INDUSTRIAL CONTROL COMMUNICATIONS, INC.

Instruction Manual

ETH-1000
Multiprotocol Ethernet / RS-485 Gateway

MILLENNIUM SERIES

March 1, 2010
ICC #10724

ETH-1000
User's Manual

Part Number 10724
Printed in U.S.A.

NOTICE TO USERS

Industrial Control Communications, Inc. reserves the right to make changes and improvements to its products without providing notice.

Industrial Control Communications, Inc. shall not be liable for technical or editorial omissions or mistakes in this manual, nor shall it be liable for incidental or consequential damages resulting from the use of information contained in this manual.

INDUSTRIAL CONTROL COMMUNICATIONS, INC.'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS. Life-support devices or systems are devices or systems intended to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs may always be present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

This user's manual may not cover all of the variations of interface applications, nor may it provide information on every possible contingency concerning installation, programming, operation, or maintenance.

The contents of this user's manual shall not become a part of or modify any prior agreement, commitment, or relationship between the customer and Industrial Control Communications, Inc. The sales contract contains the entire obligation of Industrial Control Communications, Inc. The warranty contained in the contract between the parties is the sole warranty of Industrial Control Communications, Inc., and any statements contained herein do not create new warranties or modify the existing warranty.

Any electrical or mechanical modifications to this equipment without prior written consent of Industrial Control Communications, Inc. will void all warranties and may void any UL/cUL listing or other safety certifications. Unauthorized modifications may also result in equipment damage or personal injury.

APPLICABLE FIRMWARE

Main Processor:
Modbus – BACnet Firmware Version 2.100
Metasys – Toshiba Firmware Version 2.100
Ethernet - Sullair Firmware Version 2.100

Coprocessor:
Ethernet Multiple V2.100

Usage Precautions

Operating Environment

- Please use the interface only when the ambient temperature of the environment into which the unit is installed is within the following specified temperature limits:

  Operation: -10 ~ +50°C (+14 ~ +122°F)
  Storage: -40 ~ +85°C (-40 ~ +185°F)

- Avoid installation locations that may be subjected to large shocks or vibrations.
- Avoid installation locations that may be subjected to rapid changes in temperature or humidity.

Installation and Wiring

- Proper ground connections are vital for both safety and signal reliability reasons. Ensure that all electrical equipment is properly grounded.
- Route all communication cables separate from high-voltage or noise-emitting cabling (such as ASD input/output power wiring).

TABLE OF CONTENTS

1. Introduction

Congratulations on your purchase of the ICC ETH-1000 Multiprotocol Ethernet Communications Gateway. This gateway allows information to be transferred seamlessly between various industrial Ethernet networks and one of several RS-485-based networks. In addition to the supported fieldbus protocols, the gateway hosts a USB interface for configuring the gateway via a PC.

Before using the gateway, please familiarize yourself with the product and be sure to thoroughly read the instructions and precautions contained in this manual. In addition, please make sure that this instruction manual is delivered to the end user of the gateway, and keep this instruction manual in a safe place for future reference or unit inspection.

For the latest information, support software and firmware releases, please visit http://www.iccdesigns.com.

Before continuing, please take a moment to ensure that you have received all materials shipped with your kit. These items are:

- ETH-1000 Gateway in plastic housing
- Documentation CD-ROM
- DIN rail adapter with two pre-mounted screws
- Four black rubber feet

Note that different gateway firmware versions may provide varying levels of support for the various protocols. When using this manual, therefore, always keep in mind that the firmware version indicated on your unit must be listed on page 2 for all documented aspects to apply.

This manual will primarily be concerned with the gateway's hardware specifications, installation, wiring, configuration and operational characteristics.

To maximize the abilities of your new gateway, a working familiarity with this manual will be required. This manual has been prepared for the gateway installer, user, and maintenance personnel. With this in mind, use this manual to develop a system familiarity before attempting to install or operate the gateway.

2. Features

Supported Protocols

The gateway currently provides support for the following Ethernet protocols:

- Modbus/TCP Client
- Modbus/TCP Server
- BACnet/IP Client
- BACnet/IP Server
- EtherNet/IP Client
- EtherNet/IP Server
- CSP (also known as PCCC or Allen Bradley Ethernet) Server
- CSP Client
- PROFInet IO
- Mitsubishi MELSEC Client

The gateway also currently provides support for the following RS-485 based fieldbus protocols:

- Modbus RTU Master
- Modbus RTU Slave
- Modbus RTU Sniffer
- BACnet MS/TP Client
- BACnet MS/TP Server
- Johnson Controls Metasys N2 Slave
- Toshiba ASD Protocol Master
- Sullair Supervisor Network Master

Supported Baud Rates

The gateway currently provides support for the following baud rates on the RS-485 port:

- 2400
- 4800
- 9600
- 19200
- 38400
- 57600
- 76800
- 115200

Note that not all protocols support every baud rate listed above. Refer to section 11 for more information.

Field-Upgradeable

As new firmware becomes available, the gateway can be upgraded in the field by the end-user. Refer to section 8.1 for more information.

USB Interface

The gateway can be connected to a PC via a USB mini type-B cable. This simultaneously supplies power while providing the ability to configure the gateway, monitor data, and update firmware on the device using the ICC Gateway Configuration Utility. Refer to section 8.1 for more information.

Power over Ethernet (PoE) Enabled

The gateway can be externally powered according to the PoE specification (IEEE 802.3af). Refer to section 5.2 for more information.

Alarm Evaluation with Email Notification

The gateway can autonomously monitor any database address and send emails to up to four recipients when a certain condition is detected. Alarm conditions have both value and time constraints, and can be configured to retrigger at a fixed interval as long as the alarm condition continues to be satisfied. Twenty individually-configurable alarms are available. Refer to section 10.15 for more information.

Embedded Web Server

The gateway supports real-time web browser-based interaction via an Adobe® Flash Player plug-in. This includes support for configuration, database interaction, and a dashboard GUI with multiple windows, each of which can be configured to display data in a variety of meter/graph/gauge formats. Refer to section 10 for more information.

User-Configurable Network Timeouts

The gateway can be configured to perform a specific set of actions when network communications are lost. This allows each address in the database to have its own unique “fail-safe” condition in the event of network interruption (support for this feature varies depending on the protocol). Refer to section 10.17 for more information.

Flexible Mounting Capabilities

The gateway includes all hardware for desktop, panel/wall and DIN-rail mounting capabilities. Refer to section 6.1 for more information.

3. Gateway Concepts

The ETH-1000 is a member of the Millennium Series communication gateways. Members of this family are designed to provide a uniform interface, configuration and application experience. This commonality reduces the user's learning curve, reducing commissioning time while simplifying support. All Millennium Series gateways are configured using the ICC Gateway Configuration Utility. The ETH-1000 provides simultaneous support for many different communication protocols, allowing complex interchanges of data between otherwise incompatible networks.

The heart of the Millennium Series concept is its internal database. The database is a 4 KB, byte-wise addressable data array. The database allows data to be routed from any supported network to any other supported network. Data may be stored into the database in either big-endian style (meaning that if a 16-bit or 32-bit value is stored in the database, the most significant byte will start at the lowest address) or little-endian style (meaning that if a 16-bit or 32-bit value is stored in the database, the least significant byte will start at the lowest address).

The other fundamental aspect of the Millennium Series is the concept of a configurable "service object". A service object is used for any master/client protocol to describe what service (read or write) is to be requested on the network. The gateway will cycle through the defined service objects in a round-robin fashion; however, the gateway does implement a "write first" approach. This means that the gateway will perform any outstanding write services before resuming its round-robin, read request cycle.

Additionally, the database and service objects provide the added benefit of "data mirroring", whereby current copies of data values (populated by a service object) are maintained locally within the gateway itself. This greatly reduces the request-to-response latency times on the various networks, as requests (read or write) can be entirely serviced locally, thereby eliminating the time required to execute a secondary transaction on a different network.

Regardless of their network representation, all data values are stored in the gateway's internal database as integer values (either 8-, 16- or 32-bits in length, depending on the protocol and/or object configuration). This means that even if a network variable is accessed by the gateway as a 32-bit floating-point number, this native representation will always be converted to an equivalent integer representation prior to being stored in the database. Once in the database, this value will then be accessible to the network operating on the other port of the gateway, which may then impose its own conversion process on the data. A port's conversion may be implicit (e.g. all Modbus holding registers are interpreted by the protocol as 16-bit unsigned integers) or explicit (as configured in a BACnet service object).

In order to facilitate the free scaling and conversion of native data values, a user-configurable "multiplier" and "data type" exist for some network configurations. All network values are scaled by a multiplier prior to being stored into the database or after being retrieved from the database. The data type is used to determine

how many bytes are allocated for the value in the database and whether or not to interpret the number as signed or unsigned upon retrieval from the database.

A typical use of the multiplier feature is to preserve the fractional components of a network value for insertion into the database. For example, if the floating-point value “3.19” is read by the gateway from a remote BACnet device, then we could use a multiplier value of 0.01 to preserve all of the significant digits of this value: the network representation (3.19) will be divided by the multiplier value (0.01) to obtain a resultant value of 319, which will then be inserted into the database. Similarly, when a value in the database corresponding to a specific service object is changed (which therefore requires that this updated value be written to the associated remote device on the network), the service object’s multiplier value will first be multiplied by the database value in order to obtain the resultant network value. For example, if 3000 is written to the database at a location corresponding to a certain service object on the other port, and that service object’s multiplier value is 0.1, then the database value (3000) will be multiplied by the multiplier value (0.1) to obtain the resultant network value of 300.0, which will then be written to the network as a native floating point value.

An appropriate data type should be selected based on the range of the network data values. For example, if the value of an Analog Output on a remote BACnet device can vary from –500 to 500, a 16-bit signed data type should be used. If the value can only vary from 0 to 150, for example, an 8-bit unsigned data type may be used. Care must be taken so that a signed data type is selected if network data values can be negative. For example, if 0xFF is written to the database at a location corresponding to a service object with an 8-bit unsigned data type, the resultant network value will be 255_{10} (assuming a multiplier of 1). However, if 0xFF is written to the database at a location corresponding to a service object with an 8-bit signed data type, the resultant network value will be -1_{10} (again, assuming a multiplier of 1). It is also important to select a data type large enough to represent the network data values. For example, if a value of 257 is read by the gateway from a remote device and the data type corresponding to that service object is 8-bit unsigned, the value that actually will be stored is 1 (assuming a multiplier of 1). This is because the maximum value that can be stored in 8-bits is 255. Any value higher than this therefore results in overflow.

The Millennium Series gateways also provide a powerful data-monitoring feature that allows the user to view and edit the database in real time, as well as view the status of service objects via the ICC Gateway Configuration Utility’s Monitor tab when connected via USB to a PC. The ability to interact with the database is also available via the embedded web server.

When properly configured, the gateway will become essentially “transparent” on the networks, and the various network devices can engage in seamless dialogs with each other.

4. Precautions and Specifications

DANGER!

Rotating shafts and electrical equipment can be hazardous. Installation, operation, and maintenance of the gateway shall be performed by **Qualified Personnel** only.

Qualified Personnel shall be:

- Familiar with the construction and function of the gateway, the equipment being driven, and the hazards involved.
- Trained and authorized to safely clear faults, ground and tag circuits, energize and de-energize circuits in accordance with established safety practices.
- Trained in the proper care and use of protective equipment in accordance with established safety practices.

Installation of the gateway should conform to all applicable **National Electrical Code** (NEC) *Requirements For Electrical Installation*s, all regulations of the **Occupational Safety and Health Administration**, and any other applicable national, regional, or industry codes and standards.

DO NOT install, operate, perform maintenance, or dispose of this equipment until you have read and understood all of the following product warnings and user directions. Failure to do so may result in equipment damage, operator injury, or death.

4.1 Installation Precautions

DANGER!

- Avoid installation in areas where vibration, heat, humidity, dust, metal particles, or high levels of electrical noise (EMI) are present.
- Do not install the gateway where it may be exposed to flammable chemicals or gasses, water, solvents, or other fluids.
- Where applicable, always ground the gateway to prevent electrical shock to personnel and to help reduce electrical noise.

  Note: *Conduit is not an acceptable ground.*

- Follow all warnings and precautions and do not exceed equipment ratings.

4.2 Maintenance Precautions

DANGER!

- **Do Not** attempt to disassemble, modify, or repair the gateway. Contact your ICC sales representative for repair or service information.
- If the gateway should emit smoke or an unusual odor or sound, turn the power off immediately.
- The system should be inspected periodically for damaged or improperly functioning parts, cleanliness, and to determine that all connectors are tightened securely.

4.3 Inspection

Upon receipt, perform the following checks:

- Inspect the unit for shipping damage.
- Check for loose, broken, damaged or missing parts.

Report any discrepancies to your ICC sales representative.

4.4 Maintenance and Inspection Procedure

Preventive maintenance and inspection is required to maintain the gateway in its optimal condition, and to ensure a long operational lifetime. Depending on usage and operating conditions, perform a periodic inspection once every three to six months.

Inspection Points

- Check that there are no defects in any attached wire terminal crimp points. Visually check that the crimp points are not scarred by overheating.
- Visually check all wiring and cables for damage. Replace as necessary.
- Clean off any accumulated dust and dirt.
- If use of the interface is discontinued for extended periods of time, apply power at least once every two years and confirm that the unit still functions properly.
- Do not perform hi-pot tests on the interface, as they may damage the unit.

Please pay close attention to all periodic inspection points and maintain a good operating environment.

4.5 Storage

- Store the device in a well ventilated location (in its shipping carton, if possible).
- Avoid storage locations with extreme temperatures, high humidity, dust, or metal particles.

4.6 Warranty

This gateway is covered under warranty by ICC, Inc. for a period of 12 months from the date of installation, but not to exceed 18 months from the date of shipment from the factory. For further warranty or service information, please contact Industrial Control Communications, Inc. or your local distributor.

4.7 Disposal

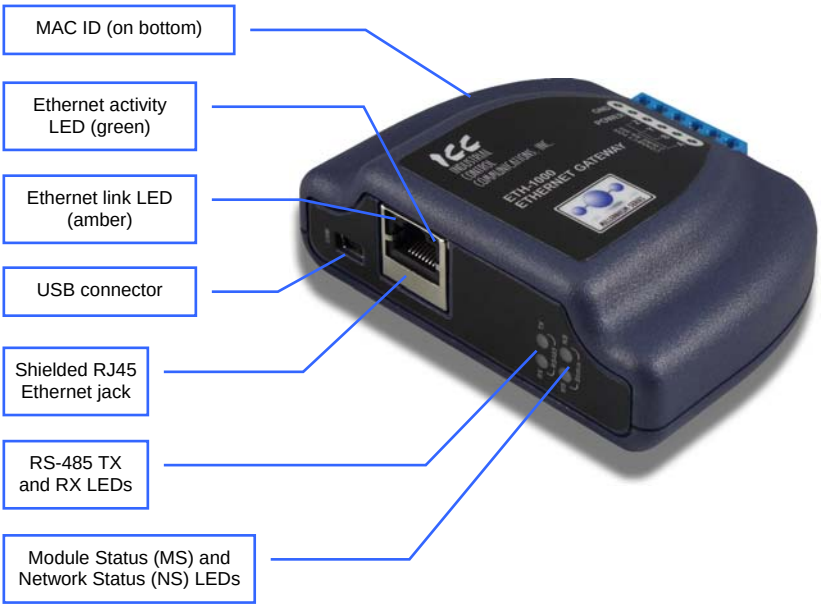
- Contact the local or state environmental agency in your area for details on the proper disposal of electrical components and packaging.
- Do not dispose of the unit via incineration.

4.8 Environmental Specifications

Item	Specification
Operating Environment	Indoors, less than 1000m above sea level, do not expose to direct sunlight or corrosive / explosive gasses
Operating Temperature	-10 ~ +50°C (+14 ~ +122°F)
Storage Temperature	-40 ~ +85°C (-40 ~ +185°F)
Relative Humidity	20% ~ 90% (without condensation)
Vibration	5.9m/s^2 {0.6G} or less (10 ~ 55Hz)
Grounding	Non-isolated, referenced to power ground
Cooling Method	Self-cooled

This device is lead-free / RoHS-compliant.

5. Gateway Overview



Gateway Overview (Front)

Gateway Overview (Back)

5.1 Power Supply Electrical Interface

When the gateway is not plugged into a PC via the USB cable, it must be powered by an external power source or via Power over Ethernet (PoE: refer to section 5.2). When using an external power source connected to the gateway's power and ground terminals, ensure that the power supply adheres to the following specifications:

Voltage rating ..... 7 - 24VDC
Minimum Current rating..... 150mA (@24VDC)

- ICC offers an optional 120VAC/12VDC power supply (ICC part number 10755) that can be used to power the gateway from a standard wall outlet.
- The power supply must be connected to the gateway's terminal block at terminals TB:5 (POWER) and TB:6 (GND) as highlighted in Figure 1.

Figure 1: Terminal Block Power Supply Connections

5.2 Power over Ethernet (PoE)

The gateway supports the IEEE 802.3af Power over Ethernet (PoE) standard as a mode A or mode B powered device (PD). In mode A, the Ethernet jack pins 1-2 (pair #2 in T568B wiring) form one side of the DC supply and pins 3-6 (pair #3 in T568B) form the other side. These are the same two pairs used for data transmission in 10Base-T and 100Base-TX, allowing the provision of both power and data over only two pairs in such networks.

In mode B, the Ethernet jack pins 4-5 (pair #1 in both T568A and T568B cabling standards) form one side of the DC supply and pins 7-8 (pair #4 in both T568A and T568B) provide the return; these are the "spare" pairs in 10BASE-T and

100BASE-TX. Mode B power transmission, therefore, requires the use of a full 4-pair Ethernet cable.

The gateway enumerates itself as a power level class 0 device (12.95W max. indicated consumption). The use of PoE endspan ("PoE switch") or midspan ("power injector") power sourcing equipment (PSE) provides for the ability to power the gateway without the necessity of connecting a dedicated power supply to the power supply terminal block.

5.3 RS-485 Port Electrical Interface

In order to ensure appropriate network conditions (signal voltage levels, etc.) when using the gateway's RS-485 port, some knowledge of the network interface circuitry is required. Refer to Figure 2 for a simplified network schematic of the RS-485 interface circuitry. The port has 4 terminals for four-wire communication. For two-wire communication, connect a jumper wire between TB:1 (A / RXD+) and TB:3 (Y / TXD+) and a wire between TB:2 (B / RXD-) and TB:4 (Z / TXD-).

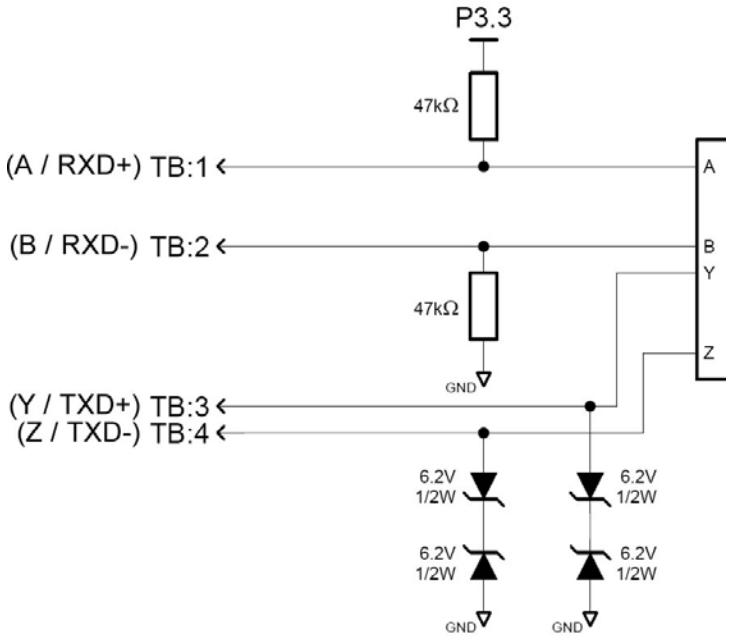


Figure 2: RS-485 Interface Circuitry Schematic

Figure 3 highlights the terminals on the gateway's terminal block that are specific to RS-485 connections.

Figure 3: Terminal Block RS-485 Connections

6. Installation

The gateway's installation procedure will vary slightly depending on the mounting method used. Before mounting the gateway, install the 4 black rubber feet (Figure 4) onto the bottom of the enclosure.

Figure 4: Rubber Feet

6.1 Mounting the Gateway

The gateway may be mounted on a panel, a wall or a DIN rail. In all cases, the gateway is mounted using the two keyhole-shaped screw holes on the bottom of the enclosure. A DIN rail adapter with two pre-mounted screws is provided for mounting the gateway on a DIN rail. The user must choose the appropriate hardware for mounting the gateway on a panel or wall. When choosing screws for panel or wall mounting, ensure the head size matches the keyhole screw holes on the back of the enclosure. The following describes the method for the two mounting options.

6.1.1 Panel / Wall Mounting

To mount the gateway on a panel or wall, drill two holes 25mm apart vertically. Screw two screws into the holes and mount the gateway on the screws.

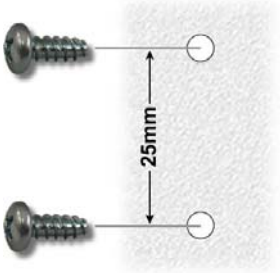


Figure 5: Panel / Wall Mounting Diagram

6.1.2 DIN Rail Mounting

The DIN rail adapter (Figure 6) can clip onto 35mm and G-type rails. To mount the gateway to a DIN rail, clip the DIN rail adapter onto the DIN rail and mount the gateway on the screws (the screws should already be seated into the adapter at the proper height). Refer to Figure 7, Figure 8, and Figure 9.

Figure 6: DIN Rail Adapter

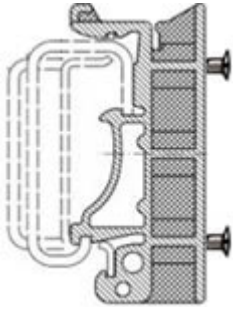


Figure 7: DIN Rail Adapter Attachment

Figure 8: Unit with Attached DIN Rail Adapter

Figure 9: Example Installation

6.2 Wiring Connections

Note that in order to power the unit, a power supply must also be installed. Refer to sections 5.1 and 5.2 for more information.

1. Mount the unit via the desired method (refer to section 6.1).
2. Connect the various networks to their respective plugs/terminal blocks. Ensure that any wires are fully seated into their respective terminal blocks, and route the network cables such that they are located well away from any electrical noise sources, such as adjustable-speed drive input power or motor wiring. Also take care to route all cables away from any sharp edges or positions where they may be pinched.
3. Take a moment to verify that the gateway and all network cables have sufficient clearance from electrical noise sources such as drives, motors, or power-carrying electrical wiring.
4. If not using PoE, connect an external power supply to the gateway's RS-485 terminal block on the terminals labeled POWER and GND. Pay particular attention to the proper polarity.

6.3 Grounding

Grounding is of particular importance for reliable, stable operation. Communication system characteristics may vary from system to system, depending on the system environment and grounding method used.

The gateway has one logic ground located on the RS-485 terminal block, which serves as the ground reference for both power and RS-485 communication signals.

CAUTION:
Note that there is a single chassis ground terminal adjacent to the RS-485 terminal block. This chassis ground terminal is NOT internally connected to the "GND" terminal on the RS-485 terminal block. Do not make any grounding connections to the chassis ground terminal.

Please be sure to consider the following general points for making proper ground connections:

Grounding method checkpoints

1. Make all ground connections such that no ground current flows through the case or heatsink of a connected electrical device.
2. Do not connect the gateway's GND terminal to a power ground or any other potential noise-producing ground connection (such as an adjustable-speed drive's "E" terminal).
3. Do not make connections to unstable grounds (paint-coated screw heads, grounds that are subjected to inductive noise, etc.)

7. LED Indicators

The gateway contains several different LED indicators, each of which conveys important information about the status of the unit and connected networks. These LEDs and their functions are summarized here.

7.1 Module/Network Status

The gateway has two dichromatic, stacked LEDs to indicate the status of the module (MS) and the status of the Ethernet/IP network server driver (NS). On startup, the LEDs blink a startup sequence: green-red-green-red. Always confirm this sequence upon powering the gateway to ensure the device is functioning properly. Refer to Table 1 and Table 2 for further details.

Table 1: Module Status LED

LED State	Indication
Off	No power
Green	Gateway has power and is functioning normally
Flashing Green	The gateway has established a USB connection
Flashing Red	A fatal error has occurred. The number of sequential blinks (followed by 2 seconds of OFF time) indicates the error code.
Alternating Red/Green	Startup sequence

Table 2: EtherNet/IP Network Status LED

LED State	Indication
Off	No IP address / no power
Green	Gateway has an IP address, one or more EtherNet/IP connections are established with a client
Flashing Green	Gateway has an IP address, no EtherNet/IP connections established with a client
Red	Critical link failure / duplicate IP address
Flashing Red	One or more EtherNet/IP connections timed-out
Alternating Red/Green	Startup sequence

7.2 RS-485 Network Status

The gateway has one red and one green LED to indicate the status of the RS-485 network.

Green (TX) LED ..... Lights when the gateway is transmitting data on the RS-485 port.

Red (RX) LED ..... Lights when the gateway is receiving data on the RS-485 port. Note that this does not indicate the validity of the data with respect to a particular protocol: only that data exists and is being detected. Also note that if a 2-wire RS-485 network is in use, that the gateway's RX LED will light in conjunction with the TX LED (as transmitting devices on 2-wire RS-485 networks also receive their own transmissions).

7.3 Ethernet Status

The Ethernet jack contains two embedded LEDs which indicate the status of the Ethernet physical layer

Amber LED..... Ethernet link: lit whenever a viable Ethernet network is connected to the port.

Green LED ..... Ethernet Activity: blinks briefly when Ethernet packets are sent or received.

8. Configuration Concepts

8.1 USB Configuration Utility

The gateway can be configured by a PC via a USB mini type-B cable. This connection provides power to the device, so there is no need for any external power supply while the gateway is attached to the PC.

The gateway is configured by the ICC Gateway Configuration Utility PC application. For information on how to install the utility, refer to the ICC Gateway Configuration Utility User's Manual.

The following will briefly describe how to configure the gateway using the configuration utility. For more information, refer to the ICC Gateway Configuration Utility User's Manual.

Manually Selecting a Device

Select the ETH-1000 from the device menu: click **Device→Select Device→ETH-1000**.

Note that when a device is selected, the utility will then automatically attempt to locate any connected devices of that type.

Automatically Connecting To a Device

If a device is already connected to the PC, you can click the **Auto Connect** button and the utility will automatically select the correct device and upload the current configuration from the connected device.

General Configuration

To configure the gateway, enable the desired Ethernet protocol(s), configure the IP address of the gateway, and configure any objects associated with the enabled Ethernet protocol(s). Then select the desired protocol, baud rate, parity, address, timeout, and scan rate/response delay for the RS-485 port, and configure any objects associated with the protocol (refer to section 8 for more information). For more information on configuring ports, refer to section 8.4.

Note that all numbers can be entered in not only decimal but also in hexadecimal by including "0x" before the hexadecimal number.

Database Endianness Selection

Select the desired endianness for how data will be stored in the database: click **Device→Database Endianness→Big Endian** to use big endian style or click **Device→Database Endianness→Little Endian** to use little endian style. Note that this is part of the configuration and therefore does not take effect until the configuration is downloaded to the device. For more information on the database endianness, refer to Appendix A: Database Endianness.

Loading a Configuration from an XML File

To load a configuration from an XML file stored on the PC, click **File→Load Configuration…** (or click the **Load Configuration** button on the toolbar).

Saving a Configuration to an XML File

To save the configuration to an XML file on the PC, click **File→Save Configuration…** (or click the **Save Configuration** button on the toolbar).

Downloading a Configuration to a Device

To download the configuration to the gateway, click **Device→Download Configuration To Device** (or click the **Download Configuration To Device** button on the toolbar).

Note that because there is a different driver firmware for each protocol, the correct firmware may not be installed on the device corresponding to your configuration. The utility may need to update the firmware on the device before the configuration can be loaded.

Updating Firmware

To update firmware on the gateway, click **Device→Update Firmware** (or click the **Update Firmware** button on the toolbar).

Note that if a newer version exists for the firmware installed on the device, a message will be displayed in the **Status** box indicating an update is available.

Resetting the Device

To reset the gateway, click **Device→Reset Device** (or click the **Reset Device** button on the toolbar).

Monitoring the Database

To monitor the gateway's database in real time, select the **Monitor** tab. Data is updated automatically to reflect the actual values in the database. Values can be edited by double clicking the data in the database. The status of service objects can also be added and viewed in this tab in the **Status** list. Section 8.5.2 describes how to view the status of a service object. For more information, refer to the ICC Gateway Configuration Utility User's Manual.

8.2 Finder Tab

The Finder tab of the configuration utility discovers all ICC communication devices on the current Ethernet subnet, regardless of whether or not their network parameters are currently compatible with the subnet upon which they reside. This tab may be used regardless of whether or not a device is connected with a USB cable, as it communicates only on the Ethernet subnet.

In order for the Finder tab to discover devices, certain UDP Ethernet traffic must be allowed in and out of the computer, and firewall applications (such as Windows Firewall) are often configured to block such traffic by default. If the Finder is unable to discover any devices on the current subnet, be sure to check the computer's firewall settings during troubleshooting, and add an exception to the firewall configuration if necessary.

All discovered devices can be organized in ascending or descending order by clicking on the desired sort header (IP Address, MAC Address, Application Firmware or Product). The rest of the information in the Finder tab is described in the following:

IP Settings Group

This section displays the IP Address information of the currently selected device. These settings may be configured by selecting either "Automatically obtain network settings via DHCP" or by selecting "Manually configure network settings" and entering values for the IP Address, Subnet Mask, and Default Gateway. Once the desired settings are configured, click **Save IP Settings** in the "Tasks" group to save the settings on the gateway.

Product Information Group

This section displays the Product Name, Application Firmware Version, and Hardware Revision for the currently selected device.

Network Information Group

This section displays the currently configured network settings for the selected device.

The "Tasks" group on the left side of the Finder tab contains buttons which perform the following actions:

Open Web Interface

Opens a web browser page of the selected device. Refer to section 10.

Open FTP Interface

Opens the computer's default FTP application, which could be either Windows Explorer, a web browser, or a 3rd-party FTP program (whatever the computer/operating system is configured for by default). This allows you to interact directly with the unit's on-board flash filesystem, enabling you to drag and drop files to/from the unit and upload new firmware. Refer to section 9.

Save IP Settings

Saves the IP settings configured in the **IP Settings** group to the gateway. This will open a dialog box which prompts for a password to save the changes. Enter the case-sensitive system password (default is "icc"), then click **Submit**.

Once the IP settings have been saved, another dialog box will open prompting to reboot the device. Click **Reboot** for the changes to take effect.

Reboot Device

Opens a dialog box which prompts for a password to reboot the device. Enter the case-sensitive system password (default is "icc"), then click **Reboot**. The reboot cycle has completed when the displayed status changes from "Rebooting" to "Ready" (note that this may require 20s or more to complete.) Clicking **Close** will then close the dialog box and cause the discovery utility to automatically rescan the network.

Refresh List

Causes the discovery utility to rescan the network.

8.3 Timeout Configuration Tab

The gateway can be configured to perform a specific set of actions when network communications are lost. This allows each address in the database to have its own unique "fail-safe" condition in the event of network interruption. Support for this feature varies depending on the protocol: refer to the protocol-specific section of this manual for further information.

Note that this feature is only used with slave/server protocols. This is not the same as the timeout value used for master/client protocols. For more information, refer to section 8.4.

There are two separate elements that comprise the timeout configuration:

- The timeout time
- Timeout Object configuration

8.3.1 Timeout Time

The timeout time is the maximum number of milliseconds for a break in network communications before a timeout will be triggered. This timeout setting is configured at the protocol level as part of the port configuration, and used by the protocol drivers themselves to determine abnormal loss-of-communications conditions and, optionally, trigger a gateway-wide timeout processing event. If it is not desired to have a certain protocol trigger a timeout processing event, then the protocol's timeout time may be set to 0 (the default value) to disable this feature. Refer to section 8.4 for details.

8.3.2 Timeout Object Configuration

A timeout object is used by the gateway as part of the timeout processing to set certain addresses of the database to "fail-safe" values. When a timeout event is triggered by a protocol, the timeout objects are parsed and the configured 8-bit,

16-bit, or 32-bit value is written to the corresponding address(es). The following describes the configurable fields of a timeout object:

Database Addr

This field is the starting address in the database where the first data element of this timeout object will begin. Depending on the designated Data Type, the maximum allowable database address is 4095, 4094, or 4092 for 8-bit, 16-bit, or 32-bit sized objects, respectively.

Data Type

This field selects the size and range of valid values for each data element in this timeout object. For instance, selecting 16-bit unsigned allows for a range of values between 0 and 65535, using 2 bytes in the database. Whereas selecting 16-bit signed allows for a range of values between -32768 and 32767, also using 2 bytes in the database. Select the desired data type from this dropdown.

Value

This is the “fail-safe” timeout value that every data element in this timeout object will be automatically written to upon processing of a timeout event triggered by a protocol.

Length

This field is the number of data elements for this timeout object. The total number of bytes modified by this timeout object is determined by the length multiplied by the number of bytes in the data type selected (1, 2 or 4).

8.4 Port Configuration Tabs Protocol Selection Group

This section describes each available field in the Protocol Selection group of the port configuration tabs. Note that support of these fields will vary by protocol, and that unsupported fields will automatically be made non-selectable within the configuration utility.

Protocol

Select the desired protocol for the port.

Baud Rate

Select the network baud rate for the port.

Parity

Select the network parity for the port.

Address

Select the network address at which the gateway will reside.

Timeout

For master/client protocols, enter the request timeout in milliseconds. This setting is the maximum amount of time that the gateway will wait for a response from a remote device after sending a request.

For slave/server protocols, this value is the maximum amount of time the protocol driver will wait in between received packets before triggering a timeout event (for network loss detection). For further timeout processing details, refer to section 8.3.

Scan Rate / Response Delay

For master/client protocols, the scan rate is the number of milliseconds the device will wait between sending requests. This is a useful feature for certain devices or infrastructure components (such as radio modems) that may not be capable of sustaining the maximum packet rates that the gateway is capable of producing. The start time for this delay is taken with respect to the moment at which the gateway is capable of sending the next packet (due to either reception or timeout of the previous request). The default setting of 0 means that the gateway will send its next request packet as soon as possible.

For slave/server protocols, the response delay is the number of milliseconds the device will wait before responding to a request. This is a useful feature for certain master devices or infrastructure components (such as radio modems) that may require a given amount of time to place themselves into a "receiving mode" where they are capable of listening for slave responses. The default setting of 0 means that the gateway will send its responses as soon as possible.

8.5 Service Object Configuration

A service object is used by the gateway to make requests on a network when a master/client protocol is enabled. Each service object defines the services (read or write) that should be performed on a range of network objects of a common type. The data from read requests is mirrored in the database starting at a user-defined address (if a read function is enabled). When a value within that address range in the database changes, a write request is generated on the network (if a write function is enabled). Depending on the protocol selected, service objects will vary slightly. Refer to section 8.7 for specific examples.

8.5.1 Description of Common Fields

This section contains general descriptions of the common service object fields, regardless of which protocol is selected. Each protocol has its own additional fields, as well as a more specific implementation of the common fields. These are discussed in section 8.7.

Description

This field is a description of the service object. It is not used by the gateway, but serves as a reference for the user.

Destination Address

This field is the network node address of the device that the gateway will send a request to.

Type

This selects the object type to use in the service object. All objects in the service object will be of this type.

Start Object

This field specifies the first instance number of the service object range.

Number of Objects

This field specifies the number of objects the service object contains in its range.

Database Address

This is the starting address in the gateway's database that is used to mirror the data on the network. The number of bytes allocated for the service object data is determined by the data type and the number of objects in the service object.

Data Type

This field specifies how many bytes are used to store each object in the service object. The data type also specifies whether the value should be treated as signed or unsigned when converting it to a real number to send over the network.

Note that each data type has its own range limitations for what can be stored in the database: 8 bits can store values up to 255, 16 bits can store values up to 65,535, and 32 bits can store values up to 4,294,967,295.

Multiplier

This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, the data is multiplied by the multiplier to produce a network value. Similarly, network values are divided by the multiplier before being stored into the database.

Note that the multiplier, coupled with the data type, imposes range limitations on network data values. For example, if the data type is 8-bit and the multiplier is 0.5, then the network data can achieve a maximum value of only 127 (since 255 is the maximum value that can be stored in 8 bits in the database).

Function Codes

This field allows you to select which function code to use for a read or write. You may also specify a read-only or a write-only service object by unchecking the checkbox next to the write or the read function, respectively.

Note that some protocols only support one read and one write function code.

8.5.2 Viewing the Status of a Service Object

The gateway provides the user the ability to debug the configured service objects while the device is running. When defining a service object, check the **Reflect Status** checkbox and enter the database address to store the status information. The status information is a 16-byte structure containing a transmission counter, a receive counter, a receive error counter, the current status, and the last error of the defined service object. This information is detailed in Appendix B: Status Information. The data contained in the status information may be viewed over the network on the other port of the gateway by mapping objects to the same database address where the status information is stored.

Alternatively, the status can be viewed in the **Monitor** tab in the **Status** list of the configuration utility. When a configuration that contains a service object status is downloaded to the device, or uploaded from the device, that address is automatically added into the **Status** list in the **Monitor** tab (status addresses can also be added manually in the **Monitor** tab by typing the address and clicking **Add Status Address**). This window will show the value of each of the counters and a translation of the current status and last error. In addition, the counters can be reset by selecting one or more entries in the **Status** list and clicking **Reset Counters**. Status addresses can also be deleted by selecting one or more entries in the **Status** list and clicking **Delete Status Address**, or all of the entries can be deleted by clicking **Delete All Status Addresses**.

8.6 General Object Editing Options

The following editing options apply for all types of configuration objects including, but not limited to, Connection Objects, Service Objects, Register Remap Objects, Timeout Objects and BACnet Objects.

Creating an Object

To create an object, populate all the fields with valid values and click the **Create** button.

Viewing an Object

Objects are listed in the object window located at the bottom of the configuration utility. To view an object, select that object's entry in the object window. This will cause all of the object configuration fields to be populated with the object's current settings.

Updating an Object

To update an object, select the object's entry in the object window, make any required changes, and then click the **Update** button.

Copying an Object

To copy an object, select the entry you wish to copy in the object window, make any required changes, and then click the **Create** button. This may be a useful feature for situations in which many objects must be configured, but only a few fields (such as the database address and type) are different.

Deleting an Object

To delete an object, select the entry you wish to delete in the object window and click the **Delete** button. Note that this action cannot be undone.

Deleting all Objects

To delete all the objects in the object window, click the **Delete All** button. Note that this action cannot be undone.

8.7 Protocol Configuration

The following section describes how to configure protocols on the gateway with the configuration utility. For more details on how to use the configuration utility, refer to the ICC Gateway Configuration Utility User's Manual.

8.7.1 Ethernet Configuration Subtab

The Ethernet configuration subtab contains Ethernet-related configuration items that are not specific to any given protocol. These settings should be appropriately configured regardless of any Ethernet protocols that may be enabled.

8.7.1.1 Authentication Group

Be sure to make a note of the new settings whenever these authentication credentials are changed, as they must be entered whenever the web page is accessed, an FTP session is initiated, or when a configuration change is performed via the Finder tab. Contact ICC for assistance if you have forgotten your customized credentials.

User Name
The username is case-sensitive and can contain letters ("a..z" and "A..Z") and numbers ("0..9"), and can be up to 80 characters in length.

Password
The password is case-sensitive and can contain letters ("a..z" and "A..Z") and numbers ("0..9"), and can be up to 80 characters in length.

8.7.1.2 Network Configuration Group

Configuration items used to modify the IP address-related parameters. Modification of these settings is consistent with the technique used with the Finder tab (refer to section 8.2).

8.7.2 BACnet /IP Client

To configure the BACnet/IP client, click on the **Ethernet Configuration** tab, check **BACnet/IP Client** in the protocol dropdown menu to enable the driver, and then click on the **BACnet/IP Client** subtab. The gateway can read and write the present value property of BACnet objects hosted by other devices on the network. This behavior is defined by configuring BACnet service objects. For more information on service objects, refer to section 8.5. Whenever the BACnet/IP client driver is enabled, the BACnet device object is always present and must be properly configured. If the BACnet/IP server is enabled, the device object configuration from the client will have priority. This section will discuss how to configure the BACnet/IP client.

Scan Rate

This is the time in milliseconds the device will wait between sending requests. This may be useful if BACnet devices that the gateway is communicating with require additional time between requests, or to reduce overall network utilization. If no additional time is required, set this field to 0.

8.7.2.1 Device Object Configuration Group

The Device Object Configuration group contains several fields that must be appropriately set for each device residing on a BACnet network.

Device Name

This field is the BACnet Device Object's name. The device name must be unique across the entire BACnet network. Enter a string of between 1 and 32 characters in length.

Instance Number

This field is the BACnet Device Object's instance number. The instance number must be unique across the entire BACnet network. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

8.7.2.2 BACnet Service Object Configuration

The following describes the configurable fields for a BACnet service object. For more information on BACnet service object editing options, refer to section 8.6.

