.

TCP Port Number specifies the port number the connection will use. The default value of 80 is typical for HTTP traffic to unencrypted servers. The value 443 is typically used for HTTPS traffic using (SSL / TLS) encryption. This can be any constant between 1 and 65534, or any Numeric memory location containing a value in that range.

The *HTTP Request* group contain the selections that build up the command that will be sent to the HTTP server.

Request Method selects which HTTP function to send:

GET is used to request data from a specified resource. The query string (name / value pairs) is sent in the URL of a GET request. The requested data will be returned in the Response Body.

HEAD is almost identical to GET, but without returning any data in the Response Body. For example, if "GET /users" returns a list of users, then "HEAD /users" will make the same request but will not return the list of users. HEAD requests are useful for checking what a GET request will return before actually making a GET request.

POST is used to send data to a server to create or update a resource. The data sent to the server with POST is stored in the Request body of the HTTP request.

PUT is also used to send data to a server to create or update a resource. The difference between POST and PUT is calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.

DELETE method deletes the specified resource.

The **Request**, often called the query-string, is a string of information that the server can use as parameters. The Request is usually one or more name and value pairs; for example, term=bluebird. Name and value pairs are separated from each other by an ampersand (&); for example, term=bluebird&source=browser-search. Components in the Request are case-sensitive. This can be text that is already stored in a system-defined String, or a user-defined String, or enter the text within double quotes in the space below.

Enable the optional **Additional User Request Header** selection to include any additional header fields in the request. Request Headers define the operating parameters of an HTTP transaction. They allow the client to pass additional information to the server with the request. A Request header consists of its case-insensitive name followed by a colon ':', then by its value (without line breaks). Leading white space before the value is ignored. Request Headers can be text that is already stored in a system-defined String, or a user-defined String, or enter the text within double quotes in the space below.

***NOTE:** If there is only one user request header entry you do not have to enter the terminating <CR><LF>, the editor will do that for you. If you have multiple user request headers you will need to place the required <CR><LF> termination characters after each header. You can use the Print to String (STRPRINT) instruction to build up the list of headers and store it in a String then reference that String here.*

HTTP, continued

Enable the optional **Request Body** selection to send data bytes immediately following the headers. This is where the data sent to the server with POST or PUT methods is stored.

Select **String** if the text is already stored in a system-defined String, or a user-defined String, or enter the text within double quotes in the box below.

Select **Numeric Data Buffer** to place the Response Body in a numeric data block (a byte buffer).

Click **Create Byte Buffer** to create a new data block of bytes to store the JSON record. You will need to specify the **Data Block Name** (1 to 16 letters) names must be unique and consist of 1 to 16 characters (A-Z, a-z; no numbers, no spaces). The default name is **HTTPCCMDBuff**, but it can be changed. You will also specify **Number of Elements** is the number of bytes (maximum of 65000) in the data block. The data blocks must be created on a DWord (4-byte) boundary. And specify whether to **Make Data Block Retentive** (retain values after power loss) will have the data block hold its state through a power cycle or a Program-to-Run mode transition. Memory NOT marked as retentive will be cleared at power up and during a Program-to-Run mode transition.

Request Body Start Address is the beginning offset into the data block where the Request Body data begins.

Number of Bytes is maximum number of bytes to allocate for use by the Request. This can be any constant value from 1 to the maximum size of the Byte buffer.

The **Buffer Range** shows the range of bytes in the buffer that will be used as defined by the Request Body Start Address and by the Number of Bytes.

***NOTE:** When using a Numeric Data Buffer to store the text for the Request Body, use the Memory View with ASCII format to see the contents of the Numeric Data Buffer in human-readable text form.*

Enable the optional **Request "Content-Type"** to include the Content-Type header with the user-supplied type and sub-type. This can be text that is already stored in a system-defined String, or a user-defined String, or enter the text within double quotes. Two of the more common Content-Types are:

application which represents application data or binary data. For instance, Content-Type: application/json; charset=utf-8 designates the content to be in the JavaScript Object Notation (JSON) format, encoded with UTF-8 character encoding.

text indicates that the content is plain text and no special software is required to read the contents. The subtype represents more specific details about the content, which can be used by the client for special processing. For instance, Content-Type: text/html indicates that the body content is text in HTML format.

The *HTTP Response* group contains selections that will handle the data that is returned from the HTTP Request. All of the HTTP Response selections are optional.

The **Response Status Code** is a numeric value returned from the server. This can be any writable numeric location. The most common Response Codes are :

200 - (OK) the request has succeeded. The information returned with the response is dependent on the method used in the request.

204 - (No Content) the server has fulfilled the request but does not need to return an entity-body, and might want to return updated meta-information.

304 - (Not Modified) if the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code.

HTTP, continued

400 - (Bad Request) the request could not be understood by the server due to malformed syntax.

401 - (Unauthorized) the request requires user authentication.

403 - (Forbidden) the server understood the request, but is refusing to fulfill it. Authorization will not help.

404 - (Not Found) the server has not found anything matching the Request.

409 - (Conflict) the request could not be completed due to a conflict with the current state of the resource.

500 - (Internal Server Error) the server encountered an unexpected condition which prevented it from fulfilling the request.

The **Response Full Status Line Text** contains the status message text that corresponds to the Response Status Code.

The **Response Header** is used to give a more detailed context of the response.

Enable the optional **Response Body** selection to store the body of the response data.

Select **String Structure** if the text is already stored in a system-defined String, or a user-defined String, or enter the text within double quotes in the box to the right.

Select **Numeric Data Block Containing Text** if the Response Body has been stored in a numeric data block (a byte buffer).

Click **Create Byte Buffer** to create a new data block of bytes to store the JSON record. You will need to specify the **Data Block Name** (1 to 16 letters) names must be unique and consist of 1 to 16 characters (A-Z, a-z; no numbers, no spaces). The default name is **HTTPCCMDBuff**, but it can be changed. You will also specify **Number of Elements** is the number of bytes (maximum of 65000) in the data block. The data blocks must be created on a DWord (4-byte) boundary. And specify whether to **Make Data Block Retentive** (retain values after power loss) will have the data block hold its state through a power cycle or a Program-to-Run mode transition. Memory NOT marked as retentive will be cleared at power up and during a Program-to-Run mode transition.

Response Body Start Address is the beginning offset into the data block where the Response Body data begins. The Buffer Range shows the range of bytes in the buffer that will be used by the Response.

Buffer Size in Bytes is maximum number of bytes to allocate for use by the Response.

Buffer Range shows the range of bytes in the buffer that will be used as defined by the Response Body Start Address and by the Buffer Size in Bytes.

Number of Bytes Received is the number of bytes of the Response Body that were stored in the byte buffer.

NOTE: *When using a Numeric Data Buffer to store the text for the Request Body, use the Memory View with ASCII format to see the contents of the Numeric Data Buffer in human-readable text form.*

The **On Success** and **On Error** parameters specify what action to perform when this instruction completes. You do not have to use the same type of selection for both **On Success** and **On Error**.

If the **Set Bit** selection is used for either **On Success** or **On Error**, the specified BIT location will be SET OFF when the instruction is first enabled and will remain OFF until the instruction completes. Once complete, the appropriate Success or Error bit location will be set ON. The specified Bit location is enabled with a SET (Latch) operation (not an OUT operation) meaning that it will remain ON even if this instruction's input logic goes OFF.

HTTP, continued

If the **JMP to Stage** selection is used for either **On Success** or **On Error** the target Stage must be in the same Program code block as this instruction; you cannot specify a target Stage that exists in a different Program code block. When the operation finishes, the target Stage will be enabled the same way as it would with a standalone Jump to Stage (JMP) instruction. The **JMP to Stage** option will only be selectable if this instruction is placed in a Program code block.

On Success selects which of the following actions to perform if the operation is successful:

Set Bit – enable this selection then specify any writable bit location.

JMP to Stage – enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error selects which of the following actions to perform if the operation is unsuccessful:

Set Bit – enable this selection then specify writable bit location.

JMP to Stage – enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Automatically create the SG box for any NEW stage number – if either the On Success or On Error selections are set to JMP to Stage, this option will be enabled which will automatically create any target stage that does not already exist.

Below this rung – the new target stage will be created on a new rung following this instruction.

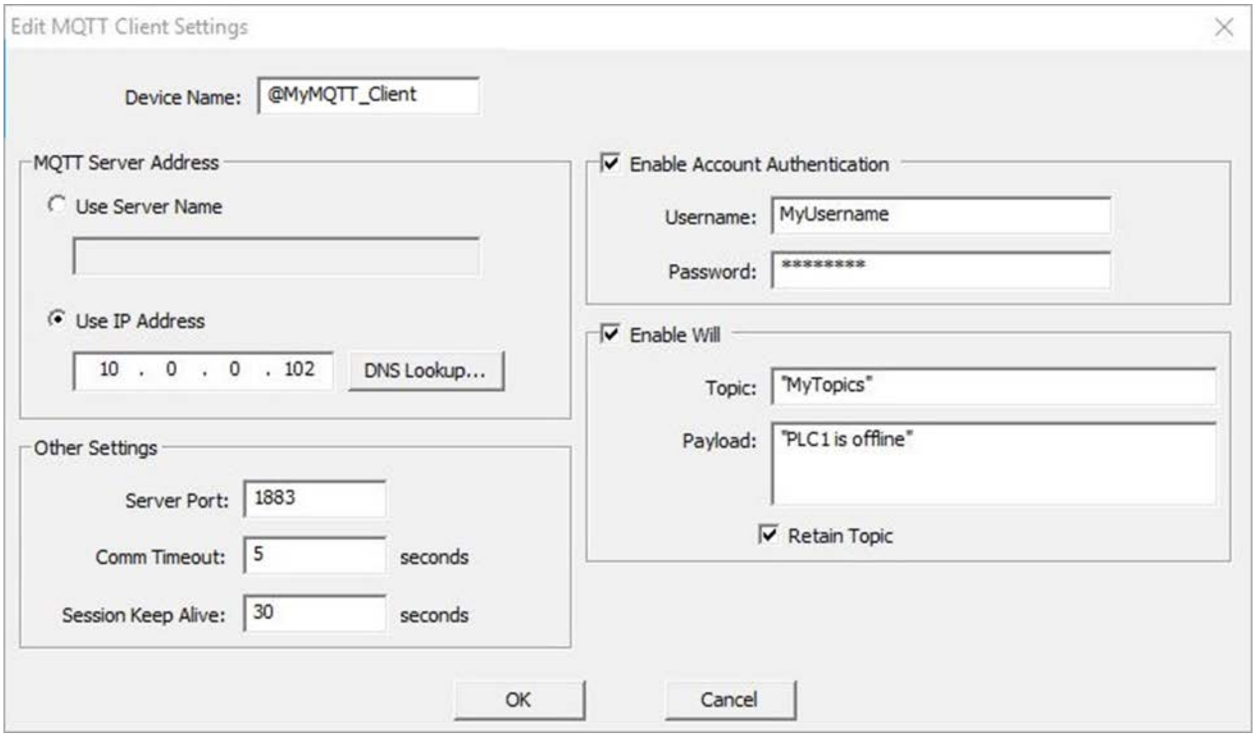
At end of code block – the new target stage will be created as the last rung of this Program.

MQTT

MQTT Client (IoT) Device

BRX PLCs with on-board Ethernet ports can connect to one or more MQTT Brokers, and once connected, these PLCs can publish messages to the Brokers and subscribe to messages from the Brokers. The parameters required to create a connection to a Broker is managed by an MQTT Client (IoT) device.

There is no limit to the number of MQTTPUB instructions that can use a single MQTT Client device, but each MQTT Client device can only provide a subscription space for 100 active topics spread across a maximum of 10 MQTTSUB instructions (for example, 2 enabled instruction with 50 Topics each, or 10 enabled instructions with 10 Topics each). To subscribe to more than 100 Topics, create multiple MQTT Client devices to the same MQTT Broker.



Edit MQTT Client Settings

Device Name: @MyMQTT_Client

MQTT Server Address

○ Use Server Name

● Use IP Address

10 . 0 . 0 . 102

Other Settings

Server Port: 1883

Comm Timeout: 5 seconds

Session Keep Alive: 30 seconds

☑ Enable Account Authentication

Username: MyUsername

Password: ********

☑ Enable Will

Topic: "MyTopics"

Payload: "PLC1 is offline"

☑ Retain Topic

Device Name is the name given to the MQTT Client device that will be referenced in IoT Publish MQTT Topics (MQTTPUB) and IoT Subscribe to MQTT Topics (MQTTSUB) instructions. Device Names can consists of 1 to 16 alphanumeric characters and must follow Nickname Rules.

MQTT Server Address selects how the connection to the MQTT Broker will be established.

Select **Use Server Name** to enter a text string that will be resolved when the MQTTPUB or MQTTSUB instruction is executed.

Select **Use IP Address** to enter the static IP address of the MQTT Broker. Clicking the **DNS Lookup** button will open the IP Address Lookup utility in Do-more! Designer that can search for the IP Address assigned to a given MQTT Broker. This requires a functional connection to a DNS Server. Enter the URL for the SMTP server then click the Lookup button. If the operation is successful click the Select button to use the IP address that was found.

Other Settings

Server Port is the IP Port number to use when creating the connection to the MQTT Broker:

1883 is the standard Port number used in connections to unencrypted MQTT Brokers.

MQTT, continued

This Server Port number can be read at runtime with the Read Device Register (DEVREAD) instruction and changed at runtime through the Write Device Register (DEVWRITE) instruction.

Comm Timeout is the amount of time (in seconds) the MQTT Client device will attempt to connect to the specified MQTT Broker before reporting an error. The default value of 5 seconds should be sufficient in most instances.

Enable Account Authentication is used if the MQTT Broker requires authentication before it will accept messages.

Username – 1 to 64 characters.

Password – 1 to 19 characters.

Last Will and Testament

MQTT is often used on unreliable networks. The Last Will and Testament (LWT) feature is used in MQTT to notify other clients about an ungracefully disconnected client. Each client can specify its Last Will message (a normal MQTT message with Topic, Payload, and , Retain flag) when connecting to a Broker. The Broker will store the message until it detects that the client has disconnected ungracefully. If the client disconnect abruptly, the Broker sends the message to all clients that are subscribed to the Topic which was specified in the Last Will message. The stored Last Will message will be discarded if a client disconnects gracefully.

According to the MQTT 3.1.1 specification the Broker will distribute the LWT of a client in the following cases:

An I/O error or network failure is detected by the Broker.

The client fails to communicate within the Keep Alive time.

The client closes the network connection without sending a DISCONNECT packet first.

The Broker closes the network connection because of a protocol error.

Enable Will to specify a Topic and a Payload that will be published as the Last Will and Testament for this MQTT Broker Device specify the following:

Topic is a simple UTF-8 string, consisting of one or more levels, which are separated by a forward slash (aka the topic level separator). A Topic is case-sensitive. Each Topic must have at least 1 character to be valid and it can contain spaces. A leading forward slash is not recommended. Published Topics cannot start with $. Wildcards (+ or #) are not supported. The Topic can be entered as a String literal (text within double quotes) up to 128 characters, or a user-defined String or a system-defined String element of up to 128 characters.

Payload – It's completely up to the client if it wants to send binary data or textual data. The only requirement is that the data be stored in a String element. Use the Print to String (STRPRINT) instruction with it's collection of String Functions to generate a text Payload. Use the Put Bytes into a String (STRPUTB) instruction to generate a Payload consisting of raw bytes of data.

The **Retain Topic** flag determines if the message will be saved by the Broker even after sending it to all current subscribers. This effectively makes the message for the Topic a "last known good value".

MQTT, continued

MQTTPUB - IoT Publish MQTT Topics

***NOTE:** This instruction can only be used with a BRX MPU or the Do-more! Simulator.*

The IoT Publish MQTT Topics (MQTTPUB) instruction is used to publish messages from a BRX MPU to an MQTT Broker. Each published message must contain a Topic, which will be used by the Broker to forward the message to interested clients, and a Payload which contains the actual data to transmit. When a client publishes a message to a MQTT Broker, the Broker will determine which other clients have subscribed to that Topic and then send the appropriate message to those clients. The client who publishes the message is only concerned about delivering the message to the Broker. From there on it is the responsibility of the Broker to deliver the message to all subscribers. The publishing client doesn't get any feedback that says subscribers have received the message.