Type

The radio buttons in this group select the BACnet object type. Choose from Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, or Binary Value.

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of between 1 and 18 characters in length.

Dest Dev Inst ("Destination Device Instance")

This field is the destination device instance of the BACnet device the gateway should send requests to for this service object. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Start Inst

This field is the starting instance number for a range of BACnet objects for this service object. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Num Insts

This field is the number of BACnet objects in this service object. Enter a value of 1 or more.

Database Addr

This field is the database address where the first BACnet object of this service object will be mapped. Enter a value between 0 and 4095 (0x0 – 0xFFF).

Note that the configuration utility will not allow entry of a starting database address that will cause the service object to run past the end of the database. The highest valid database address, therefore, will depend on the targeted data type, as well as the number of items to be accessed.

Data Type

Applies to analog objects only. This field specifies how many bytes are used to store present value data for each BACnet object in this service object, as well as whether the value should be treated as signed or unsigned when converted to a real number for transmission over the network. Select the desired data type from this dropdown menu.

Note that each data type has its own range limitations: 8-bit can have values up to 255, 16-bit can have values up to 65,535, and 32-bit can have values up to 4,294,967,295.

Multiplier

Applies to analog objects only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, the data is multiplied by the multiplier to produce a network value. Similarly, the network values are divided by the multiplier before being stored into the database.

Note that the multiplier, coupled with the data type, imposes range limitations on the network data value. For example, if the data type is 8-bit and the multiplier is 0.5, then the network data can have values only up to 127 (since 255 is the maximum value that can be stored in 8 bits).

Read Enable

Check **Read** to enable reading (the service object will continuously read from the remote device unless a pending **Write** exists).

Write Enable

Does not apply to input objects. Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted remote device).

Priority

This field is used to specify the priority associated with writes for this service object. Select the desired priority from the dropdown menu.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.3 BACnet/IP Server

To configure the BACnet/IP server, click on the **Ethernet Configuration** tab, check **BACnet/IP Server** in the protocol dropdown menu to enable the driver, and then click on the **BACnet/IP Server** subtab. The BACnet/IP server can host a wide variety of user-defined BACnet objects. Whenever the BACnet/IP server is enabled, the BACnet device object is always present and must be properly configured. If the BACnet/IP client is enabled, its device object configuration will have priority. This section will discuss how to configure the BACnet/IP server.

8.7.3.1 Device Object Configuration Group

The Device Object Configuration group contains several fields that must be appropriately set for each device residing on a BACnet network.

Device Name

This field is the BACnet Device Object's name. The device name must be unique across the entire BACnet network. Enter a string of between 1 and 16 characters in length.

Instance Number

This field is the BACnet Device Object's instance number. The instance number must be unique across the entire BACnet network. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

8.7.3.2 BACnet Object Common Configurable Fields

This section describes the common configurable fields for all BACnet objects. For more information on BACnet object editing options, refer to section 8.6.

Type

The radio buttons in this group select the BACnet object type. Choose from Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, or Binary Value.

Object Name

This field is the name of the BACnet object. Enter a string of between 1 and 16 characters in length. All object names must be unique within the gateway.

Instance

This field is the BACnet Object's instance number. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Database Addr

This field is the database address where the BACnet object's present value will reside. Enter a value between 0 and 4095 (0x0 – 0xFFF).

A note for analog objects: Depending on the designated Data Type, the maximum allowable database address is 4095, 4094, or 4092 for 8-bit, 16-bit, or 32-bit sized objects, respectively.

Multiplier

Applies to analog objects only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, the data is multiplied by the multiplier to produce a network value. Similarly, the network values are divided by the multiplier before being stored into the database.

Note that the multiplier, coupled with the data type, imposes range limitations on the network data value. For example, if the data type is 8-bit and the multiplier is 0.5, then the network data can have values only up to 127 (since 255 is the maximum value that can be stored in 8 bits).

Units

Applies to analog objects only. Select the desired units from this dropdown menu. If the desired units are not available in the dropdown menu, select **Other Units** and enter the enumerated value (as defined by the BACnet Specification) in the **Unit Value** field.

Bitmask

Applies to binary objects only. This 8-bit field specifies which bit(s) in the byte designated by the **Database Addr** that the binary object will map to. This allows up to 8 binary objects to be simultaneously assigned to one database address (each binary object mapping to a single bit of that byte in the database). It is possible to map binary objects to multiple bits within the designated database location. Such a configuration allows (for example) the modification of multiple selected database bits via a single binary output.

The effect of the **Bitmask** field when writing: When the present value property of a binary output object or binary value object is set to "active" by a BACnet client, then the bit(s) in the designated **Database Addr** indicated by a "1" in the bitmask are set. Similarly, when the present value property of the object is set to "inactive", then the bit(s) in the designated **Database Addr** indicated by a "1" in the bitmask are cleared. For binary output objects, this setting/clearing behavior is reversed if the object's **Polarity** is set to "Reversed".

The effect of the **Bitmask** field when reading: When the present value property of a binary object is read by a BACnet client, the **Bitmask** is used to determine the active/inactive state of the object by inspecting the value in the designated database address at the bit location(s) indicated in the **Bitmask**. If all of the bit locations at the designated database address indicated by a "1" in the **Bitmask** are set, then the object's state will be returned as "active". Else, the object's state will be returned as "inactive". For binary input and binary output objects, the resultant state is reversed just prior to being placed on the network if the object's **Polarity** is set to "Reversed".

Active Text

Applies to binary objects only. This field specifies the description of the object's "active" state. Enter a string of up to 8 characters in length. This field is optional and may be left blank.

Inactive Text

Applies to binary objects only. This field specifies the description of the object's "inactive" state. Enter a string of up to 8 characters in length. This field is optional and may be left blank.

Polarity

Applies to binary input and binary output objects only. This field indicates the relationship between the physical state of the object (as stored in the gateway's database) and the logical state represented by the object's present value property. If the physical state is active high, select **Normal** from this dropdown menu. If the physical state is active low, select **Reverse** from this dropdown menu. For further detail, refer to the **Bitmask** behavioral description above.

Data Type

Applies to analog objects only. This field specifies how many bytes are allocated for the present value data, as well as whether the value should be treated as signed or unsigned when converting it to a real number to send over the network. Select the desired data type from this dropdown menu.

Note that each data type has its own range limitations: 8-bit data types can have values up to 255, 16-bit data types can have values up to 65,535, and 32-bit data types can have values up to 4,294,967,295.

Relinquish Def

This field is the default value to be used for an object's present value property when all command priority values in the object's priority array are NULL. Note that this property only exists for those objects that implement a priority array (analog output, analog value, binary output and binary value objects).

8.7.4 EtherNet/IP Client

The EtherNet/IP client protocol can be used to access information on EtherNet/IP and CSP server devices which support class 1 I/O messaging and class 3 explicit messaging. Similar to other master/client protocol drivers on the gateway, the EtherNet/IP client driver uses service objects to produce and consume data and to issue explicit read and write requests to the server device. EtherNet/IP service objects also make use of an additional gateway construct called a connection object in order to target server devices. A connection object can be thought of as a communication channel or "pipe" which is created between the gateway and the server device, independent of the service objects that may later make use of that communication channel to transfer service object requests.

A connection object defines a connection to a specific endpoint (IP address). For each connection object, however, up to two underlying EtherNet/IP client connections can be simultaneously established: one connection is reserved for class 1 I/O messaging and the other connection is reserved for class 3 explicit messaging. The class 3 explicit messaging connection is capable of managing any number of service objects targeting the remote endpoint defined by the connection object (the service objects are serviced sequentially). Several different types of class 3 requests are available to match the capabilities of a variety of EtherNet/IP server devices.

Only class 3 connections can be used when connecting to CSP server devices (such as PLC-5/E or SLC-5/05 PLCs, etc.), as these devices do not support class 1 data transfers. Additionally, not all EtherNet/IP server devices support both class 1 and class 3 connections: please confirm the capabilities of the targeted server device when configuring the gateway's service objects.

Because service objects must always be configured to use a specific connection object, at least one connection object must initially be created before any service objects can be created. For more information on service objects, refer to section 8.5.

To configure the EtherNet/IP client, click on the **Ethernet Configuration** tab, check **EtherNet/IP Client** in the protocol dropdown menu to enable the driver, and then click on the **EtherNet/IP Client** subtab. This section will discuss how to configure the EtherNet/IP client.

Explicit Messaging Scan Rate

This is the time in milliseconds the device will wait between sending class 3 explicit messaging requests. If no additional time is required, set this field to 0.

8.7.4.1 Connection Object Configuration Group

Name

A unique name used for identifying the connection object. Enter a string of up to 16 characters in length.

IP Address

This is the IP address of the EtherNet/IP server (PLC or motion controller, etc.) to be targeted by the connection object. The **IP Address** must be unique in the gateway's configuration.

8.7.4.2 Service Object Configuration Group

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 18 characters in length.

Connection

Select the desired connection object from this dropdown menu (refer to section 8.7.4.1). Connection objects automatically appear in this menu after they are created.

Message Type

Select the desired message type for the service object from this dropdown menu. Class 1 is the implicit (I/O) messaging type. Produced Tag message type is another form of class 1 messaging that is used only to consume data from a - Logix PLC produced tag. Before configuring the Produced Tag message type, it is suggested that users read the ICC whitepaper titled "*Accessing Data on a Logix PLC with the ICC ETH-1000 Ethernet Gateway*", which can be found in the documents section at http://www.iccdesigns.com. All other selections indicate a specific type of class 3 explicit messaging. Only one class 1 message type is supported per connection object.

The following fields apply to Class 1 Message Types only

Pro Start Addr ("Produced Data Start Address")

This field specifies the database address that the first produced byte of I/O data is mapped to. Output data sent to the server device will be retrieved from the gateway's database starting at this address. Enter a value between 0 and 4095 (0x0 – 0xFFF).

Pro Data Size ("Produced Data Size")

This field defines the number of bytes to produce (send to the server device). Enter a value of 0 or larger.

Con Start Addr ("Consumed Data Start Address")

This field specifies the database address that the first consumed byte of data is mapped to. Input data received from the server device will be stored in the gateway's database starting at this address. Enter a value between 0 and 4095 (0x0 – 0xFFF).

Con Data Size ("Consumed Data Size")

This field defines the number of bytes to consume (receive from the server device). Enter a value of 0 or larger.

Cfg Cnxn Pt ("Configuration Connection Point")

This field specifies the assembly instance number for the configuration connection included in the initial "forward open" request. While this assembly instance number is required, no additional configuration data is included in the "forward open" request. Some EtherNet/IP servers ignore this field. Enter a value of 0 or larger.

Pro Cnxn Pt ("Produced Data Connection Point")

This field specifies the assembly instance number for the produced data connection (O→T direction). Enter a value of 0 or larger.

Con Cnxn Pt ("Consumed Data Connection Point")

This field specifies the assembly instance number for the consumed data connection (T→O direction). Enter a value of 0 or larger.

Enable Run/Idle Header

Check this checkbox to include the run/idle header in the gateway's produced data. Most EtherNet/IP server devices expect the run/idle header to be included in the data sent to them.

RPI

This field is the Requested Packet Interval and is specified in units of milliseconds. The minimum supported RPI is 10ms.

Mult

Specifies the connection timeout multiplier. The connection's timeout time is determined by multiplying the RPI times the connection timeout multiplier.

The following fields apply to CIP Generic Message Types only

Read

Check **Read** to enable reading via the "get attribute single" service. Refer to the **Triggering Mechanism** to determine the specific behavior of read services.

Write

Check **Write** to enable writing via the "set attribute single" service. If writes are enabled, then when values encompassed by this service object change in the gateway's database, these changes will be written

down to the targeted device. Refer to the **Triggering Mechanism** to determine the specific behavior of write services.

DB Addr ("Database Address")

This field specifies the address in the gateway's database where the first byte of this service object's data value will be mapped. Enter a value between 0 and 4095 (0x0 – 0xFFF).

Num Bytes

This field specifies the number of bytes starting from the designated **Database Address** that are to be accessed by this service object. By appropriately setting this field, access to any supported CIP data type (INT, DWORD, REAL, arrays etc.) is possible. Enter a value between 1 and 240 (0x1 – 0xF0).

Class

Specifies the class code of the targeted EtherNet/IP object.

Instance

Specifies the instance number of the targeted EtherNet/IP object.

Attribute

Specifies the attribute number of the targeted EtherNet/IP object.

The following fields apply to Data Table Read/Write Message Types only

Read

Check **Read** to enable reading via the "data table read" service. Refer to the **Triggering Mechanism** to determine the specific behavior of read services.

Write

Check **Write** to enable writing via the "data table write" service. If writes are enabled, then when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted device. Refer to the **Triggering Mechanism** to determine the specific behavior of write services.

DB Addr ("Database Address")

This field specifies the address in the gateway's database where the first 16-bit element of this service object's data will be mapped. Enter a value between 0 and 4094 (0x0 – 0xFFE).

Num Elements

This field specifies the number of 16-bit elements to be accessed by this service object. Enter a value between 1 and 120 (0x1 – 0x78).

Tag

This field specifies the tag name on the target device. Enter a string from 1 to 32 characters in length (note that tag names cannot contain spaces).

Offset

This field specifies the offset from the **Tag**.

The following fields apply to Typed Read/Write Message Types only

Read

Check **Read** to enable reading via the "typed read" service. Refer to the **Triggering Mechanism** to determine the specific behavior of read services.

Write

Check **Write** to enable writing via the "typed write" service. If writes are enabled, then when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted device. Refer to the **Triggering Mechanism** to determine the specific behavior of write services.

DB Addr ("Database Address")

This field specifies the address in the gateway's database where the first 16-bit element of this service object's data will be mapped. Enter a value between 0 and 4094 (0x0 – 0xFFE).

Num Elements

This field specifies the number of 16-bit elements to be accessed by this service object. Enter a value between 1 and 120 (0x1 – 0x78).

Offset

This field specifies the offset from the **File Num**.

File Num

This field specifies the file number on the target device. Enter a value between 0 and 65535 (0x0 – 0xFFFF).

The following fields apply to Produced Tag Message Types only

Produced Tag

This field specifies the produced tag name on the target device. Enter a string from 1 to 32 characters in length (note that tag names cannot contain spaces).

Con Start Addr ("Consumed Data Start Address")

This field specifies the database address that the first consumed byte of data is mapped to. Input data received from the server device will be stored in the gateway's database starting at this address. Enter a value between 0 and 4095 (0x0 – 0xFFF).

Con Data Size ("Consumed Data Size")

This field defines the number of bytes to consume (receive from the server device). Enter a value of 1 or more.

RPI

This field is the Requested Packet Interval and is specified in units of milliseconds. The minimum supported RPI is 10ms.

Mult

Specifies the connection timeout multiplier. The connection's timeout time is determined by multiplying the RPI times the connection timeout multiplier.

Triggering Mechanism Group

The triggering mechanism group provides a certain degree of control over the service object behavior (the situations in which connections are established or when data is exchanged with the remote device).

If the **Auto Trigger** checkbox is checked, then the gateway will automatically handle the service object's production/consumption or reading/writing (depending on whether the service object is a class 1 or class 3 connection).

On the other hand, there may be situations where external (manual) control over service object data transfers is desirable. This can be accomplished by manipulating bit(s) in the gateway's database, either by injecting data into the gateway via another server protocol, or by new data values being actively read into the database via service objects associated with a client protocol.

To manually activate a service object, uncheck the **Auto Trigger** checkbox and enter a database **Address** (0x0 – 0xFFF) and an 8-bit **Bitmask**. Only 1 bit may be set in the **Bitmask**, and each trigger bit at a certain **Address** may only be associated with one service object (in other words, there is a unique "one service object to one bit" association). Therefore, up to 8 **Bitmasks** (0x01, 0x02, 0x04,

0x08, 0x10, 0x20, 0x40 and 0x80) for 8 different service objects can be defined at any given database **Address**. While the definition of what constitutes a valid trigger bit is consistent for all service objects, the triggering mechanism behavior is different for class 1 and class 3 message types.

For class 1 message types

- If the **Auto Trigger** checkbox is checked, then the class 1 I/O connection will always be enabled. This means that the I/O connection will be established and production/consumption will continuously occur at regular intervals.
- If the **Auto Trigger** checkbox is unchecked, then the class 1 connection will be enabled only as long as the designated trigger bit is a "1": if the trigger bit becomes a "0", then the client will close the class 1 connection to the server. If the trigger bit transitions back to a value of "1", the client will once again reestablish the class 1 connection to the server and resume regular I/O data production/consumption.

For class 3 message types

- If the **Auto Trigger** checkbox is checked, then the behavior of the service object will depend on the **Read** and **Write** function checkboxes: if the **Read** checkbox is checked, then the service object will continuously read from the remote server unless a pending **Write** exists. If the **Write** checkbox is checked, then when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted server.
- If the **Auto Trigger** checkbox is unchecked, then only the **Read** function checkbox or the **Write** function checkbox may be checked (not both). This essentially turns the service object into a read-only or write-only service object. In this mode, the trigger bit acts like a "one-shot", meaning that when an external source sets the trigger bit to "1" in the gateway's database, then the corresponding service object request (read or write) is unconditionally generated (regardless of whether or not the service object data has changed). Once the request has been completed (either successfully or unsuccessfully), the gateway will update the database with the current data (if the service object is read-only), update the **Reflect Status** data structure (if enabled for this service object) to indicate the success or failure of the transaction, and then clear the trigger bit. In this way, a remote device can both trigger a read/write class 3 action, as well as be notified of the completion and resulting status by appropriately mapping the trigger bit, service object data, and reflect status structure into a block of data that is accessible via another network (such as a read/write service object controlled by a client driver on another network).

Note that because the gateway's entire database is initialized to "0" values after every boot-up, all defined manual trigger bits will cause their respective service objects to be disabled until explicitly enabled from an external source.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at

which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.5 EtherNet/IP Server

The EtherNet/IP server driver allows the gateway to participate in I/O messaging with other devices. While the database start addresses at which the produced and consumed data will map to are set via the configuration utility, the sizes of these exchanged data packets are determined by the client upon initial connection establishment.

To configure the EtherNet/IP server, click on the **Ethernet Configuration** tab, check **EtherNet/IP Server** in the protocol dropdown menu to enable the driver, and then click on the **EtherNet/IP Server** subtab. This section will discuss how to configure the EtherNet/IP server.

8.7.5.1 Configurable Fields

Device Name

The device name is used for identification of a device on the EtherNet/IP network. This string is accessible as the "product name" attribute of the identity object. Enter a string between 1 and 32 characters in length.

Run/Idle Flag Behavior

EtherNet/IP clients (such as PLCs) have the option of adding a 32-bit "run/idle" header to all class 1 (I/O) data packets sent to devices. Bit 0 of this header is called the "run/idle flag" by the EtherNet/IP specification, and is intended to signify when the client is in a "running" state or an "idle" state. A running state (run/idle flag = Run) is indicated whenever the client is performing its normal processing (e.g. scanning its ladder logic). An idle state (run/idle flag = Idle) is indicated otherwise. For example, Allen Bradley ControlLogix PLCs will set their run/idle flag to Idle whenever their processor keyswitch is placed in the "PROG" position, presumably in preparation to receive a new application program from RSLogix.

The behavior of EtherNet/IP devices when they receive I/O data from a controller with the run/idle flag set to Idle is not specified in the EtherNet/IP specification. The gateway allows the option of two different behavioral responses when a run/idle flag = Idle condition is received, depending on the state of the **Invoke Timeout When Run/Idle Flag = Idle** checkbox.

- If the checkbox is cleared (default setting), then the gateway will maintain the last I/O data values received from the client. For example, if a device mapped to the gateway was being commanded to run prior to the run/idle flag being set to Idle, then it will continue to run.
- If the checkbox is checked, then the gateway will invoke its user-configured timeout processing (refer to section 8.3 for further details regarding timeout processing). This setting allows the user to determine any gateway behavior they may desire.

Produced Data Start Address

This is the starting address in the database from which the gateway's produced data will be taken. Enter an address between 0 and 4095 (0x0 – 0xFFF).

Consumed Data Start Address

This is the starting address in the database at which the gateway's consumed data will be inserted. Enter an address between 0 and 4095 (0x0–0xFFF).

8.7.6 Modbus/TCP Client

The Modbus/TCP client driver can be used to access information on any device which supports the Modbus/TCP server protocol. This driver also supports the 32-bit extension to the Modbus standard (commonly referred to as the "Enron/Daniel" extension). Similar to other master/client protocol drivers on the gateway, the Modbus/TCP client driver uses service objects to issue read and write requests to the Modbus/TCP server device. Modbus/TCP service objects also make use of an additional gateway construct called a connection object in order to target Modbus/TCP server devices. A connection object can be thought of as a communication channel or "pipe" which is created between the gateway and the Modbus/TCP server device, independent of the service objects that may later make use of that communication channel to transfer service object requests. A connection object defines a connection to a specific endpoint (IP address).

Because service objects must always be configured to utilize a specific connection object, at least one connection object must initially be created before any service objects can be created. For more information on service objects, refer to section 8.5.

Each register (input or holding) in a service object is mapped to 2 bytes in the database (the data type is fixed at 16-bit). Each discrete (input or coil) is mapped starting at the least-significant bit of the byte specified by the database address and at each consecutive bit thereafter. For more information on register and discrete mapping, refer to section 11.5.1.3.

To configure the Modbus/TCP client, click on the **Ethernet Configuration** tab, check **Modbus/TCP Client** in the protocol dropdown menu to enable the driver, and then click on the **Modbus/TCP Client** subtab. This section will discuss how to configure the **Modbus/TCP Client**.

Scan Rate

This is the time in milliseconds the device will wait between sending requests. If no additional time is required, set this field to 0.

8.7.6.1 Connection Object Configuration Group

Name

A unique name used for identifying the connection object. Enter a string of up to 16 characters in length.

IP Address

This is the IP address of the Modbus/TCP server to be targeted by the connection object.

8.7.6.2 Service Object Configuration Group

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 18 characters in length.

Connection

Select the desired connection object from this dropdown menu (refer to section 8.7.6.1). Connection objects automatically appear in this menu after they are created.

Unit ID

This field is used primarily for intra-system routing purposes. Many Modbus TCP servers ignore this field.

Type

Select the desired function type from this dropdown menu. This field designates the Modbus data type that this service object will access on the remote server.

Start Register / Start Discrete

For holding register and input register types: this field defines the starting register number for a range of registers associated with this service object. Enter a value between 1 and 65535.

For coil status and input status types: this field defines the starting discrete number for a range of discretes associated with this service object. Enter a value between 1 and 65535.

Num Registers / Num Discretes

For holding register and input register types: This field defines the number of registers associated with this service object. Enter a value between 1 and 125.

For coil status and input status types: This field defines the number of discretes associated with this service object. Enter a value between 1 and 2000.

DB Addr ("Database Address")

This field defines the database address where the first register/discrete of this service object will be mapped. Enter a value between 0 and 4095. Note that the configuration utility will not allow entry of a starting database address that will cause the service object to run past the end of the database. The highest valid database address, therefore, will depend on the targeted data type, as well as the number of items to be accessed.

Multiplier

Applies to register types only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, raw data is

multiplied by the multiplier to produce a network value. Similarly, network values are divided by the multiplier before being stored into the database.

Note that the multiplier imposes range limitations on network data values. For example, if the multiplier is 0.01, then the network data can achieve a maximum value of only 655 (since 65535 is the maximum value that can be stored in 16 bits in the database).

Read Enable and Function Code Selection

Check **Read** to enable reading (the service object will continuously read from the server unless a pending **Write** exists). When reads are enabled, the desired read **Function Code** can be selected in the drop-down box.

Write Enable and Function Code Selection

Applies to holding register and coil status types only. Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted server). When writes are enabled, the desired write **Function Code** can be selected in the drop-down box.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.6.2.1 32-Bit Extension Options

Applies to register types only. If the target registers are associated with the Enron/Daniel extension to the Modbus specification, or are represented by 32-bit values, check the **Enable Enron/Daniel** checkbox to enable the 32-bit extension option. The following describes each of the extension options:

Floating Point

Enable **Floating Point** if the transmitted values are encoded in IEEE 754 floating point format.

Big Endian

Enable **Big Endian** if the transmitted values are encoded in big-endian, 16-bit word order, i.e. the most significant 16-bit word is before the least significant 16-bit word.

Word-Size Reg

Enable **Word-Size Reg** if each target register is 16-bits wide, but two 16-bit registers comprise one 32-bit value. If not enabled, each of the target registers is assumed to be 32-bits wide.

Note that when **Word-Size Reg** is enabled, the **Num Regs** field name changes to **Num Reg Pairs**, indicating the number of pairs of 16-bit wide registers to address. When enabled, each register pair will use two register addresses and the selected **Data Type** will be applicable for the register pair, not the individual registers. For example, if the **Start Reg** is 100, **Num Reg Pairs** is 2, and **Data Type** is 32-bit Unsigned, then register numbers 100 – 103 will be accessed by the service object, with registers 100 and 101 stored as the first 32-bit Unsigned value and registers 102 and 103 stored as the next 32-bit Unsigned value in the gateway's database.

Word Count

Enable **Word Count** to encode the number of 16-bit words to be transferred in the Modbus "quantity of registers" field. If not enabled, the number of 32-bit registers will be used in the "quantity of registers" field.

Data Type

This field specifies how many bytes are used to store data for each register (or register pair) in this service object, as well as whether the value should be treated as signed or unsigned when converted to a floating point number for transmission over the network. Select the desired data type from this dropdown menu.

Note that each data type has different range limitations: 16-bit data types can represent values up to 65,535, and 32-bit data types can represent values up to 4,294,967,295.

8.7.7 Modbus/TCP Server

To configure the Modbus/TCP server, click on the **Ethernet Configuration** tab, check **Modbus/TCP Server** in the "Protocol" dropdown menu to enable the driver, and then click on the **Modbus/TCP Server** subtab. By default, the gateway's entire database is accessible via the register mapping mechanism discussed in section 11.5.2.2.

8.7.7.1 Protocol Selection Group

Timeout Time

Refer to section 8.3.1.

8.7.7.2 Connection Timeout Options Group

This group contains a selection to enable the gateway's Modbus "supervisory timer" function. This timer provides the ability for the gateway to monitor timeout occurrences between successive Modbus/TCP socket connections, as opposed to the standard timeout functionality (refer to section 8.3), which monitors timeout occurrences only within the scope of each client socket connection. While this feature provides an additional level of fail-safe functionality for those applications that require it, there are several ramifications that must be understood prior to enabling this capability. Before enabling this timer, therefore, it is suggested that users read the ICC whitepaper titled "*A Discussion of Modbus/TCP Server-Side Timeout Processing*", which can be found in the documents section at http://www.iccdesigns.com.

8.7.7.3 Register Remap Object

Optionally, registers can be remapped to different database addresses from their default mapping using a register remap object. It also allows the user to map a register that is not mapped into the database by default (any register above 2048) to an address in the database. The register remap object can remap a range of consecutive registers to any starting address in the database (as long as the entire range is within the database).

Note that registers can be accessed as either holding registers or input registers.

The following describes the configurable fields for a register remap object. For more information on register remap object editing options, refer to section 8.6.

Description

This field is a description of the register remap object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 18 characters in length.

Start Reg

This field is the starting register number for a range of registers to be remapped. Enter a value between 1 and 65535 (0x1 – 0xFFFF).

Num Regs

This field is the number of registers to remap. Enter a value of 1 or more.

Database Addr

This field is the database address where the remapping begins. Enter a value between 0 and 4094 (0x0 – 0xFFE).

Register Type

Select the appropriate register type(s) for the register remap. Both Holding Register and Input Register can be selected together or individually.

8.7.7.3.1 32-Bit Extension Options

If the target registers are associated with the Enron/Daniel extension to the Modbus specification, or are represented by 32-bit values, check the **Enable Enron/Daniel 32-bit Extension** checkbox to enable the 32-bit extension option. The following describes each of the extension options:

Floating Point

Enable **Floating Point** if the transmitted values are to be encoded in IEEE 754 floating point format.

Big Endian

Enable **Big Endian** if the transmitted values are to be encoded in big-endian, 16-bit word order, i.e. the most significant 16-bit word is before the least significant 16-bit word.

Word-Size Register

Enable **Word-Size Register** if each target register is 16-bits wide, but two 16-bit registers are to comprise one 32-bit value. If not enabled, each of the target registers is assumed to be 32-bits wide.

Note that when **Word-Size Register** is enabled, the **Num Regs** field name changes to **Num Reg Pairs**, indicating the number of pairs of 16-bit wide registers to be addressed. When enabled, each register pair will use two register addresses and the selected **Data Type** will be applicable for the register pair, not the individual registers. For example, if the **Start Reg** is 100, **Num Reg Pairs** is 2, and **Data Type** is 32-bit Unsigned, then register numbers 100 – 103 will be remapped, with registers 100 and 101 representing the first 32-bit Unsigned value and registers 102 and 103 representing the next 32-bit Unsigned value in the gateway's database.

Word Count

Enable **Word Count** to interpret the Modbus "quantity of registers" field as the number of 16-bit words to be transferred. If not enabled, the "quantity of

registers" field will be interpreted as the number of 32-bit registers to be transferred.

Data Type

This field specifies how many bytes are used to store data for each register (or register pair), as well as whether the internal value should be treated as signed or unsigned when converted to a floating point number for transmission over the network. Select the desired data type from this dropdown menu.

Note that each data type has different range limitations: 16-bit data types can represent values up to 65,535, and 32-bit data types can represent values up to 4,294,967,295.

8.7.8 Mitsubishi MELSEC Client

The MELSEC client protocol can be used to access information on Mitsubishi PLCs and motion controllers which support the MELSEC protocol using 3E and 1E frames. Similar to other master/client protocol drivers on the gateway, the MELSEC client driver uses service objects to issue read and write requests to the MELSEC server device. MELSEC service objects also make use of an additional gateway construct called a connection object in order to target MELSEC server devices. A connection object can be thought of as a communication channel or "pipe" which is created between the gateway and the MELSEC server device, independent of the service objects that may later make use of that communication channel to transfer service object requests.

A connection object defines a connection to a specific endpoint (IP address and port). Multiple connections can be established with a single remote MELSEC server device residing at a given IP address: as long as the assigned port number in each connection object is unique, then the connection objects will be accessing non-conflicting endpoints, and will therefore be independently managed within the gateway.

Because service objects must always be configured to utilize a specific connection object, at least one connection object must initially be created before any service objects can be created. For more information on service objects, refer to section 8.5.

Figure 10 provides a graphical representation of the MELSEC client concept, and demonstrates the associations among service objects, connection objects, MELSEC servers and internal server devices. This example system contains five service objects, four connection objects, and three MELSEC servers (MELSEC-capable PLC's or motion controllers). The "internal devices" in each of the MELSEC servers can be any of the supported internal device types (data registers, internal relays, motion registers, etc.)

The blue connections in Figure 10 show a situation in which two gateway service objects have been mapped to two internal device types on a single physical MELSEC server. In this case, both service objects share a single connection object. The advantage of this configuration is that connection object usage is minimized, thereby allowing connections to more physical servers. The disadvantage is that these service objects must be serialized (first one is dealt with, and then the other), as a single connection object can only process one request at a time.

The red connections in Figure 10 show a similar situation (two service objects mapped to two internal device types on a single physical MELSEC server). In this case, however, two separate connection objects have been defined for MELSEC server #3 (each with a different destination port number). This allows each service object associated with MELSEC server #3 to have its own dedicated connection object, which improves performance by allowing these service objects to be processed in parallel.

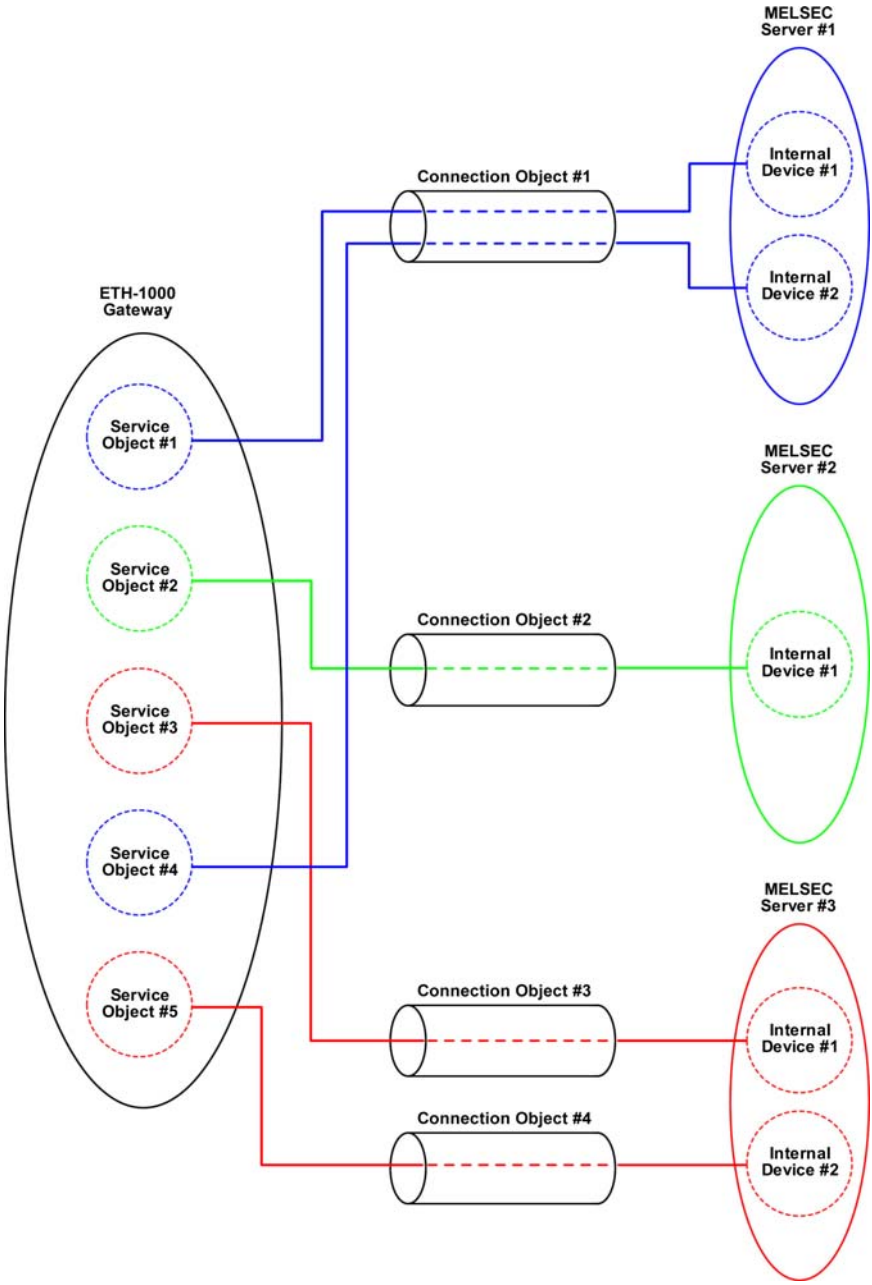


Figure 10: MELSEC Client Concept

To configure the MELSEC client, click on the **Ethernet Configuration** tab, check **MELSEC Client** in the protocol dropdown menu to enable the driver, and then click on the **MELSEC** subtab. This section will discuss how to configure the MELSEC client.

Scan Rate

This is the time in milliseconds the device will wait between sending requests. If no additional time is required, set this field to 0.

8.7.8.1 Connection Object Configuration Group

Name

A unique name used for identifying the connection object. Enter a string of up to 16 characters in length.

IP Address

This is the IP address of the MELSEC server (PLC or motion controller, etc.) to be targeted by the connection object. The **IP Address** and **Port** must be a unique combination.

Port

This is the TCP port of the MELSEC server that the connection object will connect to. Enter a number between 1 and 65535 (0x1 – 0xFFFF).

Frame Type

This drop-down box designates the frame type to use when communicating with the targeted MELSEC server. If the required **Frame Type** is unknown, *Auto-Detect* can be selected to automatically determine the frame type when the gateway establishes initial communication with the server. Once the frame type has been successfully detected, the gateway will then remember this type and use it as the initial preference for subsequent connection attempts. If both 3E and 1E frames are supported by the targeted MELSEC server, then 3E frames are given priority. Note that the auto-detection procedure may cause a communication error to be indicated on the server device during the initial connection attempt, but this error can be cleared by power-cycling the server. To determine the required **Frame Type** for your equipment, please consult the MELSEC server device user's manual or contact Mitsubishi for assistance.

8.7.8.2 Service Object Configuration Group

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 18 characters in length.

Connection

Select the desired connection object from this dropdown menu (refer to section 8.7.8.1). Connection objects automatically appear in this menu after they are created.

Device Code

Select the desired device code (MELSEC server internal device type identifier) for the service object from this dropdown menu.

If the desired device code is not listed in the dropdown menu, select **Other Device Code**, enter the enumerated value in the **Code Value** field. Note that device codes are not case-sensitive. Also note that if the device code entered into the **Other Device Code** field is recognized by the configuration utility, then the standard device code text will be displayed in the MELSEC Service Object List when the service object is created.

Function

Check **Read** to enable reading (the service object will continuously read from the server unless a pending **Write** exists).

Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted MELSEC server).

Starting Point

This field is the starting point in a device's range of points. Enter a value between 0 and 16777215 (0x0 – 0xFFFFFF). If the targeted server uses 1E frames and the Device Code or Code Value is set to a bit device, the starting point should be a multiple of 16.

Num Words

This field is the number of words from the **Starting Point** that are to be accessed by this service object. Enter a value between 1 and 722 (0x1–0x2D2). If the targeted server uses 1E frames, then the maximum value should be limited to 256 words. If a larger value is entered, the driver will internally limit the value to 256. The targeted server may also impose additional limitations (please refer to the MELSEC server equipment user's manual.)

Database Addr

This field is the database address where the first point of this service object will be mapped. Enter a value between 0 and 4094 (0x0 – 0xFFE).

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at

which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.9 Profinet IO

The Profinet IO driver allows the gateway to participate in cyclic data exchange with a Profinet controller. While the database start addresses at which the produced and consumed data will map to are set via the configuration utility, the sizes of these exchanged data packets are determined by the controller upon initial connection establishment.

To configure Profinet IO, click on the **Ethernet Configuration** tab, check **Profinet IO** in the protocol dropdown menu to enable the driver, and then click on the **Profinet IO** subtab. This section will discuss how to configure Profinet IO.

8.7.9.1 Configurable Fields

Device Name

The device name (or station name) must be unique across the entire Profinet network because it is used by controllers to uniquely identify Profinet devices. Enter a string between 1 and 32 characters in length.

Produced Data Start Address

This is the starting address in the database from which the gateway's produced data will be taken. Enter an address between 0 and 4095 (0x0 – 0xFFF).

Consumed Data Start Address

This is the starting address in the database at which the gateway's consumed data will be inserted. Enter an address between 0 and 4095 (0x0–0xFFF).

8.7.10 Modbus RTU Master

To configure the gateway for Modbus RTU Master, click on the **RS-485 Configuration** tab and select **Modbus RTU Master** in the protocol dropdown menu. The Modbus RTU Master protocol uses service objects to make requests. For more information on service objects, refer to section 8.5. Each register (input or holding) in a service object is mapped to 2 bytes in the database (the data type is fixed at 16-bit). Each discrete (input or coil) is mapped starting at the least-significant bit of the byte specified by the database address and at each consecutive bit thereafter. For more information on register and discrete mapping, refer to section 11.5.1.3.

8.7.10.1 Protocol Selection Group

Protocol

Select **Modbus RTU Master** from this dropdown menu.

Baud Rate

Select the desired network baud rate from this dropdown menu.

Parity

Select the desired network parity and number of stop bits from this dropdown menu.

Timeout

This is the time in milliseconds that the device will wait for a response from a remote slave after sending a request.

Scan Rate

This is the time the device will wait between sending requests. This may be useful if slave devices require additional time between requests. If no additional delay time is needed, set this field to 0. For more information, refer to section 8.4.

8.7.10.2 Modbus Service Object Configuration

This section describes the configurable fields for a Modbus service object. For more information on Modbus service object editing options, refer to section 8.6.

Type

This group designates the Modbus data type that this service object will access. Choose from Holding Register, Input Register, Coil Status, or Input Status.

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 16 characters in length.

Dest Address

This field indicates the destination address of the remote slave device on the network that will be accessed by this service object. Enter a value between 0 and 247. Note that address 0 is defined by Modbus as the broadcast address: if this address is used, the **Read** function checkbox must be unchecked, as attempts to read a service object targeting destination address 0 will invariably time out.

Start Reg / Start Discrete

For holding register and input register types: this field defines the starting register number for a range of registers associated with this service object. Enter a value between 1 and 65535.

For coil status and input status types: this field defines the starting discrete number for a range of discretes associated with this service object. Enter a value between 1 and 65535.

Num Regs / Num Discretes

For holding register and input register types: This field defines the number of registers associated with this service object. Enter a value between 1 and 125.

For coil status and input status types: This field defines the number of discretes associated with this service object. Enter a value between 1 and 2000.

Database Addr

This field defines the database address where the first register/discrete of this service object will be mapped. Enter a value between 0 and 4095. Note that the configuration utility will not allow entry of a starting database address that will cause the service object to run past the end of the database. The highest valid database address, therefore, will depend on the targeted data type, as well as the number of items to be accessed.

Multiplier

Applies to register types only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, raw data is multiplied by the multiplier to produce a network value. Similarly, network values are divided by the multiplier before being stored into the database.

Note that the multiplier imposes range limitations on network data values. For example, if the multiplier is 0.01, then the network data can achieve a maximum value of only 655 (since 65535 is the maximum value that can be stored in 16 bits in the database).

Read Enable and Function Code Selection

Check **Read** to enable reading (the service object will continuously read from the slave unless a pending **Write** exists). When reads are enabled, the desired read **Function Code** can be selected in the drop-down box.

Write Enable and Function Code Selection

Applies to holding register and coil status types only. Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted slave). When writes are enabled, the desired write **Function Code** can be selected in the drop-down box.

Group Multiple Writes

Applies to holding register and coil status types with writes enabled only. This checkbox is used to indicate whether the gateway should group writes to multiple holding registers or coils into one packet, or send separate write packets for each one. Check this box to enable the grouping of multiple writes into one write packet.

For holding register types: note that this feature is only available with function code **16 (Preset Multiple Registers)**.

For coil status types: note that this feature is always enabled with function code **15 (Force Multiple Coils)**.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.10.2.1 32-Bit Extension Options

Applies to register types only. If the target registers are associated with the Enron/Daniel extension to the Modbus specification, or are represented by 32-bit values, check the **Enable Enron/Daniel** checkbox to enable the 32-bit extension option. The following describes each of the extension options:

Floating Point

Enable **Floating Point** if the transmitted values are encoded in IEEE 754 floating point format.

Big Endian

Enable **Big Endian** if the transmitted values are encoded in big-endian, 16-bit word order, i.e. the most significant 16-bit word is before the least significant 16-bit word.

Word-Size Reg

Enable **Word-Size Reg** if each target register is 16-bits wide, but two 16-bit registers comprise one 32-bit value. If not enabled, each of the target registers is assumed to be 32-bits wide.

Note that when **Word-Size Reg** is enabled, the **Num Regs** field name changes to **Num Reg Pairs**, indicating the number of pairs of 16-bit wide registers to address. When enabled, each register pair will use two register addresses and the selected **Data Type** will be applicable for the register pair, not the individual registers. For example, if the **Start Reg** is 100, **Num Reg Pairs** is 2, and **Data Type** is 32-bit Unsigned, then register numbers 100 – 103 will be accessed by the service object, with registers 100 and 101 stored as the first 32-bit Unsigned value and registers 102 and 103 stored as the next 32-bit Unsigned value in the gateway's database.

Word Count

Enable **Word Count** to encode the number of 16-bit words to be transferred in the Modbus "quantity of registers" field. If not enabled, the number of 32-bit registers will be used in the "quantity of registers" field.

Data Type

This field specifies how many bytes are used to store data for each register (or register pair) in this service object, as well as whether the value should be treated as signed or unsigned when converted to a floating point number for transmission over the network. Select the desired data type from this dropdown menu.

Note that each data type has different range limitations: 16-bit data types can represent values up to 65,535, and 32-bit data types can represent values up to 4,294,967,295.

8.7.10.3 Configuration Example

This example will configure the gateway for end-to-end communication using Modbus/TCP Server and Modbus RTU Master.

Let's assume that we have an installation that has recently been upgraded to use Ethernet communications and will now be controlled by a PLC that uses Modbus/TCP. However, we wish to retain our existing Modbus RTU-capable adjustable-speed drives, because we are familiar with them and know them to be reliable. Let's further assume that the registers used to communicate to an existing representative drive are holding registers 2001 – 2100.

Configure the Ethernet port using the above requirements

- Connect the ETH-1000 to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).
- Click on the **Ethernet Configuration** tab.
- Click **Use a static IP address**, and enter the IP address settings that your PLC is configured to communicate with.

- Check the **Modbus/TCP Server** checkbox in the **Protocol** dropdown menu, and then click on the **Modbus/TCP** subtab.
- Because the desired register range (2001 – 2100) exceeds the default register access mapping of the database, we must create a register remap object to expose a portion of the database at the alternate register range:
    - We can create a single register remap object to remap holding registers 2001 – 2100 to database addresses 0 – 199.
        - Enter “2001” into the **Start Reg** field.
        - Enter “100” into the **Num Regs** field.
        - Enter “0” into the **Database Addr** field.
        - Click **Create**.

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **Modbus RTU Master** from the protocol dropdown menu.
- Enter the **Baud Rate** and **Parity** settings to match that of the drive.
- Create a Service Object to read and write the holding registers:
    - We can create a single service object to provide read and write access to holding registers 2001 - 2100.
        - Select **Holding Register** from the **Type** selection group.
        - Enter the address of the drive into the **Dest Address** field.
        - Enter “2001” into the **Start Reg** field.
        - Enter “100” into the **Num Regs** field.
        - Enter “0” into the **Database Addr** field.
        - Click **Create**.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway with your PLC.

8.7.11 Modbus RTU Slave

To configure the gateway for Modbus RTU Slave, click on the **RS-485 Configuration** tab and select **Modbus RTU Slave** in the protocol dropdown menu. By default, the gateway's entire database is accessible via the register mapping discussed in section 11.5.2.2.

8.7.11.1 Protocol Selection Group

Protocol

Select **Modbus RTU Slave** from this dropdown menu.

Baud Rate

Select the desired network baud rate from this dropdown menu.

Parity

Select the desired network parity and number of stop bits from this dropdown menu.

Address

This field is the slave address at which the device will reside on the network. Enter a value between 1 and 247.

Timeout Time

Refer to section 8.3.1.

Response Delay

This field is used to set the time, in milliseconds, the device waits before responding to master requests. This may be useful if the Modbus master communicating to the gateway requires additional time before it can process a response to its request. If no delay is required, set this field to 0.

8.7.11.2 Register Remap Object

Optionally, registers can be remapped to different database addresses from their default mapping using a register remap object. It also allows the user to map a register that is not mapped into the database by default (any register above 2048) to an address in the database. The register remap object can remap a range of consecutive registers to any starting address in the database (as long as the entire range is within the database).

Note that registers can be accessed as either holding registers or input registers. Accessing either type refers to the same register on the gateway.

The following describes the configurable fields for a register remap object. For more information on register remap object editing options, refer to section 8.6.

Type

This group designates the Modbus register type(s) that this object will remap. Choose Holding Register and/or Input Register to assign which register type(s) to remap.

Description

This field is a description of the register remap object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 16 characters in length.

Start Reg

This field is the starting register number for a range of registers to be remapped. Enter a value between 1 and 65535 (0x1 – 0xFFFF).

Num Regs

This field is the number of registers to remap. Enter a value of 1 or more.

Database Addr

This field is the database address where the remapping begins. Enter a value between 0 and 4094 (0x0 – 0xFFE).

Multiplier

This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, raw data is multiplied by the multiplier to produce a network value. Similarly, network values are divided by the multiplier before being stored into the database.

Note that the multiplier imposes range limitations on network data values. For example, if the multiplier is 0.01, then the network data can achieve a maximum value of only 655 (since 65535 is the maximum value that can be stored in 16 bits in the database).

8.7.11.2.1 32-Bit Extension Options

If the target registers are associated with the Enron/Daniel extension to the Modbus specification, or are represented by 32-bit values, check the **Enable Enron/Daniel** checkbox to enable the 32-bit extension option. The following describes each of the extension options:

Floating Point

Enable **Floating Point** if the transmitted values are to be encoded in IEEE 754 floating point format.

Big Endian

Enable **Big Endian** if the transmitted values are to be encoded in big-endian, 16-bit word order, i.e. the most significant 16-bit word is before the least significant 16-bit word.

Word-Size Register

Enable **Word-Size Register** if each target register is 16-bits wide, but two 16-bit registers are to comprise one 32-bit value. If not enabled, each of the target registers is assumed to be 32-bits wide.

Note that when **Word-Size Register** is enabled, the **Num Regs** field name changes to **Num Reg Pairs**, indicating the number of pairs of 16-bit wide registers to be addressed. When enabled, each register pair will use two register addresses and the selected **Data Type** will be applicable for the register pair, not the individual registers. For example, if the **Start Reg** is 100, **Num Reg Pairs** is 2, and **Data Type** is 32-bit Unsigned, then register numbers 100 – 103 will be remapped, with registers 100 and 101 representing the first 32-bit Unsigned value and registers 102 and 103 representing the next 32-bit Unsigned value in the gateway's database.

Word Count

Enable **Word Count** to interpret the Modbus "quantity of registers" field as the number of 16-bit words to be transferred. If not enabled, the "quantity of registers" field will be interpreted as the number of 32-bit registers to be transferred.

Data Type

This field specifies how many bytes are used to store data for each register (or register pair), as well as whether the internal value should be treated as signed or unsigned when converted to a floating point number for transmission over the network. Select the desired data type from this dropdown menu.

Note that each data type has different range limitations: 16-bit data types can represent values up to 65,535, and 32-bit data types can represent values up to 4,294,967,295.

8.7.11.3 Configuration Example

This example will configure the gateway for end-to-end communication using Modbus RTU Slave and EtherNet/IP Server.

Say, for instance, we wish to share data between two PLC's. One PLC only supports Modbus, while the other only supports EtherNet/IP. Since the PLC that supports Modbus is already the lone master for the Modbus network, we cannot set the gateway as a Modbus Master. Therefore the gateway must be set as a Modbus Slave on the RS-485 port. Let's assume that the Modbus PLC exchanges information from different floors of a building. There are 3 floors. Floor 1 has 3 registers at address 1000, 1001, and 1002 for monitoring the floor status

and 3 registers at address 1003, 1004, and 1005 for executing commands on the floor. Similarly floors 2 and 3 have the same registers for monitoring and commanding starting at register 2000 for floor 2 and starting at 3000 for floor 3.

Configure the Ethernet port using the above requirements

- Connect the gateway to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).
- Click on the **Ethernet Configuration** subtab.
- Click **Use a static IP address** and enter the IP address settings that your PLC is configured to communicate with.
- Check the checkbox next to **EtherNet/IP Server** in the **Protocol** dropdown and then click on the **EtherNet/IP Server** subtab.
- Enter "0" into the **Produced Data Start Address** field.
- Enter "100" into the **Consumed Data Start Address** field.

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **Modbus RTU Slave** from the protocol dropdown menu.
- Enter the **Baud Rate** and **Parity** settings to match that of the PLC.
- Enter the **Address** for the gateway to reside at on the network.
- Create register remap objects to map the registers into produced and consumed data blocks, since these must be exchanged on the EtherNet/IP side of the gateway. The produced data will start at database address 0 and the consumed data will start at database address 100.

    - Remap floor 1's monitor data registers:
        - Enter "1000" into the **Start Reg** field.
        - Enter "3" into the **Num Regs** field.
        - Enter "0" into the **Database Addr** field.
        - Click **Create**.

    - Remap floor 1's command data registers:
        - Enter "1003" into the **Start Reg** field.
        - Enter "3" into the **Num Regs** field.
        - Enter "100" into the **Database Addr** field.
        - Click **Create**.

    - Remap floor 2's monitor data registers:
        - Enter "2000" into the **Start Reg** field.
        - Enter "3" into the **Num Regs** field.
        - Enter "6" into the **Database Addr** field.
        - Click **Create**.

    - Remap floor 2's command data registers:
        - Enter "2003" into the **Start Reg** field.
        - Enter "3" into the **Num Regs** field.
        - Enter "106" into the **Database Addr** field.
        - Click **Create**.

- Remap floor 3's monitor data registers:
  - Enter "3000" into the **Start Reg** field.
  - Enter "3" into the **Num Regs** field.
  - Enter "12" into the **Database Addr** field.
  - Click **Create**.

- Remap floor 1's command data registers:
  - Enter "3003" into the **Start Reg** field.
  - Enter "3" into the **Num Regs** field.
  - Enter "112" into the **Database Addr** field.
  - Click **Create**.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway with your PLCs.

Where are the monitor and command values?

Modbus Register	EtherNet/IP Byte
Floor 1 Monitor Data 1 (Register 1000)	Produced Data Byte 1 – Lower byte Produced Data Byte 2 – Upper byte
Floor 1 Monitor Data 2 (Register 1001)	Produced Data Byte 3 – Lower byte Produced Data Byte 4 – Upper byte
Floor 1 Monitor Data 3 (Register 1002)	Produced Data Byte 5 – Lower byte Produced Data Byte 6 – Upper byte
Floor 2 Monitor Data 1 (Register 2000)	Produced Data Byte 7 – Lower byte Produced Data Byte 8 – Upper byte
Floor 2 Monitor Data 2 (Register 2001)	Produced Data Byte 9 – Lower byte Produced Data Byte 10 – Upper byte
Floor 2 Monitor Data 3 (Register 2002)	Produced Data Byte 11 – Lower byte Produced Data Byte 12 – Upper byte
Floor 3 Monitor Data 1 (Register 3000)	Produced Data Byte 13 – Lower byte Produced Data Byte 14 – Upper byte
Floor 3 Monitor Data 2 (Register 3001)	Produced Data Byte 15 – Lower byte Produced Data Byte 16 – Upper byte
Floor 3 Monitor Data 3 (Register 3002)	Produced Data Byte 17 – Lower byte Produced Data Byte 18 – Upper byte
Floor 1 Command Data 1 (Register 1003)	Consumed Data Byte 1 – Lower byte Consumed Data Byte 2 – Upper byte
Floor 1 Command Data 2 (Register 1004)	Consumed Data Byte 3 – Lower byte Consumed Data Byte 4 – Upper byte
Floor 1 Command Data 3 (Register 1005)	Consumed Data Byte 5 – Lower byte Consumed Data Byte 6 – Upper byte
Floor 2 Command Data 1 (Register 2003)	Consumed Data Byte 7 – Lower byte Consumed Data Byte 8 – Upper byte
Floor 2 Command Data 2 (Register 2004)	Consumed Data Byte 9 – Lower byte Consumed Data Byte 10 – Upper byte
Floor 2 Command Data 3 (Register 2005)	Consumed Data Byte 11 – Lower byte Consumed Data Byte 12 – Upper byte
Floor 3 Command Data 1 (Register 3003)	Consumed Data Byte 13 – Lower byte Consumed Data Byte 14 – Upper byte
Floor 3 Command Data 2 (Register 3004)	Consumed Data Byte 15 – Lower byte Consumed Data Byte 16 – Upper byte
Floor 3 Command Data 3 (Register 3005)	Consumed Data Byte 17 – Lower byte Consumed Data Byte 18 – Upper byte

Note that the database is assumed to be little endian in this example.

8.7.12 Modbus RTU Sniffer

To configure the gateway for Modbus RTU Sniffer, click on the **RS-485 Configuration** tab and select **Modbus RTU Sniffer** in the protocol dropdown menu. The Modbus RTU Sniffer driver is passive (listen only), and uses service objects to define what registers to log values for from the network traffic. For more information on service objects, refer to section 8.5. Each register (input or holding) in a service object is mapped to 2 bytes in the database (the data type is fixed at 16-bit). For more information on register mapping, refer to section 11.5.1.3.

8.7.12.1 Protocol Selection Group

Protocol

Select **Modbus RTU Sniffer** from this dropdown menu.

Baud Rate

Select the desired network baud rate from this dropdown menu.

Parity

Select the desired network parity and number of stop bits from this dropdown menu.

8.7.12.2 Modbus Sniffer Service Object Configuration

This section describes the configurable fields for a Modbus sniffer service object. For more information on Modbus service object editing options, refer to section 8.6.

Type

This group designates the Modbus register type that this service object will log (capture data for). Choose from Holding Register or Input Register.

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 16 characters in length.

Dest Address

This field indicates the node address of the remote slave device on the network that contains the register(s) to be logged by this service object. Enter a value between 0 and 247. Note that address 0 is defined by Modbus as the broadcast address: if this address is used, the **Read** function checkbox must be unchecked, since slaves cannot respond to broadcast messages.

Note that using a destination address of 0 will configure the service object to only log broadcast messages; however, if a destination address other than 0 is used,

broadcast messages will also be logged for that service object as well as requests targeted specifically at the defined destination address.

Start Reg

This field defines the starting register number for a range of registers associated with this service object. Enter a value between 1 and 65535.

Num Regs

This field defines the number of registers associated with this service object. Enter a value between 1 and 125.

Database Addr

This field defines the database address where the first register of this service object will be mapped. Enter a value between 0 and 4095. Note that the configuration utility will not allow entry of a starting database address that will cause the service object to run past the end of the database. The highest valid database address, therefore, will depend on the targeted data type, as well as the number of items to be accessed.

Multiplier

This field is the amount that associated network values are scaled by prior to being stored into the database. Network values are divided by the multiplier before being stored into the database.

Note that the multiplier imposes range limitations on network data values. For example, if the multiplier is 0.01, then the network data can achieve a maximum value of only 655 (since 65535 is the maximum value that can be stored in 16 bits in the database).

Read Enable and Function Code Selection

Check **Read** to enable read function logging (the service object will log reads from the master to the slave). When reads are enabled, the desired read **Function Code** can be selected in the drop-down box.

Write Enable and Function Code Selection

Applies to holding register only. Check **Write** to enable write function logging (the service object will log writes from the master to the slave). When writes are enabled, the desired write **Function Code** can be selected in the drop-down box.

Note that the Modbus sniffer driver allows for both function codes 6 and 16 to be logged simultaneously so that if a register is written using either of these two function codes, it will be logged into the gateway's database.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at

which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

Note that the reflect status information for the Modbus sniffer driver is slightly different than that of the Modbus RTU master driver, because the sniffer driver does not actually transmit any requests itself. The status information should be interpreted from the perspective of the network master (as if the master were updating the status information). For example, when the master transmits a request to read a register, the TX Counter is incremented, and when the slave responds, the RX Counter is incremented.

8.7.12.3 Configuration Example

This example will configure the gateway for communication using the Modbus RTU Sniffer driver.

Say, for instance, we wish to monitor the communication between an adjustable-speed drive (the slave) and a PLC (the master), storing the transferred data values in the gateway's database for access by another network on the gateway. This scenario allows the gateway to expose data values on the Modbus network in a non-intrusive manner, which simplifies installation and nearly eliminates integration effort when applied to an already-functioning Modbus network. In this case, we wish to monitor the drive's output frequency, output current and output voltage, located at input registers 201, 202, and 203, respectively. We'd also like to monitor the frequency command (as commanded by the master) at holding register 14.

Configure the gateway using the above requirements

- Connect the gateway to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **Modbus RTU Sniffer** from the protocol dropdown menu.
- Enter the **Baud Rate** and **Parity** settings to match that of the Modbus network.
- Create Service Objects to log data from the desired registers:
    - We can create one service object to monitor the output frequency, output current and output voltage.
        - Select **Input Register** from the **Type** selection group.
        - Enter the address of the drive into the **Dest Address** field.
        - Enter "201" into the **Start Reg** field.
        - Enter "3" into the **Num Regs** field.
        - Enter "0" into the **Database Addr** field.
        - Click **Create**.

- To monitor the drive's frequency command, we must create a second service object for that register.
  - Select **Holding Register** from the **Type** selection.
  - Enter the address of the drive into the **Dest Address** field.
  - Enter "14" into the **Start Reg** field.
  - Enter "1" into the **Num Regs** field.
  - Enter "6" into the **Database Addr** field.
  - Click **Create**.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway to the Modbus network.

Where are the monitor values?

Drive's Modbus Register	Database Address
output frequency (input register 201)	0 & 1
output current (input register 202)	2 & 3
output voltage (input register 203)	4 & 5
frequency command (holding register 14)	6 & 7

8.7.13 BACnet MS/TP Client

BACnet MS/TP Client can be configured on the RS-485 port of the gateway by selecting **BACnet MS/TP Client** from the protocol dropdown menu. The gateway can read and write the present value property of BACnet objects hosted by other devices on the network. This behavior is defined by configuring BACnet service objects. For more information on service objects, refer to section 8.5. Whenever the BACnet MS/TP client driver is enabled, the BACnet device object is always present and must be properly configured. This section will discuss how to configure the BACnet MS/TP client.

8.7.13.1 Protocol Selection Group

This section describes the fields that must be configured on the RS-485 port.

Protocol

Select **BACnet MS/TP Client** from this dropdown menu.

Baud Rate

Select the network baud rate from this dropdown menu.

Address

This field is the node address that the gateway will reside at on the network. Enter a value between 0 and 127.

Scan Rate

This is the time the device will wait between sending requests. This may be useful if BACnet devices that the gateway is communicating with require additional time between requests. If no additional time is required, set this field to 0.

8.7.13.2 Device Object Configuration Group

The Device Object Configuration group contains several fields that must be appropriately set for each device residing on a BACnet network.

Device Name

This field is the BACnet Device Object's name. The device name must be unique across the entire BACnet network. Enter a string of between 1 and 16 characters in length.

Instance Number

This field is the BACnet Device Object's instance number. The instance number must be unique across the entire BACnet network. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Max Master

This field is the highest allowable address for MS/TP master nodes on the network. Any address higher than this will not receive the token from the gateway. Enter a value between 0 and 127. Note that this value must be greater than or equal to the configured Address for the gateway. If the highest address on the network is unknown, set this field to 127.

Configuration tip: The Address and Max Master fields greatly affect network performance. For best results, set all device addresses consecutively, starting with address 0, ending with a device with a configurable Max Master field at the highest address. Then set that device's Max Master field to its network address. This will prevent any unnecessary poll for master packets on the network and thereby maximize efficiency.

8.7.13.3 BACnet Service Object Configuration

The following describes the configurable fields for a BACnet service object. For more information on BACnet service object editing options, refer to section 8.6.

Type

The radio buttons in this group select the BACnet object type. Choose from Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, or Binary Value.

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of between 1 and 16 characters in length.

Dest Dev Inst ("Destination Device Instance")

This field is the destination device instance of the BACnet device the gateway should send requests to for this service object. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Note that the gateway uses this value for dynamic device binding to determine the address of the destination device. If the destination device does not support dynamic device binding, then static device binding must be used. For more information on device binding, refer to section 11.6.2.3.

Use Static Device Binding

This checkbox is used to manually define the destination device network address. This feature must be used for all MS/TP slave devices, and for any MS/TP master devices that do not support dynamic device binding. For more information on device binding, refer to section 11.6.2.3.

Dest Address

*Note that this field is available only when the **Use Static Device Binding** checkbox is checked.* This field is used to manually define the address of the BACnet device that the gateway should target for this service object. Enter a value between 0 and 127.

Start Inst

This field is the starting instance number for a range of BACnet objects for this service object. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Num Insts

This field is the number of BACnet objects in this service object. Enter a value of 1 or more.

Database Addr

This field is the database address where the first BACnet object of this service object will be mapped. Enter a value between 0 and 4095 (0x0 – 0xFFF).

Note that the configuration utility will not allow entry of a starting database address that will cause the service object to run past the end of the database. The highest valid database address, therefore, will depend on the targeted data type, as well as the number of items to be accessed.

Data Type

Applies to analog objects only. This field specifies how many bytes are used to store present value data for each BACnet object in this service object, as well as whether the value should be treated as signed or unsigned when converted to a real number for transmission over the network. Select the desired data type from this dropdown menu.

Note that each data type has its own range limitations: 8-bit can have values up to 255, 16-bit can have values up to 65,535, and 32-bit can have values up to 4,294,967,295.

Multiplier

Applies to analog objects only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, the data is multiplied by the multiplier to produce a network value. Similarly, the network values are divided by the multiplier before being stored into the database.

Note that the multiplier, coupled with the data type, imposes range limitations on the network data value. For example, if the data type is 8-bit and the multiplier is 0.5, then the network data can have values only up to 127 (since 255 is the maximum value that can be stored in 8 bits).

Read Enable and Function Code Selection

Check **Read** to enable reading (the service object will continuously read from the remote device unless a pending **Write** exists). When reads are enabled, the desired read **Function Code** can be selected in the drop-down box.

Write Enable and Function Code Selection

Does not apply to input objects. Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted remote device). When writes are enabled, the desired write **Function Code** can be selected in the drop-down box.

Priority

This field is used to specify the priority associated with writes for this service object. Select the desired priority from the dropdown menu.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.13.4 Configuration Example

This example will configure the gateway for end-to-end communication using BACnet/IP Server and BACnet MS/TP Client.

Let's assume that we have an installation that has recently been upgraded to use Ethernet communications and will now be controlled by a PLC that uses BACnet/IP. However, we wish to retain our existing BACnet MS/TP-capable adjustable-speed drives, because we are familiar with them and know them to be reliable. Let's further assume that the PLC was programmed to monitor analog inputs 401 and 402 and binary inputs 1101 and 1102, and to command analog outputs 250 and 251 and binary outputs 1670 and 1671. We will also assume that all analog objects are 16-bit, unsigned values.

Configure the Ethernet port using the above requirements

- Connect the gateway to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).
- Click on the **Ethernet Configuration** subtab.
- Click **Use a static IP address** and enter the IP address settings that your PLC is configured to communicate with.
- Check the checkbox next to **BACnet/IP Server** in the **Protocol** dropdown menu and then click on the **BACnet/IP** subtab.
- Create the BACnet objects
    - Create the analog input objects
      For the first analog input, enter the following:
        - Select **Analog Input** from the **Type** selection group.

- Enter “401” into the **Instance** field.
- Enter “0” into the **Database Addr** field (the addresses for all BACnet/IP objects are arbitrary at this point, but must match the BACnet MS/TP configuration which will be completed later.)
- Select **16-bit Unsigned** from the **Data Type** dropdown.
- Click **Create**.

For the second analog input, enter the following:

- Select **Analog Input** from the **Type** selection group.
- Enter “402” into the **Instance** field.
- Enter “2” into the **Database Addr** field.
- Select **16-bit Unsigned** from the **Data Type** dropdown.
- Click **Create**.

o Create the binary input objects

For the first binary input, enter the following:

- Select **Binary Input** from the **Type** selection group.
- Enter “1101” into the **Instance** field.
- Enter “100” into the **Database Addr** field.
- Enter “0x01” into the **Bitmask** field (this maps the binary input to the least-significant bit of the byte at database address 100.)
- Click **Create**.

For the second binary input, enter the following:

- Select **Binary Input** from the **Type** selection group.
- Enter “1102” into the **Instance** field.
- Enter “100” into the **Database Addr** field.
- Enter “0x02” into the **Bitmask** field (this maps the binary input to the second least-significant bit of the byte at database address 100.)
- Click **Create**.

o Create the analog output objects

For the first analog output, enter the following:

- Select **Analog Output** from the **Type** selection group.
- Enter “250” into the **Instance** field.
- Enter “200” into the **Database Addr** field.
- Select **16-bit Unsigned** from the **Data Type** dropdown.
- Click **Create**.

For the second analog output, enter the following:

- Select **Analog Output** from the **Type** selection group.
- Enter “251” into the **Instance** field.
- Enter “202” into the **Database Addr** field.
- Select **16-bit Unsigned** from the **Data Type** dropdown.
- Click **Create**.

o Create the binary output objects

For the first binary output, enter the following:

- Select **Binary Output** from the **Type** selection group.
- Enter “1670” into the **Instance** field.
- Enter “300” into the **Database Addr** field.

- Enter “0x01” into the **Bitmask** field (this maps the binary output to the least-significant bit of the byte at database address 300.)
- Click **Create**.

For the second binary output, enter the following:

- Select **Binary Output** from the **Type** selection group.
- Enter “1671” into the **Instance** field.
- Enter “300” into the **Database Addr** field.
- Enter “0x02” into the **Bitmask** field (this maps the binary output to the second least-significant bit of the byte at database address 300.)
- Click **Create**.

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **BACnet MS/TP Client** from the protocol dropdown menu.
- Enter the **Baud Rate** and **Parity** settings to match that of the drive.
- Enter the **Address** at which the gateway will reside on the network.
- Enter a **Device Name,** device **Instance Number**, and the **Max Master** for the gateway.
- Create Service Objects to correspond to the BACnet/IP objects that were created in the previous step:
  - We can create a single service object for both analog inputs
    - Select **Analog Input** from the **Type** selection group.
    - Enter the device instance of the drive in the **Dest Dev Inst** field.
    - Enter “401” into the **Start Inst** field.
    - Enter “2” into the **Num Insts** field.
    - Enter “0” into the **Database Addr** field.
    - Select **16-bit Unsigned** from the **Data Type** dropdown.
    - Click **Create**.
  - We can also create a single service object for both binary inputs
    - Select **Binary Input** from the **Type** selection group.
    - Enter the device instance of the drive in the **Dest Dev Inst** field.
    - Enter “1101” into the **Start Inst** field.
    - Enter “2” into the **Num Insts** field.
    - Enter “100” into the **Database Addr** field.
    - Click **Create**.
  - Similarly, we can create a single service object for both analog outputs
    - Select **Analog Output** from the **Type** selection group.
    - Enter the device instance of the drive in the **Dest Dev Inst** field.
    - Enter “250” into the **Start Inst** field.
    - Enter “2” into the **Num Insts** field.
    - Enter “200” into the **Database Addr** field.
    - Select **16-bit Unsigned** from the **Data Type** dropdown.
    - Click **Create**.

- We can also create a single service object for both binary outputs
  - Select **Binary Output** from the **Type** selection group.
  - Enter the device instance of the drive in the **Dest Dev Inst** field.
  - Enter “1670” into the **Start Inst** field.
  - Enter “2” into the **Num Insts** field.
  - Enter “300” into the **Database Addr** field.
  - Click **Create**.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway with your PLC.

8.7.14 BACnet MS/TP Server

BACnet MS/TP Server can be configured on the RS-485 port of the gateway by selecting **BACnet MS/TP Server** from the protocol dropdown menu. The BACnet MS/TP server can host a wide variety of user-defined BACnet objects. Whenever the BACnet MS/TP server is enabled, the BACnet device object is always present and must be properly configured. This section will discuss how to configure the BACnet MS/TP server.

8.7.14.1 Protocol Selection Group

This section describes the fields that must be configured on the RS-485 port.

Protocol

Select **BACnet MS/TP Server** from this dropdown menu.

Baud Rate

Select the network baud rate from this dropdown menu.

Address

This field is the node address that the gateway will reside at on the network. Enter a value between 0 and 127.

8.7.14.2 Device Object Configuration Group

The Device Object Configuration group contains several fields that must be appropriately set for each device residing on a BACnet network.

Device Name

This field is the BACnet Device Object's name. The device name must be unique across the entire BACnet network. Enter a string of between 1 and 16 characters in length.

Instance Number

This field is the BACnet Device Object's instance number. The instance number must be unique across the entire BACnet network. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Max Master

This field is the highest allowable address for master nodes on the network. Any address higher than this will not receive the token from the gateway. Enter a value between 0 and 127. Note that this value must be greater than or equal to the configured Address for the gateway. If the highest address on the network is unknown, set this field to 127.

Configuration tip: The Address and Max Master fields greatly affect network performance. For best results, set all device addresses consecutively, starting

with address 0, ending with a device with a configurable Max Master field at the highest address. Then set that device's Max Master field to its address. This will prevent any unnecessary poll for master packets on the network and thereby maximize efficiency.

8.7.14.3 BACnet Object Common Configurable Fields

This section describes the common configurable fields for all BACnet objects. For more information on BACnet object editing options, refer to section 8.6.

Type

The radio buttons in this group select the BACnet object type. Choose from Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, or Binary Value.

Object Name

This field is the name of the BACnet object. Enter a string of between 1 and 16 characters in length. All object names must be unique within the gateway.

Instance

This field is the BACnet Object's instance number. Enter a value between 0 and 4194302 (0x0 – 0x3FFFFE).

Database Addr

This field is the database address where the BACnet object's present value will reside. Enter a value between 0 and 4095 (0x0 – 0xFFF).

A note for analog objects: Depending on the designated Data Type, the maximum allowable database address is 4095, 4094, or 4092 for 8-bit, 16-bit, or 32-bit sized objects, respectively.

Multiplier

Applies to analog objects only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, the data is multiplied by the multiplier to produce a network value. Similarly, the network values are divided by the multiplier before being stored into the database.

Note that the multiplier, coupled with the data type, imposes range limitations on the network data value. For example, if the data type is 8-bit and the multiplier is 0.5, then the network data can have values only up to 127 (since 255 is the maximum value that can be stored in 8 bits).

Units

Applies to analog objects only. Select the desired units from this dropdown menu. If the desired units are not available in the dropdown menu, select **Other**

Units and enter the enumerated value (as defined by the BACnet Specification) in the **Unit Value** field.

Bitmask

Applies to binary objects only. This 8-bit field specifies which bit(s) in the byte designated by the **Database Addr** that the binary object will map to. This allows up to 8 binary objects to be simultaneously assigned to one database address (each binary object mapping to a single bit of that byte in the database). It is possible to map binary objects to multiple bits within the designated database location. Such a configuration allows (for example) the modification of multiple selected database bits via a single binary output.

The effect of the **Bitmask** field when writing: When the present value property of a binary output object or binary value object is set to "active" by a BACnet client, then the bit(s) in the designated **Database Addr** indicated by a "1" in the bitmask are set. Similarly, when the present value property of the object is set to "inactive", then the bit(s) in the designated **Database Addr** indicated by a "1" in the bitmask are cleared. For binary output objects, this setting/clearing behavior is reversed if the object's **Polarity** is set to "Reversed".

The effect of the **Bitmask** field when reading: When the present value property of a binary object is read by a BACnet client, the **Bitmask** is used to determine the active/inactive state of the object by inspecting the value in the designated database address at the bit location(s) indicated in the **Bitmask**. If all of the bit locations at the designated database address indicated by a "1" in the **Bitmask** are set, then the object's state will be returned as "active". Else, the object's state will be returned as "inactive". For binary input and binary output objects, the resultant state is reversed just prior to being placed on the network if the object's **Polarity** is set to "Reversed".

Active Text

Applies to binary objects only. This field specifies the description of the object's "active" state. Enter a string of up to 8 characters in length. This field is optional and may be left blank.

Inactive Text

Applies to binary objects only. This field specifies the description of the object's "inactive" state. Enter a string of up to 8 characters in length. This field is optional and may be left blank.

Polarity

Applies to binary input and binary output objects only. This field indicates the relationship between the physical state of the object (as stored in the gateway's database) and the logical state represented by the object's present value property. If the physical state is active high, select **Normal** from this dropdown menu. If the physical state is active low, select **Reverse** from this dropdown menu. For further detail, refer to the **Bitmask** behavioral description above.

Data Type

Applies to analog objects only. This field specifies how many bytes are allocated for the present value data, as well as whether the value should be treated as signed or unsigned when converting it to a real number to send over the network. Select the desired data type from this dropdown menu.

Note that each data type has its own range limitations: 8-bit data types can have values up to 255, 16-bit data types can have values up to 65,535, and 32-bit data types can have values up to 4,294,967,295.

Relinquish Def

This field is the default value to be used for an object's present value property when all command priority values in the object's priority array are NULL. Note that this property only exists for those objects that implement a priority array (analog output, analog value, binary output and binary value objects).

8.7.14.4 Configuration Example

This example will configure the gateway for end-to-end communication using BACnet MS/TP Server and EtherNet/IP Server.

Say, for instance, we wish to share data between two PLC's. One PLC only supports BACnet, while the other only supports EtherNet/IP. Let's assume that the BACnet PLC exchanges information from different floors of a building. There are 3 floors; floor 1 has 3 analog values at instances 1000, 1001, and 1002 for monitoring the floor status and 3 analog values at instances 1003, 1004, and 1005 for executing commands on the floor. Similarly, floors 2 and 3 have the same analog values for monitoring and commanding starting at instance 2000 for floor 2, and starting at instance 3000 for floor 3.

Configure the Ethernet port using the above requirements

- Connect the gateway to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).
- Click on the **Ethernet Configuration** subtab.
- Click **Use a static IP address** and enter the IP address settings that your PLC is configured to communicate with.
- Check the checkbox next to **EtherNet/IP Server** in the **Protocol** dropdown menu, and then click on the **EtherNet/IP Server** subtab.
- Enter "0" into the **Produced Data Start Address** field.
- Enter "100" into the **Consumed Data Start Address** field.

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **BACnet MS/TP Server** from the protocol dropdown menu.
- Enter the **Baud Rate** settings to match that of the PLC.
- Enter the **Address** at which the gateway will reside on the network.
- Enter a **Device Name,** device **Instance Number**, and the **Max Master** for the gateway.

- Create BACnet objects to map the data from the PLC into produced and consumed data blocks which will then be exchanged on the EtherNet/IP side of the gateway. The produced data will start at database address 0 and the consumed data will start at database address 100.

    - Create objects for floor 1's monitor data
      For the first object, enter the following:
        - Select **Analog Value** from the **Type** selection group.
        - Enter "F1 Mon Data 1" into the **Object Name** field.
        - Enter "1000" into the **Instance** field.
        - Enter "0" into the **Database Addr** field.
        - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
        - Enter "1" into the **Multiplier** field.
        - Select **No Units (95)** from the **Units** dropdown menu.
        - Click **Create**.

      Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

    - Create objects for floor 1's command data
      For the first object, enter the following:
        - Select **Analog Value** from the **Type** selection group.
        - Enter "F1 Cmd Data 1" into the **Object Name** field.
        - Enter "1003" into the **Instance** field.
        - Enter "100" into the **Database Addr** field.
        - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
        - Enter "1" into the **Multiplier** field.
        - Select **No Units (95)** from the **Units** dropdown menu.
        - Click **Create**.

      Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

    - Create objects for floor 2's monitor data
      For the first object, enter the following:
        - Select **Analog Value** from the **Type** selection group.
        - Enter "F2 Mon Data 1" into the **Object Name** field.
        - Enter "2000" into the **Instance** field.
        - Enter "12" into the **Database Addr** field.
        - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
        - Enter "1" into the **Multiplier** field.
        - Select **No Units (95)** from the **Units** dropdown menu.
        - Click **Create**.

      Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

- Create objects for floor 2's command data
  For the first object, enter the following:
  - Select **Analog Value** from the **Type** selection group.
  - Enter "F2 Cmd Data 1" into the **Object Name** field.
  - Enter "2003" into the **Instance** field.
  - Enter "112" into the **Database Addr** field.
  - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
  - Enter "1" into the **Multiplier** field.
  - Select **No Units (95)** from the **Units** dropdown menu.
  - Click **Create**.

  Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

- Create objects for floor 3's monitor data
  For the first object, enter the following:
  - Select **Analog Value** from the **Type** selection group.
  - Enter "F3 Mon Data 1" into the **Object Name** field.
  - Enter "3000" into the **Instance** field.
  - Enter "24" into the **Database Addr** field.
  - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
  - Enter "1" into the **Multiplier** field.
  - Select **No Units (95)** from the **Units** dropdown menu.
  - Click **Create**.

  Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

- Create objects for floor 3's command data:
  For the first object, enter the following:
  - Select **Analog Value** from the **Type** selection group.
  - Enter "F3 Cmd Data 1" into the **Object Name** field.
  - Enter "3003" into the **Instance** field.
  - Enter "124" into the **Database Addr** field.
  - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
  - Enter "1" into the **Multiplier** field.
  - Select **No Units (95)** from the **Units** dropdown menu.
  - Click **Create**.

  Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway with your PLC.

Where are the monitor and command values?