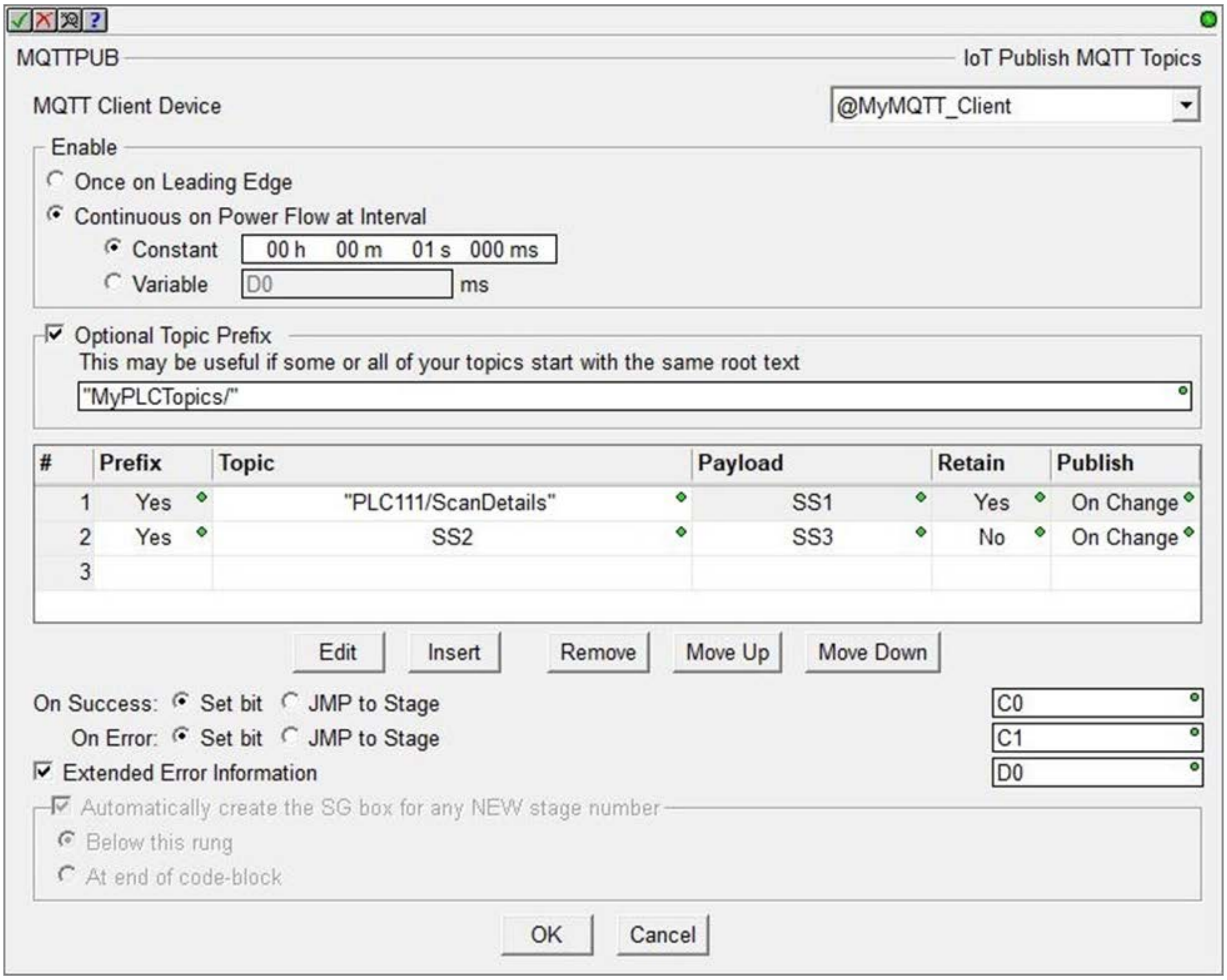
Quality of Service

Quality of Service (QoS) is a major feature of MQTT in that it makes communication in unreliable networks much easier because the protocol handles retransmission and guarantees the delivery of the message regardless of the unreliability of the underlying transport. The QoS level is an agreement between sender and receiver of a message regarding the guarantee of delivering a message.

There are 3 QoS levels in MQTT: At most once (0), At least once (1), and Exactly once (2). **The MQTTPUB instruction publishes all messages with a QoS of 0 (at most once).** The client sends a message but does not wait for an acknowledgment from the Broker.

And remember that in a MQTT system there are always two different parts of delivering a message: publishing client to Broker then Broker to subscribing client(s). The QoS level for publishing client to Broker is set when the client connects to the Broker. When the Broker transfers a message to a subscribing client it uses the QoS of the subscription made by the client.

MQTT, continued



MQTTPUB — IoT Publish MQTT Topics

MQTT Client Device: @MyMQTT_Client

Enable

○ Once on Leading Edge

● Continuous on Power Flow at Interval

● Constant: 00 h 00 m 01 s 000 ms

○ Variable: D0 ms

☑ Optional Topic Prefix

This may be useful if some or all of your topics start with the same root text

"MyPLCTopics/"

#	Prefix	Topic	Payload	Retain	Publish
1	Yes	"PLC111/ScanDetails"	SS1	Yes	On Change
2	Yes	SS2	SS3	No	On Change
3					

On Success: ● Set bit ○ JMP to Stage — C0

On Error: ● Set bit ○ JMP to Stage — C1

☑ Extended Error Information — D0

☑ Automatically create the SG box for any NEW stage number

● Below this rung

○ At end of code-block

MQTT Client Device selects which of the preconfigured MQTT Client devices this instruction will connect to.

Click **create device** to open the Add Device dialog of the System Configuration where a new MQTT Client device can be created.

Enable selects how often the Topics in this instruction will be published. Choose one of the following:

Select the **Once on Leading Edge** option to have this instruction make exactly one attempt to publish each of the Topics in the instruction.

Select the **Continuous on Power Flow at Interval** option to have this instruction attempt to publish each of the Topics in the instruction when the instruction is first enabled, then as long as this instruction remains enabled, each time the specified time interval is reached, the list of Topics is reprocessed. Refer to the Publish Interval Setting below which determines if a Topic is published only if it has changed, or published every time the list is processed.

The following options select how much time (in milliseconds) to wait between successive runs. A value of 0ms means the instruction will re-run immediately. Be aware of setting an interval value that is very low because many Brokers will protect themselves against overloading by rejecting or ignoring messages.

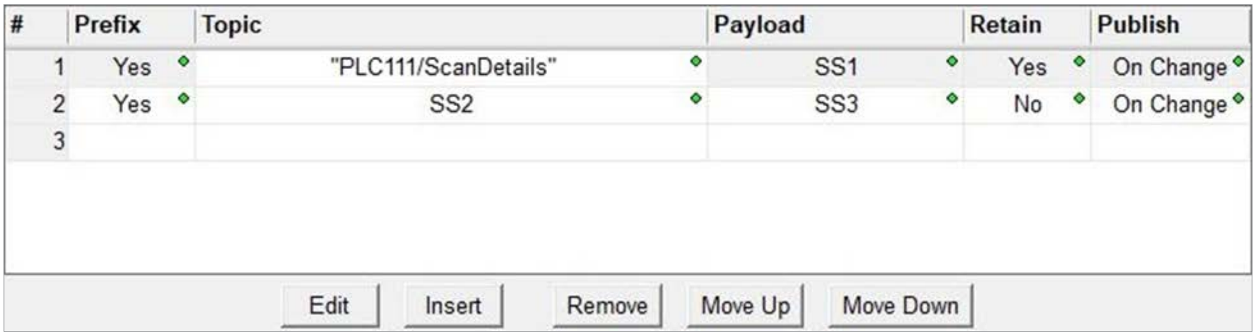
Constant means the interval time is a fixed value specified as Hours / Minutes / Seconds / Milliseconds.

MQTT, continued

Variable means this can be any writable numeric location that contains a value between 0 and 2,147,483,647.

The next section contains a list of up to 50 Topics that get published to the selected MQTT Broker.

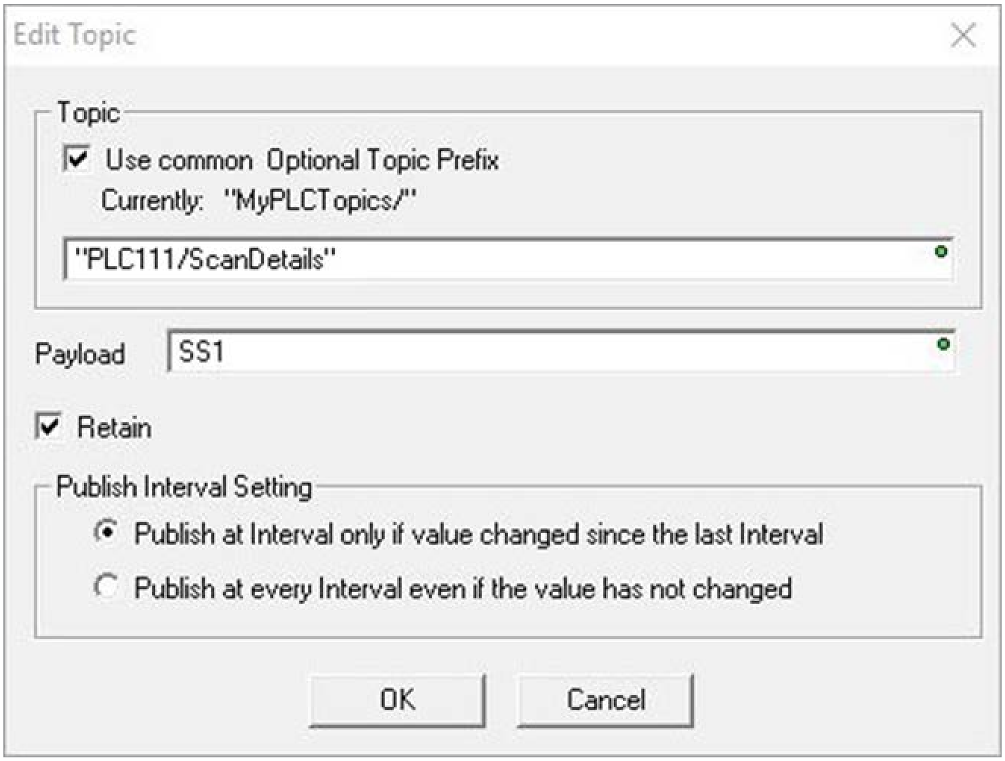
Enabling the **Optional Topic Prefix** allows Topics to use repeated text. If there are multiple Topics that begin with the same text, you can put the repeated text in this Optional Topic Prefix and then select that optional text when the Topic is created and added to the list. This can be any string literal (text enclosed with double quotes) of any user-defined or system-defined String. Remember, Topics are case sensitive.



#	Prefix	Topic	Payload	Retain	Publish
1	Yes	"PLC111/ScanDetails"	SS1	Yes	On Change
2	Yes	SS2	SS3	No	On Change
3					

The buttons below the list are functions which manage the rows in the instruction: **Edit** opens the editor for the currently selected row, **Insert** adds an empty row before the currently selected row and opens the editor for this new row, **Remove** deletes the currently selected row from the table, and **Move Up** / **Move Down** shifts the currently selected row up one or down one row respectively in the table.