BACnet Object	EtherNet/IP Byte
Floor 1 Monitor Data 1 (AV1000)	Produced Data Byte 1 – Low byte Produced Data Byte 2 – Middle byte Produced Data Byte 3 – High byte Produced Data Byte 4 – Upper byte
Floor 1 Monitor Data 2 (AV1001)	Produced Data Byte 5 – Low byte Produced Data Byte 6 – Middle byte Produced Data Byte 7 – High byte Produced Data Byte 8 – Upper byte
Floor 1 Monitor Data 3 (AV1002)	Produced Data Byte 9 – Low byte Produced Data Byte 10 – Middle byte Produced Data Byte 11 – High byte Produced Data Byte 12 – Upper byte
Floor 2 Monitor Data 1 (AV2000)	Produced Data Byte 13 – Low byte Produced Data Byte 14 – Middle byte Produced Data Byte 15 – High byte Produced Data Byte 16 – Upper byte
Floor 2 Monitor Data 2 (AV2001)	Produced Data Byte 17 – Low byte Produced Data Byte 18 – Middle byte Produced Data Byte 19 – High byte Produced Data Byte 20 – Upper byte
Floor 2 Monitor Data 3 (AV2002)	Produced Data Byte 21 – Low byte Produced Data Byte 22 – Middle byte Produced Data Byte 23 – High byte Produced Data Byte 24 – Upper byte
Floor 3 Monitor Data 1 (AV3000)	Produced Data Byte 25 – Low byte Produced Data Byte 26 – Middle byte Produced Data Byte 27 – High byte Produced Data Byte 28 – Upper byte
Floor 3 Monitor Data 2 (AV3001)	Produced Data Byte 29 – Low byte Produced Data Byte 30 – Middle byte Produced Data Byte 31 – High byte Produced Data Byte 32 – Upper byte
Floor 3 Monitor Data 3 (AV3002)	Produced Data Byte 33 – Low byte Produced Data Byte 34 – Middle byte Produced Data Byte 35 – High byte Produced Data Byte 36 – Upper byte
Floor 1 Command Data 1 (AV1003)	Consumed Data Byte 1 – Low byte Consumed Data Byte 2 – Middle byte Consumed Data Byte 3 – High byte Consumed Data Byte 4 – Upper byte
Floor 1 Command Data 2 (AV1004)	Consumed Data Byte 5 – Low byte Consumed Data Byte 6 – Middle byte Consumed Data Byte 7 – High byte Consumed Data Byte 8 – Upper byte

BACnet Object	EtherNet/IP Byte
Floor 1 Command Data 3 (AV1005)	Consumed Data Byte 9 – Low byte Consumed Data Byte 10 – Middle byte Consumed Data Byte 11 – High byte Consumed Data Byte 12 – Upper byte
Floor 2 Command Data 1 (AV2003)	Consumed Data Byte 13 – Low byte Consumed Data Byte 14 – Middle byte Consumed Data Byte 15 – High byte Consumed Data Byte 16 – Upper byte
Floor 2 Command Data 2 (AV2004)	Consumed Data Byte 17 – Low byte Consumed Data Byte 18 – Middle byte Consumed Data Byte 19 – High byte Consumed Data Byte 20 – Upper byte
Floor 2 Command Data 3 (AV2005)	Consumed Data Byte 21 – Low byte Consumed Data Byte 22 – Middle byte Consumed Data Byte 23 – High byte Consumed Data Byte 24 – Upper byte
Floor 3 Command Data 1 (AV3003)	Consumed Data Byte 25 – Low byte Consumed Data Byte 26 – Middle byte Consumed Data Byte 27 – High byte Consumed Data Byte 28 – Upper byte
Floor 3 Command Data 2 (AV3004)	Consumed Data Byte 29 – Low byte Consumed Data Byte 30 – Middle byte Consumed Data Byte 31 – High byte Consumed Data Byte 32 – Upper byte
Floor 3 Command Data 3 (AV3005)	Consumed Data Byte 33 – Low byte Consumed Data Byte 34 – Middle byte Consumed Data Byte 35 – High byte Consumed Data Byte 36 – Upper byte

Note that the database is assumed to be little endian in this example.

8.7.15 Metasys N2 Slave

Johnson Controls Metasys N2 slave can be configured on the RS-485 port of the gateway by selecting **Metasys N2 Slave** from the protocol dropdown menu. The Metasys N2 slave driver can host a wide variety of user-defined N2 objects. This section will discuss how to configure the Metasys N2 driver.

8.7.15.1 Protocol Selection Group

This section describes the fields that must be configured on the RS-485 port.

Protocol

Select **Metasys N2 Slave** from this dropdown menu.

Address

This field is the station address that the gateway will reside at on the network. Enter a value between 1 and 255.

Timeout Time

Refer to section 8.3.1.

Response Delay

This field is used to set the time, in milliseconds, the device waits before responding to master requests. This may be useful if the Metasys master communicating to the gateway requires additional time before it can process a response to its request. If no delay is required, set this field to 0.

8.7.15.2 Metasys Object Common Configurable Fields

This section describes the common configurable fields for all Metasys objects. For more information on Metasys object editing options, refer to section 8.6.

Type

The radio buttons in this group select the Metasys object type. Choose from Analog Input, Analog Output, Binary Input, or Binary Output.

Object Name

This field is a description of the Metasys object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 16 characters in length.

Instance

This field is the Metasys object's instance number. Metasys allows a maximum of 256 instances of each object type. Enter a value between 1 and 256 (0x1 – 0x100).

Database Addr

This field is the database address where the Metasys object's current value will reside. Enter a value between 0 and 4095 (0x0 – 0xFFF).

A note for analog objects: Depending on the designated Data Type, the maximum allowable database address is 4095, 4094, or 4092 for 8-bit, 16-bit, or 32-bit sized objects, respectively.

Multiplier

Applies to analog objects only. This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, the data is multiplied by the multiplier to produce a network value. Similarly, the network values are divided by the multiplier before being stored into the database.

Note that the multiplier, coupled with the data type, imposes range limitations on the network data value. For example, if the data type is 8-bit and the multiplier is 0.5, then the network data can have values only up to 127 (since 255 is the maximum value that can be stored in 8 bits).

Bitmask

Applies to binary objects only. This 8-bit field specifies which bit(s) in the byte designated by the **Database Addr** that the binary object will map to. This allows up to 8 binary objects to be simultaneously assigned to one database address (each binary object mapping to a single bit of that byte in the database). It is possible to map binary objects to multiple bits within the designated database location. Such a configuration allows (for example) the modification of multiple selected database bits via a single binary output.

The effect of the **Bitmask** field when writing: When the current state of a binary output object is overridden to "1" by a Metasys master, then the bit(s) in the designated **Database Addr** indicated by a "1" in the bitmask are set. Similarly, when the current state of the object is overridden to "0", then the bit(s) in the designated **Database Addr** indicated by a "1" in the bitmask are cleared.

The effect of the **Bitmask** field when reading: When the current state of a binary object is read by a Metasys master, the **Bitmask** is used to determine the state of the object by inspecting the value in the designated database address at the bit location(s) indicated in the **Bitmask**. If all of the bit locations at the designated database address indicated by a "1" in the **Bitmask** are set, then the object's state will be returned as "1". Else, the object's state will be returned as "0".

Data Type

Applies to analog objects only. This field specifies how many bytes are allocated for the object's current value, as well as whether the value should be treated as signed or unsigned when converting it to a real number to send over the network. Select the desired data type from this dropdown menu.

Note that each data type has its own range limitations: 8-bit data types can have values up to 255, 16-bit data types can have values up to 65,535, and 32-bit data types can have values up to 4,294,967,295.

8.7.15.3 Configuration Example

This example will configure the gateway for end-to-end communication using Metasys N2 and EtherNet/IP Server.

Say, for instance, we wish to share data between a Metasys N2 building automation system (BAS) and a PLC which supports EtherNet/IP. Let's assume that the BAS exchanges information from different floors of a building. There are 3 floors; floor 1 has 3 analog output object instances (AO1, AO2, and AO3) which the BAS updates with floor status information (perhaps actual temperatures), and 3 analog input object instances (AI1, AI2, and AI3) which the BAS reads to execute commands on the floor (perhaps to set thermostat setpoints). Similarly, floors 2 and 3 have the same analog objects for monitoring and commanding: AO4..AO6 and AI4..AI6 for floor 2, and AO7..AO9 and AI7..AI9 for floor 3.

Configure the Ethernet port using the above requirements

- Connect the gateway to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).
- Click on the **Ethernet Configuration** subtab.
- Click **Use a static IP address** and enter the IP address settings that your PLC is configured to communicate with.
- Check the checkbox next to **EtherNet/IP Server** in the **Protocol** dropdown menu, and then click on the **EtherNet/IP Server** subtab.
- Enter "0" into the **Produced Data Start Address** field.
- Enter "100" into the **Consumed Data Start Address** field.

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **Metasys N2 Slave** from the protocol dropdown menu.
- Enter the **Address** at which the gateway will reside on the network.
- Create Metasys objects to map the data from the PLC into produced and consumed data blocks which will then be exchanged on the Ethernet side of the gateway. The produced data will start at database address 0 and the consumed data will start at database address 100.

    - Create output objects for floor 1's monitor data
      For the first object, enter the following:
        - Select **Analog Output** from the **Type** selection group.
        - Enter "F1 Mon Data 1" into the **Object Name** field.
        - Enter "1" into the **Instance** field.
        - Enter "0" into the **Database Addr** field.
        - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
        - Enter "1" into the **Multiplier** field.

- Click **Create**.

Repeat these steps for the other two AO objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

o Create input objects for floor 1's command data
For the first object, enter the following:
- Select **Analog Input** from the **Type** selection group.
- Enter "F1 Cmd Data 1" into the **Object Name** field.
- Enter "1" into the **Instance** field.
- Enter "100" into the **Database Addr** field.
- Select **32-bit Unsigned** from the **Data Type** dropdown menu.
- Enter "1" into the **Multiplier** field.
- Click **Create**.

Repeat these steps for the other two AI objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

o Create output objects for floor 2's monitor data
For the first object, enter the following:
- Select **Analog Output** from the **Type** selection group.
- Enter "F2 Mon Data 1" into the **Object Name** field.
- Enter "4" into the **Instance** field.
- Enter "12" into the **Database Addr** field.
- Select **32-bit Unsigned** from the **Data Type** dropdown menu.
- Enter "1" into the **Multiplier** field.
- Click **Create**.

Repeat these steps for the other two AO objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

o Create input objects for floor 2's command data
For the first object, enter the following:
- Select **Analog Input** from the **Type** selection group.
- Enter "F2 Cmd Data 1" into the **Object Name** field.
- Enter "4" into the **Instance** field.
- Enter "112" into the **Database Addr** field.
- Select **32-bit Unsigned** from the **Data Type** dropdown menu.
- Enter "1" into the **Multiplier** field.
- Click **Create**.

Repeat these steps for the other two AI objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

- Create output objects for floor 3's monitor data
  For the first object, enter the following:
  - Select **Analog Output** from the **Type** selection group.
  - Enter "F3 Mon Data 1" into the **Object Name** field.
  - Enter "7" into the **Instance** field.
  - Enter "24" into the **Database Addr** field.
  - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
  - Enter "1" into the **Multiplier** field.
  - Click **Create**.

  Repeat these steps for the other two AO objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

- Create input objects for floor 3's command data:
  For the first object, enter the following:
  - Select **Analog Input** from the **Type** selection group.
  - Enter "F3 Cmd Data 1" into the **Object Name** field.
  - Enter "7" into the **Instance** field.
  - Enter "124" into the **Database Addr** field.
  - Select **32-bit Unsigned** from the **Data Type** dropdown menu.
  - Enter "1" into the **Multiplier** field.
  - Click **Create**.

  Repeat these steps for the other two objects, increasing the **Object Name** by 1, **Instance** by 1, and **Database Addr** by 4 each time.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway with your PLC.

Where are the monitor and command values?

Metasys Object	EtherNet/IP Byte
Floor 1 Monitor Data 1 (AO1)	Produced Data Byte 1 – Low byte Produced Data Byte 2 – Middle byte Produced Data Byte 3 – High byte Produced Data Byte 4 – Upper byte
Floor 1 Monitor Data 2 (AO2)	Produced Data Byte 5 – Low byte Produced Data Byte 6 – Middle byte Produced Data Byte 7 – High byte Produced Data Byte 8 – Upper byte
Floor 1 Monitor Data 3 (AO3)	Produced Data Byte 9 – Low byte Produced Data Byte 10 – Middle byte Produced Data Byte 11 – High byte Produced Data Byte 12 – Upper byte
Floor 2 Monitor Data 1 (AO4)	Produced Data Byte 13 – Low byte Produced Data Byte 14 – Middle byte Produced Data Byte 15 – High byte Produced Data Byte 16 – Upper byte
Floor 2 Monitor Data 2 (AO5)	Produced Data Byte 17 – Low byte Produced Data Byte 18 – Middle byte Produced Data Byte 19 – High byte Produced Data Byte 20 – Upper byte
Floor 2 Monitor Data 3 (AO6)	Produced Data Byte 21 – Low byte Produced Data Byte 22 – Middle byte Produced Data Byte 23 – High byte Produced Data Byte 24 – Upper byte
Floor 3 Monitor Data 1 (AO7)	Produced Data Byte 25 – Low byte Produced Data Byte 26 – Middle byte Produced Data Byte 27 – High byte Produced Data Byte 28 – Upper byte
Floor 3 Monitor Data 2 (AO8)	Produced Data Byte 29 – Low byte Produced Data Byte 30 – Middle byte Produced Data Byte 31 – High byte Produced Data Byte 32 – Upper byte
Floor 3 Monitor Data 3 (AO9)	Produced Data Byte 33 – Low byte Produced Data Byte 34 – Middle byte Produced Data Byte 35 – High byte Produced Data Byte 36 – Upper byte
Floor 1 Command Data 1 (AI1)	Consumed Data Byte 1 – Low byte Consumed Data Byte 2 – Middle byte Consumed Data Byte 3 – High byte Consumed Data Byte 4 – Upper byte
Floor 1 Command Data 2 (AI2)	Consumed Data Byte 5 – Low byte Consumed Data Byte 6 – Middle byte Consumed Data Byte 7 – High byte Consumed Data Byte 8 – Upper byte

Metasys Object	EtherNet/IP Byte
Floor 1 Command Data 3 (AI3)	Consumed Data Byte 9 – Low byte Consumed Data Byte 10 – Middle byte Consumed Data Byte 11 – High byte Consumed Data Byte 12 – Upper byte
Floor 2 Command Data 1 (AI4)	Consumed Data Byte 13 – Low byte Consumed Data Byte 14 – Middle byte Consumed Data Byte 15 – High byte Consumed Data Byte 16 – Upper byte
Floor 2 Command Data 2 (AI5)	Consumed Data Byte 17 – Low byte Consumed Data Byte 18 – Middle byte Consumed Data Byte 19 – High byte Consumed Data Byte 20 – Upper byte
Floor 2 Command Data 3 (AI6)	Consumed Data Byte 21 – Low byte Consumed Data Byte 22 – Middle byte Consumed Data Byte 23 – High byte Consumed Data Byte 24 – Upper byte
Floor 3 Command Data 1 (AI7)	Consumed Data Byte 25 – Low byte Consumed Data Byte 26 – Middle byte Consumed Data Byte 27 – High byte Consumed Data Byte 28 – Upper byte
Floor 3 Command Data 2 (AI8)	Consumed Data Byte 29 – Low byte Consumed Data Byte 30 – Middle byte Consumed Data Byte 31 – High byte Consumed Data Byte 32 – Upper byte
Floor 3 Command Data 3 (AI9)	Consumed Data Byte 33 – Low byte Consumed Data Byte 34 – Middle byte Consumed Data Byte 35 – High byte Consumed Data Byte 36 – Upper byte

Note that the database is assumed to be little endian in this example.

8.7.16 Toshiba ASD Master

To configure the gateway for Toshiba ASD Master, click on the **RS-485 Configuration** tab and select **Toshiba ASD Master** in the protocol dropdown menu. The Toshiba ASD Master protocol uses service objects to make requests. For more information on service objects, refer to section 8.5. Each parameter in a service object is mapped to 2 bytes in the database (the data size is fixed at 16-bit, as this is the native data size of Toshiba ASD parameters). For more information on parameter mapping, refer to section 11.9.3.

8.7.16.1 Protocol Selection Group

Protocol

Select **Toshiba ASD Master** from this dropdown menu.

Baud Rate

Select the desired network baud rate from this dropdown menu.

Parity

Select the desired network parity and number of stop bits from this dropdown menu.

Timeout

This is the time in milliseconds that the device will wait for a response from a drive after sending a request.

Scan Rate

This is the time the device will wait between sending requests. This may be useful if drives require additional time between requests. If no additional delay time is needed, set this field to 0. For more information, refer to section 8.5.1.

8.7.16.2 Toshiba Service Object Configuration

This section describes the configurable fields for a Toshiba service object. For more information on Toshiba service object editing options, refer to section 8.4.

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 16 characters in length.

Dest Address

This field indicates the destination address of the drive on the network that will be accessed by this service object. Enter a value between 0 and 63 to target a specific drive. A value of 255 (0xFF) can also be entered in this field. Address 255 designates the broadcast address in the Toshiba ASD protocol. If a broadcast service object is configured, then the **Read** function checkbox must be

unchecked, as attempts to read a service object targeting destination address 255 will invariably time out.

Start Param

This field defines the starting parameter number for a range of drive parameters associated with this service object. Enter a value between 0 and FF99. For example, the drive's output frequency typically resides at parameter FE00.

For configuration parameters (i.e. those parameters which are not used for drive control or monitoring), do not include the leading "F" character which some documentation may include. If the leading "F" character is included in the string entered into the Start Param field, then the parameter can be read, but the drive will reject writing to the parameter. For example, some Toshiba documentation may indicate that the "deceleration time 1" configuration parameter is "F010". This should be entered into the Start Param field as "0010" (or just "10", as the configuration utility will automatically add "0" characters to the beginning of parameter numbers when necessary).

Num Params

This field defines the number of parameters associated with this service object. Enter a value between 1 and 125.

As an example, if you wish to access both "acceleration time 1" and "deceleration time 1" via a single service object, then enter "9" in the Start Param field and "2" in the Num Params field. This will cause the service object to access both parameters 0009 and 0010 (which some Toshiba documentation may describe as parameters F009 and F010, respectively).

Database Addr

This field defines the database address where the first parameter of this service object will be mapped. Enter a value between 0 and 4094. Note that the configuration utility will not allow entry of a starting database address that will cause the service object to run past the end of the database. The highest valid database address, therefore, depends on the number of parameters to be accessed.

Multiplier

This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, raw data is multiplied by the multiplier to produce a network value (to be sent to a drive). Similarly, network values (read from a drive) are divided by the multiplier before being stored into the database.

Note that the multiplier imposes range limitations on network data values. For example, if the multiplier is 0.01, then the network data can achieve a maximum value of only 655 (since 65535 is the maximum value that can be stored in 16 bits in the database).

Read Enable and Function Code Selection

Check **Read** to enable reading (the service object will continuously read from the drive unless a pending **Write** exists). When reads are enabled, the desired read **Function Code** can be selected in the drop-down box. When connected to the drives via a 2-wire RS-485 network, Toshiba recommends use of the "G" read function code. When connected to the drives via a 4-wire RS-485 network, either the "G" or "R" function codes can be used.

Write Enable and Function Code Selection

Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted drive). When writes are enabled, the desired write **Function Code** can be selected in the drop-down box. The "P" function code writes to the drive's volatile RAM memory only, and is typically used when frequently writing to configuration parameters in order to prevent damage to the drive's EEPROM memory (which has a limited write count lifecycle). The "W" function code writes to both the drive's volatile RAM memory as well as its non-volatile EEPROM memory. Drive command parameters (command word, frequency command etc.) exist in RAM only, so either write function code can be safely used when writing to them.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.16.3 Configuration Example

This example will configure the gateway for end-to-end communication using Toshiba ASD Master and EtherNet/IP Server.

Say, for instance, we wish to communicate to a Toshiba G7 adjustable-speed drive from a PLC that supports EtherNet/IP. We wish to monitor the output frequency, status word, output current, and DC bus voltage of the drive, located at parameters FE00, FE01, FE03 and FE04, respectively. To run the drive, we need to be able to write to the RS-485 command word and frequency command, located at parameters FA04 and FA05, respectively.

Configure the Ethernet port using the above requirements

- Connect the DNET-1000 to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).
- Click on the **Ethernet Configuration** subtab.
- Click **Use a static IP address** and enter the IP address settings that your PLC is configured to communicate with.
- Check the checkbox next to **EtherNet/IP Server** in the **Protocol** dropdown menu, and then click on the **EtherNet/IP Server** subtab.
- Enter "0" into the **Produced Data Start Address** field.

- Enter “16” into the **Consumed Data Start Address** field.

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **Toshiba ASD Master** from the protocol dropdown menu.
- Enter the **Baud Rate** and **Parity** settings to match that of the drive.
- Create Service Objects to read and write the desired parameters. Because the drive status parameters are located at parameter numbers FE00, FE01, FE03 and FE04, we could decide to retrieve these by defining just one service object which reads a quantity of 5 parameters, starting with parameter FE00. Obviously, this would include parameter FE02, which we are not interested in. This unnecessary parameter in the produced data block could just be ignored when read by the PLC. This approach results in a slightly faster and simpler configuration of the gateway, but at the expense of a slightly less efficient use of the RS-485 network bandwidth and special processing required in the PLC. For the purposes of this example, therefore, we will define two service objects for reading the drive status parameters: one which accesses parameters FE00 and FE01, and one which accesses parameters FE03 and FE04.

    - Create one service object to monitor the output frequency and drive status word.
        - Enter the address of the drive into the **Dest Address** field.
        - Enter “FE00” into the **Start Param** field.
        - Enter “2” into the **Num Params** field.
        - Enter “0” into the **Database Addr** field.
        - Uncheck the “write” function code check box (these are monitor-only parameters, so there will be no need to write to them)
        - Click **Create**.

    - Create a second service object to monitor the output current and DC bus voltage.
        - Enter the address of the drive into the **Dest Address** field.
        - Enter “FE03” into the **Start Param** field.
        - Enter “2” into the **Num Params** field.
        - Enter “4” into the **Database Addr** field.
        - Uncheck the “write” function code check box
        - Click **Create**.

    - Create a final service object for the RS-485 command word and frequency command.
        - Enter the address of the drive into the **Dest Address** field.
        - Enter “FA04” into the **Start Param** field.
        - Enter “2” into the **Num Params** field.
        - Enter “16” into the **Database Addr** field.
        - Ensure that the “write” function code check box is checked, and then select the desired **Write Function Code**. Because this service object will be used to write to

drive command registers (which exist only in RAM), either "P" or "W" will work fine: we will choose "P" from the dropdown menu.

- Click **Create**.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect to the gateway with your PLC.

Where are the monitor and command values?

Drive Parameter	EtherNet/IP Byte
Output frequency (FE00)	Produced Data Byte 1 – Lower byte Produced Data Byte 2 – Upper byte
Drive status word (FE01)	Produced Data Byte 3 – Lower byte Produced Data Byte 4 – Upper byte
Output current (FE03)	Produced Data Byte 5 – Lower byte Produced Data Byte 6 – Upper byte
DC bus voltage (FE04)	Produced Data Byte 7 – Lower byte Produced Data Byte 8 – Upper byte
RS-485 command word (FA04)	Consumed Data Byte 1 – Lower byte Consumed Data Byte 2 – Upper byte
RS-485 frequency command (FA05)	Consumed Data Byte 3 – Lower byte Consumed Data Byte 4 – Upper byte

Note that the database is assumed to be little endian in this example.

8.7.17 Sullair Supervisor Master

To configure the gateway for Sullair Supervisor Master, click on the **RS-485 Configuration** tab and select **Sullair Master** in the protocol dropdown menu. The Sullair Master protocol uses service objects to make requests. For more information on service objects, refer to section 8.5. Except for display parameters, each parameter in a Sullair supervisor service object is mapped to 2 bytes in the database (the data size is fixed at 16-bit). For more information on parameter mapping, refer to section 11.10.2

8.7.17.1 Protocol Selection Group

Protocol

Select **Sullair Master** from this dropdown menu.

8.7.17.2 Sullair Service Object Configuration

This section describes the configurable fields for a Sullair service object. For more information on Sullair service object editing options, refer to section 8.6.

Description

This field is a description of the service object. It is not used on the gateway, but serves as a reference for the user. Enter a string of up to 16 characters in length.

Dest Address

This field indicates the destination address of the controller on the network that will be accessed by this service object. Enter a value between 1 and 16 to target a specific controller.

Start Param

This field defines the starting parameter number for a range of controller parameters associated with this service object. Enter a value between 0 and 124. For example, the controller's unload pressure value resides at parameter 5.

Num Params

This field defines the number of parameters associated with this service object. Enter a value between 1 and 125.

As an example, if you wish to access all the net status parameters via a single service object, then enter "100" in the Start Param field and "7" in the Num Params field. This will cause the service object to access all parameters that are updated via the net status message.

Database Addr

This field defines the database address where the first parameter of this service object will be mapped. Enter a value between 0 and 4094. Note that the configuration utility will not allow entry of a starting database address that will

cause the service object to run past the end of the database. The highest valid database address, therefore, depends on the number of parameters to be accessed.

Multiplier

This field is the amount that associated network values are scaled by prior to being stored into the database or after being retrieved from the database. Upon retrieval from the database, raw data is multiplied by the multiplier to produce a network value (to be sent to a controller). Similarly, network values (read from a controller) are divided by the multiplier before being stored into the database. This value is ignored for display parameters.

Note that the multiplier imposes range limitations on network data values. For example, if the multiplier is 0.01, then the network data can achieve a maximum value of only 655 (since 65535 is the maximum value that can be stored in 16 bits in the database).

Read Enable and Function Code Selection

Check **Read** to enable reading (the service object will continuously read from the controller unless a pending **Write** exists). When reads are enabled, the desired read **Function Code** can be selected in the drop-down box.

Write Enable and Function Code Selection

Check **Write** to enable writing (when values encompassed by this service object change in the gateway's database, these changes will be written down to the targeted controller). When writes are enabled, the desired write **Function Code** can be selected in the drop-down box.

Service Object Status

If it is desired to reflect the status of this service object, check the **Reflect Status** checkbox and enter a database **address** between 0 and 4080 (0x0 – 0xFF0) at which to store the status information. For more information on reflecting the status of service objects, refer to section 8.5.2.

8.7.17.3 Configuration Example

This example will configure the gateway for accessing a Supervisor controller via the Sullair Master driver. Say, for instance, we wish to monitor P1 – P4, T1 – T5, and the run status. These data items are located at parameters 107 – 110, 111 – 115, and 103 respectively (refer to section 11.10 for a list of Supervisor parameters with indexes of 100+). We also wish to control the unload pressure, load pressure delta, and unload time, located at parameters 5, 6, and 7 respectively.

Configure the gateway using the above requirements

- Connect the gateway to the PC via a USB mini type-B cable.
- Open the configuration utility and select the ETH-1000 (see section 8.1 for more information on selecting a device).

Configure the RS-485 port using the above requirements

- Click on the **RS-485 Configuration** tab.
- Select **Sullair Master** from the protocol dropdown menu.
- Create Service Objects to read and write the desired parameters. Because the pressure and temperature parameters are located at contiguous indexes (107 – 115), we can retrieve these by defining just one service object which reads a quantity of 9 parameters, starting with parameter 107. Similarly, the parameters we wish to control are also at contiguous indexes (5 – 7), enabling us to define a single service object for these parameters as well.

    - Create one service object to monitor the pressure and temperature parameters.
        - Enter the address of the controller into the **Dest Address** field.
        - Enter "107" into the **Start Param** field.
        - Enter "9" into the **Num Params** field.
        - Enter "0" into the **Database Addr** field.
        - Uncheck the "write" function code check box (these are monitor-only parameters, so there will be no need to write to them)
        - Click **Create**.

    - Create a second service object to monitor the run status of the controller.
        - Enter the address of the controller into the **Dest Address** field.
        - Enter "103" into the **Start Param** field.
        - Enter "1" into the **Num Params** field.
        - Enter "18" into the **Database Addr** field.
        - Uncheck the "write" function code check box
        - Click **Create**.

    - Create a final service object for the unload pressure, load pressure delta, and unload time.
        - Enter the address of the controller into the **Dest Address** field.
        - Enter "5" into the **Start Param** field.
        - Enter "3" into the **Num Params** field.
        - Enter "32" into the **Database Addr** field.
        - Ensure that the "write" function code check box is checked (as we do wish to modify these parameters over the network).
        - Click **Create**.

Finishing Up

- Download the configuration to the gateway (see section 8.1 for more information on downloading a configuration to a device).
- Connect the gateway to the Supervisor network.

Where are the monitor and command values?

Controller Parameter (Parameter Index)	Database Address
P1 (107)	0 & 1
P2 (108)	2 & 3
P3 (109)	4 & 5
P4 (110)	6 & 7
T1 (111)	8 & 9
T2 (112)	10 & 11
T3 (113)	12 & 13
T4 (114)	14 & 15
T5 (115)	16 & 17
run status (103)	18 & 19
unload pressure (5)	32 & 33
load pressure delta (6)	34 & 35
unload time (7)	36 & 37

9. Interacting With the Filesystem

The gateway's on-board filesystem is used to store files for use by the application firmware. Currently, the application firmware's main use of the filesystem is to store XML-encoded configuration files that dictate the characteristics of the various protocols. Each protocol that requires configuration will have its own XML file stored on the filesystem. For easy identification, the filename will begin with the corresponding protocol which it configures. For example, a BACnet configuration file's filename will begin with "bacnet", and an EtherNet/IP file will begin with "eip".

Whenever the configuration for a specific protocol is completed, it is suggested that a backup copy of the configuration file be downloaded from the unit to a PC. One reason for this is in case it becomes necessary to restore a previous configuration at a later time. Another reason is that it may be desirable to load multiple units with the same configuration, as a downloaded configuration file can be uploaded again to any compatible unit, allowing the user to easily clone multiple units with the same configuration. While the majority of the configuration files that are available via FTP are recreated whenever a configuration is downloaded via USB from the configuration utility, some of the files are available solely over Ethernet. For example, the dashboard and alarm configuration can only be configured via the embedded web server, and therefore these specific configuration files are only available via FTP.

Each time the gateway boots up, it will interrogate the filesystem for the configuration files required by the protocols currently operating in the unit. If it does not find a required file, it will create one and initialize it with factory-default values. Therefore, if it is ever desired to reset a protocol's configuration to factory-default values, this can be easily accomplished by simply deleting the appropriate configuration file from the filesystem and rebooting the unit.

Note that the application firmware uses specific filenames for the configuration files. This means that if a file with a different filename is loaded onto the unit, it will be stored correctly, but will not be used by the application firmware. Similarly, if an existing configuration file's filename is changed, then the unit will again create a default configuration file at next boot-up, which will be stored in the filesystem alongside the file with the changed name.

Configuration files are only read by the protocol drivers at unit boot-up. Therefore, if a new configuration file is loaded onto a unit's filesystem, that unit must be rebooted for the configuration file's settings to take effect. Rebooting a unit can be performed by:

- power-cycling the gateway,
- selecting "Reset Device" from the configuration utility, or
- selecting the "Reboot Device" button in the Finder tab of the configuration utility

Interacting with the filesystem is performed by use of the File Transfer Protocol (FTP). Using FTP allows the user to interact with the files on the gateway's

filesystem in the same manner as though they were traditional files stored on a local or remote PC. While there are many different FTP applications available, the following sections will provide general examples of using some of the most commonly-available ones.

9.1 Initiating FTP via the Finder Tab

After discovering all gateways on the current subnet as described in section 8.2, select the target gateway and then click on the "Open FTP Interface" button. This will open the computer's default FTP application, which could be Windows Explorer, a web browser, or a 3rd-party FTP program (whatever the computer/operating system is configured for by default). This example will assume that a web browser (Microsoft Internet Explorer) is configured as the default FTP application.

An authentication dialog will appear (refer to Figure 11). Enter the currently-configured user name and case-sensitive password (defaults are "root" and "icc", respectively), then click "Log On."

Figure 11: FTP Authentication

The web browser will then display the filesystem's contents (refer to Figure 12.) FTP access via a web browser allows viewing and downloading files to a computer, but does not allow advanced file manipulation such as cut, paste, drag-and-drop, etc. For advanced file manipulation abilities, use of a different FTP application is required.

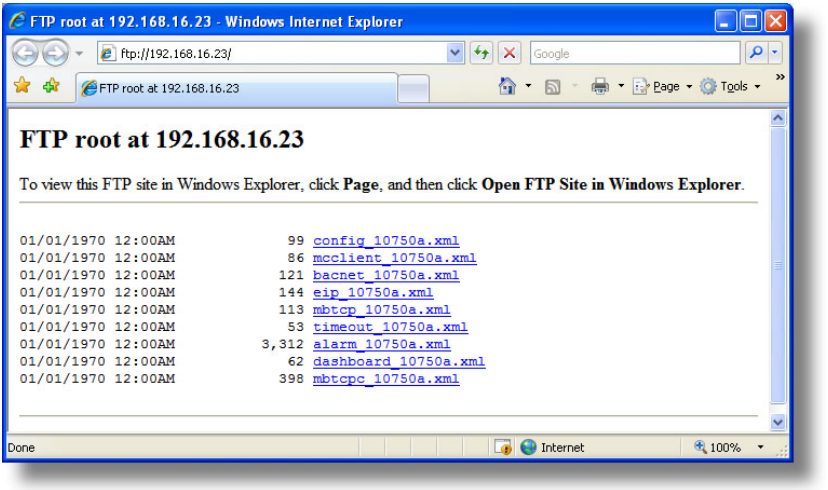


Figure 12: FTP Navigation with Internet Explorer

9.2 Using FTP with Windows Explorer

To use FTP with Microsoft Windows Explorer, first open either “Windows Explorer” or “My Computer”. Refer to Figure 13. Please note that the indicated procedure, prompts and capabilities outlined here can vary depending on such factors as the installed operating system, firewalls and service packs.

Figure 13: Accessing Windows Explorer

In the “Address” field, type in “ftp://root@” and then the IP address of the target gateway (if the user name has been changed from its default, then replace “root” in “ftp://root@” with the new user name.) Refer to Figure 14.

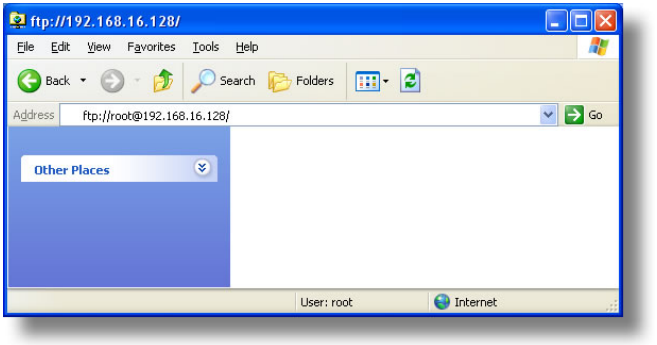


Figure 14: FTP Navigation with Windows Explorer

You will then be presented with an authentication dialog (refer to Figure 15.) The user name will already be filled-in. Enter the case-sensitive password (default is "icc") and click "Log On."

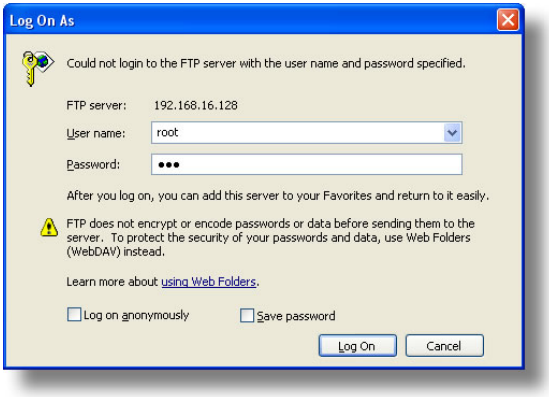


Figure 15: FTP Authentication

Windows Explorer will then display the filesystem's contents (refer to Figure 16.) You can now perform normal file manipulation actions on the available files (cut, copy, paste, open, rename, drag-and-drop transfers etc.) in the same manner as though you were manipulating any traditional file stored on your computer's hard drive.

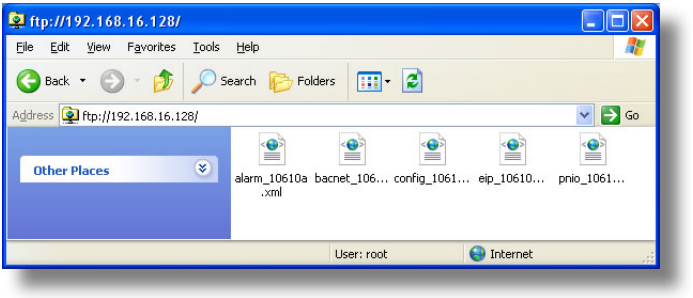


Figure 16: File Access with Windows Explorer

9.3 Using FTP with a Windows Command Prompt

To use FTP with a Windows command (DOS) prompt, first open a command prompt by either selecting *Start…All Programs…Accessories…Command Prompt*, or by selecting *Start…Run* and typing "cmd" in the "Run" dialog.

Once the command prompt opens, type "ftp" and the IP address of the target gateway. The FTP client will connect to the unit and then prompt for the username and case-sensitive password (defaults are "root" and "icc", respectively). Upon successful entry of the authentication information, you will be presented with an "ftp>" prompt. Refer to Figure 17.

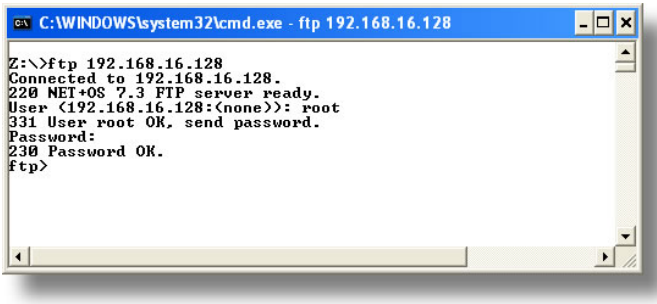


Figure 17: FTP Initiation and Authentication

At this point, you can use standard Unix-style file and directory manipulation commands to perform such actions as listing files (Figure 18), copying files to your computer (Figure 19), and copying files to the unit (Figure 20).

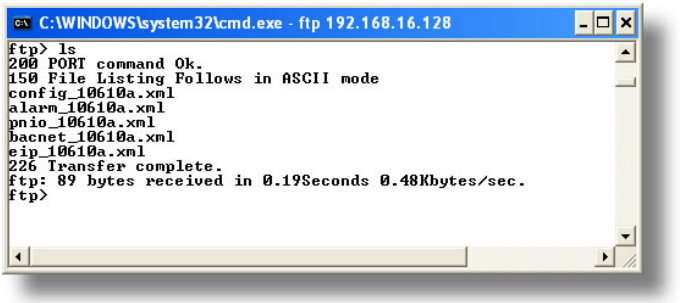


Figure 18: Listing Files with "ls" Command

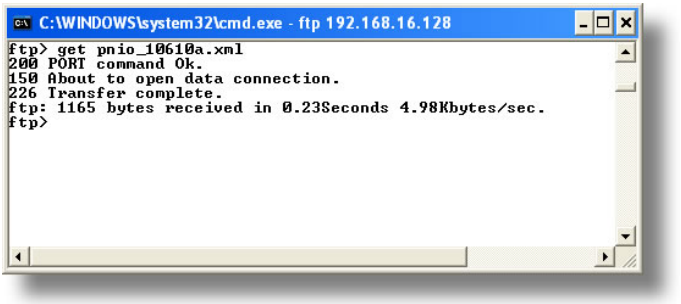


Figure 19: Copying a File from the Unit With "get" Command

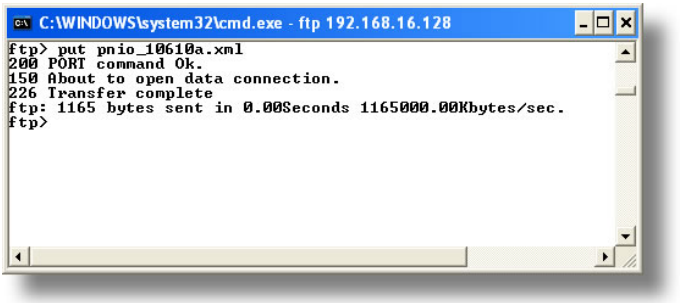


Figure 20: Copying a File to the Unit With "put" Command

9.4 Using FTP with Core FTP LE

Core FTP LE (Lite) is a 3rd-party FTP application that can be downloaded for free from http://www.coreftp.com. Core FTP is just one example of the various commercial and freeware FTP client applications available on the internet.

After installing Core FTP LE, run the program. If the "Site Manager" window (Figure 21) does not automatically open, open it by choosing "File…connect".

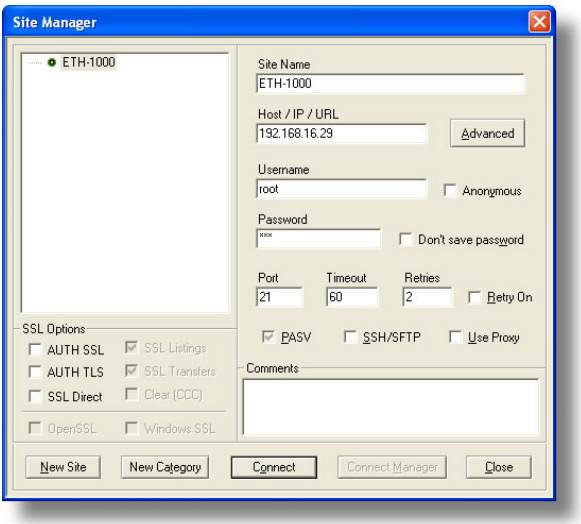


Figure 21: Core FTP Site Manager

Click on the "New Site" button, then enter a Site Name, IP Address, user name (default is "root") and case-sensitive password (default is "icc"). The "Port", "Timeout", and "Retries" fields should already contain the default values. Click the "Connect" button when done.

Core FTP LE will then try to connect and authenticate to the FTP server, and if successful, will populate the right-hand side of the main page with the unit's filesystem contents. Refer to Figure 22.

Files can be easily downloaded from the unit by choosing the appropriate destination folder on your computer in the left-hand side of the main page, choosing the file to download, and then clicking the "download" button in the right-hand (source) side. Similarly, files can be easily uploaded to the unit by choosing the file to upload and then clicking the "upload" button in the left-hand (source) side of the main page.

Like most 3rd-party FTP client applications, Core FTP LE has a wide array of configuration and file management capabilities, which are beyond the scope of this manual. Refer to the program's Help file for more detailed instructions.

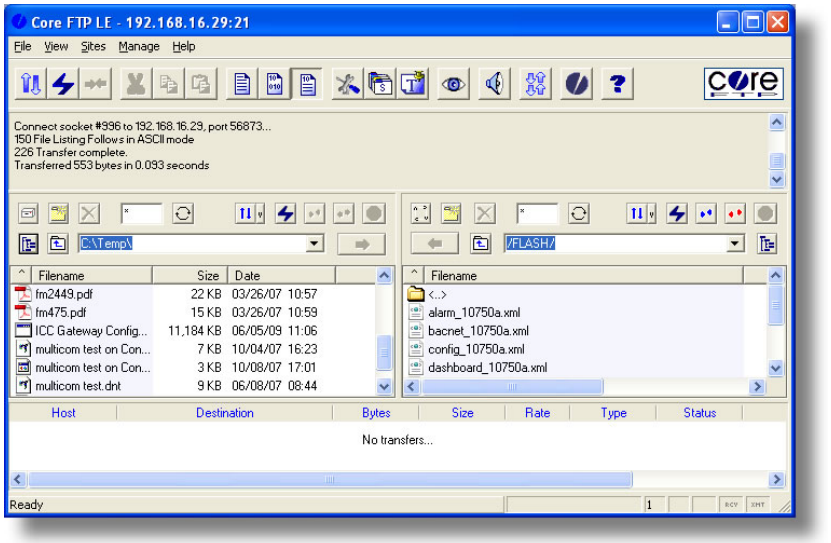


Figure 22: Core FTP in "Connected" State

10. Embedded Web Server

10.1 Overview

In addition to the ability to configure the gateway over USB via the ICC Gateway Configuration Utility, all Ethernet drivers may also be configured through the gateway's embedded web server (also known as an HTTP server). This dual-access capability allows users to access the gateway's data in a graphical manner with web browsers such as Microsoft Internet Explorer or Mozilla Firefox. In this way, the gateway can be monitored, configured and controlled from across the room or from across the globe.

In order to view the gateway's web page, the free Adobe (formerly Macromedia) Flash Player browser plug-in is required. If the plug-in is not already installed on your computer, then your browser will automatically be redirected to the appropriate Adobe download web site when you initially attempt to access the gateway's web page. Alternatively, the plug-in can be downloaded directly by going to http://www.adobe.com, and choosing the "get Adobe Flash Player" link. Always ensure that you have the latest version of the Flash Player installed: if some aspect of the web page does not appear to be displayed properly, installing the latest Flash Player update usually resolves the problem.

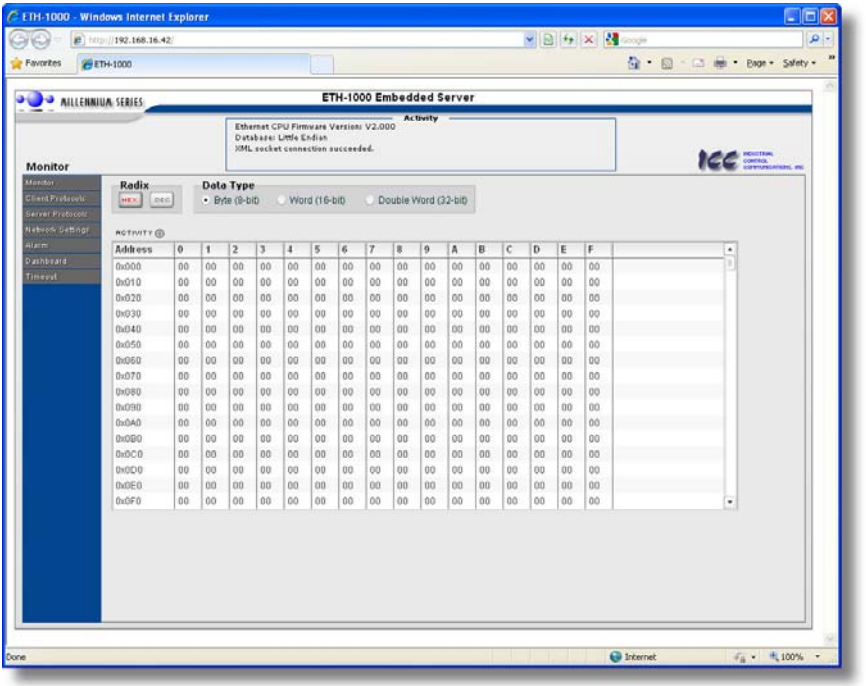


Figure 23: Embedded Web Server

To access the gateway's embedded web server, either use the finder tab in the configuration utility (refer to section 8.2) and select the "Open Web Interface" button when the target unit is highlighted, or just directly enter the target unit's IP address into the address (URL) field of your web browser. Refer to Figure 23 for a representative screenshot of the web server interface.

In order to access the web server and view the parameter values, destination TCP ports 80 and 2000 must be accessible from the client computer. If an "XML socket connection failed" error message is displayed in the activity window, and no parameter values are shown, this is typically indicative of port 2000 being blocked by a firewall or Ethernet router situated between the client computer and the gateway.

10.2 Authentication

For security, the gateway requires valid user authentication whenever the web page is accessed. The authentication request will appear as a browser popup box that will request entry of a user name and password. Refer to Figure 24.

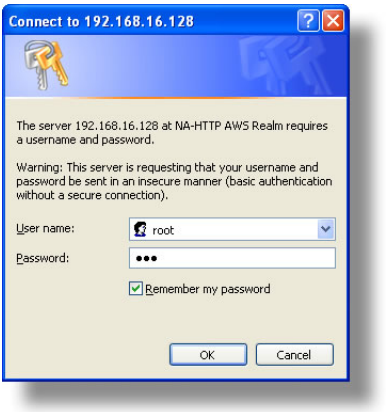


Figure 24: Web Server Authentication

The factory-default user name is "root", and the password is "icc". Note that the username and password are case-sensitive, and that once authenticated, the authentication will remain in effect from that point until all browser windows are closed. The authentication credentials can also be changed from their default settings (refer to section 8.7.1.1.)

10.3 Navigation Menu Tree

The web interface is structured as a navigation menu tree accessible on the left-hand side of the web page, where each menu contains information common to a specific feature or protocol. Refer to Figure 25. To change to a different configuration menu, just click on the title of the menu you wish to view. Some menus can be expanded to display submenus. The title of the currently-selected menu appears above the navigation menu tree.

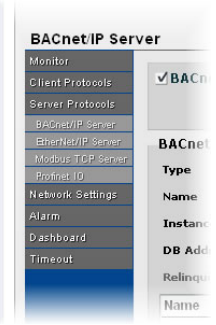


Figure 25: Navigation Menu Tree

10.4 Activity Window

Figure 26 shows the Activity Window, which is located at the top of the web page. This window displays various informational messages regarding the status of the gateway or web browser session.

Figure 26: Activity Window

10.5 Monitor Menu

10.5.1 Activity Indicator

There is an “activity” indicator located in the upper-left hand corner of the Monitor Menu, which blinks periodically to show the status of data communication between the web browser and the gateway. If you do not observe the activity indicator blink at all for several seconds or more, it is possible that the web browser may have lost contact to the web server due to a device reset or a

network problem: to reestablish communications, select “refresh” on your web browser.

10.5.2 Database

The database displays the live values from the associated device's internal database. The alignment (byte, word or double word) and radix (hex or decimal) of the displayed data can be adjusted via the appropriate controls in the “Radix” and “Data Type” selection groups. Database values can also be directly edited by clicking on the desired location and entering the desired value.

10.5.2.1 Radix Selection

The radix selection buttons provide the ability to change the data display and entry radix between decimal and hexadecimal formats.

When “DEC” is selected, the current values will be displayed in decimal, and modified values must be entered in decimal format.

Similarly, when “HEX” is selected, the current values will be displayed in hexadecimal, and modified values must be entered in hexadecimal format.

10.5.2.2 Data Type Selection

The alignment of the data displayed can be set to byte (8-bit), word (16-bit), or double word (32-bit). To change the data type, select the radio button of the desired type: the database columns and displayed values will be automatically readjusted according to the selected data type.

10.6 MELSEC Client Menu

The MELSEC Client menu provides access to the same general configuration items accessible via the MELSEC subtab in the configuration utility. Refer to Figure 27.

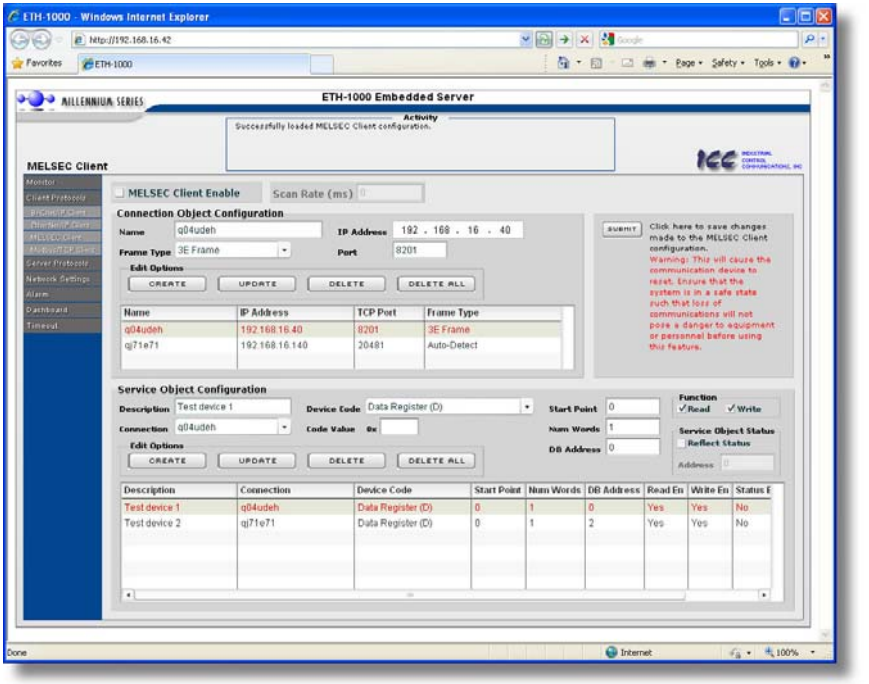


Figure 27: MELSEC Client Menu

10.6.1 MELSEC Client Enable

Check to enable the MELSEC client driver.

10.6.2 Connection Object Configuration

For more information regarding MELSEC connection object configuration, refer to section 8.7.8.1.

10.6.3 Service Object Configuration

For information regarding MELSEC service object configuration, refer to section 8.7.8.2.

10.6.4 Submitting Changes

Whenever the MELSEC client configuration has been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.7 EtherNet/IP Client Menu

The EtherNet/IP Client menu provides access to the same general configuration items accessible via the EtherNet/IP Client subtab in the configuration utility. Refer to Figure 28.

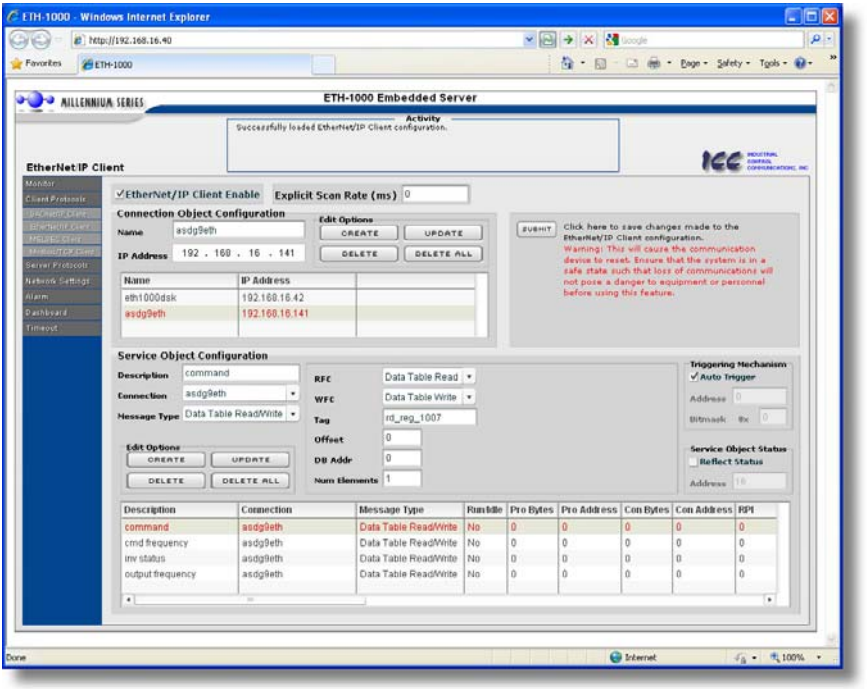


Figure 28: EtherNet/IP Client Menu

10.7.1 EtherNet/IP Client Enable

Check to enable the EtherNet/IP client driver.

10.7.2 Connection Object Configuration

For more information regarding EtherNet/IP Client connection object configuration, refer to section 8.7.4.1.

10.7.3 Service Object Configuration

For information regarding EtherNet/IP service object configuration, refer to section 8.7.4.2.

10.7.4 Submitting Changes

Whenever the EtherNet/IP client configuration has been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.8 Modbus/TCP Client Menu

The Modbus/TCP Client menu provides access to the same general configuration items accessible via the Modbus/TCP Client subtab in the configuration utility. Refer to Figure 29.

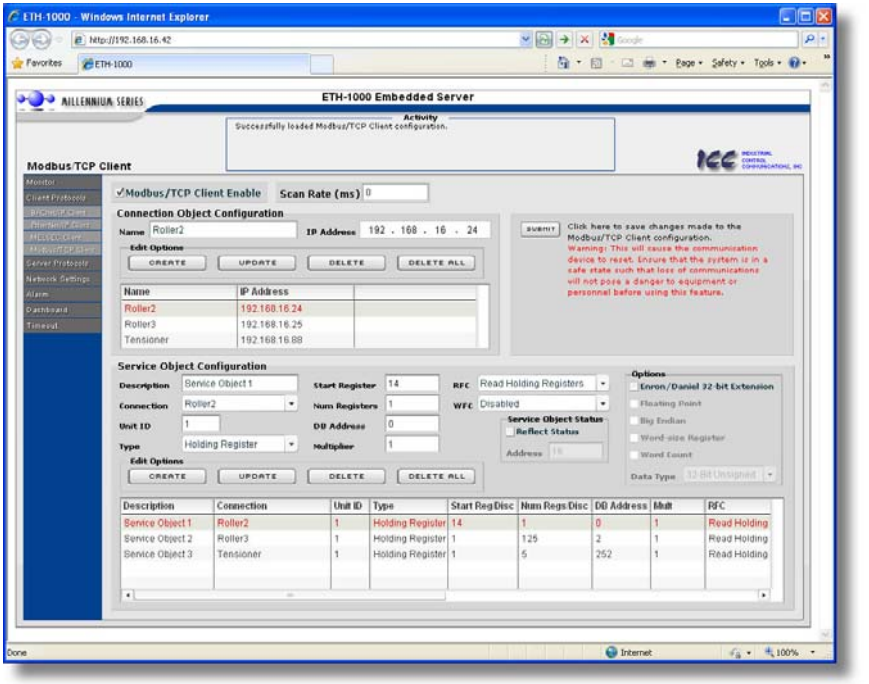


Figure 29: Modbus/TCP Client Menu

10.8.1 Modbus/TCP Client Enable

Check to enable the Modbus/TCP client driver.

10.8.2 Connection Object Configuration

For more information regarding Modbus/TCP Client connection object configuration, refer to section 8.7.6.1.

10.8.3 Service Object Configuration

For information regarding Modbus/TCP service object configuration, refer to section 8.7.6.2.

10.8.4 Submitting Changes

Whenever the Modbus/TCP client configuration has been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.9 BACnet/IP Client Menu

The BACnet/IP Client menu provides access to the same general configuration items accessible via the BACnet/IP Client subtab in the configuration utility. Refer to Figure 30.

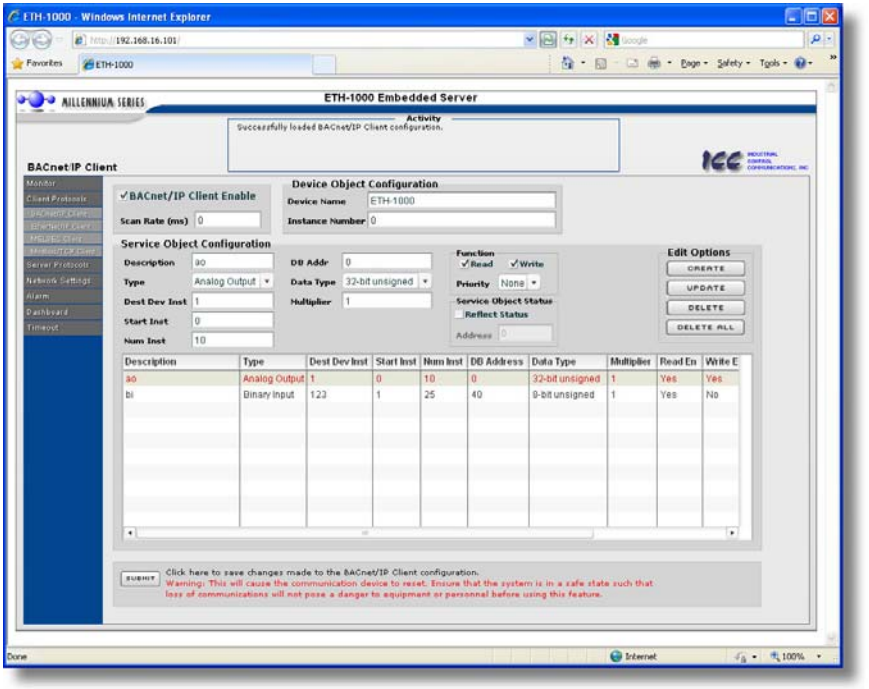


Figure 30: BACnet/IP Client Menu

10.9.1 BACnet/IP Client Enable

Check to enable the BACnet/IP client driver.

10.9.2 Device Object Configuration

For information regarding these fields, refer to section 8.7.2.1.

10.9.3 Service Object Configuration

For information regarding BACnet/IP service object configuration, refer to section 8.7.2.2.

10.9.4 Submitting Changes

Whenever the BACnet/IP client configuration has been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.10 BACnet/IP Server Menu

The BACnet/IP Server menu provides for the configuration of the gateway as a server on a BACnet/IP network. Refer to Figure 31.

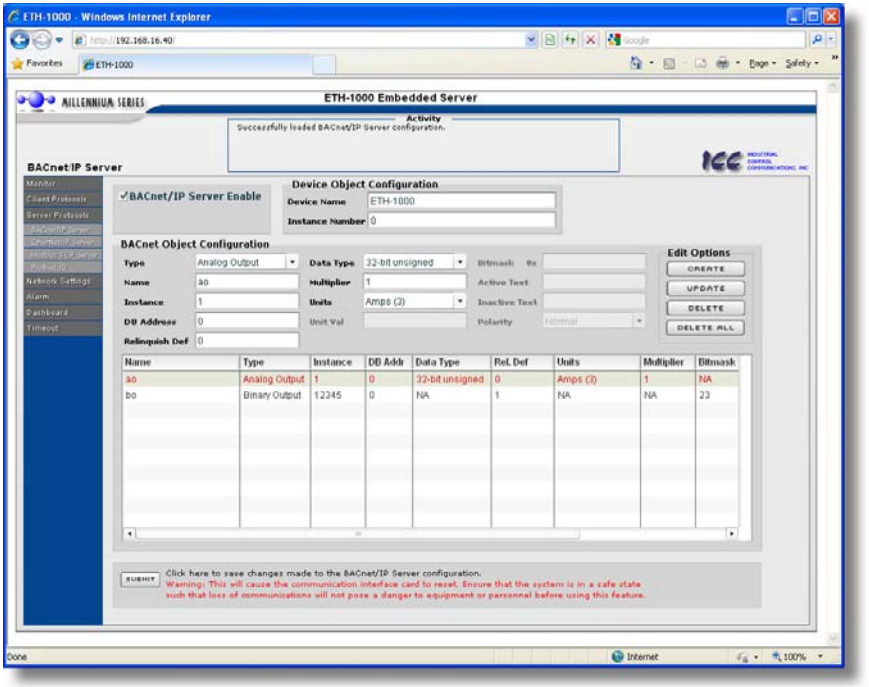


Figure 31: BACnet/IP Server Menu

10.10.1 BACnet/IP Server Enable

Check to enable the BACnet/IP server driver.

10.10.2 Device Object Configuration

For information regarding these fields, refer to section 8.7.3.1.

10.10.3 BACnet Object Configuration

For detailed information regarding BACnet object configuration fields (Type, Name, Instance etc.), refer to section 8.7.3.2. For detailed information regarding object editing options (create, update etc.), refer to section 8.6.

10.10.4 Submitting Changes

The “submit” button must be clicked in order to write changed settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.11 EtherNet/IP Server Menu

The EtherNet/IP Server menu provides access to the same general configuration items accessible via the EtherNet/IP Server subtab in the configuration utility. Refer to Figure 32.

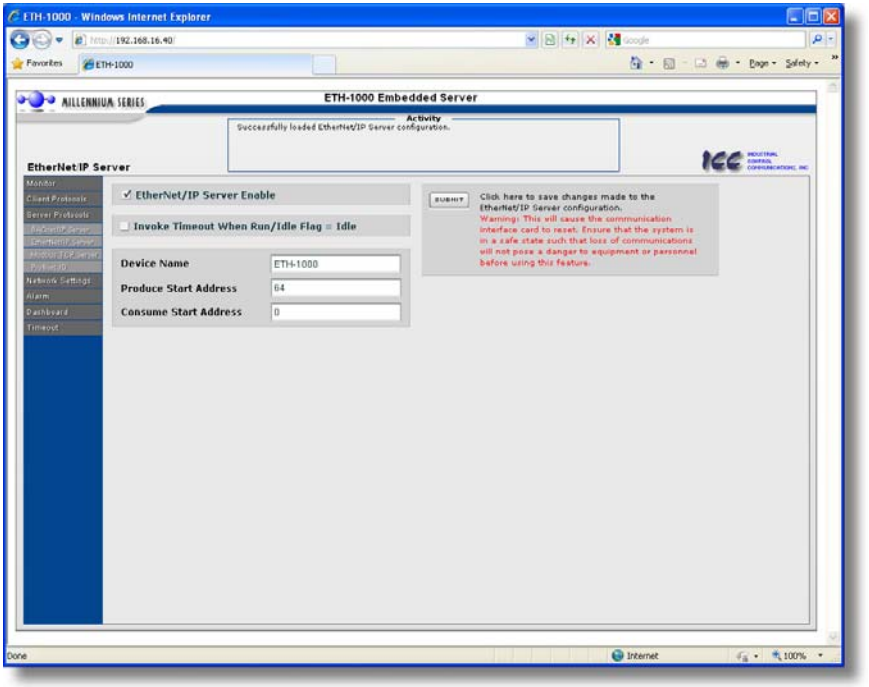


Figure 32: EtherNet/IP Server Menu

10.11.1 EtherNet/IP Enable

Check to enable the EtherNet/IP server driver.

10.11.2 Run/Idle Flag Behavior

For more information, refer to **Run/Idle Flag Behavior** in section 8.7.5.1.

10.11.3 Device Identification

For more information, refer to **Device Name** in section 8.7.5.1.

10.11.4 Class 1 (I/O) Data Configuration

For more information, refer to **Produced Data Start Address** and **Consumed Data Start Address** in section 8.7.5.1.

10.11.5 Submitting Changes

Whenever any of the EtherNet/IP server configuration elements have been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.12 Modbus/TCP Server Menu

The Modbus/TCP Server menu provides access to the same general configuration items accessible via the Modbus/TCP subtab in the configuration utility. Refer to Figure 33.

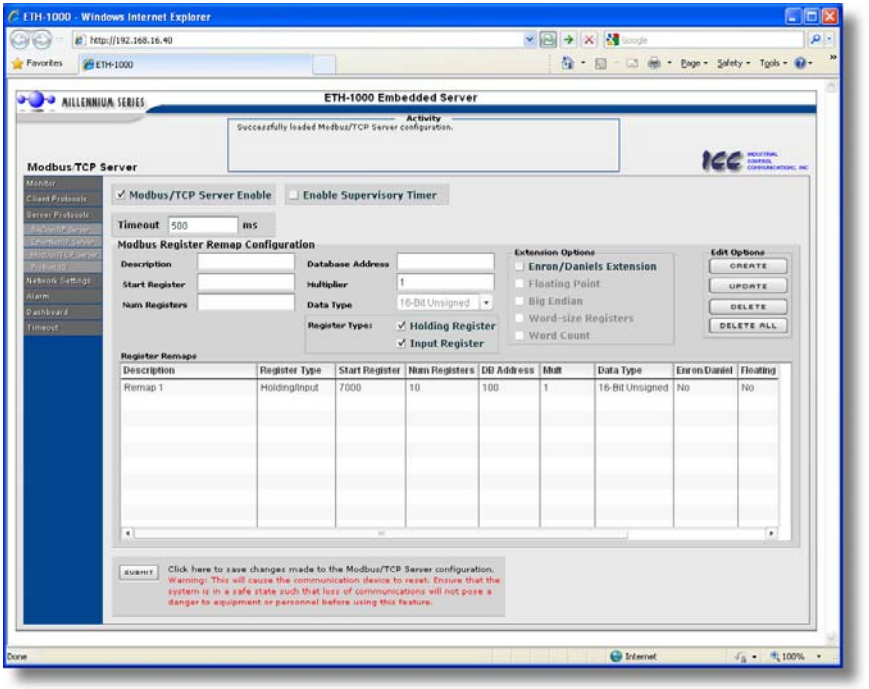


Figure 33: Modbus/TCP Server Menu

10.12.1 Modbus/TCP Server Enable

Check to enable the Modbus/TCP server driver.

10.12.2 Timeout Time Setting

For more information, refer to section 8.3.1.

10.12.3 Register Remap Configuration

For more information, refer to section 8.7.7.3.

10.12.4 Submitting Changes

Whenever the Modbus/TCP server configuration has been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.13 Profinet IO Menu

The Profinet IO menu provides access to the same general configuration items accessible via the Profinet IO subtab in the configuration utility. Refer to Figure 34.

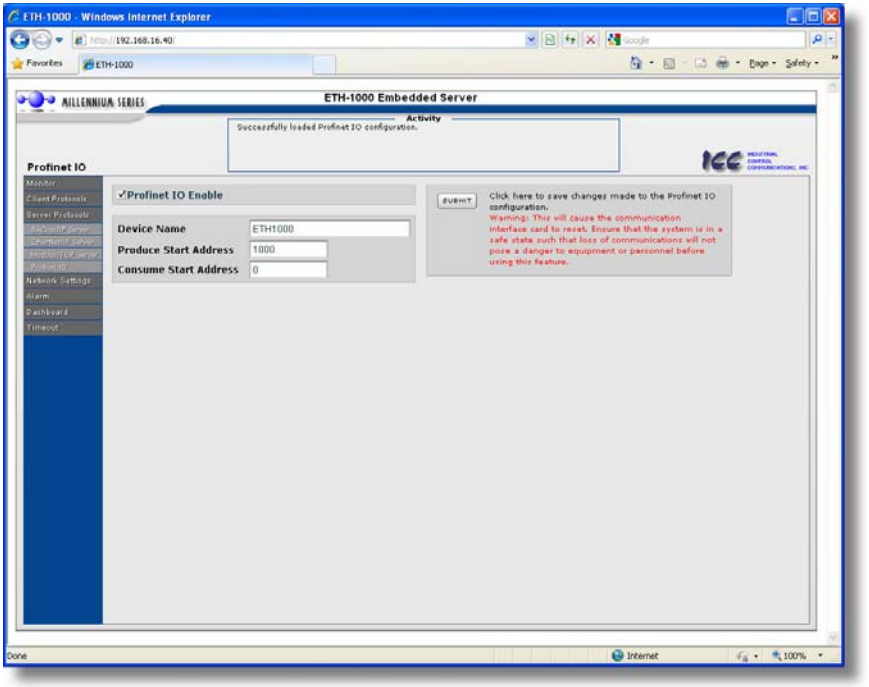


Figure 34: Profinet IO Menu

10.13.1 Profinet IO Enable

Check to enable the Profinet IO driver.

10.13.2 Device Identification

For more information, refer to **Device Name** in section 8.7.9.1.

10.13.3 Cyclic (I/O) Data Configuration

For more information, refer to **Produced Data Start Address** and **Consumed Data Start Address** in section 8.7.9.1.

10.13.4 Submitting Changes

Whenever any of the Profinet IO configuration elements have been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.14 Network Settings Menu

The network settings menu provides access to the same general Ethernet configuration items accessible via the Ethernet Configuration subtab in the configuration utility. Refer to Figure 35.

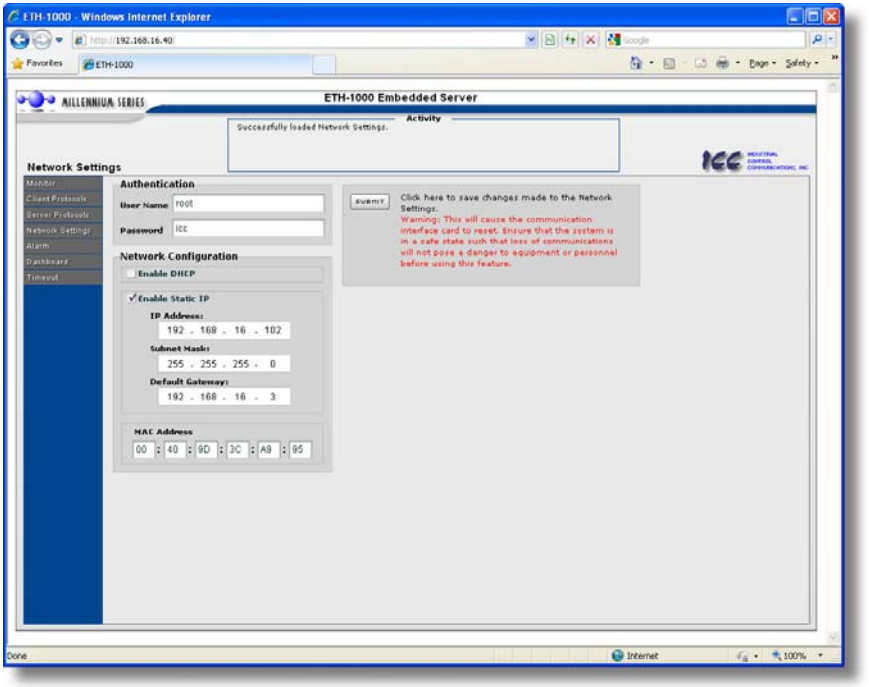


Figure 35: Network Settings Menu

10.14.1 Authentication

Refer to section 8.7.1.1.

10.14.2 Network Configuration

Figure 36 shows the configuration items used to modify the IP address-related parameters. For more information on **DHCP** and **Static IP** configuration settings, refer to section 10.14.2

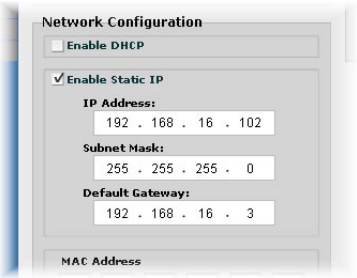


Figure 36: Network Configuration

10.14.3 MAC Address

Figure 37 shows the entry boxes that are used to view the unique MAC address of the gateway.

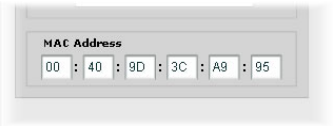


Figure 37: MAC Address

10.14.4 Submitting Changes

Whenever any of the network settings have been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.15 Alarm Menu

The alarm menu provides a configurable mechanism by which the gateway can autonomously monitor any database address and send emails to up to four recipients when a certain condition is detected. The alarm conditions have both value and time constraints, and can be configured to retrigger at a fixed interval as long as the alarm condition continues to be satisfied. Twenty individually-configurable alarms are available. Refer to Figure 38.

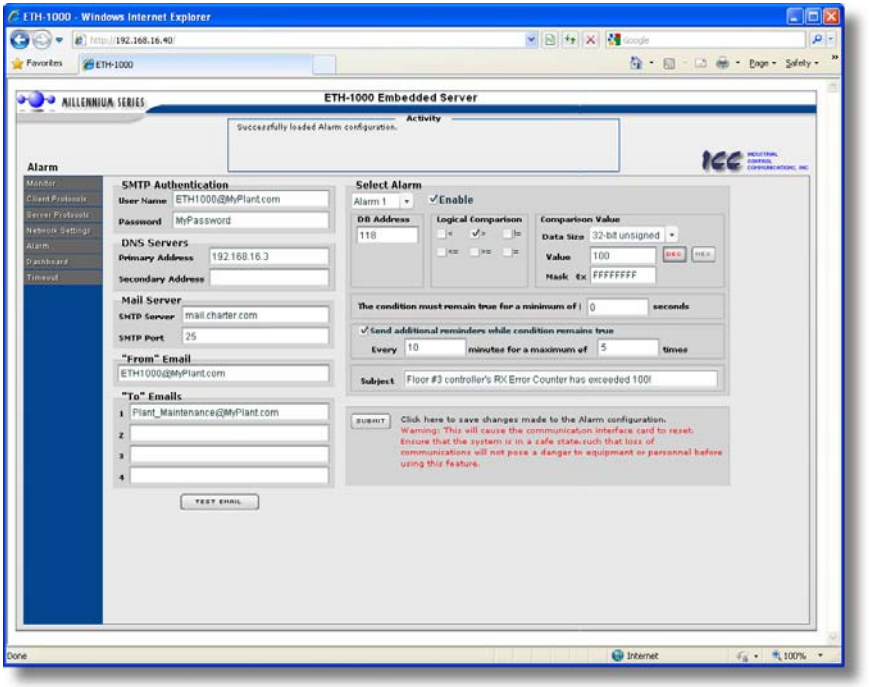


Figure 38: Alarm Menu

10.15.1 Email Configuration

In order for an alarm trigger to successfully send a notification email, some network settings must first be configured properly (refer to Figure 39.)

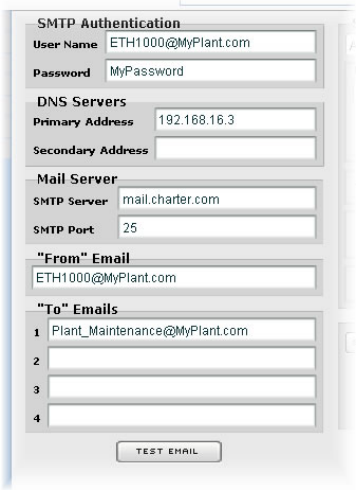


Figure 39: Email Configuration

SMTP Authentication: Some email servers require that clients wishing to send emails first authenticate themselves. If the email server in use requires authentication, then enter the appropriate user name and password. If the email server in use does not require authentication, then these entries can be disregarded.

DNS Servers: Enter the dotted-decimal IP addresses of the primary and secondary DNS servers which will be used to resolve the configured SMTP server name. Only the primary DNS server is required, but if a secondary DNS server is entered, then it will be used if the primary server is inaccessible.

Mail Server: Enter the SMTP server address as a name or as a dotted-decimal IP address, and the SMTP port (default=25) that the SMTP server listens for incoming emails on.

"From" Email: Enter the email address that will appear as the sender's email address in the email headers.

"To" Emails: Up to four recipients can be designated to receive alarm emails. Blank entries will not be processed by the gateway.

"Test Email" Button: When the "Test Email" button is pressed, the gateway will use the information currently entered in the above-mentioned fields to send a test email. Note that you do not have to first "submit" the settings to the gateway's filesystem (refer to section 10.15.3) in order to test them: fields can be changed and retested on-the-fly without affecting the operation of the gateway's control protocols. When the correct settings have been confirmed with a successfully-sent test email, "submit" the changes at that time to commit them to the gateway's filesystem: any changes made prior to submitting as described in section 10.15.3 are temporary only and will be lost if a different configuration menu is selected or if the web browser is closed.

When a test email transmission is initiated, completes successfully, or fails due to an error, the activity window (refer to section 10.4) will display appropriate messages such as those shown in Figure 40 and Figure 41.

Although the test email is sent immediately, note that due to internet and/or email server delays, it may take several minutes to receive test emails.

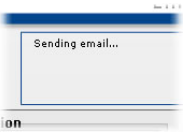


Figure 40: Activity Window at Test Email Initiation

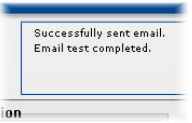


Figure 41: Activity Window at Test Email Successful Completion

10.15.2 Alarm Configuration

The gateway supports twenty independently-configurable alarms. As shown in Figure 42, each alarm has a variety of configuration elements, which will be explained further below.

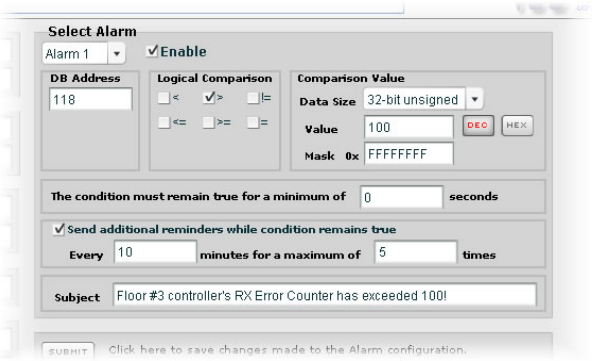


Figure 42: Alarm Configuration Box

Alarm Selection: This drop-down box allows the selection of one of the twenty available alarms. When an alarm is selected, that alarm's current configuration parameters will be populated in the alarm configuration box.

"Enable" Check Box: If checked, this alarm is active and will be evaluated every second. If unchecked, this alarm is inactive and will therefore not be evaluated.

DB Address: Enter the database address that this alarm will continuously monitor. If the "Data Size" (see below) is set to anything other than 8-bit, then

this address designates the database starting location for the multi-byte element to be evaluated. For multi-byte elements, whether this designated address represents the element's high byte or low byte depends upon the current database endianness setting.

Logical Comparison: Choose a comparison operator which will be used to compare the current value of the indicated "Data Size" with the reference "Comparison Value". Available selections are "less than" (<), "less than or equal to" (<=), "greater than" (>), "greater than or equal to" (>=), "not equal to" (!=), and "equal to" (=).

Comparison Value: The reference comparison value is comprised of three subcomponents: a "Data Size" field, a "Mask" field, and a "Value" field. Each time the alarm is evaluated, the current value at the indicated "DB Address" is first bit-wise "AND"ed with the "Mask" field. The resulting derived value is then compared with the "Value" field by way of the "Logical Comparison" operator. While the "Mask" field is always a hexadecimal number, the display and entry radix of the "Value" field can be changed between decimal and hexadecimal with the associated "DEC" and "HEX" buttons.

Database values that correspond to "analog" process variables (e.g. frequencies, voltages, counters etc.) should typically have their "Mask" fields set to all F's (0xFF for 8-bit, 0xFFFF for 16-bit, and 0xFFFFFFFF for 32-bit), which causes all data bits to be retained for the "Value" field comparison. For values that correspond to "enumerated" process variables (e.g. status words where each bit of the database value indicates a different item), however, the "Mask" can be chosen to single out one or more specific data bits of the value.

Note that the alarms will evaluate the designated database location regardless of the meaning of the value contained there. What this means is that alarms can react to not only process values that are being read and/or written via the various connected networks, but also to ancillary items such as "reflect status" counters or error indicators. As an example, the alarm shown in Figure 42 targets the 32-bit "RX Error Counter" element of a status information structure that was presumably placed at database address 110 (the "RX Error Counter" element is located 8 bytes in from the start of the structure, thereby resulting in a required address of 118: refer to Appendix B: Status Information). If the service object associated with this status information structure accumulates more than 100 errors, then this alarm will trigger and notify the appropriate plant maintenance personnel to perform an inspection.

The Condition Must Remain True For A Minimum Of: Alarm analysis processing is performed by the gateway once per second. Enter the number of seconds that the condition must be continuously evaluated as "true" for the alarm to be triggered. A time of 0 seconds means that just a single evaluation of "true" will immediately trigger the alarm.

Send Additional Reminders While The Condition Remains True: If this check box is unchecked, then only one email transmission event will occur when an alarm condition is triggered: further email transmissions will not be attempted

for this alarm unless the alarm condition is first evaluated as “false” (which resets the alarm), and then once again is triggered by a subsequent event.

If this check box is checked, then as long as the alarm condition continues to be evaluated as “true”, subsequent email transmissions will be automatically retriggered every indicated number of minutes for a maximum of the indicated number of times. If at any time during the subsequent transmissions the alarm condition is evaluated as “false”, then the alarm will be reset and email transmissions for this alarm will stop (until the next time the alarm is triggered, of course).

Subject: Enter a string of up to 128 characters in length which will appear in the “subject” line of the alarm email. The body of the alarm email is empty.

10.15.3 Submitting Changes

Whenever any of the Alarm configuration elements have been changed, the “submit” button must be clicked in order to write these settings to the gateway’s filesystem.

Note that because these configuration elements are read from the filesystem only when the gateway boots up, the act of submitting configuration changes will also reset the gateway. Please allow 20 seconds for the gateway to reboot, at which time it will then be operating with the recently-submitted configuration.

10.16 Dashboard Menu

The Dashboard Menu provides access to a variety of gauges, meters and graphs that can be configured to provide an at-a-glance graphical overview of critical application variables in real-time. A total of 10 gauge windows are available (four at a time), and each gauge window can be configured to display any database value via one of six different gauge types. User-defined engineering units, data type, scaling and range limits are also configurable. Refer to Figure 43.