Clicking the **Edit** (on an existing Topic) or **Insert** button will open a sub-editor where the contents of the message are managed. The **Topic** group specifies the Topic to publish:



Edit Topic

Topic

☑ Use common Optional Topic Prefix

Currently: "MyPLCTopics/"

"PLC111/ScanDetails"

Payload: SS1

☑ Retain

Publish Interval Setting

● Publish at Interval only if value changed since the last Interval

○ Publish at every Interval even if the value has not changed

MQTT, continued

If an **Optional Topic Prefix** was created on the main dialog, you can enable the **Use Common Optional Topic Prefix** to prepend the prefix text to the Topic text to construct the full name of the Topic.

The **Topic** itself is a simple UTF-8 string, consisting of one or more levels, which are separated by a forward slash (aka the topic level separator). A Topic is case-sensitive. Each Topic must have at least 1 character to be valid and it can contain spaces. A leading forward slash is not recommended. Published Topics cannot start with $. MQTTPUB does not support Wildcards (+ or #) in Topics. The Topic can be entered as a String literal (text within double quotes) up to 128 characters, or a user-defined String or a system-defined String element.

The **Payload** is the data to be published to the Broker. MQTT protocol itself is data-agnostic, meaning the format of the data totally depends on the how the Payload will be used by the subscribing clients. It's completely up to the publishing client if it wants to send binary data or textual data. The only requirement for the instruction is that the data be a String literal (text within double quotes) up to 128 characters, or stored in a String element. Use the Print to String (STRPRINT) instruction with its collection of String Functions to embed PLC data into ASCII text Payload. Use the Put Bytes into a String (STRPUTB) instruction to generate a Payload consisting of raw bytes of data.

The **Retain** flag determines if the message will be saved by the Broker even after sending it to all current subscribers. This effectively makes the message for the Topic a "last known good value". Normally if a publisher publishes a message to a Topic, and no one is currently subscribed to that Topic, the message is simply discarded by the Broker. However, the publisher can tell the Broker to keep the last message for that Topic by setting the Retain flag. New clients that subscribe to a Topic that is Retained will receive the "last known good value" for that Topic as soon as they subscribe. Without retained messages, any new subscribers would have to wait for the status to change before it received a message. Only one message is Retained per Topic. The next message published on that Topic replaces the last Retained message for that Topic.

To **Delete a retained message** send another Retained message with an empty Payload (a zero-byte String, for example "") to the same Topic where the previous Retained message is stored. The Broker deletes the Retained message meaning new subscribers will no longer get a "last known good value" message for that Topic.

Publish Interval Setting is only available if the **Enable** selection is set to Continuous on Power Flow at Interval. The selection determines whether or not the Topic will be published each time the MQTTPUB executes regardless of whether the Topic or the Payload changed since the last interval:

Publish at interval only if value changed since the last interval means each time the MQTTPUB instruction processes the entry list the Topic and Payload will be published only if either the Topic or the Payload have changed since the last time the list was processed.

Publish at every interval even if value has not changed means each time the MQTTPUB instruction processes the entry list the Topic and Payload will be published, even if neither the Topic or Payload have changed since the last time the list was processed.

Depending on the **Enable** selection, this instruction will run to completion exactly one time, or run to completion multiple times as long as the instruction is enabled. Each time the execution of this instruction completes it is deemed a Success if at least one of the Topics in the list was successfully published. If none of the Topics were successfully published the instruction is deemed to be in Error. The **On Success** and **On Error** parameters specify what action to perform when this instruction completes. You do not have to use the same type of selection for both **On Success** and **On Error**.

MQTT, continued

If the **Set Bit** selection is used for either **On Success** or **On Error**, the specified BIT location will be SET OFF when the instruction is first enabled and will remain OFF until the instruction completes. Once complete, the appropriate Success or Error bit location will be set ON. The specified Bit location is enabled with a SET (Latch) operation (not an OUT operation) meaning that it will remain ON even if this instruction's input logic goes OFF.

If the **JMP to Stage** selection is used for either **On Success** or **On Error** the target Stage must be in the same Program code block as this instruction; you cannot specify a target Stage that exists in a different Program code block. When the operation finishes, the target Stage will be enabled the same way as it would with a standalone Jump to Stage (JMP) instruction. The **JMP to Stage** option will only be selectable if this instruction is placed in a Program code block.

On Success selects which of the following actions to perform if the operation is successful:

Set Bit – enable this selection then specify any writable bit location.

JMP to Stage – enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error selects which of the following actions to perform if the operation is unsuccessful:

Set Bit – enable this selection then specify writable bit location.

JMP to Stage – enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

The **Extended Error Information** is a DWord location that contains a combination value. Using the :SD cast here provides a good method for easily separating the high / low values. For example, V100:SD would place the high word in V101 and the low Word in V100).

The high Word is the entry number of the first Topic that failed. There may have been multiple failures, but only the entry number of the first failure will be listed in the Return Code. That error must be corrected before failures for any later entry numbers will be shown.

The low Word contains the error code itself from the following table:

MQTT Error Codes

Value in Low Word	Error	Description
0	No Error	
57	Unexpected MQTT Response	The response from the MQTT Broker was not properly formed.
58	MQTT Broker Rejected	The MQTT Broker refused the Topic; could be a bad username or password, or a security violation on the Broker itself.
59	Out of Resources (MQTTSUB only)	There are more than 10 MQTTSUB instructions using the same MQTT Client device, or there are more than 100 Topics using the same MQTT Client device.
60	Invalid Topic	This typically means the Topic is empty.
61	Duplicate Topic	The Topic is already subscribed to in a different MQTTSUB instruction.

The last error code reported by the MQTT Broker will be in **$LastProtoError** (DST38).

Automatically create the SG box for any NEW stage number – if either the On Success or On Error selections are set to JMP to Stage, this option will be enabled which will automatically create any target stage that does not already exist.

Below this rung – the new target stage will be created on a new rung following this instruction.

MQTT, continued

At end of code block – the new target stage will be created as the last rung of this Program.

Do-more Logger

Another tool that is useful for debugging problems with getting MQTTPUB to function properly from the PLC's perspective is Do-more! Logger. Setting the system status bit **$EnableMsgDump** (ST36) ON will cause the MQTTPUB instruction to automatically echo the MQTT communication traffic between the Do-more! CPU and the MQTT Broker to the Do-more! Logger. Do-more! Logger will capture and display this conversation which allows to programmer to see what changes need to be made to correct any errors.

So between the High Word (:W1) of the Extended Error Information value telling you which entry number failed, and the Low Word of the Extended Error Information value telling you why that entry failed, and the value in **$LastProtoError** (DST38) giving you the MQTT Broker's error value, and Do-more! Logger displaying the communication traffic, you should be able to resolve any fault conditions.

MQTTSUB - IoT Subscribe to MQTT Topics

NOTE: *This instruction can only be used with a BRX MPU or the Do-more! Simulator.*

The IoT Subscribe to MQTT Topics (MQTTSUB) instruction is used to subscribe to Topics in an MQTT Broker. An MQTTSUB instruction can contain up to 50 subscriptions for a Broker. After a client successfully subscribes to one or more Topics, as long as the instruction remains enabled, it will receive every published message matching the Topic of the subscription.

When the MQTTSUB instruction is disabled it will automatically attempt to unsubscribe from any Topics to which it is currently subscribed.