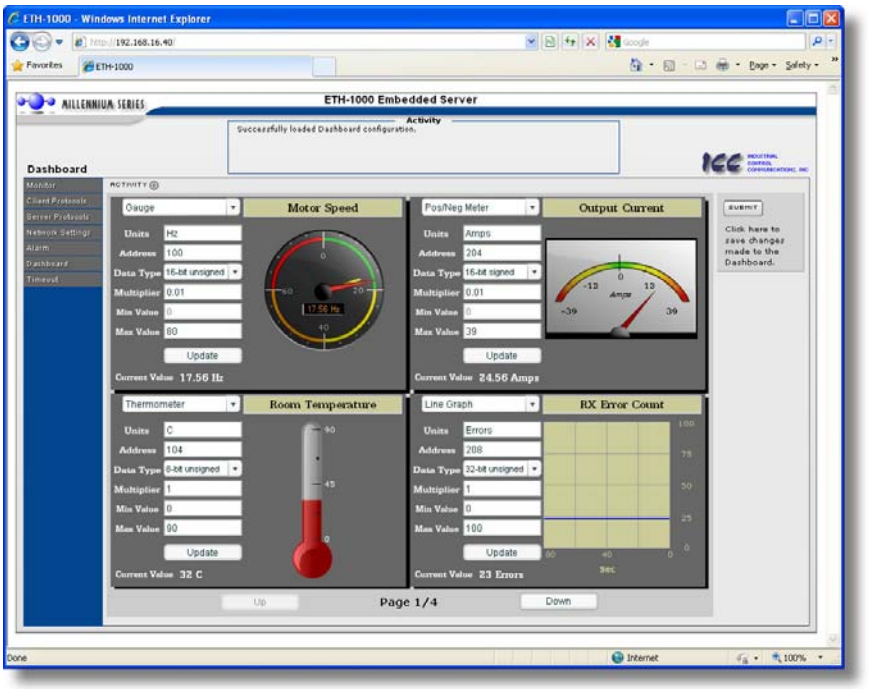


Figure 43: Dashboard Menu

10.16.1 Gauge Window Navigation

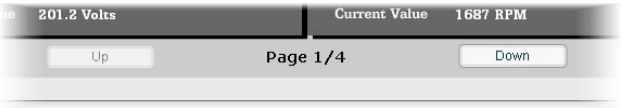


Figure 44: Gauge Window Navigation

Figure 44 shows the two buttons that provide for navigation of the gauge windows. Gauge windows are displayed four at a time in the Dashboard menu, and by clicking the “Up” or “Down” buttons, the gauge windows will scroll in the corresponding direction, two at a time.

10.16.2 Gauge Window Configuration

Each of the gauge windows can be independently configured to display database values with a variety of flexible configuration options. While the behavior and presentation may vary slightly depending on the specific gauge chosen, all of the gauges share the following common elements (refer to Figure 45 for an example):

Gauge Selector: A drop-down selection box in the upper left-hand corner of the gauge window, which allows the user to select the type of gauge that will be displayed.

Title: A text entry box located above the gauge, in which the user can enter a descriptive gauge title comprised of up to 16 characters.

Units: A text entry box in which the user can enter an engineering units string comprised of up to 8 characters. This units string will be appended to all locations in the gauge window that display the designated current value.

Address: The designated database address whose value is to be reflected on the gauge. Valid addresses are 0..4095. The special address designation of -1 disables a gauge, and is the default address setting for all gauges (i.e. all gauges are disabled by default.)

Data Type: Designates the data type (size and signed vs. unsigned) of the data read from the configured address.

Multiplier: The multiplier value is a floating-point number that is used to scale the raw value contained in the database. As its name suggests, the multiplier value is multiplied by the current raw value in order to calculate the gauge's indicated value. Negative multiplier values can also be used if desired.

Min Value: The gauge's minimum indicated value. Negative values can be used if desired (e.g. if a negative Multiplier attribute is used to generate a negative indicated value). Not all gauges allow adjustment of the min value.

Max Value: The gauge's maximum indicated value. Similar to the Min Value attribute, negative values can be used if desired. Indicated value characteristics can even be inverted by setting the Max Value attribute to a value less than the Min Value attribute.

Update Button: Clicking the update button will apply the current configuration attribute settings to the gauge. Note, however, that simply updating the gauge's current display properties does not write these settings to the gateway's filesystem. To save the current configuration of all the gauge windows to the filesystem, the Dashboard menu's "submit" button must be selected (refer to section 10.16.3).

Current Value: The current indicated value is numerically displayed with the configured Units string at the bottom of each gauge window.

The following is a summary of the different available gauge types:

Gauge: Refer to Figure 45. This type of meter implements a rotary dial-type display format. The indicated value and units are shown numerically on the face of the gauge, and via the red indicator needle. The yellow needle shows the previous indicated value, thereby providing a simple historical reference. The "Min Value" attribute is not configurable; this gauge always starts at 0.

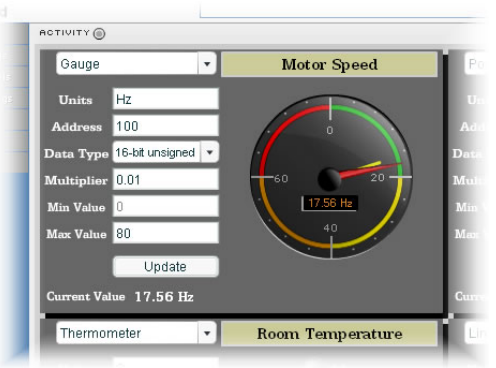


Figure 45: Gauge

BarGraph: Refer to Figure 46. This type of meter implements a linear bar graph display format. Hovering the mouse pointer over the red portion of the graph pops up a tooltip which displays the current indicated value and units.

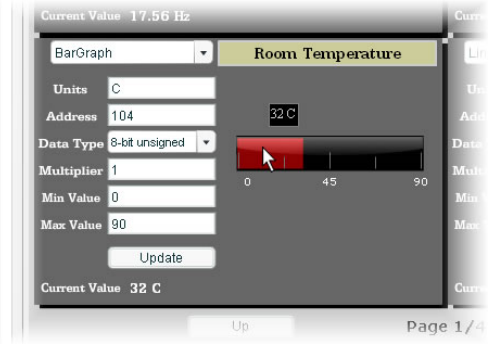


Figure 46: BarGraph

Meter: Refer to Figure 47. This type of meter implements a common panel meter-type display format. The units string is shown on the face of the meter.

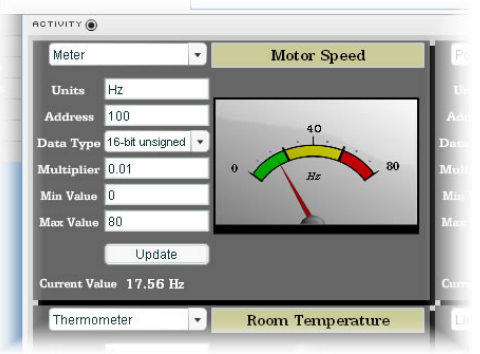


Figure 47: Meter

Pos/Neg Meter: Refer to Figure 48. Similar to the "meter" gauge, this type of meter also implements a common panel meter-type display format, but in this instance the indicated value can be positive or negative (two's complement interpretation). Because the meter placard is always centered around zero, the "Min Value" attribute is not configurable, and the "Max Value" attribute is used for both the maximum positive indicated value as well as the maximum negative indicated value.

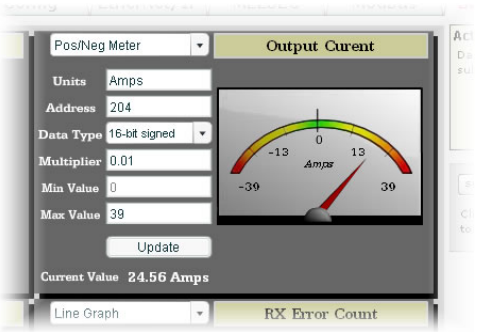


Figure 48: Pos/Neg Meter

Thermometer: Refer to Figure 49. This type of meter implements the universally-identifiable thermometer display format. Hovering the mouse pointer over the red "mercury" portion of the graph pops up a tooltip which displays the current indicated value and units.

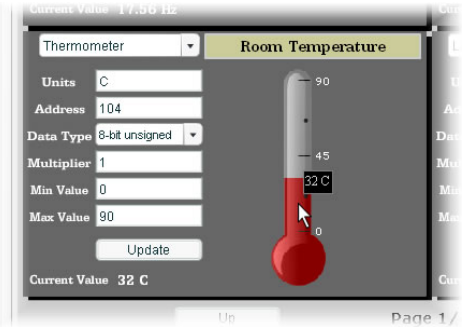


Figure 49: Thermometer

Line Graph: Refer to Figure 50. This type of graph implements a continuously-scrolling historical data logging line graph. Up to 80 seconds worth of historical data is available. Hovering the mouse pointer anywhere on the graph displays a vertical reference line at the corresponding time, and pops up a tooltip which displays the indicated value at that time.

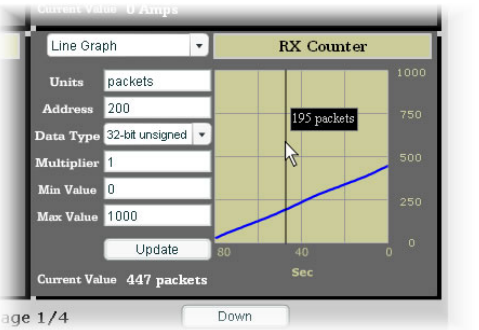


Figure 50: Line Graph

⚠ *Gauge Usage Tip*

At times, it may be convenient to zoom in on a particular gauge or meter in order to more clearly see the indicator, or to fill the computer screen with a particular gauge's image. This can be easily accomplished with the web browser's Flash Player plug-in by right-clicking on the gauge and selecting the desired zoom level (refer to Figure 51).

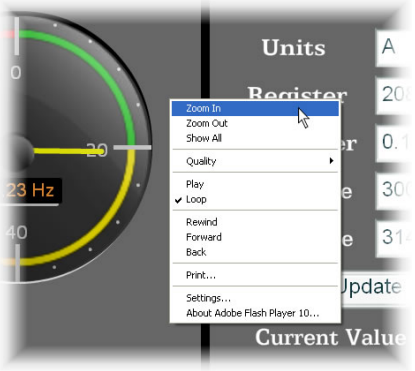


Figure 51: Zooming

10.16.3 Submitting Changes

Whenever any of the gauge window configuration items in the Dashboard menu have been changed, the "submit" button must be selected in order to write these settings to the gateway's filesystem. Note that submitting the Dashboard menu configuration does not require rebooting of the gateway: the changes take effect immediately, and the gateway continues its operation without interruption.

10.17 Timeout Menu

The Timeout menu provides access to configuration items related to timeout processing. Refer to Figure 52.

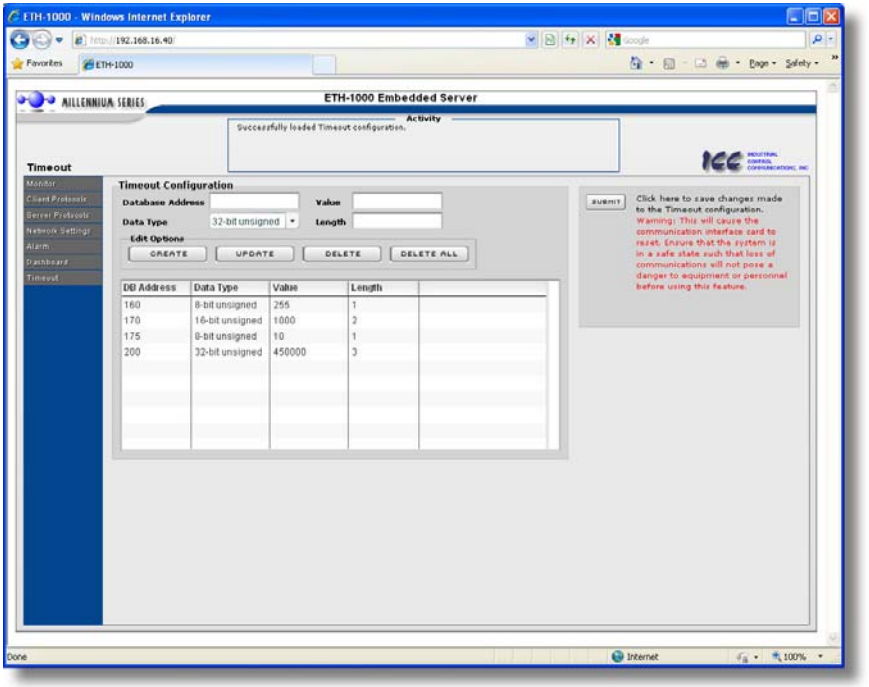


Figure 52: Timeout Menu

10.17.1 Timeout Object Configuration

For detailed information regarding timeout configuration, refer to section 8.3.

11. Protocol-Specific Information

This section will discuss topics that are specific to each of the supported protocols.

11.1 EtherNet/IP

11.1.1 Overview

The EtherNet/IP protocol is an application-level protocol implemented on top of the Ethernet TCP/IP and UDP/IP layers. It shares its object model with ControlNet and DeviceNet through the common Control and Information Protocol (CIP). This protocol allows the transfer of data and I/O over Ethernet.

EtherNet/IP incorporates both the TCP and UDP layers of Ethernet in the transmission of data. Because TCP/IP is a point-to-point topology, EtherNet/IP uses this layer only for explicit messaging; i.e. those messages in which the data field carries both protocol information and instructions for service performance. With explicit messaging, nodes must interpret each message, execute the requested task and generate responses. These types of messages can be used to transmit configuration, control and monitor data.

The UDP/IP protocol layer, which has the ability to multi-cast, is used for implicit (I/O) messaging. With I/O messaging, the data field contains only real-time I/O data; no protocol information is sent because the meaning of the data is pre-defined at the time the connection is established, which in turn minimizes the processing time of the node during run-time. I/O messages are short and have low overhead, which allows for the time-critical performance needed by controllers.

Some other notes of interest are:

- The gateway supports the EtherNet/IP protocol (release 1.0), administered by the Open DeviceNet Vendor Association (ODVA).
- This product has been self-tested by ICC, Inc. and found to comply with ODVA EtherNet/IP Conformance Test Software Version A-6.
- For EtherNet/IP server, the I/O connection sizes for assembly instances 100 and 150 are limited by the gateway only in that they must lie entirely within the database boundaries.
- The gateway's product type code is 12 (Communications Adapter).
- EtherNet/IP server supports unconnected messages (UCMM), and up to 16 simultaneous class 1 (I/O) or class 3 (explicit) connections.
- EtherNet/IP client can simultaneously support a class 1 (I/O) and class 3 (explicit) connection per connection object. Each class 3 connection can support multiple service objects.

- EtherNet/IP client supports up to 4 connection objects and up to 64 service objects.
- The class 1 and class 3 connections operate independently of each other. This applies to both the server and client drivers.
- In the server driver, point-to-point class 1 connected messages will be produced targeting the IP address of the device that instantiated the connection, port 0x08AE (port 2222).
- If a class 1 connection's consuming half (O→T) times out, then the producing half (T→O) will also time-out and will stop producing.
- If a class 1 or class 3 connection timeout occurs, the driver will trigger a timeout event as described in section 8.3. For EtherNet/IP client, class 3 connections timeout after 30 seconds. Class 1 timeout values are calculated by multiplying the **RPI** and **Multiplier**.
- For class 1 client connections, if the **API** (actual packet interval) returned by the server is within ±10ms of the client's configured **RPI** setting, then the connection is successful. This ±10ms restriction is enforced in order to notify the user (by way of an unsuccessful connection) that the server's data rate capabilities "substantially" differ from the configured client settings. The connections will produce and consume data according to the **API**.
- EtherNet/IP client supports Produced Tag connections to access data on a -Logix PLC.

11.1.2 EtherNet/IP Client

The EtherNet/IP client supports both explicit and I/O messaging and can communicate with both EtherNet/IP server and CSP (PCCC) server devices. The following explicit messaging commands are supported:

- Get Attribute Single – Returns the contents of the specified attribute.
- Set Attribute Single – Modifies the contents of the specified attribute.
- Data Table Read – Reads data associated with the tag name.
- Data Table Write – Writes data associated with the tag name.
- Typed Read – Reads a block of data from the file number and offset.
- Typed Write – Writes a block of data to the file number and offset.

11.1.2.1 EtherNet/IP Client Service Objects

The EtherNet/IP client protocol uses service objects to describe what services the gateway should perform. If the service object is configured for a class 1 message type, the gateway will produce and consume data. If the service object is configured for a produced tag message type, the gateway will only consume data. If the service object is configured for an explicit messaging type, the gateway will generate a request. The service objects are activated according to

the **Triggering Mechanism**. For more information on configuring EtherNet/IP service objects, refer to section 8.7.4.2.

11.1.3 EtherNet/IP Server

The EtherNet/IP server supports both explicit and I/O messaging. The I/O messaging is entirely user-configurable, and is invoked when a client opens a connection to the gateway using assembly instances 100 and 150.

The following sections demonstrate specific examples of how to use EtherNet/IP to transfer data between the gateway and Allen-Bradley Logix-brand PLCs.

11.1.3.1 ControlLogix Examples: Setup

This section will demonstrate how to initially setup a ControlLogix PLC (such as a 1756-L61) coupled with a 1756-ENBT/A communication interface (adjust this procedure according to your specific equipment). Later sections will provide specific read/write examples using this configuration with I/O or explicit messaging.

1) Run RSLogix 5000, and create a new configuration.

2) To add a 1756-ENBT/A to your I/O configuration, first switch to offline mode.

3) Right click on the I/O Configuration node in the controller organizer view and choose “New Module…”

4) The “Select Module” window will open.

5) Under “Communications”, select “1756-ENBT/A”, and click OK. Refer to Figure 53.

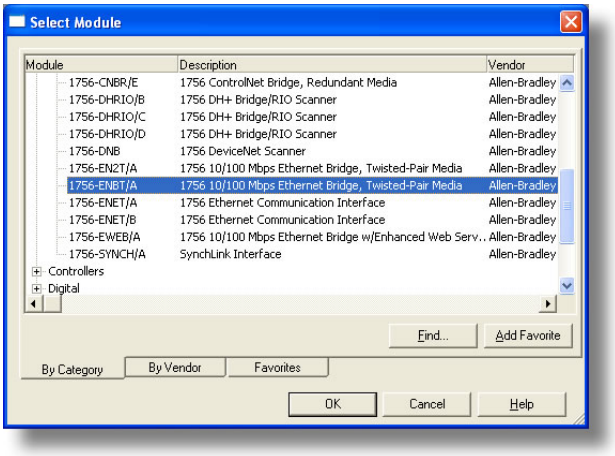


Figure 53: Adding a New 1756-ENBT/A Module

6) The “New Module” window will open. Refer to Figure 54.

7) Assign the Ethernet module a name (we will use “EIP”) and an IP address, deselect “Open Module Properties”, and click OK.

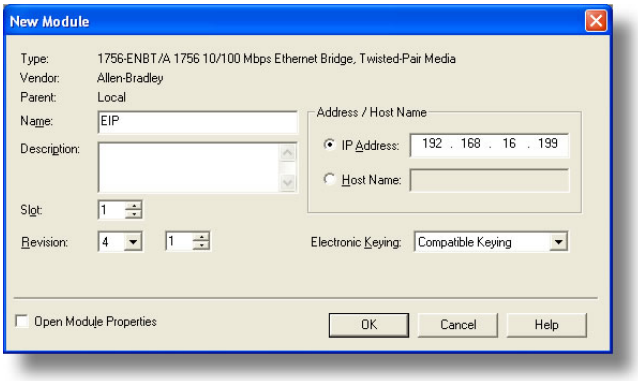


Figure 54: Configuring the New Module

8) Download the configuration.

9) Switch to online mode. Right click on the 1756-ENBT/A module in the I/O Configuration and choose “Properties”.

10) Select the Port Configuration tab from the Module Properties dialog box.

11) Confirm that the IP Address, Subnet Mask and Gateway Address fields are configured correctly. The IP Address must match the IP Address entered when the new module was first created. Refer to Figure 55.

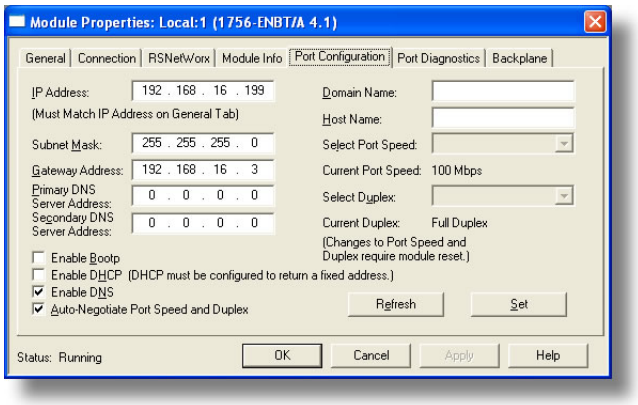


Figure 55: Confirming the Module's Properties

12) Apply any changes to the settings using the “Set” button.

13) You should now be able to confirm that the 1756-ENBT/A module is configured properly by (for example) opening the module's web interface in a web browser.

11.1.3.2 ControlLogix Example: I/O Messaging

This section will demonstrate how to setup and use an EtherNet/IP I/O connection via vendor-specific assembly instances 100 & 150. EtherNet/IP I/O messaging allows the gateway's addresses to be directly mapped into tags in the ControlLogix PLC. Once an I/O connection is established, it is automatically synchronized at an interval defined by the Requested Packet Interval (RPI).

1) Switch to offline mode.

2) Right click on the 1756-ENBT/A node under the I/O Configuration in the controller organizer view and choose "New Module…"

3) Choose "Generic Ethernet Module" in the Select Module dialog box and click "OK". Refer to Figure 56.

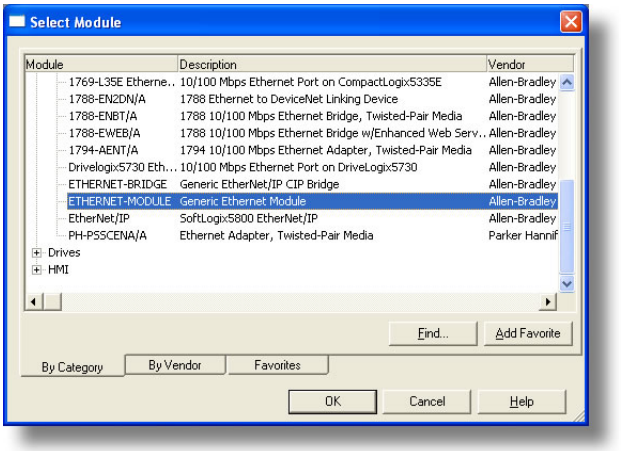


Figure 56: Adding a New Generic Ethernet Module

4) The module properties dialog box will open (refer to Figure 57). Enter a Name and Description which will allow easy identification of the gateway on the network (the tags created in RSLogix 5000 will be derived from this Name). Because the data type of the gateway's database is entirely subject to interpretation, an appropriate "Comm Format" must be chosen which makes the transferred data meaningful to the application. For this specific example, the "Comm Format" is set to "Data-SINT". Enter the IP address of the targeted gateway.

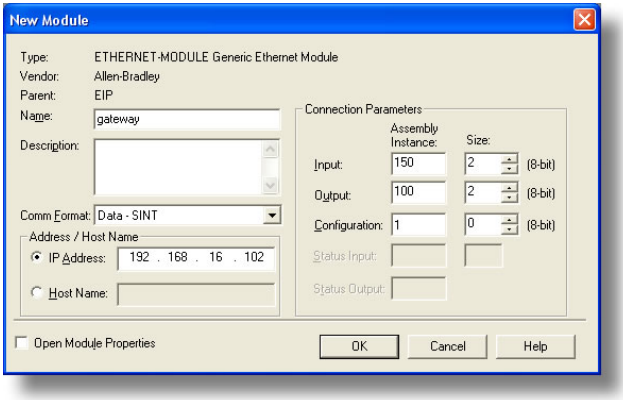


Figure 57: Gateway Module Properties

In the "Connection Parameters" portion of the dialog box, enter the following information:

Input: The Input Assembly is the collection of monitor data that is produced by the gateway and is received as an input to the PLC. The starting address for produced data is specified by the Produced Data Start Address (refer to section 8.7.5.1). The Input Assembly Instance must be set to 150 when connecting to the vendor-specific I/O assembly instances, and the size must be set to the number of elements (each of "Comm Format" type) that we wish to receive from the gateway. Notice that the "number of bits" specified to the right of the "size" selection box changes according to the selected "Comm Format" (the "number of bits" label shown in Figure 57 is "8-bit"). As the gateway will reject any connection requests that attempt to access data past the upper boundary of the database, Equation 1 can be used to determine if the overall requested size is valid. If the result of Equation 1 is less than or equal to 4095 (4kbytes), then the gateway will not reject the connection request.

$$size \times \left(\frac{number\ of\ bits}{8}\right) + Produced\ Data\ Start\ Address \qquad \textbf{Equation 1}$$

Output: The Output Assembly is the collection of command & configuration data that is sent as an output from the PLC and consumed by the gateway. The starting address for consumed data is specified by the Consumed Data Start Address (refer to section 8.7.5.1). The Output Assembly Instance must be set to 100 when connecting to the vendor-specific I/O assembly instances, and the size must be set to the number of elements (each of "Comm Format" type) that we wish to send to the gateway. In a similar manner as the input assembly previously discussed, Equation 2 can be used to determine if the overall requested size is valid. If the result of Equation 2 is less than or equal to 4095 (4kbytes), then the gateway will not reject the connection request.

$$size\ x\ (\frac{number\ of\ bits}{8}) + Consumed\ Data\ Start\ Address$$ **Equation 2**

Configuration: The Configuration Assembly Instance is unused, and its instance number and size are therefore irrelevant (you can just enter "1" and "0", respectively).

When done, click "OK".

5) You should now see the new module (named "ETHERNET-MODULE gateway") in the 1756-ENBT/A branch under the I/O Configuration in the controller organizer view. Right click on this new module, choose "Properties", and select the Connection tab. Refer to Figure 58.

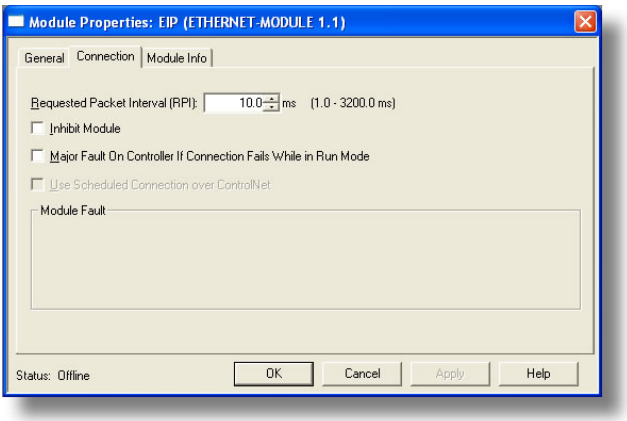


Figure 58: Gateway Module Properties Connection Tab

Confirm the setting of the Requested Packet Interval (RPI). The RPI defines the amount of time (in milliseconds) between data exchanges across an I/O connection. The smallest RPI supported by the gateway is 10ms.

When done, click "OK".

6) After adding the I/O Module to the configuration, the full I/O Configuration tree should appear similar to Figure 59.

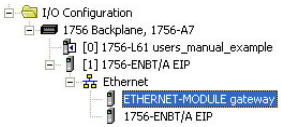


Figure 59: I/O Configuration Tree

7) Switch to online mode and download the project to the PLC. Verify that the newly-added gateway is available and operating correctly by observing any indications shown on the gateway's icon. When the gateway's icon is selected, its status and any available error messages will be displayed in

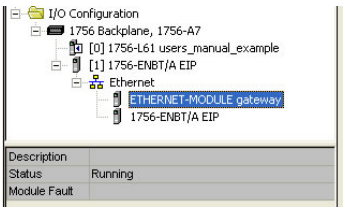


Figure 60: Online Module Status

the area below the project tree. Refer to Figure 60. Also confirm that the gateway's "Network Status" LED should be solid green, indicating an "online/connected" state.

By double-clicking "Controller Tags" in the project tree, it is possible to view the newly-added tags. Refer to Figure 61. The gateway:C configuration tag is unused, the gateway:I tag allows viewing of the input data, and the gateway:O tag allows modification of the output data. These tags will be synchronized with the gateway at whatever rate was established for the module's RPI. If desired, we can directly interact with these tags in order to read and write data on the gateway.

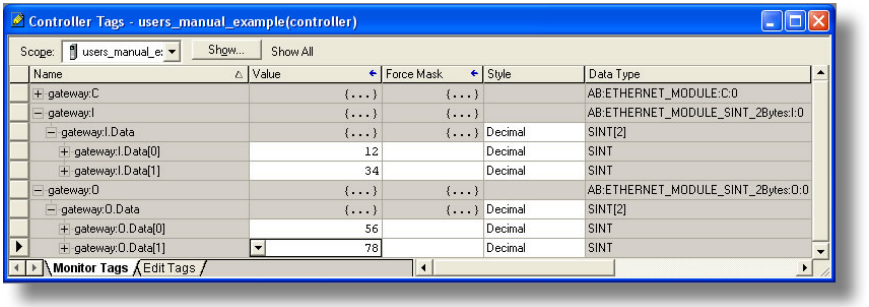


Figure 61: Controller Tags for I/O Access

11.1.3.3 Explicit Messaging Reference and Examples

Class 3 (explicit messaging) can serve both as a direct method of accessing the database and as a bypass mechanism to access data on devices connected to the gateway by reference to "tag names". Tags are read via the EtherNet/IP "data table read" service, and tags are written via the EtherNet/IP "data table write" service. Note that an underscore ("_") is used to separate fields in the tag names.

To read data from the gateway, the application PLC program must reference a "source element" from which to start reading and the "number of elements" to read. The "source element" will be a tag name constructed according to the naming conventions shown in the following sections.

In a similar manner, to write data to the gateway, the application PLC program must reference a "destination element" to which to start writing and the "number of elements" to write. Again, the "destination element" will be a tag name constructed according to the naming convention shown in the following sections.

Whether reading or writing, the "number of elements" can be any quantity from 1 to the maximum allowable length.

11.1.3.3.1 Database Access Tag Reference

Any given database address can be accessed with its own unique tag name with one PLC instruction. Tag names are generated according to the following structure:

[target prefix]_[data type]_[address]

Where
[target prefix] is a 2-character sequence "DB" to indicate database access.

[data type] is a 1-character field that specifies the data type of the values in the response. Enter "b" or "B" to indicate byte (8-bit). Enter "w" or "W" to indicate word (16-bit). Enter "d" or "D" to indicate double word (32-bit). Note that the storage/source element on the requester must also be of the same data size.

[address] is a 1- to 4-character field ("0", "1"…"4094", "4095") corresponding to the referenced database address. Specify the value in decimal format.

Examples
Access bytes starting at database address 0... .....DB_b_0
Access words starting at database address 2048 .....DB_w_2048
Access double words starting at database address 4092.....DB_d_4092

11.1.3.3.2 ControlLogix Example: Read Database Data

This example program will show how to continuously read a block of data from the gateway's internal database with a single MSG instruction. Only one read request is outstanding at any given time.

1) Create new Tags.

a) Double click "Controller Tags" in the controller organizer view.

b) The "Controller Tags" window appears. Refer to Figure 62.

c) Select the "Edit Tags" tab at the bottom.

d) Create a new tag by entering "connection" in the first blank Name field, and change its Data Type to "MESSAGE". This tag will contain configuration information for the MSG instruction.

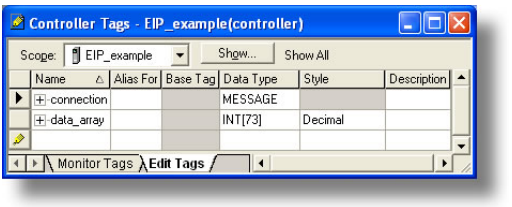


Figure 62: Create New Tags

e) Select the "Monitor Tags" tab. Expand the "connection" tag by clicking on the "+" sign next to the tag name. Scroll down to the connection.UnconnectedTimeout field and change its value from the default 30000000 (30s in 1uS increments) to 1000000 (1s). This value

determines how long to wait before timing out and retransmitting a connection request if a connection failure occurs. Refer to Figure 63.

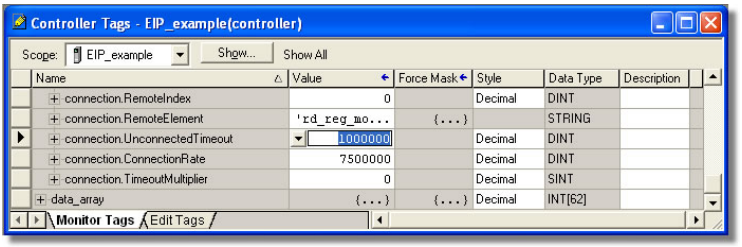


Figure 63: Reduce the UnconnectedTimeout Value

f) Collapse the "connection" tag again by clicking on the "-" sign next to the tag name.

g) Select the "Edit Tags" tab again. Create another new tag by entering "data_array" in the next blank Name field, and change its Data Type by typing in "INT[73]" in the Data Type field. This tag is an array of INTs that will be able to hold up to 73 16-bit values from the gateway. Always make sure that the destination tag size is large enough to hold all elements to be read.

2) Add a MSG instruction to the main program.

a) Double click "MainRoutine" under Tasks …MainTask …MainProgram in the controller organizer view.

b) Right click on the first ladder logic rung in the MainRoutine window and select "Add Ladder Element..."

c) The "Add Ladder Element" window appears.

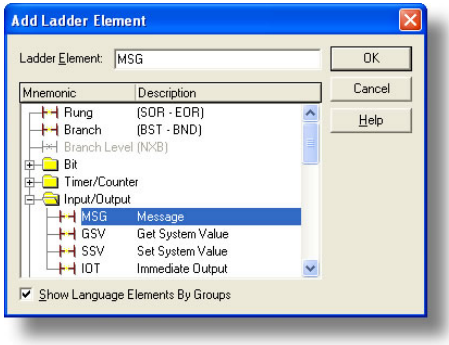


Figure 64: Adding a MSG Instruction

d) Select the "MSG" instruction in the Input/Output folder. Refer to Figure 64.

e) Click OK.

3) Add an XIO element to the main program.

a) Right click on the ladder logic rung containing the MSG instruction in the MainRoutine window and select “Add Ladder Element...” again.

b) The “Add Ladder Element” window appears.

c) Select the “XIO” element in the Bit folder. Refer to Figure 65.

d) Click OK.

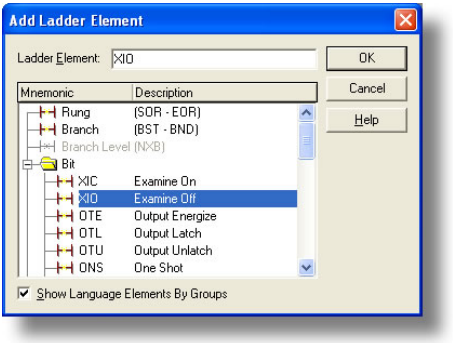


Figure 65: Adding an XIO Element

4) Configure the MSG instruction.

a) Edit the “Message Control” field on the MSG instruction to use the previously-created “connection” tag. Refer to Figure 66.

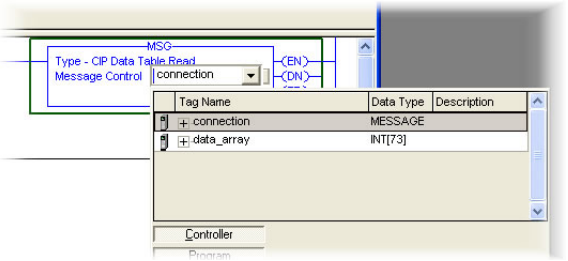


Figure 66: MSG Instruction Tag Assignment

b) Click the message configuration button (“…”) in the MSG instruction. The “Message Configuration” window will open. Refer to Figure 67.

Figure 67: MSG Instruction Configuration

c) “Configuration” tab settings:

i) Change the “Message Type” to “CIP Data Table Read”.

ii) In the "Source Element" field, enter the read tag you wish to access (refer to section 11.1.3.3.1). In this example, we will be reading a total of 21 16-bit words beginning at DB_w_2050 (database address 2050).

iii) Enter the Number Of Elements to read. In this example, we will read 21 16-bit words.

iv) For the Destination Element, either directly type in “data_array[50]”, or select element #50 in the data_array tag via the drop-down box. The destination could be any offset in the data_array tag, as long as the offset plus the Number Of Elements (21) does not exceed the tag’s defined size (73).

d) “Communication” tab settings (refer to Figure 68):

i) Enter the Path to the gateway. A typical path is formatted as “*Local_ENB,2,target_IP_address*”, where:

- *Local_ENB* is the name of the 1756-ENBx module in the local chassis (we named ours “EIP” in section 11.1.3.1),
- *2* is the Ethernet port of the 1756-ENBx module in the local chassis, and
- *target_IP_address* is the IP address of the target node.

In our example, this path would be entered as “EIP,2,192.168.16.102”.

ii) If “Cache Connections” is enabled (checked), the connection remains open after transmission. If disabled (unchecked), the connection is opened before and closed after every transmission. For efficiency, it is recommended to enable “Cache Connections”.

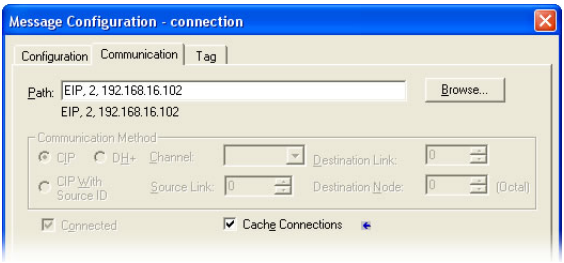


Figure 68: Setting the Communication Path

e) Click “OK” to close the MSG Configuration dialog. At this stage, MainRoutine should look like Figure 69.

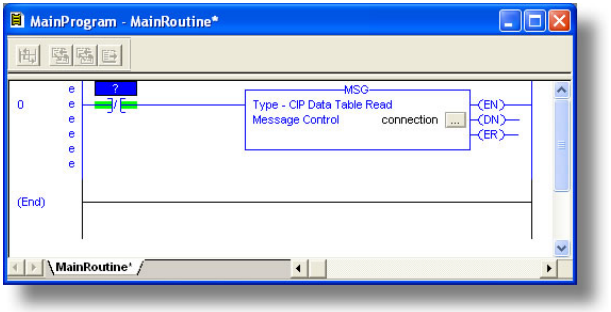


Figure 69: MainRoutine

5) Assign a tag to the XIO element.

a) Double-click on the XIO element located to the left of the MSG block. In the drop-down box, double-click on the "connection.EN" field. Refer to Figure 70. This configuration causes the MSG instruction to automatically retrigger itself when it completes. While this is acceptable for the purposes of this example, it can produce high network utilization. In actual practice, it may be desirable to incorporate additional logic elements to allow triggering the MSG instruction at a specific rate or under specific conditions.

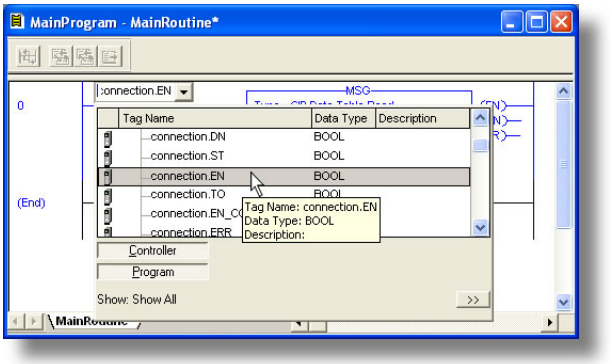


Figure 70: Configure XIO Element

6) The program is now complete. Refer to Figure 71.

7) Save, download and run the program.

a) To view the values of the data being read from the gateway, double-click "Controller Tags" in the controller organizer view.

b) Select the "Monitor Tags" tab.

c) Expand the data_array tag. Refer to Figure 72.

d) 21 16-bit values starting at database address #2050 are being continuously read from the gateway's database and placed in the 21 sequential offsets of data_array starting at the 50th offset (data_array[50]). In Figure 72, we can see that data_array[50] has a value of 11860, data_array[58] has a value of 3558, etc.

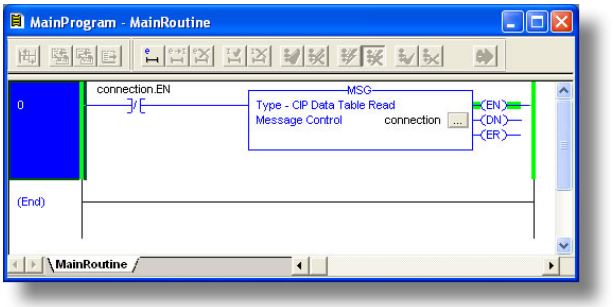


Figure 71: Complete Program

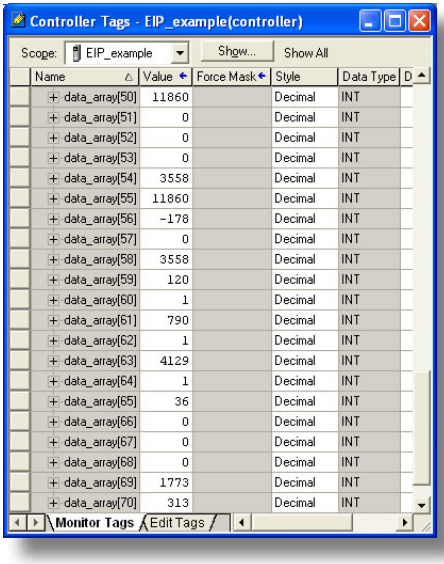


Figure 72: Viewing the Values

11.1.3.3.3 ControlLogix Example: Multiple MSG Instructions

At times, reading from different groups of data may be necessary. To accomplish this task, multiple MSG instructions will need to be implemented in the PLC program.

The configuration and execution for implementing multiple MSG instructions is in general identical to that required for implementing just one MSG instruction. Each MSG instruction will require its own message controller tag. In the case of read MSG instructions, more than one instruction may use the same Destination Element tag, but the storage locations must not overlap. Figure 73 shows an example of two MSG instructions, each accessing different read tags. It is evident from this logic that "rd_connection" and "rd_connection2" are the two independent message controller tags created for these instructions.

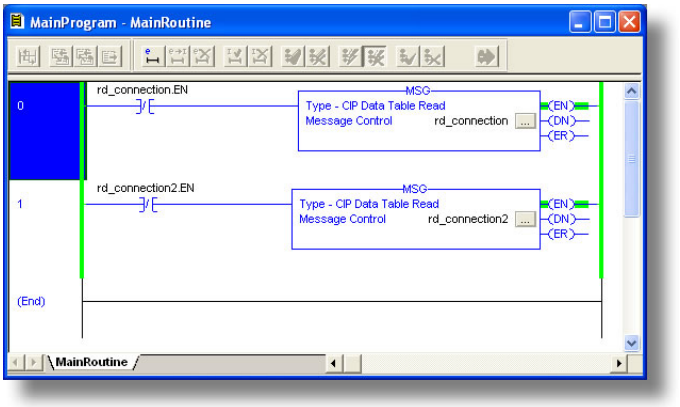


Figure 73: Reading Via Multiple MSG Instructions

11.1.3.3.4 ControlLogix Example: Reading and Writing

Often times, applications may need to both read data from and write data to the gateway. At a minimum, this will require two MSG instructions and two message controller tags. Figure 74 shows an example of three MSG instructions, one for reading and two for writing. The only item of note that differentiates this example from the multiple-read example in section 11.1.3.3.3 is the addition of the en_xx_wr XIC elements. The reason for the addition of these elements is that while reading from a remote device is often continuously performed (monitoring), data is typically written to the remote device only when necessary (i.e. when the value to write has changed). This conserves network bandwidth on the target device. The en_xx_wr elements in this example, therefore, would typically be replaced in an actual application program by user-provided logic that controls the conditions under which write operations would be performed.

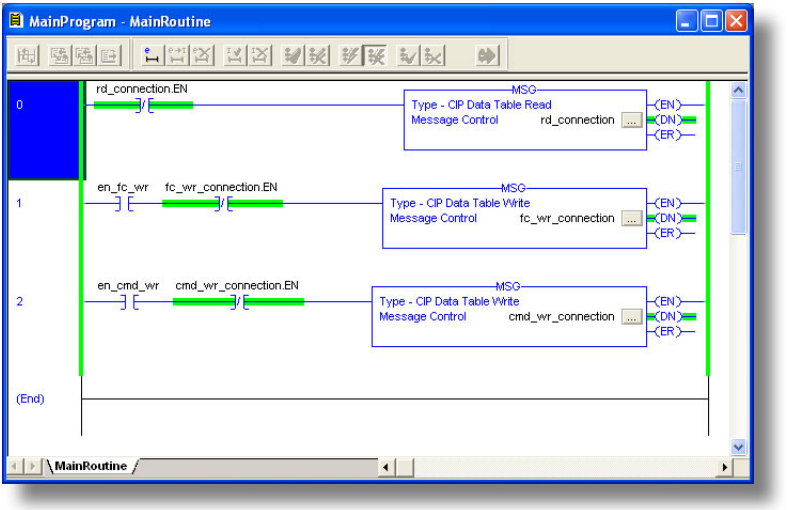


Figure 74: Reading and Writing via MSG Instructions

Figure 75 shows the configuration details of the example fc_wr_connection MSG instruction. Note that the chosen "Message Type" is "CIP Data Table Write", and that this instruction will be writing 21 16-bit values.

Figure 75: MSG Configuration for Writing

Note that when writing data via explicit messaging, use caution to ensure that the commanded addresses are not also simultaneously being commanded in the background via I/O messaging. Indeterminate behavior can occur if MSG instructions and background I/O data transfers are both writing to the same addresses. In other words, if the I/O messaging example procedure detailed in section 11.1.3.2 has already been implemented, and the same program is now being modified to implement explicit messaging, then it is recommended to inhibit the target module by selecting the "Inhibit Module" checkbox in the Connection tab of the Module Properties dialog.

11.1.3.3.5 MELSEC Access/Bypass Request

Any device data on a MELSEC server can be accessed via EtherNet/IP explicit messaging with the knowledge of some MELSEC connection object information. Note that the MELSEC bypass request does not require the use of MELSEC service objects. Tag names are generated according to the following structure:

[target prefix]_[cnxn name]_[device code]_[starting point][_bit flag]

Where

[target prefix] is a 2-character sequence “MC” to indicate MELSEC bypass request.

[cnxn name] is a 1- to 16-character field for the MELSEC connection object name. See “Name” in section 8.7.8.1

[device code] is a field of up to 4 characters in length that corresponds to the device code on the MELSEC device. This device code must be specified in hexadecimal radix format (do not prepend the code with the “0x” characters typically associated with hexadecimal notation). Enter either a 2-character hexadecimal value for 3E frames or a 4-character hexadecimal value for 1E frames. Device codes are not limited to known device codes (refer to Table 6 on page 184); any device code supported by the targeted server may be specified. If specifying a MELSEC word device, the storage/source element on the requester must be a 16-bit data type. If specifying a MELSEC bit device, the storage/source element on the requester must be an 8-bit data type. If the device code is a bit device but is unknown by the gateway, the bit flag must be set (see bit flag below). Device codes are not case-sensitive (i.e. “C3”=”c3”).

[starting point] is a 1- to 8-character decimal value corresponding to the starting point of the device code.

[_bit flag] is an optional 2-character field “_b” which indicates whether or not an unknown device code targets a bit device. If the device code is unknown to the gateway and this field is omitted, then it is assumed that the unknown device code targets a word device. This field is only relevant when the targeted device code is unknown to the gateway: if the device code is known, then the presence or absence of this field is ignored by the gateway.

Examples

Access Request	Tag
Using connection object “Q04”, target Data Register (0xA8) starting at device point 0	MC_Q04_A8_0
Using connection object “Q04”, target Input Relay (0x9C) starting at device point 1000	MC_Q04_9C_1000
Using connection object “QJ71E71”, target unknown word device (0xFF) starting at device point 125	MC_ QJ71E71_FF_125
Using connection object “QJ71E71”, target unknown bit device (0xFE) starting at device point 1250	MC_ QJ71E71_FE_1250_b

Note

If accessing a bit device on a MELSEC server, each bit is actually encoded as an entire byte by the server. If the targeted bit is 0, then the resultant byte is 0; else the resultant byte is non-zero.

Please refer to the MELSEC server equipment user’s manual for limitations on the number of points processed per communication.

11.1.3.3.6 ControlLogix Example: Read MELSEC Device Data

This example program will demonstrate bypass access to device data on a MELSEC server. The configuration and execution for implementing a MSG instruction is in general identical to that in section 11.1.3.3.2. There are, however, two discrepancies to note:

1. The *Source Element* must take the form of a MELSEC tag name structure. In this example, the “cnxn_name” field is “Q04” (as derived from the connection object’s name shown in Figure 76), the device code is a8 (data register), and the starting point is 0. Refer to Figure 77.
2. The data type of the *Destination Element* depends on the device code in use. If the device code targets a word device, then the *Destination Element* must be specified as type INT. If the device code targets a bit device, however, then the *Destination Element* must be specified as type SINT.

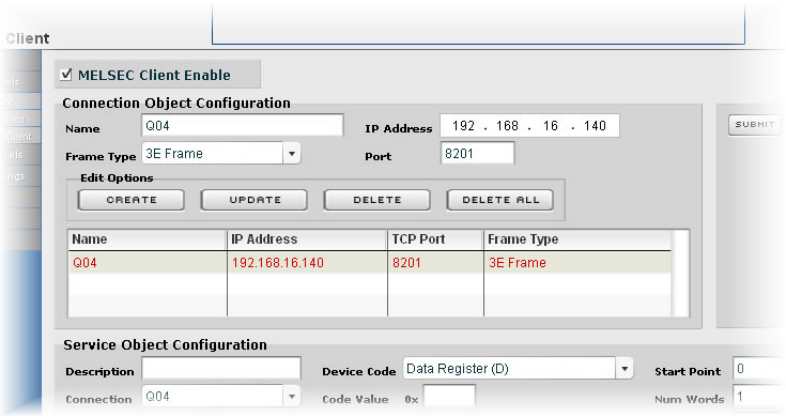


Figure 76: MELSEC Connection Object

Figure 77: MELSEC Bypass MSG Instruction Configuration

This example of directly reading MELSEC device data via explicit messaging can easily be extended to accesses via multiple MSG instructions and to writing data by following the examples detailed in sections 11.1.3.3.3 and 11.1.3.3.4, respectively.

11.2 Allen Bradley CSP

11.2.1 Overview

Ethernet-enabled Allen-Bradley legacy PLCs (such as the PLC5E and SLC-5/05 series) use a protocol called CSP (Client Server Protocol) to communicate over the Ethernet network. The flavor of CSP used by these PLCs is also known as "PCCC" (Programmable Controller Communication Commands) and "AB Ethernet". The gateway supports CSP for direct connectivity to these PLCs.

If a connection timeout or socket-level error occurs, the driver will trigger a timeout event as described in section 8.3

11.2.2 Tag Reference

Data is read from and written to the gateway via CSP by reference to an integer "file/section number" and an "offset/element" within that file. Reading is performed via the CSP "PLC5 Read" (DF1 protocol typed read) service, and writing is performed via the CSP "PLC5 Write" (DF1 protocol typed write) service.

The formula to calculate which database address is targeted in the gateway is provided in Equation 3.

$$target\ address = (file\ number\ -10) \times 100 + offset \qquad \textbf{Equation 3}$$

In Equation 3, "target address" $\in$[0…4095], "file number" $\in$[10…50] (which means N10…N50), and "offset" is restricted only by the limitations of the programming software (but is a value of 4095 max). Table 3 provides some examples of various combinations of file/section numbers and offsets/elements which can be used to access database addresses. Note that there are multiple different combinations of file/section numbers and offsets/elements that will result in the same database address being accessed.

Table 3: CSP Target Address Examples

File/Section Number	Offset/Element	Start Target Address
N10	2	2
N12	62	262
N11	162	262
N27	98	1798
N20	798	1798
N50	95	4095
N10	4095	4095

11.2.3 SLC-5/05 Example: Read Database

This example program will show how to continuously read a block of database addresses from the gateway with a single MSG instruction. Only one read request is outstanding at any given time.

1) Run RSLogix 500, and create a new configuration.

2) Create a control and a data file.

a) Right click Data Files and select New… The "Create Data File" dialog box appears (refer to Figure 78).

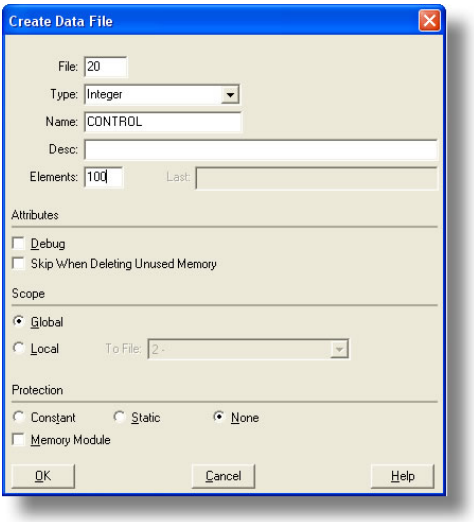


Figure 78: Creating a Control File

b) To create a control file, enter a file number (e.g. 20), set the type to "Integer", enter a descriptive name (e.g. "CONTROL"), and enter a number of elements (e.g. 100). Click OK to create the file. The control file is used to store configuration information pertaining to the functionality of the MSG instruction which will perform the data read.

c) Follow the same procedure to create a data file. This file will be used to store the incoming data read from the gateway. Enter a file number (e.g. 18), set the type to "Integer", enter a descriptive name (e.g. "DATA"), and enter a number of elements (e.g. 200). Refer to Figure 79. Click OK to create the file.

3) Add a MSG instruction to the program.

a) If not already visible, double-click “LAD2” under Project…Program Files in the controller organizer view to bring up the ladder logic program.

b) Right click on the default rung number on the left-hand side of the LAD2 window and select “Insert Rung”.

c) Right click on the rung number of the new editable rung and select “Append Instruction”.

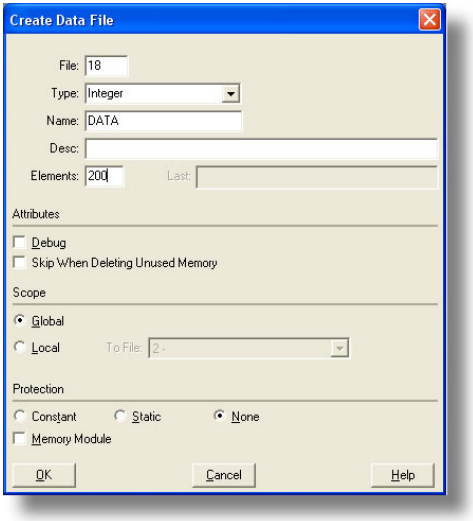


Figure 79: Creating a Data File

d) Select the “MSG” instruction from the “Input/Output” classification, then click OK. Refer to Figure 80.

4) Add an XIO element to the program.

a) Right click on the rung number of the rung currently being edited and select “Append Instruction” again.

b) Select the “XIO” instruction from the “Bit” classification, then click OK. Refer to Figure 81.

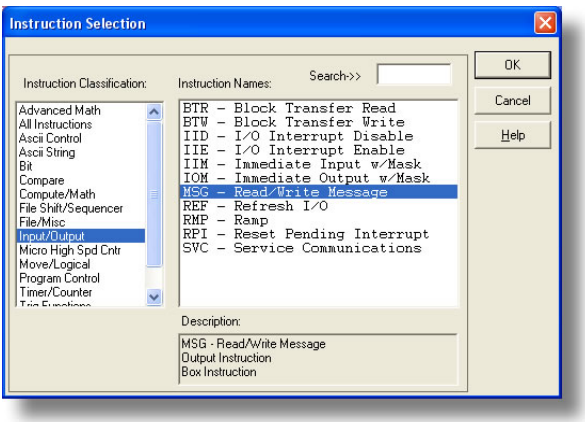


Figure 80: MSG Instruction Selection

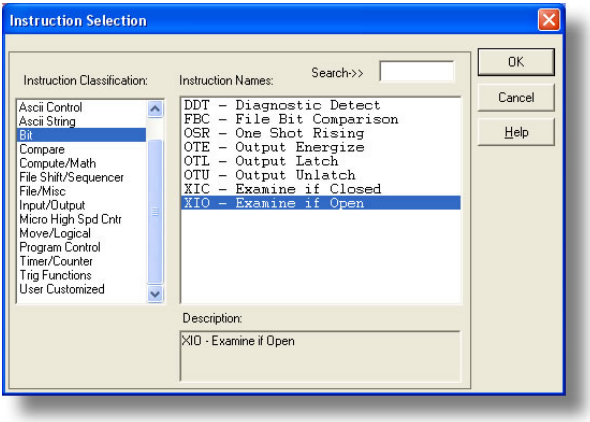


Figure 81: XIO Instruction Selection

5) Configure the MSG instruction.

a) Set the “Read/Write” field to “Read”, “Target Device” field to “PLC5”, “Local/Remote” field to “Local”, and “Control Block” to “N20:0”.

b) Upon hitting the <ENTER> key while in the “Control Block” entry box, the MSG Properties dialog box should appear (or it can be opened by clicking on the “Setup Screen” button at the bottom of the MSG instruction). Refer to Figure 82.

c) In this example, we will be reading a total of 25 16-bit word values from the database beginning at N30:50 (address 2050). To configure this, under “This Controller” set the “Data Table Address” field to N18:1, set the “Size in Elements field” to 25, and set the “Channel” field to 1 (Ethernet).

d) Under “Target Device”, set the “Data Table Address” field to N30:50 (starting target address=2050) and set the “MultiHop” field to Yes to cause the “MultiHop” tab to appear.

e) Under the “MultiHop” tab settings, set the “To Address” in the first row to the gateway’s IP address, and the “To Address” in the second row to 0. Refer to Figure 83.

f) Close the dialog box. At this point, the program should appear as shown in Figure 84.

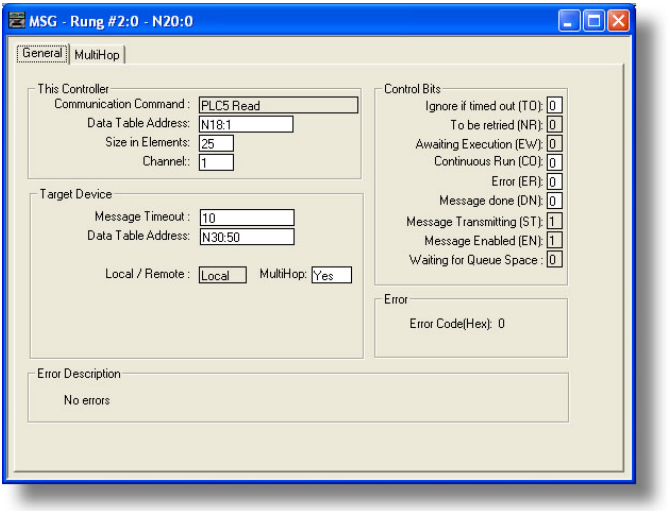


Figure 82: MSG Configuration, "General" Tab

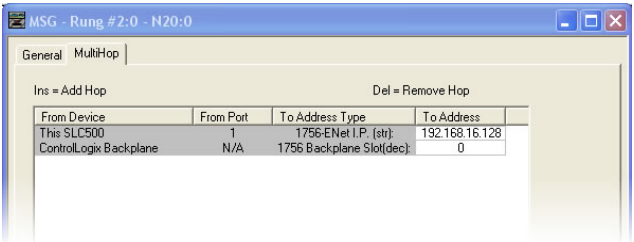


Figure 83: MSG Configuration, "MultiHop" Tab

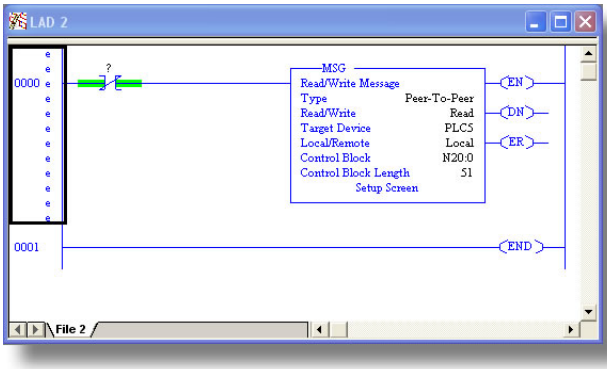


Figure 84: PLC Program after MSG Instruction Configuration

6) Assign a tag to the XIO element.

a) Double-click on the XIO element located to the left of the MSG block. Type in N20:0/15 (MSG instruction's enable bit). This configuration causes the MSG instruction to automatically retrigger itself when it completes. While this is acceptable for the purposes of this example, it can produce high network utilization. In actual practice, it may be desirable to incorporate additional logic elements to allow triggering the MSG instruction at a specific rate or under specific conditions.

7) The program is now complete. Refer to Figure 85.

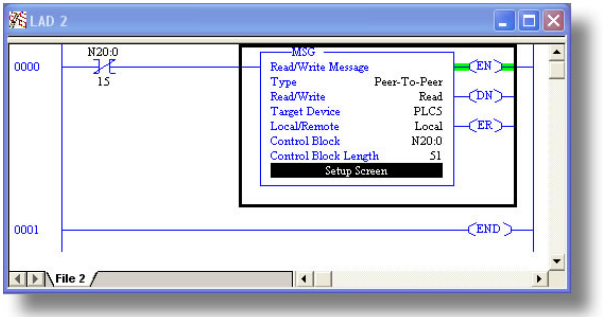


Figure 85: Completed PLC Program

8) Save, download, and run the program.

a) To view the 16-bit data values being read from the gateway, double-click the data file N18 under "Data Files" in the controller organizer view. 25 16-bit word values starting at database address #2050 are being continuously read from the gateway and placed in the 25 sequential offsets of N18 starting at N18:1. Refer to Figure 86. We can see that N18:9 (database address 2068) has a value of 2525, N18:12 (database address 2074) has a value of 610, etc.

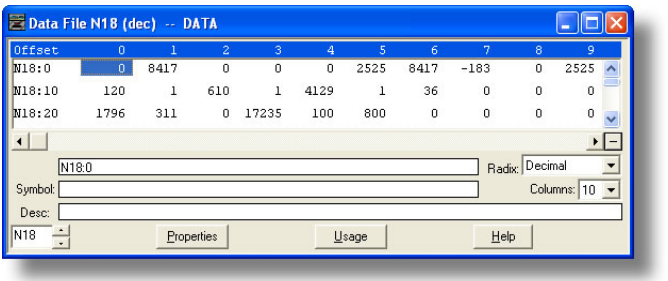


Figure 86: Monitoring the Data Being Read from the Gateway

11.2.4 SLC-5/05 Example: Multiple MSG Instructions

At times, reading from different non-contiguous address of the database may be necessary. To accomplish this task efficiently, multiple MSG instructions can be implemented in the PLC program.

The configuration and execution for implementing multiple MSG instructions is in general identical to that required for implementing just one MSG instruction. Each MSG instruction will require its own message control file. In the case of read MSG instructions, more than one instruction may use the same data file to store the received 16-bit word values, but the storage locations must not overlap. Figure 87 shows an example of two MSG instructions, each accessing different target integer files. It is evident from this logic that N20 and N21 are the two independent message control files created for these instructions.

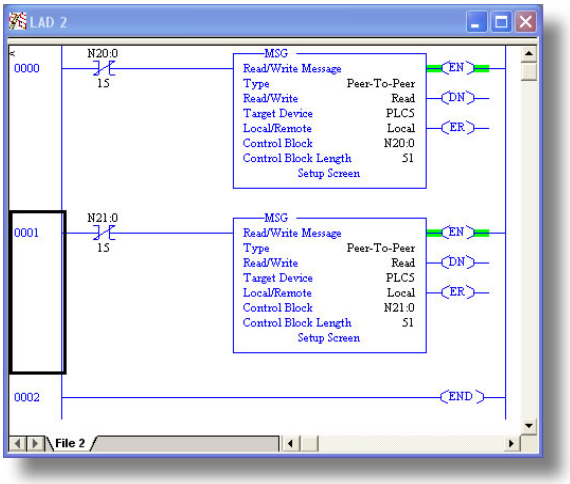


Figure 87: Multiple MSG Instructions

11.2.5 SLC-5/05 Example: Reading and Writing

Often times, applications may need to both read data from and write data to the gateway. At a minimum, this will require two MSG instructions and two message control files. Figure 88 shows an example of two MSG instructions, one for reading and one for writing. Note that the “Read/Write” field of each of the MSG instructions is set according to their function.

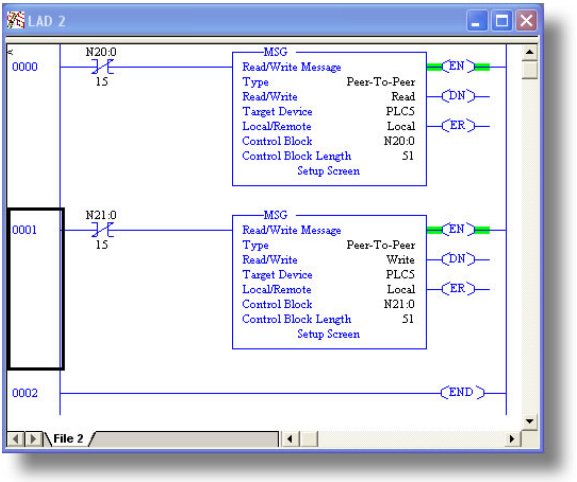


Figure 88: Reading and Writing via MSG Instructions

Figure 89 shows the configuration details of the "write" MSG instruction. Note that this instruction will only be writing to one 16-bit word value (N27:98 = database address 1798). The source Data Table Address in this case is N18:30.

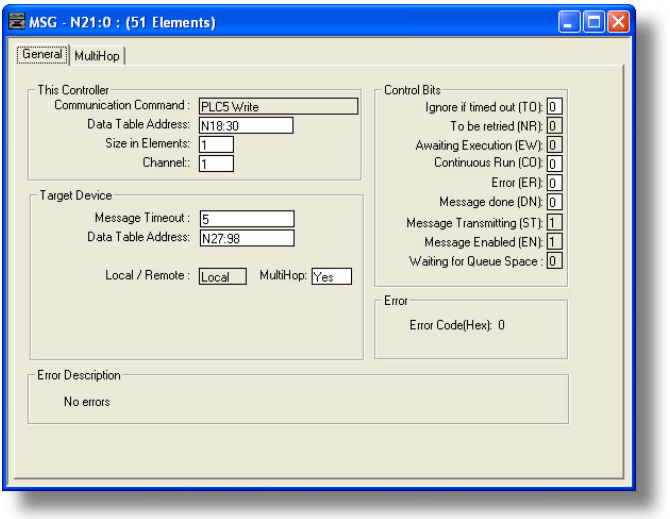


Figure 89: MSG Configuration for Writing

11.3 Modbus/TCP

11.3.1 Modbus/TCP Client

11.3.1.1 Overview

The gateway supports Schneider Electric's Modbus/TCP client protocol, release 1.0. The gateway is conformance class 0 and partial class 1 and class 2 compliant. Other notes of interest are:

- Supported Modbus client functions are indicated in Table 4.

Table 4: Supported Modbus/TCP Client Functions

Function Code	Function	Modbus/TCP Class
01	Read coils	1
02	Read input status	1
03	Read multiple registers	0
04	Read input registers	1
06	Write single register	1
15	Force multiple coils	2
16	Write multiple registers	0

- Supports up to 4 simultaneous connections (defined by the connection objects) and up to 64 service objects.
- 32-bit register accesses are supported in a variety of options and formats.

11.3.1.2 Modbus/TCP Client Service Objects

The Modbus/TCP client protocol uses service objects to describe what services the gateway should perform. For each service object, the gateway will continually read the device data defined within the service object from the designated server, storing the value(s) in the database (if the read function is enabled). When data in the database changes where the service object is mapped, a write request is generated to the designated server notifying it of the changed device data value(s) (if the write function is enabled). For more information on configuring Modbus/TCP service objects, refer to section 8.7.6.2.

11.3.1.3 Register and Discrete Mapping

Refer to section 11.5.1.3.

11.3.2 Modbus/TCP Server

11.3.2.1 Overview

The gateway supports Schneider Electric's Modbus/TCP server protocol, release 1.0. The gateway is conformance class 0 and partial class 1 and class 2 compliant, and allows up to 8 simultaneous Modbus/TCP server connections (sockets). Other notes of interest are:

- Supported Modbus server functions are indicated in Table 5.

Table 5: Supported Modbus/TCP Server Functions

Function Code	Function	Modbus/TCP Class
01	Read coils	1
02	Read input status	1
03	Read multiple registers	0
04	Read input registers	1
05	Write coil	1
06	Write single register	1
15	Force multiple coils	2
16	Write multiple registers	0

- Database data can be accessed as either holding registers (4X references) or input registers (3X references). For example, accessing database address 1300 involves accessing holding register 41301 or input register 31301 (i.e. offset 1301).
- Specific bits within the database can be accessed as either coils (0X references) or discrete inputs (1X references).
- Configuration tip: Improved network utilization may be obtained by appropriately grouping contiguous register assignments in the database. In this way, the "read multiple registers", "read input registers" and "write multiple registers" functions can be used to perform transfers of larger blocks of registers using fewer Modbus transactions compared to a situation where the read/write registers were arranged in an alternating or scattered fashion.
- The "unit identifier" (UI) field of the request packets is ignored.
- The socket timeout time is determined by the "timeout" setting on the web server's "Modbus" tab or via the "timeout" setting of the protocol selection group in the configuration utility (refer to section 8.7.7.1.) This means that if a particular open socket experiences no activity for more than the timeout time setting, then the gateway assumes that the client or network has experienced some sort of unexpected problem, and will close that socket.
- Because the socket timeout determination is performed on a per-socket basis, note that a certain degree of caution must be exercised when using the network timeout feature to avoid "nuisance" timeouts from occurring.

Specifically, do not perform inadvisable behavior such as sending a request from the master device to the gateway, and then closing the socket prior to successfully receiving the unit's response. The reason for this is because the gateway will then experience an error when attempting to respond via the now-closed socket, which will immediately trigger the timeout action. Always be sure to manage socket life cycles "gracefully", and do not abandon outstanding requests.

- If a socket timeout occurs (regardless of whether it was due to a communication lapse or abnormal socket error), the driver will trigger a timeout event as described in section 8.3.
- 32-bit register accesses are supported in a variety of options and formats.

11.3.2.2 Holding & Input Register Mappings

The Modbus/TCP server driver provides read/write support for holding registers (4X references) and read-only support for input registers (3X references). Both holding registers and input registers access the same data. For example, reading Holding Register 4 returns the same data as reading Input Register 4. By default, registers are mapped into the database using the following scheme:

Register 1 is mapped to address 0,
Register 2 is mapped to address 2,
Register 3 is mapped to address 4,
…

Arithmetically, the register-to-address relationship can be described via Equation 4:

$$address = 2 \times (register - 1) \qquad \textbf{Equation 4}$$

Additionally, a register remap object can be created to map a register to a different address in the database, or to map a register that is outside of the default mapping into the database. Refer to section 8.7.7.3 for more information on configuring register remap objects.

For clarity, let's use Equation 4 in a calculation example with a remap object. Let's assume we have defined a register remap object to remap register 25 to database address 62. This means that instead of register 25 mapping to address 48 (as it would with the default mapping), it will now map to address 62. Now say we wish to read registers 24 and 25. We already know that register 25 maps to database address 62, so we must use Equation 4 to calculate what address register 24 is mapped to. Using the equation, we can determine that register 24 is mapped to database address 46. So reading registers 24 and 25 will return data from addresses 46 and 62 in the database, respectively.

11.3.2.3 Coil & Discrete Input Mappings

The Modbus/TCP server driver provides read/write support for coils (0X references) and read-only support for discrete inputs (1X references). These will

collectively be referred to from here on out as simply “discretes”. Accessing discretes does not reference any new physical data: discretes are simply indexes into various bits of existing registers. What this means is that when a discrete is accessed, that discrete is resolved by the gateway into a specific register, and a specific bit within that register. The pattern of discrete-to-register/bit relationships can be described as follows:

Discrete 1...16 map to register #1, bit0...bit15 (bit0=LSB, bit15=MSB)
Discrete 17...32 map to register #2, bit0...bit15, and so on.

Arithmetically, the discrete-to-register/bit relationship can be described as follows: For any given discrete, the register in which that discrete resides can be determined by Equation 5:

$$register = \left\lfloor \frac{discrete + 15}{16} \right\rfloor \qquad \textbf{Equation 5}$$

Where the bracket symbols “⌊ ⌋” indicate the “floor” function, which means that any fractional result (or “remainder”) is to be discarded, with only the integer value being retained.

Also, for any given discrete, the targeted bit in the register in which that discrete resides can be determined by Equation 6:

$$bit = (discrete - 1)\ \%\ 16 \qquad \textbf{Equation 6}$$

Where “discrete” $\in$[1…65535], “bit” $\in$[0…15], and “%” is the modulus operator, which means that any fractional result (or “remainder”) is to be retained, with the integer value being discarded (i.e. it is the opposite of the “floor” function).

For clarity, let’s use Equation 5 and Equation 6 in a calculation example. Say, for instance, that we are going to read coil #34. Using Equation 5, we can determine that coil #34 resides in register #3, as ⌊3.0625⌋ = ⌊3 r1⌋ = 3. Then, using Equation 6, we can determine that the bit within register #3 that coil #34 targets is (34-1)%16 = 1, as 33%16 = mod(2 r1) = 1. Therefore, reading coil #34 will return the value of register #3, bit #1.

Note that discretes are mapped to registers, not database addresses. The location of a given register in the database determines what physical address the discrete will access. Because of this, it is possible to indirectly remap discretes using register remap objects. If a register has been remapped to an alternate database address, then the discretes that map to that register will also be remapped to that alternate address.

11.4 MELSEC Client

11.4.1 Overview

The Mitsubishi MELSEC communication protocol (also known as MC protocol) is an application-level protocol implemented on top of the Ethernet TCP/IP and UDP/IP layers. It is typically used to read and write data from/to a programmable controller CPU using 3E and 1E frames.

The gateway supports both Device Memory Batch Read (0401) and Batch Write (1401) commands for both Bit and Word access for 3E frames. For 1E frames, Device Memory Batch Read Bit (0), Batch Read Word (1), Batch Write Bit (2), and Batch Write Word (3) commands are supported. All MELSEC communication is conducted over only TCP/IP.

Some other notes of interest are:

- Supports auto-detection of frame type. If a MELSEC server device supports both 3E and 1E frames, 3E frames will be given priority. Otherwise, the frame type can be selected manually. Please refer to the MELSEC server equipment user's manual or contact Mitsubishi for assistance.
- Supports up to 8 simultaneous connections (defined by the connection objects) and up to 64 service objects.
- The option to specify the device code allows users to target any device even if the device is unknown by the gateway.
- Device data can be accessed without configuring service objects by using EtherNet/IP explicit messaging. Refer to section 11.1.3.3.5.

11.4.2 MELSEC Service Objects

The MELSEC client protocol uses service objects to describe what services the gateway should perform. For each service object, the gateway will continually read the device data defined within the service object from the designated server, storing the value(s) in the database (if the read function is enabled). When data in the database changes where the service object is mapped, a write request is generated to the designated server notifying it of the changed device data value(s) (if the write function is enabled). For more information on configuring MELSEC service objects, refer to section 8.7.8.2.

11.4.3 MELSEC Device Codes

For user convenience, the gateway provides internal support for a certain number of device codes (refer to Table 6). However, this in no way limits the variety of device codes that the gateway can target. The user has the ability to specify any device code that is supported by the targeted MELSEC server equipment (refer to sections 8.7.8.2 and 11.1.3.3.5). Note that device codes are not case-sensitive (i.e. “C3” is interpreted the same as “c3”).

Table 6: MELSEC Device Code List

Device	Type	3E Frame Device Code	1E Frame Device Code
Annunciator (F)	Bit	93_H	N/A
Counter Coil (CC)	Bit	$C3_H$	4343_H
Counter Contact (CS)	Bit	$C4_H$	4353_H
Counter Current Value (CN)	Word	$C5_H$	$434E_H$
Data Register (D)	Word	$A8_H$	4420_H
Direct Input (DX)	Bit	$A2_H$	N/A
Direct Output (DY)	Bit	$A3_H$	N/A
Edge Relay (V)	Bit	94_H	N/A
Index Register (Z)	Word	CC_H	N/A
Input (X)	Bit	$9C_H$	5820_H
Internal Relay (M)	Bit	90_H	$4D20_H$
Latch Relay (L)	Bit	92_H	N/A
Link Register (W)	Word	$B4_H$	5720_H
Link Relay (B)	Bit	$A0_H$	4220_H
Link Special Register (SW)	Word	$B5_H$	N/A
Link Special Relay (SB)	Bit	$A1_H$	N/A
Motion Register (#)	Word	$E0_H$	N/A
Output (Y)	Bit	$9D_H$	5920_H
Retentive Timer Coil (SC)	Bit	$C6_H$	N/A
Retentive Timer Contact (SS)	Bit	$C7_H$	N/A
Retentive Timer Current Value (SN)	Word	$C8_H$	N/A
Special Register (SD)	Word	$A9_H$	N/A
Special Relay (SM)	Bit	91_H	N/A
Step Relay (S)	Bit	98_H	N/A
Timer Coil (TC)	Bit	$C0_H$	5443_H
Timer Contact (TS)	Bit	$C1_H$	5453_H
Timer Current Value (TN)	Word	$C2_H$	$544E_H$

11.5 Modbus RTU

11.5.1 Modbus RTU Master

11.5.1.1 Overview

The gateway supports the Modbus RTU master protocol on its RS-485 port. Some notes of interest are:

- Supported Modbus master functions are indicated in Table 7.

Table 7: Supported Modbus RTU Master Functions

Function Code	Function
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
06	Preset Single Register
15	Force Multiple Coils
16	Force Multiple Registers

- Requests are fully configurable through service objects.
- 32-bit register accesses are supported in a variety of options and formats.
- The following point types are supported in Modbus Service Objects:
    - Holding Register
    - Input Register
    - Coil Status
    - Input Status

11.5.1.2 Modbus Service Objects

The Modbus RTU master driver uses service objects to describe what services the gateway should perform. For each service object, the gateway will continually read the registers or discretes defined within the service object from the designated slave, storing the value(s) in the database (if the read function is enabled). When data in the database changes where the registers or discretes are mapped, a write request is generated to the designated slave notifying it of the changed register or discrete value(s) (if the write function is enabled). For more information on configuring Modbus service objects, refer to section 8.7.10.2.

11.5.1.3 Register and Discrete Mapping

Holding and Input Registers

Modbus registers are mapped in the database as 2-byte values. This means that each register in a service object takes up two database addresses. For example if a service object's starting register is "1", the number of registers is "5", and the database address is "100", then registers 1 through 5 will be mapped at database addresses 100 through 109 (register 1 mapped at address 100 and 101, register 2 mapped at address 102 and 103 and so on).

Coils and Discrete Inputs

Coils and Discrete Inputs, from here on collectively referred to as "discretes", are mapped on a bit-by-bit basis in the database starting with the least significant bit of the database byte. For example, if a service object's starting discrete is "1", the number of discretes is "19", and the database address is "320", then discrete 1 through 8 will be mapped to bit 0 through 7, respectively, at address 320, discrete 9 through 16 will be mapped to bit 0 through 7, respectively, at address 321, and discrete 17 through 19 will be mapped to bit 0 through 2, respectively, at address 322. The remaining 5 bits in the byte at address 322 are unused.

11.5.2 Modbus RTU Slave

11.5.2.1 Overview

The gateway supports the Modbus RTU slave protocol on its RS-485 port. Some notes of interest are:

- Supported Modbus slave functions are indicated in Table 8.

Table 8: Supported Modbus RTU Slave Functions

Function Code	Function
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
08	Diagnostics (Subfunction 0 only)
15	Force Multiple Coils
16	Force Multiple Registers

- Database data can be accessed as either holding registers (4X references) or input registers (3X references). For example, accessing database address 1300 involves accessing holding register 41301 or input register 31301 (i.e. offset 1301).

- Specific bits within the database can be accessed as either coils (0X references) or discrete inputs (1X references).
- 32-bit register accesses are supported in a variety of options and formats.
- Because the transaction is handled locally within the gateway, write data checking is not available. For example, if a write is performed to a register with a data value that is out-of-range of the corresponding data element, no Modbus exception will be immediately returned.
- Configuration tip: Improved network utilization may be obtained by appropriately grouping contiguous register assignments in the database. In this way, the “read multiple registers”, “read input registers” and “write multiple registers” functions can be used to perform transfers of larger blocks of registers using fewer Modbus transactions compared to a situation where the read/write registers were arranged in an alternating or scattered fashion.

11.5.2.2 Holding & Input Register Mappings

The Modbus RTU slave driver provides read/write support for holding registers (4X references) and read-only support for input registers (3X references). Both holding registers and input registers access the same data. For example, reading Holding Register 4 returns the same data as reading Input Register 4. By default, registers are mapped into the database using the following scheme:

Register 1 is mapped to address 0,
Register 2 is mapped to address 2,
Register 3 is mapped to address 4,
…

Arithmetically, the register-to-address relationship can be described via Equation 7:

$$address = 2 \times (register - 1) \qquad \textbf{Equation 7}$$

Additionally, a register remap object can be created to map a register to a different address in the database, or to map a register that is outside of the default mapping into the database. Refer to section 8.7.11.2 for more information on configuring register remap objects.

For clarity, let’s use Equation 7 in a calculation example with a remap object. Let’s assume we have defined a register remap object to remap register 25 to database address 62. This means that instead of register 25 mapping to address 48 (as it would with the default mapping), it will now map to address 62. Now say we wish to read registers 24 and 25. We already know that register 25 maps to database address 62, so we must use Equation 7 to calculate what address register 24 is mapped to. Using the equation, we can determine that register 24 is mapped to database address 46. So reading registers 24 and 25 will return data from addresses 46 and 62 in the database, respectively.