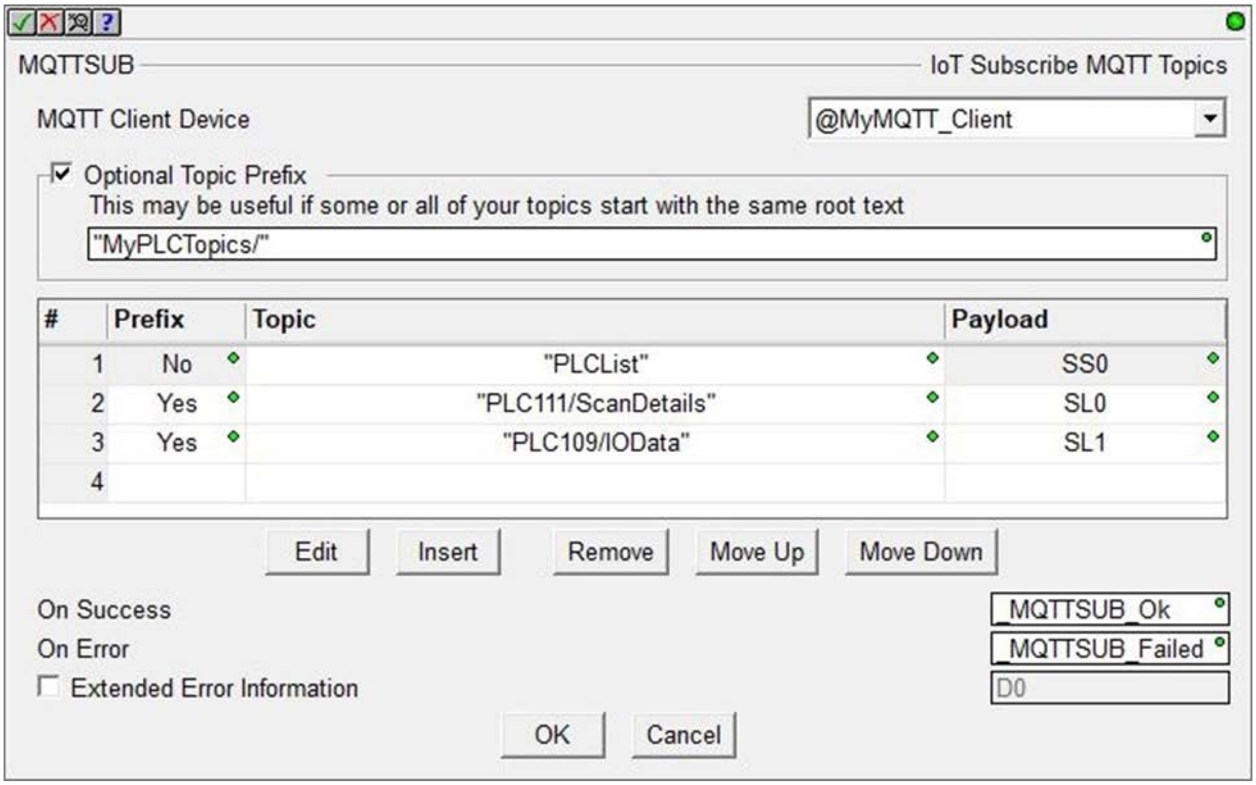
Quality of Service

Quality of Service (QoS) is a major feature of MQTT in that it makes communication in unreliable networks much easier because the protocol handles retransmission and guarantees the delivery of the message regardless of the unreliability of the underlying transport. The QoS level is an agreement between sender and receiver of a message regarding the guarantee of delivering a message.

There are 3 QoS levels in MQTT: At most once (0), At least once (1), and Exactly once (2). **The MQTTSUB instruction subscribes to all messages with a QoS of 0 (at most once)**, which means that a subscription message from the Broker will be delivered only once; the Broker will not continue retrying the transmission until it gets an acknowledgment message from the receiver.

NOTE: *Although RUN-mode edits are allowed on existing MQTTSUB instructions, doing so will likely cause a "Duplicate MQTTSUB topic ..." warning message as the PLC works through resubscribing the Topics in the message list. The re-subscribes work properly after the RUN-mode edit; this is just a warning caused by a temporary situation. You can clear this warning message on the System Status tab of the System Information utility.*

MQTT, continued



MQTTSUB — IoT Subscribe MQTT Topics

MQTT Client Device: @MyMQTT_Client

☑ Optional Topic Prefix

This may be useful if some or all of your topics start with the same root text

"MyPLCTopics/"

#	Prefix	Topic	Payload
1	No	"PLCList"	SS0
2	Yes	"PLC111/ScanDetails"	SL0
3	Yes	"PLC109/IOData"	SL1
4			

On Success: _MQTTSUB_Ok

On Error: _MQTTSUB_Failed

☐ Extended Error Information: D0

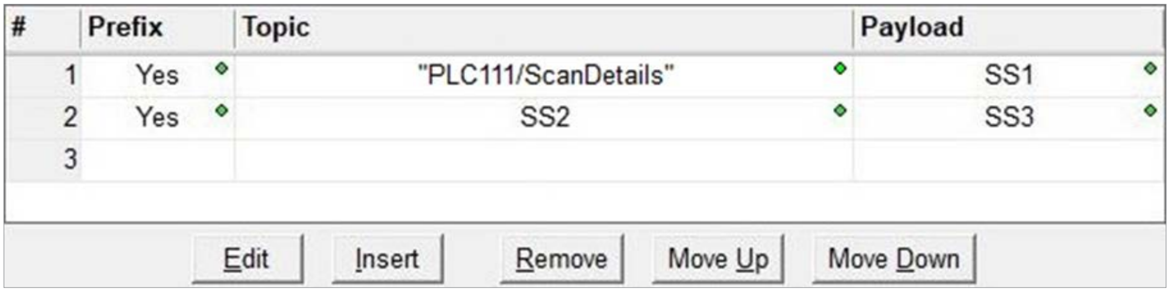
MQTT Client Device selects which of the preconfigured MQTT Client devices this instruction will connect to.

Click **create device** to open the Add Device dialog of the System Configuration where a new MQTT Client device can be created.

***NOTE:** Each MQTT Client device can only provide a subscription space for 100 concurrent topics spread across a maximum of 10 active MQTTSUB instructions; for example, 2 enabled instructions with 50 Topics each, or 10 enabled instructions with 10 Topics each. To subscribe to more than 100 Topics, create multiple MQTT Client devices to the same MQTT Broker.*

The next section contains a list of up to 50 Topics that get subscribed to from the selected MQTT Broker.

Enabling the **Optional Topic Prefix** allows Topics to use repeated text. If there are multiple Topics that begin with the same text, you can put the repeated text in this Optional Topic Prefix and then select that optional text when the Topic is created and added to the list. This can be any string literal (text enclosed with double quotes) of any user-defined or system-defined String. Remember, Topics are case sensitive.

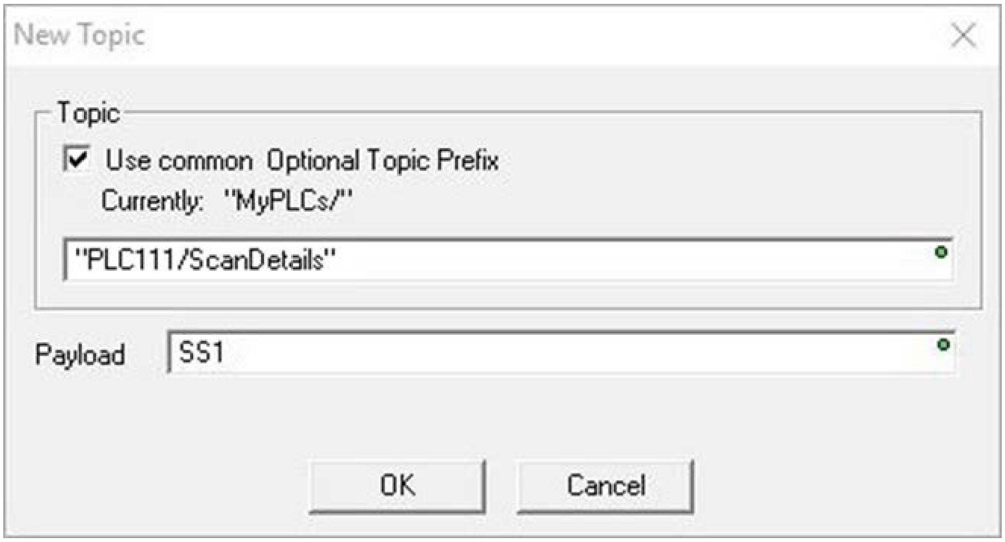


#	Prefix	Topic	Payload
1	Yes	"PLC111/ScanDetails"	SS1
2	Yes	SS2	SS3
3			

MQTT, continued

The buttons below the list are functions which manage the rows in the instruction: **Edit** opens the editor for the currently selected row, **Insert** adds an empty row before the currently selected row and opens the editor for this new row, **Remove** deletes the currently selected row from the table, and **Move Up** / **Move Down** shifts the currently selected row up one or down one row respectively in the table.

Clicking the **Edit** (on an existing Topic) or **Insert** button will open a sub-editor where the contents of the message are managed. The **Topic** group specifies the Topic to which to subscribe:



New Topic

Topic

☑ Use common Optional Topic Prefix

Currently: "MyPLCs/"

"PLC111/ScanDetails"

Payload SS1

If an **Optional Topic Prefix** was created on the main dialog, you can enable the **Use Common Optional Topic Prefix** to prepend the prefix text to the Topic text to construct the full name of the Topic.

The **Topic** itself is a simple UTF-8 string, consisting of one or more levels, which are separated by a forward slash (aka the topic level separator). A Topic is case-sensitive. Each Topic must have at least 1 character to be valid and it can contain spaces. A leading forward slash is not recommended. Published Topics cannot start with $. MQTTSUB does not support Wildcards (+ or #) in Topics. The Topic can be entered as a String literal (text within double quotes) up to 128 characters, or a user-defined String or a system-defined String element.

The **Payload** is the memory location in the PLC where the data from the subscribed Topic will be stored. MQTT protocol itself is data-agnostic, meaning the format of the data totally depends on the how the data was published to the Broker. It's completely up to the publisher if it wants to send binary data or textual data. The only requirement for the instruction is that the data be stored in a String element. Use the Convert String to Integer (STR2INT) or Convert String to Real (STR2REAL) instructions to convert the contents of the Strings to numbers if needed. Use the Get Bytes Out of a String (STRGETB) instruction to extract raw bytes of data into a byte buffer.

***NOTE:** The maximum length of the Topic (including the Optional Topic Prefix if selected) plus the Payload is 1024 characters.*

When this instruction is first enabled the On Success and On Error indicators will be turned OFF. Then the instruction will process each subscribe entry in the list. The On Success will be turned ON if at least one of the Topics in the instruction was successfully subscribed to. If none of the Topics were successfully subscribed the On Error will be turned ON.

MQTT, continued

When this instruction is disabled, it will process each entry in the list of Topics and attempt to unsubscribe each of them. The On Success will be turned ON if at least one of the Topics in the instruction was successfully unsubscribed. If none of the Topics were successfully unsubscribed the On Error will be turned ON.

On Success – select any writable bit location.

On Error – select any writable bit location.

The **Extended Error Information** is a DWord location that contains a combination value. Using the :SD cast here provides a good method for easily separating the high / low values. For example, V100:SD would place the high word in V101 and the low Word in V100).

The high Word is the entry number of the first Topic that failed. There may have been multiple failures, but only the entry number of the first failure will be listed in the Return Code. That error must be corrected before failures for any later entry numbers will be shown.

The low Word contains the error code itself from the following table:

MQTT Error Codes

Value in Low Word	Error	Description
0	No Error	
57	Unexpected MQTT Response	The response from the MQTT Broker was not properly formed.
58	MQTT Broker Rejected	The MQTT Broker refused the Topic; could be a bad username or password, or a security violation on the Broker itself.
59	Out of Resources (MQTTSUB only)	There are more than 10 MQTTSUB instructions using the same MQTT Client device, or there are more than 100 Topics using the same MQTT Client device.
60	Invalid Topic	This typically means the Topic is empty.
61	Duplicate Topic	The Topic is already subscribed to in a different MQTTSUB instruction.

The last error code reported by the MQTT Broker will be in **$LastProtoError** (DST38).

Do-more Logger

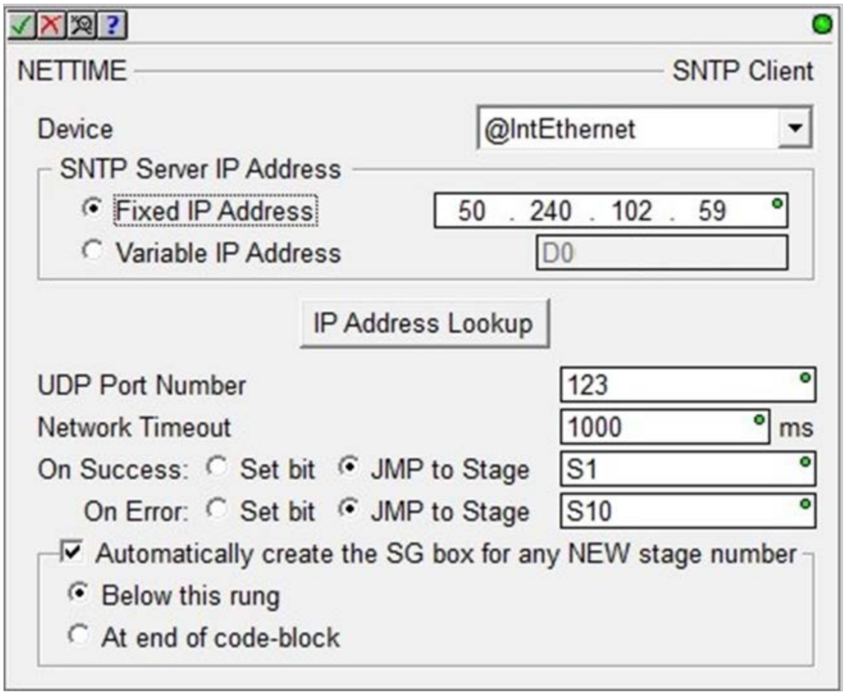
Another tool that is useful tool for debugging problems with getting MQTTSUB to function properly from the PLC's perspective is Do-more! Logger. Setting the system status bit **$EnableMsgDump** (ST36) ON will cause the MQTTSUB instruction to automatically echo the MQTT communication traffic between the Do-more! CPU and the MQTT Broker to the Do-more! Logger. Do-more! Logger will capture and display this conversation which allows to programmer to see what changes need to be made to correct any errors.

So between the High Word (:W1) of the Extended Error Information value telling you which entry number failed, and the Low Word of the Extended Error Information value telling you why that entry failed, and the value in **$LastProtoError** (DST38) giving you the MQTT Broker's error value, and Do-more! Logger displaying the communication traffic, you should be able to resolve any fault conditions.

NETTIME (SNTP Client)

The SNTP Client (NETTIME) instruction is used to retrieve clock and calendar information from a Time Server using the Simple Network Time Protocol (SNTP). This instruction can use either an NTP Server or an SNTP Server as the time server. After the date and time have been successfully read from the time server, the retrieved values will be used to recalculate the local time ($Now) and set the PC's real-time clock to this calculated value. This instruction can be used in conjunction with TimeSync operation to synchronize the real-time clocks in multiple Do-more! PLCs with the date and time of a master PLC.

The local time ($Now) is calculated by applying a local Time Zone offset value ($TimeZone, DST384), and optionally adjusting for Daylight Saving Time ($SummerTime, ST768) to the value retrieved via the NETTIME instruction.



NETTIME — SNTP Client

Device: @IntEthernet

SNTP Server IP Address

● Fixed IP Address: 50 . 240 . 102 . 59

○ Variable IP Address: D0

UDP Port Number: 123

Network Timeout: 1000 ms

On Success: ○ Set bit ● JMP to Stage: S1

On Error: ○ Set bit ● JMP to Stage: S10

☑ Automatically create the SG box for any NEW stage number

● Below this rung

○ At end of code-block

***NOTE:** Use the F9 key to open the Default Element Selection Tool (the Element Picker or the Element Browser) or use the Down-Arrow key (Auto-Complete) on any parameter field to see a complete list of the memory locations that are valid for that parameter of the instruction.*

Parameters:

Device selects which of the Ethernet Devices to use. For more information on configuring devices go to the Device Configuration Section under System Configuration.

@IntEthernet will use the on-board Ethernet port on the Do-more! CPU.

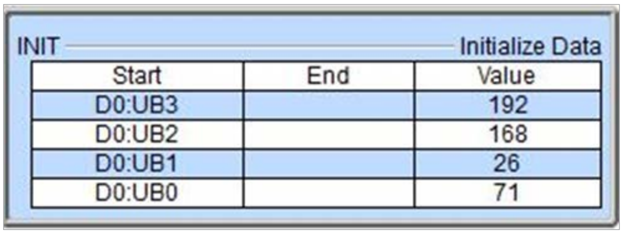
'No devices available' indicates that there are no Devices that can execute this instruction, i.e. the CPU does not have an on-board Ethernet port.

SNTP Server IP Address is the TCP/IP Address of the SNTP server to contact for the clock and calendar information. The default SNTP Server IP Address of 12.69.41.165 is an SNTP server (ntp2.hosteng.com) at Host Engineering. While it is reasonable to use the default time server for testing purposes, be aware that the uptime of the default time server can not be guaranteed. It is recommended that you install your own time server or select a publicly available time server for long-term use.

NETTIME (SNTP Client), continued

Fixed Address: select this option to enter the static TCP/IP address assigned to the SNTP server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots. Invoking the Element Browser (F9) for this field will bring up the IP Address Lookup utility that can find the IP address for a given SNTP Server name.

Variable IP Address: select this option if the IP address resides in a numeric memory location in the CPU. This can be any readable DWord numeric location. This allows the IP address to be changed at runtime. Each octet of the IP address is stored in one byte of the Variable Address location. In the example below the Initialize Data instruction will use the :UB operator to place the 4 octet values of IP address 192.168.26.71 into the 4 Bytes of the DWord location D0 in the proper order.

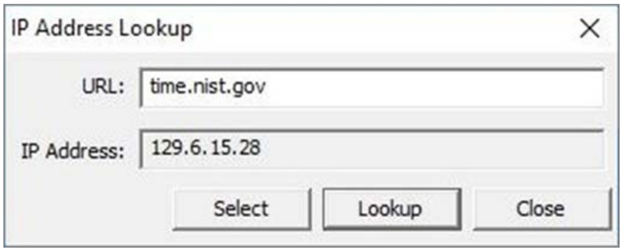


INIT — Initialize Data

Start	End	Value
D0:UB3		192
D0:UB2		168
D0:UB1		26
D0:UB0		71

The ‘IP Address’ format selection in a Data View can be used to see the IP address stored in the DWord location in the traditional dot-decimal notation (000.000.000.000).

Clicking the **IP Address Lookup** button will open a dialog where the name of an SNTP server can be entered and Do-more! Designer will attempt to find the TCP/IP address assigned to that name using the Domain Name System protocol (DNS).



IP Address Lookup

URL: time.nist.gov

IP Address: 129.6.15.28

The server name needs to consist only of the server name, not a fully qualified URL. For example, the Name field should contain only the text “time.nist.gov”, not “https://time.nist.gov”.

UDP Port Number is the UDP/IP port number the SNTP Server uses to communicate. The SNTP protocol uses UDP Port Number 123 by default. This value can be any constant in the range of 1 to 65535, or any readable numeric location containing a value in that range.

Network Timeout specifies the maximum amount of time in milliseconds to allow the instruction to wait before signaling a completion. This value can be any constant in the range of 1 to 65535.

The **On Success** and **On Error** parameters specify what action to perform when this instruction completes. You do not have to use the same type of selection for both On Success and On Error.

If the **Set Bit** selection is used for either **On Success** or **On Error**, the specified BIT location will be SET OFF when the instruction is first enabled and will remain OFF until the instruction completes. Once complete, the appropriate Success or Error bit location will be set ON. The specified Bit location is enabled with a SET (Latch) operation meaning that it will remain ON even if the input logic for the instruction goes OFF.

NETTIME (SNTP Client), continued

If the **JMP to Stage** selection is used for either **On Success** or **On Error** the target Stage must be in the same Program code block as this instruction; you cannot specify a target Stage that exists in a different Program code block. When the operation finishes, the target Stage will be enabled the same way as a standalone Jump to Stage (JMP) instruction would do it. The JMP to Stage option will only be available if this instruction is placed in a Program code block.

On Success - selects which of the following actions to perform if the operation is successful:

SET Bit - enable this selection then specify any writable bit location.

JMP to Stage - enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error - selects which of the following actions to perform if the Close Device operation is unsuccessful:

SET Bit - enable this selection then specify writable bit location.

JMP to Stage - enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Automatically create the SG box for any NEW stage number - if either the **On Success** or **On Error** selections are set to **JMP to Stage**, this option will be enabled which will automatically create any target stage that does not already exist.

Below this rung - the new target stage will be created on a new rung following this instruction.

At end of code block - the new target stage will be created as the last rung of this Program.

TCP/IP Raw Data

Do-more! PLCs can be used to create custom protocols for various types of devices. For devices that utilize TCP/IP based protocols, this section of the manual will give you direction on how to get started. You will need to have a thorough understanding of the protocol that you are attempting to write. Showing how to fully implement a protocol is beyond the scope of this manual.

Basic steps to implement a Server (Slave) protocol:

1. Create a TCP/IP Server Raw socket device. Enter the correct port number that your Server is transmitting on. See Do-more! Designer help topic DMD0248.
2. Put a TCPLISTEN command into $MAIN that will start the Program to handle the communications with your TCP Server. See Do-more! Designer help topic DMD0072.
3. In the Program block above, put the appropriate STREAMIN and STREAMOUT instructions for communications. See Do-more! Designer help topics DMD0303 and DMD0304.

Basic steps to implement a Client (Master) protocol:

1. Create a TCP/IP Client Raw socket device.
2. When $YourTCPClientNameHere.Connected is true, execute an OPENTCP command. See Do-more! Designer help topic DMD0069.
3. Execute a STREAMOUT command with your data. See Do-more! Designer help topic DMD0304.

UDP/IP Raw Data

Do-more! PLCs can be used to create custom protocols for various types of devices. For devices that utilize UDP/IP based protocols, this section of the manual will give you direction on how to get started. You will need to have a thorough understanding of the protocol that you are attempting to write. Showing how to fully implement a protocol is beyond the scope of this manual.

Basic steps to implement an UDP/IP protocol:

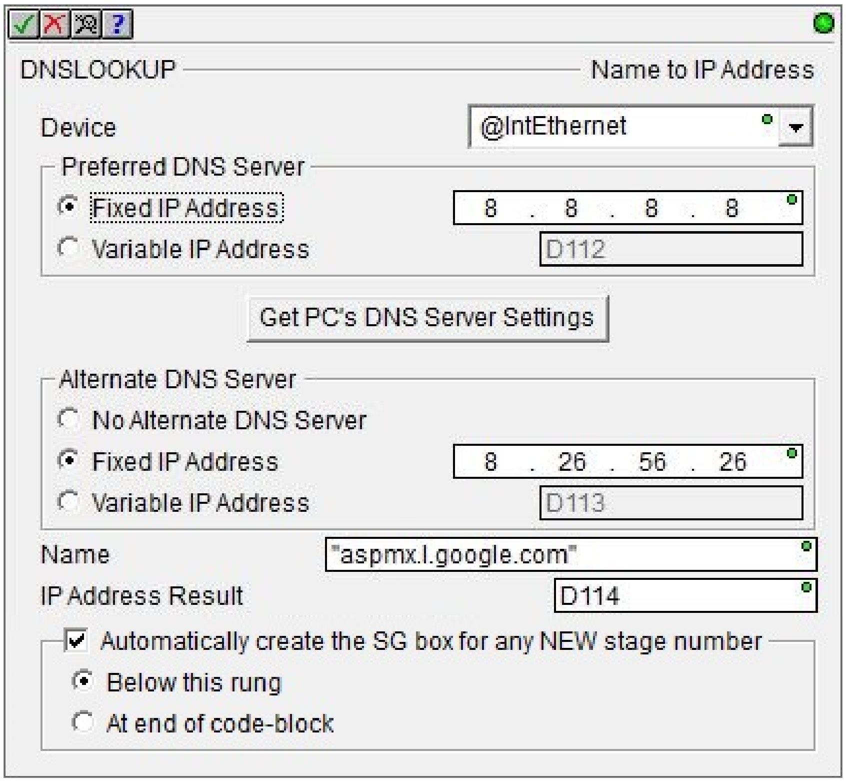
1. Create an UDP/IP Raw socket device. Enter the correct port number that your Server is transmitting or listening on. See Do-more! Designer help topic DMD0248.
2. When $YourUDPClientNameHere.PacketAvailable is true, execute a PACKETIN command. See Do-more! Designer help topic DMD0301.
3. When you want to send data to the server, execute a PACKETOUT. See Do-more! Designer help topics DMD0302.

DNS Lookup – Name to IP Address

The Name to IP Address (DNSLOOKUP) instruction is used to find the IP Address assigned to the named device using the Domain Name System protocol (DNS). This instruction uses the on-board Ethernet port and requires that the TCP/IP network be setup so the CPU has a functional connection to a DNS Server.

The Name needs to consist only of the server name, not a fully qualified URL. For example, the Name field should contain only the text “mail.google.com”, not “https://mail.google.com”. The timeout value (including retries) for the DNS Lookup operation is fixed in the DNS client at 126 seconds.

If the DNS Lookup operation is successful, the IP address will be stored in a DWord location. Use the “IP Address” format in a Data View to see the returned IP Address in the traditional dot-decimal notation (000.000.000.000).



DNSLOOKUP — Name to IP Address

Device: @IntEthernet

Preferred DNS Server
● Fixed IP Address: 8 . 8 . 8 . 8
○ Variable IP Address: D112

Get PC's DNS Server Settings

Alternate DNS Server
○ No Alternate DNS Server
● Fixed IP Address: 8 . 26 . 56 . 26
○ Variable IP Address: D113

Name: "aspmx.l.google.com"
IP Address Result: D114

☑ Automatically create the SG box for any NEW stage number
● Below this rung
○ At end of code-block

DNS Lookup, continued

***NOTE:** Use the F9 key to open the Default Element Selection Tool (the Element Picker or the Element Browser) or use the Down-Arrow key (Auto-Complete) on any parameter field to see a complete list of the memory locations that are valid for that parameter of the instruction.*

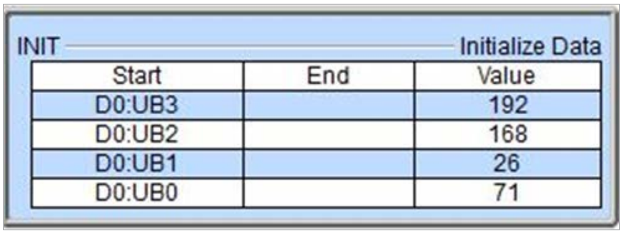
Parameters:

Device selects which of the Ethernet Devices to use. Currently this instruction can only use @IntEthernet which is the on-board Ethernet port on the Do-more! CPU. If the only entry in the list is no devices available this indicates that there are no Devices that can execute this instruction, most likely cause is the CPU does not have an on-board Ethernet port.

Preferred DNS Server: the TCP/IP Address of the first DNS server to contact for resolving the Name to an IP Address. The default value is 8.8.8.8 (decimal 34744072) which is Google's public DNS Server.

Fixed Address: the IP Address assigned to the DNS Server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address: the IP Address resides in a memory location in the CPU. This allows the IP Address to be changed at runtime. This can be any readable DWord numeric location. Each octet of the IP Address is stored in one byte of the Variable Address location. In the example below the Initialize Data instruction will use the :UB operator to place the 4 octet values of IP Address 192.168.26.71 into the 4 Bytes of the DWord location D0 in the proper order.


INIT — Initialize Data

Start	End	Value
D0:UB3		192
D0:UB2		168
D0:UB1		26
D0:UB0		71

The 'IP Address' format selection in a Data View can be used to see the IP address stored in the DWord location in the traditional dot-decimal notation (000.000.000.000).

Click the **Get PC's DNS Server Settings** button to retrieve the Preferred DNS Server and the Alternate DNS Server IP Addresses from the network configuration of the PC running Do-more! Designer and use those IP Addresses in this instruction.

Alternate DNS Server: the TCP/IP Address of an alternate DNS server to contact if the preferred DNS server cannot resolve the Name to an IP Address. The default value is the Secondary DNS Server IP address from the network setup on the computer that is running Do-more! Designer.

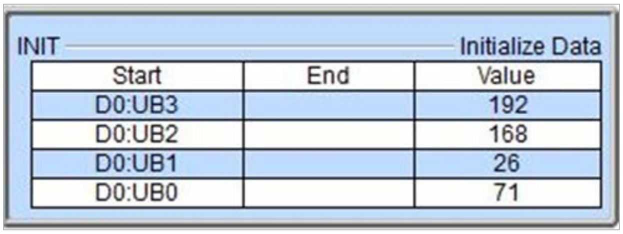
No Alternate DNS Server: select this option to only use the Preferred DNS Server to resolve the Name.

Fixed Address: the IP Address assigned to the Alternate DNS Server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Variable IP Address: the IP Address resides in a memory location in the CPU. This allows the IP Address to be changed at runtime. This can be any readable DWord numeric location. Each octet of the IP Address is stored in one byte of the Variable Address location. In the example below

DNS Lookup, continued

the Initialize Data instruction will use the :UB operator to place the 4 octet values of IP Address 192.168.26.71 into the 4 Bytes of the DWord location D0 in the proper order.



INIT — Initialize Data

Start	End	Value
D0:UB3		192
D0:UB2		168
D0:UB1		26
D0:UB0		71

The 'IP Address' format selection in a Data View can be used to see the IP Address stored in the DWord location in the traditional dot-decimal notation (000.000.000.000).

Name: specifies the name to resolve to an IP Address. This can be a string literal (text within double quotes), or any system-defined Short Strings, or system-defined Long Strings, or user-defined Strings. The Name is not a fully qualified URL like "https://mail.google.com"; use only the "mail.google.com".

IP Address Result: designates a location to store the IP Address if a DNS server can successfully resolve the Name. This can be any writable numeric location.

The **On Success** and **On Error** parameters specify what action to perform when this instruction completes. You do not have to use the same type of selection for both On Success and On Error.

If the **Set Bit** selection is used for either **On Success** or **On Error**, the specified BIT location will be SET OFF when the instruction is first enabled and will remain OFF until the instruction completes. Once complete, the appropriate Success or Error bit location will be set ON. The specified Bit location is enabled with a SET (Latch) operation meaning that it will remain ON even if the input logic for the instruction goes OFF.

If the **JMP to Stage** selection is used for either **On Success** or **On Error** the target Stage must be in the same Program code block as this instruction; you cannot specify a target Stage that exists in a different Program code block. When the operation finishes, the target Stage will be enabled the same way as a standalone Jump to Stage (JMP) instruction would do it. The JMP to Stage option will only be available if this instruction is placed in a Program code block.

On Success - selects which of the following actions to perform if the operation is successful:

SET Bit - enable this selection then specify any writable bit location.

JMP to Stage - enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error - selects which of the following actions to perform if the Close Device operation is unsuccessful:

SET Bit - enable this selection then specify writable bit location.

JMP to Stage - enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Automatically create the SG box for any NEW stage number - if either the **On Success** or **On Error** selections are set to **JMP to Stage**, this option will be enabled which will automatically create any target stage that does not already exist.

Below this rung - the new target stage will be created on a new rung following this instruction.

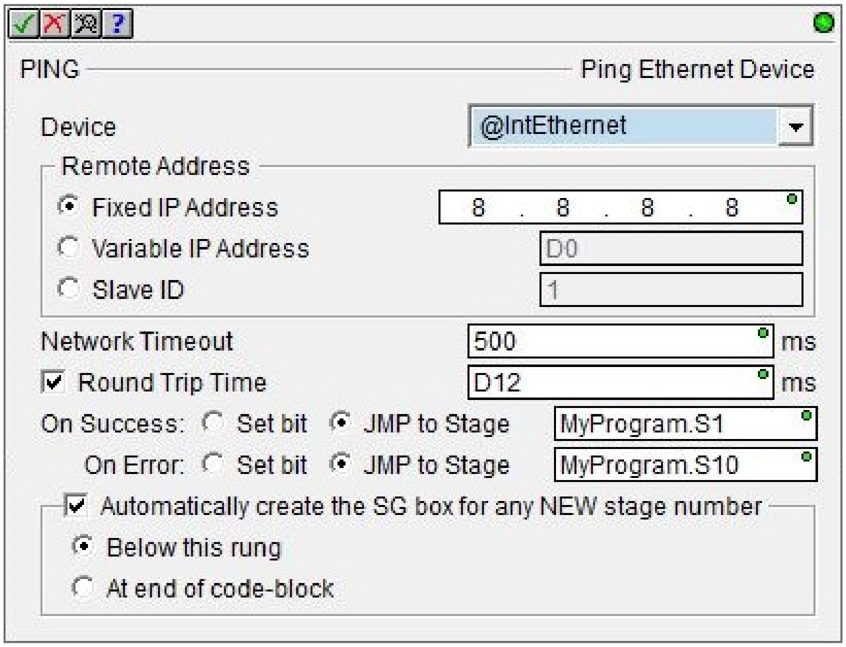
At end of code block - the new target stage will be created as the last rung of this Program.

Ping – Ping Ethernet Device

The Ping Ethernet Device (PING) instruction uses the Internet Control Message protocol (ICMP) to determine if the device with the specified IP Address is available on the network, and optionally to report the amount of time required to get a response from that device.

The amount of time required to complete the PING operation can optionally be stored in a numeric location. The round trip time begins accumulating when the PING packet is actually sent and ends with either a response from the remote device or a timeout. If the PING operation is successful, the Round Trip Time will be total amount of time (in milliseconds) that elapsed after the PING packet was sent and the response packet was received. If the PING operation failed, the Round Trip Time will be the total mount of time (in milliseconds) that elapsed while waiting for the timeout to occur.

***NOTE:** Care must be taken to NOT run this instruction at an uncontrolled pace. If this happens the network hardware could perceive it as a Ping Flood (where ICMP packets are sent as fast as possible without waiting for replies). This could lead to a DENIAL-OF-SERVICE for legitimate PING operations.*



PING — Ping Ethernet Device

Device: @IntEthernet

Remote Address
- ● Fixed IP Address: 8 . 8 . 8 . 8
- ○ Variable IP Address: D0
- ○ Slave ID: 1

Network Timeout: 500 ms

☑ Round Trip Time: D12 ms

On Success: ○ Set bit ● JMP to Stage: MyProgram.S1

On Error: ○ Set bit ● JMP to Stage: MyProgram.S10

☑ Automatically create the SG box for any NEW stage number
- ● Below this rung
- ○ At end of code-block

***NOTE:** Use the F9 key to open the Default Element Selection Tool (the Element Picker or the Element Browser) or use the Down-Arrow key (Auto-Complete) on any parameter field to see a complete list of the memory locations that are valid for that parameter of the instruction.*

Parameters:

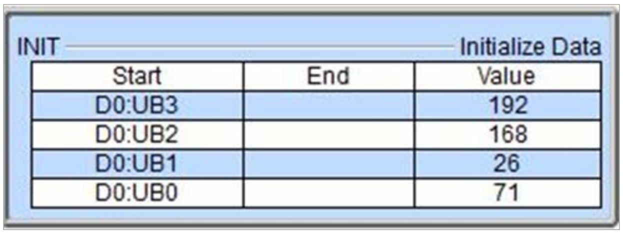
Device selects which of the Ethernet Devices to use. Currently this instruction can only use @IntEthernet which is the on-board Ethernet port on the Do-more! CPU. If the only entry in the list is no devices available this indicates that there are no Devices that can execute this instruction, most likely cause is the CPU does not have an on-board Ethernet port.

Remote Address: the TCP/IP Address of the remote network device to attempt to locate.

Fixed Address: the IP Address assigned to the DNS Server. IP addresses are canonically represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots.

Ping, continued

Variable IP Address: the IP Address resides in a memory location in the CPU. This allows the IP Address to be changed at runtime. This can be any readable DWord numeric location. Each octet of the IP Address is stored in one byte of the Variable Address location. In the example below the Initialize Data instruction will use the :UB operator to place the 4 octet values of IP Address 192.168.26.71 into the 4 Bytes of the DWord location D0 in the proper order.



INIT — Initialize Data

Start	End	Value
D0:UB3		192
D0:UB2		168
D0:UB1		26
D0:UB0		71

The 'IP Address' format selection in a Data View can be used to see the IP address stored in the DWord location in the traditional dot-decimal notation (000.000.000.000).

Network Timeout: specifies the amount of time (in milliseconds) to wait on a response from the network device. This can be any constant value between 1 and 65535, or any readable numeric location containing a value in that range.

Round Trip Time (optional): if enabled, designates a location to store the total amount of time (in milliseconds) that elapsed during the Ping operation. This can be any writable numeric location.

The **On Success** and **On Error** parameters specify what action to perform when this instruction completes. You do not have to use the same type of selection for both On Success and On Error.

If the **Set Bit** selection is used for either **On Success** or **On Error**, the specified BIT location will be SET OFF when the instruction is first enabled and will remain OFF until the instruction completes. Once complete, the appropriate Success or Error bit location will be set ON. The specified Bit location is enabled with a SET (Latch) operation meaning that it will remain ON even if the input logic for the instruction goes OFF.

If the **JMP to Stage** selection is used for either **On Success** or **On Error** the target Stage must be in the same Program code block as this instruction; you cannot specify a target Stage that exists in a different Program code block. When the operation finishes, the target Stage will be enabled the same way as a standalone Jump to Stage (JMP) instruction would do it. The JMP to Stage option will only be available if this instruction is placed in a Program code block.

On Success - selects which of the following actions to perform if the operation is successful:

SET Bit - enable this selection then specify any writable bit location.

JMP to Stage - enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

On Error - selects which of the following actions to perform if the Close Device operation is unsuccessful:

SET Bit - enable this selection then specify writable bit location.

JMP to Stage - enable this selection then specify any Stage number from S0 to S127 in the current Program code block.

Automatically create the SG box for any NEW stage number - if either the **On Success** or **On Error** selections are set to **JMP to Stage**, this option will be enabled which will automatically create any target stage that does not already exist.

Below this rung - the new target stage will be created on a new rung following this instruction.

At end of code block - the new target stage will be created as the last rung of this Program.