11.5.2.3 Coil & Discrete Input Mappings

The Modbus RTU slave driver provides read/write support for coils (0X references) and read-only support for discrete inputs (1X references). These will collectively be referred to from here on out as simply “discretes”. Accessing discretes does not reference any new physical data: discretes are simply indexes into various bits of existing registers. What this means is that when a discrete is accessed, that discrete is resolved by the gateway into a specific register, and a specific bit within that register. The pattern of discrete-to-register/bit relationships can be described as follows:

Discrete 1...16 map to register #1, bit0...bit15 (bit0=LSB, bit15=MSB)
Discrete 17...32 map to register #2, bit0...bit15, and so on.

Arithmetically, the discrete-to-register/bit relationship can be described as follows: For any given discrete, the register in which that discrete resides can be determined by Equation 8:

$$register = \left\lfloor \frac{discrete + 15}{16} \right\rfloor \qquad \textbf{Equation 8}$$

Where the bracket symbols “$\lfloor\ \rfloor$” indicate the “floor” function, which means that any fractional result (or “remainder”) is to be discarded, with only the integer value being retained.

Also, for any given discrete, the targeted bit in the register in which that discrete resides can be determined by Equation 9:

$$bit = (discrete - 1)\ \%\ 16 \qquad \textbf{Equation 9}$$

Where “discrete” $\in$[1…65535], “bit” $\in$[0…15], and “%” is the modulus operator, which means that any fractional result (or “remainder”) is to be retained, with the integer value being discarded (i.e. it is the opposite of the “floor” function).

Conversely, for any bit in a register, the targeted discrete corresponding to that bit can be calculated by Equation 10:

$$discrete = 16 \times (register - 1) + bit + 1 \qquad \textbf{Equation 10}$$

For clarity, let’s use Equation 8 and Equation 9 in a calculation example. Say, for instance, that we are going to read coil #34. Using Equation 8, we can determine that coil #34 resides in register #3, as $\lfloor 3.0625 \rfloor = \lfloor 3\ r1 \rfloor = 3$. Then, using Equation 9, we can determine that the bit within register #3 that coil #34 targets is (34-1)%16 = 1, as 33%16 = mod(2 r1) = 1. Therefore, reading coil #34 will return the value of register #3, bit #1.

Note that discretes are mapped to registers, not database addresses. The location of a given register in the database determines what physical address the discrete will access. Because of this, it is possible to indirectly remap discretes using register remap objects. If a register has been remapped to an alternate database address, then the discretes that map to that register will also be remapped to that alternate address.

11.5.3 Modbus RTU Sniffer

11.5.3.1 Overview

The gateway supports a Modbus RTU sniffer driver on its RS-485 port. This driver enables fully non-intrusive insight into any existing Modbus RTU network consisting of a master and at least one slave. The driver can be configured to "sniff" the requests of the master and log the responses of the slave(s) into the database. Some notes of interest are:

- Supported Modbus functions are indicated in Table 9.

Table 9: Supported Modbus RTU Sniffer Functions

Function Code	Function
03	Read Holding Registers
04	Read Input Registers
06	Preset Single Register
16	Force Multiple Registers

- The filtering of specific actions targeting registers of interest is fully configurable through service objects.
- Both Holding and Input Registers are supported in Modbus Service Objects.
- The Modbus Sniffer Service Objects are identical to those of the Modbus Master Service Objects with the exception that instead of the gateway itself generating requests, it must rely on the existing Modbus master to make requests on its behalf. Therefore, if the master never reads or writes a certain register that is configured in a service object on the gateway, the value of that register will never be updated. For more information on Modbus Service Objects, refer to section 11.5.1.2.
- The Modbus Sniffer driver never transmits on the Modbus network being sniffed.

11.6 BACnet

The gateway supports BACnet/IP server (via Ethernet) and both BACnet MS/TP client and server drivers (via RS-485). Both MS/TP client and server drivers act as masters on the network, meaning they are actively involved in token management.

11.6.1 Protocol Implementation Conformance Statement

BACnet Protocol

Date:	August 22, 2008
Vendor Name:	ICC, Inc.
Product Name:	Millennium Series Ethernet Gateway
Product Model Number:	ETH-1000
Applications Software Version:	V2.100
Firmware Revision:	V2.100
BACnet Protocol Revision:	2

Product Description:

The ETH-1000 is an Ethernet to RS-485 multiprotocol gateway. This product supports native BACnet, connecting directly to the IP and/or MS/TP LAN using baud rates of 4800, 9600, 19200, 38400, 57600, 76800, and 115200. The device can be configured as a BACnet/IP server, MS/TP client and/or MS/TP server.

BACnet Standard Device Profile (Annex L):

☐ BACnet Operator Workstation (B-OWS)
☐ BACnet Building Controller (B-BC)
☐ BACnet Advanced Application Controller (B-AAC)
☒ BACnet Application Specific Controller (B-ASC)
☐ BACnet Smart Sensor (B-SS)
☐ BACnet Smart Actuator (B-SA)

BACnet Interoperability Building Blocks Supported (Annex K):

☒ Data Sharing – ReadProperty-A (DS-RP-A)
☒ Data Sharing – ReadProperty-B (DS-RP-B)
☒ Data Sharing – ReadPropertyMultiple-B (DS-RPM-B)
☒ Data Sharing – WriteProperty-A (DS-WP-A)
☒ Data Sharing – WriteProperty-B (DS-WP-B)
☒ Data Sharing – WritePropertyMultiple-B (DS-WPM-B)
☒ Data Sharing – COV-B (DS-COV-B) (BACnet/IP only)
☒ Device Management – Dynamic Device Binding-A (DM-DDB-A)
☒ Device Management – Dynamic Device Binding-B (DM-DDB-B)
☒ Device Management – Dynamic Object Binding-B (DM-DOB-B)
☒ Device Management – DeviceCommunicationControl-B (DM-DCC-B)
☒ Device Management – ReinitializeDevice-B (DM-RD-B)

Segmentation Capability:

None

☐ Segmented requests supported Window Size _____
☐ Segmented responses supported Window Size _____

Standard Object Types Supported:

See “Object Types/Property Support Table” for object details.

Data Link Layer Options:

☒ BACnet IP, (Annex J)
☐ BACnet IP, (Annex J), Foreign Device
☐ ISO 8802-3, Ethernet (Clause 7)
☐ ANSI/ATA 878.1, 2.5 Mb. ARCNET (Clause 8)
☐ ANSI/ATA 878.1, RS-485 ARCNET (Clause 8), baud rate(s) _____
☒ MS/TP master (Clause 9), baud rate(s): 4800, 9600, 19200, 38400, 57600, 76800, 115200
☐ MS/TP slave (Clause 9), baud rate(s): _____
☐ Point-To-Point, EIA 232 (Clause 10), baud rate(s): _____
☐ Point-To-Point, modem, (Clause 10), baud rate(s): _____
☐ LonTalk, (Clause 11), medium: _____
☐ Other: _____

Device Address Binding:

Is static device binding supported? (This is currently for two-way communication with MS/TP slaves and certain other devices.) ☒ Yes ☐ No

Networking Options:

☐ Router, Clause 6 - List all routing configurations
☐ Annex H, BACnet Tunneling Router over IP
☐ BACnet/IP Broadcast Management Device (BBMD)
Does the BBMD support registrations by Foreign Devices? ☐ Yes ☐ No

Character Sets Supported:

Indicating support for multiple character sets does not imply that they can all be supported simultaneously.

☒ ANSI X3.4 ☐ IBM™/Microsoft™ DBCS ☐ ISO 8859-1
☐ ISO 10646 (UCS-2) ☐ ISO 10646 (UCS-4) ☐ JIS C 6226

If this product is a communication gateway, describe the types of non-BACnet equipment/networks(s) that the gateway supports:

Refer to Section 11 for other supported protocols.

Datatypes Supported:
The following table summarizes the datatypes that are accepted (in the case of a write property service) and returned (in the case of a read property service) when targeting the present value property of each supported object type.

Object Type	Service	
	Read Property	**Write Property**
Analog Output **Analog Value**	Real	Real, Unsigned, Integer, Null
Analog Input	Real	N/A
Binary Output **Binary Value**	Enumerated	Enumerated, Boolean, Real, Unsigned, Integer, Null
Binary Input	Enumerated	N/A

Object Types/Property Support Table

The following table summarizes the Object Types/Properties supported.

Property	Object Type						
	Device	Binary Input	Binary Output	Binary Value	Analog Input	Analog Output	Analog Value
Object Identifier	R	R	R	R	R	R	R
Object Name	R	R	R	R	R	R	R
Object Type	R	R	R	R	R	R	R
System Status	R						
Vendor Name	R						
Vendor Identifier	R						
Model Name	R						
Firmware Revision	R						
App Software Revision	R						
Protocol Version	R						
Protocol Revision	R						
Services Supported	R						
Object Types Supported	R						
Object List	R						
Max APDU Length	R						
Segmentation Support	R						
APDU Timeout	R						
Number APDU Retries	R						
Max Master	R						
Max Info Frames	R						
Device Address Binding	R						
Database Revision	R						
Present Value		R	W	W	R	W	W
Status Flags		R	R	R	R	R	R
Event State		R	R	R	R	R	R
Out-of-Service		R	R	R	R	R	R
Units					R	R	R
Priority Array			R	R		R	R
Relinquish Default			R	R		R	R
Polarity		R	R				
Inactive Text		R	R	R			
Active Text		R	R	R			

R – readable using BACnet services
W – readable and writable using BACnet services

11.6.2 BACnet/IP and BACnet MS/TP Client

11.6.2.1 Overview

The gateway supports BACnet IP client on its Ethernet port and BACnet MS/TP client on its RS-485 port. Some notes of interest are:

- The gateway supports reading and writing the present value property of BACnet objects in devices on the network.
- Requests are fully configurable through service objects.
- Supported BACnet objects include:
    - Analog Input
    - Analog Output
    - Analog Value
    - Binary Input
    - Binary Output
    - Binary Value
- Supported MS/TP baud rates include:
    - 4800
    - 9600
    - 19200
    - 38400
    - 57600
    - 76800
    - 115200
- MS/TP static device binding is supported.

11.6.2.2 BACnet Service Objects

The BACnet Client protocol uses service objects to describe what services the gateway should perform. For each service object, the gateway will continually read the present value of the defined BACnet object within the service object from the designated device, storing the value(s) in the database (if the read function is enabled). When data in the database changes where the BACnet objects are mapped, a write request is generated to the designated device notifying it of the changed present value(s) of the BACnet object(s) (if the write function is enabled). For more information on configuring BACnet MS/TP service objects, refer to section 8.7.13.3. For more information on configuring BACnet/IP service objects, refer to section 8.7.2.2.

11.6.2.3 Device Binding

Dynamic Device Binding

(MS/TP only) In order for a BACnet client to request data from other devices, it must first learn what addresses those devices are located at on the network. BACnet client devices can use dynamic device binding to learn the addresses of other devices on the network. This is done by sending a Who-Is request on the network. Any devices whose device instance falls within the range of the Who-Is

request will respond with an I-Am response, informing the client of what network address its device instance is associated with. By default, the gateway will use dynamic device binding if a service object is not configured to use static device binding.

Static Device Binding

(MS/TP only) Not all BACnet devices support dynamic device binding. If the gateway needs to request data from an MS/TP slave, or an MS/TP master that doesn't support dynamic device binding, then static device binding must be used. Static device binding allows the user to manually define the information that the client would normally acquire using dynamic device binding. The only additional information the user must define is the network address of the destination device. This feature may also be useful if the destination device instance is unknown, but the network address of the device is known. In this case, an arbitrary device instance may be used (as long as it does not conflict with any other device instances in other defined service objects) and the destination address must be set to the network address of the device.

11.6.2.4 BACnet Object Mapping

Analog Objects

Analog objects are mapped in the database as either an 8-bit, 16-bit, or 32-bit value, depending on the data type selected. This means that each analog object in a service object consumes one, two, or four database addresses, respectively. For example, if a service object's starting analog output instance is "1", the number of instances is "5", the database address is "100", and the data type is "32-bit Unsigned", then AO 1 through 5 will be mapped at database addresses 100 through 119 (AO 1 is mapped at address 100 through 103, AO 2 is mapped at address 104 and 107, and so on).

Binary Objects

Binary objects are mapped on a bit-by-bit basis in the database starting with the least significant bit of the database byte. For example, if a service object's starting binary output instance is "1", the number of instances is "12", and the database address is "240", then BO 1 through 8 will be mapped to bit 0 through 7, respectively, at address 240, and BO 9 through 12 will be mapped to bit 0 through 3, respectively, at address 241. The remaining 4 bits in the byte at address 241 are unused.

11.6.3 BACnet/IP and BACnet MS/TP Server

11.6.3.1 Overview

The gateway supports BACnet IP server on its Ethernet port and BACnet MS/TP server on its RS-485 port. Some notes of interest are:

- Fully configurable BACnet objects.
- Supported BACnet objects include:
    - Analog Input
    - Analog Output
    - Analog Value
    - Binary Input
    - Binary Output
    - Binary Value
- Supported MS/TP baud rates include:
    - 4800
    - 9600
    - 19200
    - 38400
    - 57600
    - 76800
    - 115200
- The gateway supports the BACnet/IP (Annex J) protocol over Ethernet via UDP port 47808.
- Binary Objects support custom Active and Inactive Text.
- BACnet/IP supports COVs.

11.6.3.2 BACnet Objects

The BACnet server hosts BACnet objects which contain many different properties for any BACnet client on the network to access. The gateway supports seven different BACnet objects: Device, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, and Binary Value. All supported properties of these objects are readable, while only the present value property is writable (for Outputs and Values only). Refer to section 11.6.1 for the list of properties each object supports. The objects and their properties are configured using the configuration utility (or, optionally, via the web interface for BACnet IP). Refer to sections 8.7.2 and 8.7.14 for more information on configuring BACnet objects.

11.6.3.3 Supported BACnet Services

This section details the BACnet services that are supported:

Read Property

This service is used to request data from the gateway about one of its BACnet object's properties.

Read Property Multiple

This service is used to request data from the gateway about several of its BACnet objects' properties.

Write Property

This service is used to send data to the gateway to change the value of one of its BACnet object's properties.

Note that write priorities are ignored by the gateway.

Write Property Multiple

This service is used to send data to the gateway to change the value of several of its BACnet objects' properties.

Note that write priorities are ignored by the gateway.

Dynamic Device Binding

This service is used to discover the gateway on the network. Upon receiving a Who-Is request on the network, the gateway will generate an I-Am response, if its instance number is included in the range of the request. This allows other devices to resolve the gateway's network address.

Dynamic Object Binding

This service is used to discover the gateway's objects on the network. Upon receiving a Who-Has request on the network, the gateway will generate an I-Have response, if that object exists on the device. This allows other devices to resolve which devices on the network have specific BACnet objects.

Device Communication Control

This service is used to halt responses to requests directed at the gateway for a defined amount of time or indefinitely. Once communication is disabled, until the defined amount of time has expired the device will only respond to a device communication control service that re-enables communication, or a reinitialize device service that resets the device. This service is generally only used for commissioning purposes.

Reinitialize Device

This service is used to reset the device. The gateway does not distinguish between a warm and cold restart. This service is password protected. To successfully reset the gateway, "icc" must be used as the password.

11.7 Profinet IO

The gateway supports both cyclic data exchange and acyclic read/write requests. The cyclic data exchange is a high speed operation that bypasses the TCP/IP layer.

Some other notes of interest are:

- 9 standard I/O modules are available for selection by the controller. The standard I/O modules follow the naming conventions of "IN:XXXX, OUT:YYYY", where XXXX is the number of input bytes to be sent to the controller, and YYYY is the number of output bytes to be sent to the gateway.
- When using standard I/O modules, up to 512 bytes can be sent to the gateway, and up to 512 bytes can be retrieved from the gateway.
- To improve efficiency, the GSDML file can be modified by the user to support almost any combination of up to 1400 input bytes and up to 1400 output bytes. Refer to the Profinet controller user's manual for the maximum input and output data lengths that are supported.
- The cyclic data connection sizes are limited by the gateway only in that they must lie entirely within the database boundaries.
- No explicit module selection is required on the interface card: the module will be selected automatically according to the controller's configuration.
- The Profinet IO driver does not trigger timeout events (section 8.3).
- Any database value can be accessed via Profinet acyclic services. The record number/index is equivalent to the database address and the length is specified in terms of bytes to access.

11.7.1 Customizing the GSDML

The GSDML xml file can be customized to fit the exact needs of the user. All that's required is a text editor (such as WordPad) and the latest ETH-1000 GSDML file, available from http://www.iccdesigns.com. The following example will demonstrate how to modify the number of input and output bytes for an existing module using a text editor and the Step 7 Hardware Configuration.

11.7.1.1 GSDML Example

This section will demonstrate how to modify an existing module to consume 123 (input) bytes and produce 45 (output) bytes.

1) Download the latest ETH-1000 GSDML file from http://www.iccdesigns.com.
2) Open the GSDML file with a text editor such as WordPad.
3) Choose any module to edit. In this example, we'll modify module VM_1.
4) Modify the ModuleIdentNumber from "0x00020002" to "0x01230045". Notice that the first half of the ModuleIdentNumber represents the number of input

bytes (0123) and the second half represents the number of output bytes (0045).

5) Locate the last reference of IN2_OUT2 and change the 'Value' to "IN: 0123, OUT: 0045".

6) Save the GSDML file.

7) Open the (Step 7) Hardware Configuration.

8) Navigate to *Options->Install GSD File…* and click Browse.

9) Select the directory containing the customized GSDML file.

10) Navigate to *Options->Update Catalog*.

11) Verify that the new module ("IN: 0123, OUT: 0045") is now listed in the Virtual IO. If not, it may be necessary to close and reopen the Hardware Configuration and repeat step 10.

12) Your new module is now ready to be inserted into Slot 1.

11.8 Metasys N2 Slave

The gateway supports the Johnson Controls Metasys N2 slave driver on its RS-485 port, and supports N2 analog input, analog output, binary input and binary output object types.

11.8.1 Overview

Some notes of interest are:

- Fully configurable N2 objects.
- The Metasys device type for the gateway is VND.
- Network characteristics are fixed at 9600 baud, 8 data bits, 1 start bit, 1 stop bit and no parity according to the Metasys N2 specification.
- Connect the N2 bus wiring to the gateway's RS-485 by using twisted-pair cable connected as shown in Figure 90. Connect the N2+ wire to terminal "A", the N2- wire to terminal "B", and the network ground wire to terminal "GND". Also install jumper wires connecting terminal "A" to terminal "Y", and terminal "B" to terminal "Z". Continue this connection scheme throughout the remainder of the network. Always connect each unit in a daisy-chain fashion, without drop lines, star configurations, etc. For further N2 network wiring requirements and procedures, please refer to the appropriate JCI network installation documentation.

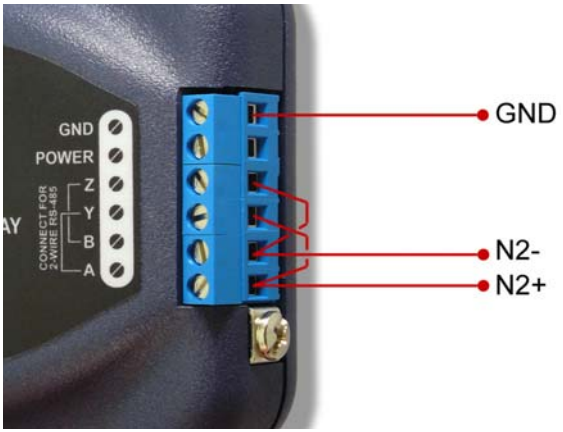


Figure 90: N2 Bus Cable Connection

11.8.2 Metasys Objects

- **Analog input** (AI) objects are used for monitoring analog status items. AI objects support low alarm limits, low warning limits, high warning limits, high alarm limits and differential values. Change of state (COS), alarm and

warning functions can also be enabled. An AI object will accept an override command, but will not change its actual value or indicate override active. A "multiplier value" is associated with the object, and is multiplied to the point's value to produce the floating-point AI value sent to the NCU (AI value = [database value] x multiplier).

- **Analog output** (AO) objects are used for setting and monitoring analog control and configuration items. An AO value can be modified by issuing an override command. Issuing a release command will not cause the AO to automatically return to its pre-override value, nor will the AO automatically return to its pre-override value after a certain time period of no communication. A "multiplier value" is associated with the object, and the floating-point AO value is divided by this multiplier to produce the result that is then stored in the gateway's database (database value = [AO value] / multiplier).

- **Binary input** (BI) objects are used for monitoring discrete (digital) status items. BI objects support COS, alarm enabling and normal/alarm status indications. A BI object will accept an override command, but will not change its actual value or indicate override active. A "bitmask" is associated with the object, and is used to determine the current state of the BI by inspecting the database data at the bit location(s) indicated in the bit mask. If all of the bit locations of the database data value indicated by a "1" in the bit mask are set, then the BI's current state is set to "1". Else, it is set to "0".

- **Binary output** (BO) points are used for setting and monitoring discrete control and configuration items. A BO value can be modified by issuing an override command. Issuing a release command will not cause the BO to automatically return to its pre-override value, nor will the BO return to its pre-override value after a certain time period of no communication. A "bitmask" is associated with the object, and is used to determine the current state of the BO by modifying the database at the bit location(s) indicated in the bit mask. When the BO's current state is set to "1" by the NCU, then the bit(s) of the database data value indicated by a "1" in the bit mask are set. Similarly, when the BO's current state is set to "0" by the NCU, then the bit(s) of the database data value indicated by a "1" in the bit mask are cleared.

11.9 Toshiba ASD Master

11.9.1 Overview

The gateway supports the Toshiba ASD Master protocol on its RS-485 port. This protocol allows direct connection to Toshiba adjustable-speed drives with RS-485 ports that support the Toshiba protocol, such as the G7/Q7/H7 and AS1/FS1/G9/H9/Q9 families. Some notes of interest are:

- Supported function codes are indicated in Table 10.

Table 10: Supported Toshiba ASD Master Functions

Function Code	Function
R	RAM read for 4-wire RS-485 networks
G	RAM read for 2-wire RS-485 networks
W	RAM & EEPROM write
P	RAM-only write

- Requests are fully configurable through service objects.
- Up to 125 parameters can be requested per service object.
- Note that Toshiba 7-series drives (G7/Q7/H7 etc.) configured for 2-wire mode (F821=0) shipped prior to early 2006 may exhibit an issue that can cause their RS-485 ports to stop communicating after a certain amount of time. Please contact Toshiba technical support to confirm your configuration prior to using 2-wire RS-485 mode on these drives.
- If a 2-wire RS-485 drive network is desired, then the drive(s) must be properly configured for 2-wire RS-485. Note that this may involve hardware configuration in addition to parameter changes. For example, G7/Q7/H7-series drives have duplex selection jumpers located on the drive's control board near the communication ports. For these drives, both jumpers must be placed in the "HALF" position for successful 2-wire operation. Refer to Figure 91 for an example detailed view of correctly-positioned duplex selection jumpers.

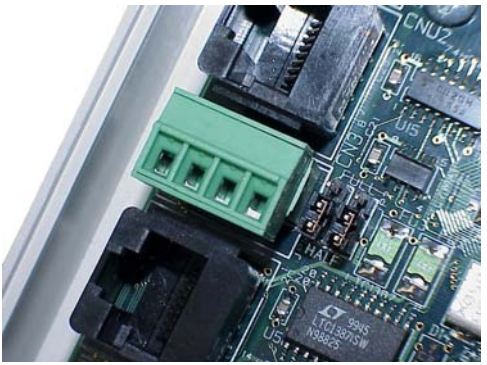


Figure 91: RS-485 Terminal Block (CN3) and Duplex Selection Jumpers

- The Toshiba RS-485 terminal block connections for G7/Q7/H7/W7 drives are shown in Figure 92 for reference only. Because there are several possible

RS-485 port configurations & options available for the various Toshiba drives, please refer to the relevant Toshiba documentation for your drive.

- When using the “W” function code to write drive configuration parameters, be sure to follow Toshiba’s guidelines regarding the number of times a specific parameter can be written without risk of EEPROM damage.

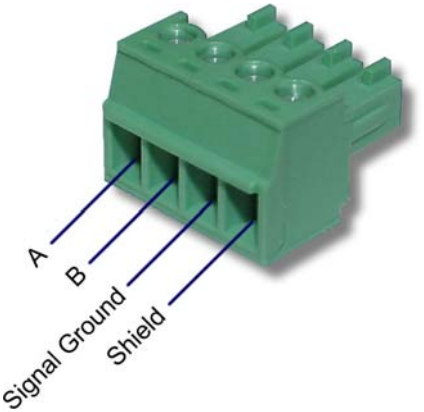


Figure 92: G7/Q7/H7/W7 RS-485 Terminal Block (CN3) Connections

11.9.2 Toshiba Service Objects

The Toshiba ASD master driver uses service objects to describe what services the gateway should perform. For each service object, the gateway will continually read the parameters defined within the service object from the designated drive, storing the value(s) in the database (if the read function is enabled). When data in the database changes where the parameters are mapped, a write request is generated to the designated drive notifying it of the changed parameter value(s) (if the write function is enabled). For more information on configuring Toshiba ASD service objects, refer to section 8.7.16.2.

11.9.3 Parameter Mapping

Drive parameters are mapped in the database as 2-byte values. This means that each parameter in a service object takes up two database addresses. For example, if a service object’s starting parameter is “FE00”, the number of parameters is “5”, and the database address is “100”, then parameters FE00 through FE04 will be mapped at database addresses 100 through 109 (parameter FE00 mapped at addresses 100 and 101, parameter FE01 mapped at addresses 102 and 103, and so on).

11.10 Sullair Supervisor Master

- The gateway acts as a Sullair Supervisor Protocol network monitor device (master) via its RS-485 port. It can automatically adapt to the Supervisor network configuration (sequencing or non-sequencing/slave mode).
- Any numerically-addressed parameter defined by the Supervisor protocol is directly accessible (machine type = parameter #1, etc.). However, some Supervisor data objects are not natively numerically-addressed. For these data objects, the additional parameter numbers indicated in Table 11 have been assigned.

Table 11: Additional Supervisor Parameter Assignments

Parameter Number	Item	Note	Source
100	Capacity		Net / Quick Status
101	P2		
102	Sequence Hours		
103	Run Status	0 = E-stop 1 = Manual stop 2 = Remote stop 3 = Standby 4 = Starting 5 = Unloaded 6 = Loaded 7 = Trim 8 = Full load 9 = Remote disable	
104	Mode	0 = Auto 1 = Continuous	
105	Fault Status	0 = No Fault 1 = Faulted	
106	Sequencing Status	0 = Not Sequencing 1 = Sequencing	
107	P1		Info status
108	P2		
109	P3		
110	P4		
111	T1		
112	T2		
113	T3		
114	T4		
115	T5		
116	ID		
117	Analog Shutdown		
118	Relay Outputs		
119	Digital Shutdown		

Parameter Number	Item	Note	Source
120	Digital Inputs		
121	Run Time		
122	Load Time		
123	Display 1	1st Line of Display	
124	Display 2	2nd Line of Display	

- The baud rate is fixed at 9600 baud.
- The gateway Supervisor interface is primarily a system monitor and configuration device. As such, the following native Supervisor network commands are not accessible:

  S – Stop
  L – Load (modulate)
  T – Trim (modulate)
  D – Display message
  C – Cont run mode
  U – Unload
  F – Full load
  E – Emergency stop
  A – Auto run mode

- Requests are fully configurable through service objects.
- Up to 125 parameters can be requested per service object.

11.10.1 Sullair Service Objects

The Sullair Supervisor master driver uses service objects to describe what services the gateway should perform. For each service object, the gateway will continually read the parameters defined within the service object from the designated controller, storing the value(s) in the database (if the read function is enabled). When data in the database changes where the parameters are mapped, a write request is generated to the designated controller notifying it of the changed parameter value(s) (if the write function is enabled). For more information on configuring Sullair Supervisor service objects, refer to section 8.7.17.2.

11.10.2 Parameter Mapping

All but the two display parameters (indexes 123 & 124) are mapped in the database as 2-byte values. This means that each parameter in a service object takes up two database addresses. For example, if a service object's starting parameter is "10", the number of parameters is "5", and the database address is "100", then parameters 10 through 14 will be mapped at database addresses 100 through 109 (parameter 10 mapped at addresses 100 and 101, parameter 11 mapped at addresses 102 and 103, and so on). Each display parameter is mapped into the database as a 20-byte ASCII character array. In other words, each display parameter takes up 20 database addresses.

Note that because display parameters differ in size from all other parameters, a single service object cannot contain both types of parameters.

12. Troubleshooting

Although by no means exhaustive, the following table provides possible causes behind some of the most common errors experienced when using the gateway.

Problem	Symptom	Solution
The gateway will not turn on.	All LEDs are off and the gateway shows no activity.	• Confirm that power is connected to the correct inputs on the RS-485 terminal block. • If using Power-over-Ethernet (PoE), check that all cables are full inserted and that the Power Sourcing Equipment (PSE) is powered on. • If firmware was being updated, it may have been corrupted. Unplug and reconnect the USB cable and run the configuration utility. Follow the utility instructions to restore the firmware.
No communications between the RS-485 network and the gateway.	The gateway's RS-485 TX and RX LEDs are blinking slowly, sporadically, or not at all	• Check connections and orientation of wiring between the network and the gateway. • Confirm that the protocol, baud rate, parity, and address settings on the RS-485 port match your network configuration.
No communication between the Ethernet network and the gateway.	Communications cannot be established, or the Ethernet "activity" LED flashes only infrequently or not at all.	• Confirm that the gateway is powered (refer to section 5.1). • Confirm that the destination IP address programmed into the controller equipment or computer matches that of the gateway, as displayed by the Finder utility. • Confirm that intermediate firewalls or routers have been configured to allow access to the interface via the applicable TCP/UDP ports. • If attempting to access the web server on a computer whose web browser is configured to use a proxy server, ensure that the proxy server is accessible to the computer, and that the gateway is accessible to the proxy server.

Problem	Symptom	Solution
Firmware-generated error	The module status LED is flashing red. The number of times the LED flashes indicates an error code.	• 4 flashes indicate there is no more space left in Object Memory. Delete some configuration objects from the configuration utility. • Any other number of flashes indicates an internal device error. Please contact ICC for further assistance.
The device will not connect to the PC with the USB cable.	The USB cable is plugged into both the PC and the device and the module status LED is not flashing green. The configuration utility may indicate a "Device Communication Error".	• Unplug and reconnect the USB cable. • Reinstall the ICC Gateway Configuration Utility. • Reinstall the ICC USB device drivers

13. Appendix A: Database Endianness

A key feature of the Millennium Series gateways is the ability to change the byte order storage scheme for data in the database between big endian and little endian. The database endianness is the convention used to store multi-byte data to or retrieve multi-byte data from the database. The selected endianness affects the end-to-end consistency of multi-byte data between the two networks on the gateway.

To better understand how this byte-ordering scheme works, the following explains how the gateway stores and retrieves multi-byte data to and from the database. Data is stored into the database starting at the low address and filled to higher addresses. The endianness determines whether the most-significant or least-significant bytes are stored first.

Let's look at some examples that demonstrate this.

This example shows how the hex value 12345678 is stored into the database using a big endian byte order. Since the hex value 12 is the most significant byte, it is stored at address "a", the lowest address.

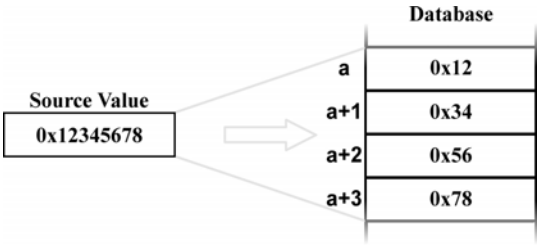


Figure 93: Big Endian Storage

This other example shows how the hex value 12345678 is stored into the database using a little endian byte order. Since the hex value 78 is the least significant byte, it is stored at the lowest address.

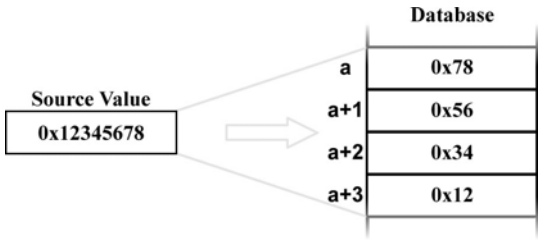


Figure 94: Little Endian Storage

Similarly, data is retrieved from the database starting at the low address. The endianness decides whether the first byte is the least-significant byte or the most-significant byte of the multi-byte number.

Here are some examples that demonstrate this.

This example shows how the hex value 12345678 is retrieved from the database using a big endian byte order. Since the hex value 12 is at address “a”, the lowest address, it is the most significant byte.

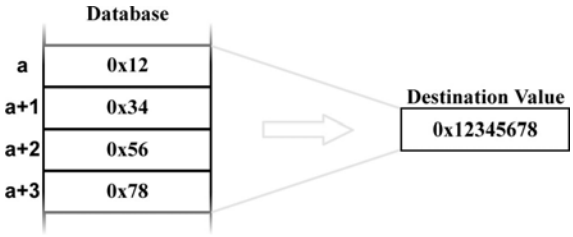


Figure 95: Big Endian Retrieval

This other example shows how the hex value 12345678 is retrieved from the database using a little endian byte order. Since the hex value 78 is at the lowest address, it is the least significant byte.

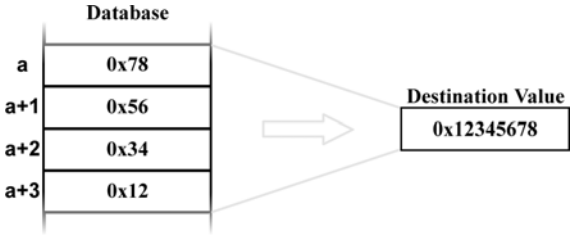


Figure 96: Little Endian Retrieval

The above examples illustrate the data movement to and from the gateway’s internal database. This idea helps explain the data movement, as a whole, from one port to the other on the gateway between two different networks. Because networks vary in the manner that they exchange data, endianness selection must be part of the gateway’s configuration in order to ensure coherent multi-byte data exchange. There are two data exchange methods used by the supported networks of the gateway.

The first method is used in those networks that define a byte order for how to interpret multi-byte data within an array of bytes. Profibus, for example, defines a big-endian order for multi-byte data, while DeviceNet defines a little-endian order for multi-byte data. These networks exchange I/O data by means of a “bag of bytes” approach, whereas the gateway need not concern itself with where individual values are delimited within the array of bytes itself (as this is determined by the sending or receiving nodes on the networks). The bytes are simply stored into the database in the order they were received. Gateway endianness selection therefore has no effect on data storage or retrieval with a “bag of bytes” protocol driver.

The other method is that used by networks that exchange data by means of an “object value” system, whereas data is exchanged by addressing a certain object to read or write data. Modbus for example, uses registers, while BACnet uses objects such as analog values to exchange data. When multi-byte values are received by the gateway, the bytes must be stored into the database in the order defined by the endianness selected. Likewise, when retrieving multi-byte values from the database for the gateway to transmit, the endianness selected will determine how the data is reconstructed when read from the database.

The selection of the correct byte ordering is crucial for coherent interaction between these two types of networks on the gateway. The following presents examples of how the database endianness affects end-to-end communication between networks and when each byte-ordering scheme should be used.

13.1 Ex: Modbus - Profibus

This example shows the interaction between a network using an object value method (Modbus) and one using a bag of bytes method (Profibus) to exchange data. The gateway reads holding registers 1 and 2 from the Modbus network, stores the data into the database, and then sends the 4 bytes of input data onto the Profibus network. Figure 97 shows this data movement for the gateway's database configured as big endian. Because the Profibus specification defines multi-byte values within the byte array to be interpreted as big endian, it is recommended that the database be configured for big-endian byte order when using Profibus. In the example, holding register 1 has a value of 0x1234 and holding register 2 has a value of 0x5678. When the Profibus device receiving the input data from the gateway recombines the two pairs of 2-byte values, the resulting data is 0x1234 and 0x5678, thus successfully receiving the correct values for holding registers 1 and 2.

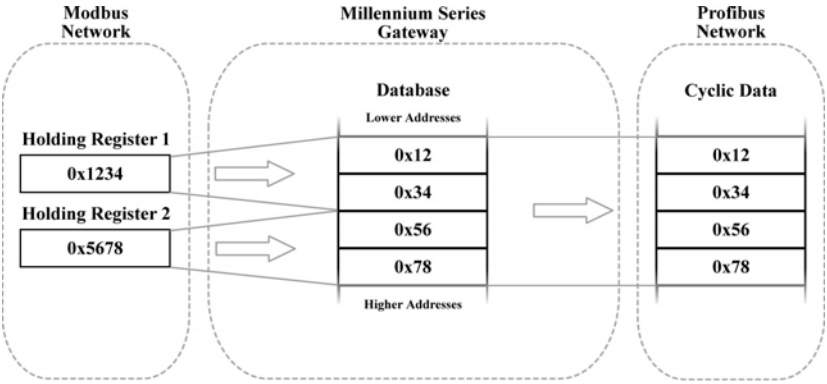


Figure 97: Modbus - Profibus Big Endian

In contrast, Figure 98 shows the effects of configuring the database for little-endian byte order. Holding registers 1 and 2 again have values of 0x1234 and 0x5678, respectively. However, when the Profibus device receiving the input data from the gateway interprets these values, the resulting pairs of 2-byte values become 0x3412 and 0x7856, thus receiving incorrect values for holding registers 1 and 2. Note that in both examples, the Profibus network data is always identical, byte-for-byte, to the gateway's database. For this reason it is important to configure gateways that use a bag-of-bytes style network, such as the PBDP-1000, to use the same endianness as defined for that network.

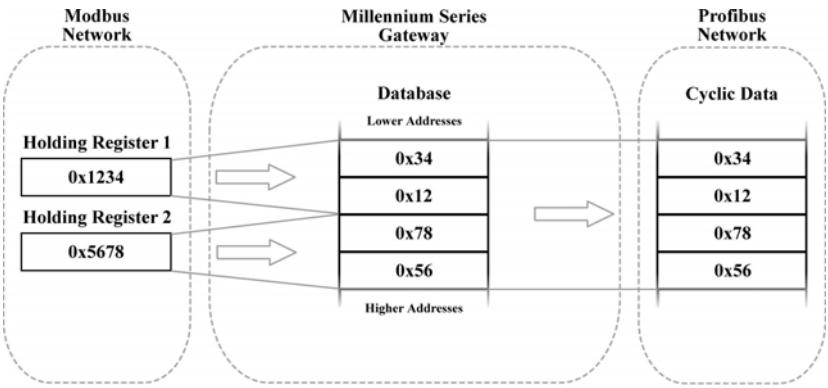


Figure 98: Modbus - Profibus Little Endian

13.2 Ex: Modbus - DeviceNet

This example shows the interaction between a network using an object value method (Modbus) and one using a bag of bytes method (DeviceNet) to exchange data. The gateway reads holding registers 1 and 2 from the Modbus network, stores the data into the database, and then sends the 4 bytes of input data onto the DeviceNet network. Figure 97 shows this data movement for the gateway's database configured as little endian. Because the DeviceNet specification defines multi-byte values within the byte array to be interpreted as little endian, it is recommended that the database be configured for little-endian byte order when using DeviceNet. In the example, holding register 1 has a value of 0x1234 and holding register 2 has a value of 0x5678. When the DeviceNet device receiving the input data from the gateway recombines the two pairs of 2-byte values, the resulting data is 0x1234 and 0x5678, thus successfully receiving the correct values for holding registers 1 and 2.

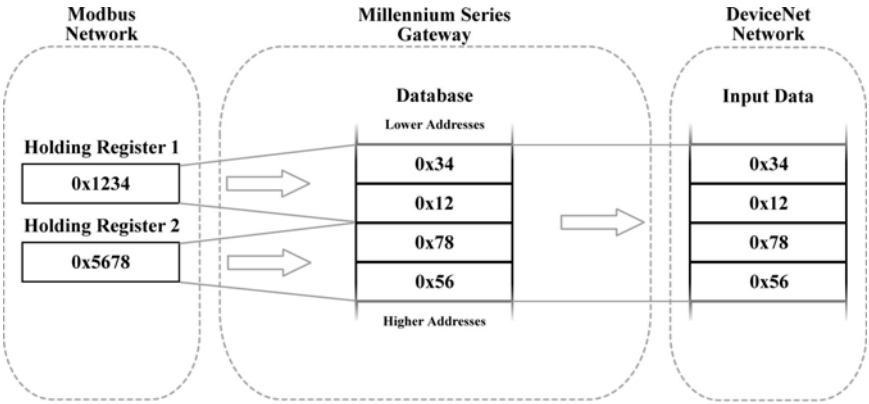


Figure 99: Modbus - DeviceNet Little Endian

In contrast, Figure 98 shows the effects of configuring the database for big-endian byte order. Holding registers 1 and 2 again have values of 0x1234 and 0x5678, respectively. However, when the DeviceNet device receiving the input data from the gateway interprets these values, the resulting pairs of 2-byte values become 0x3412 and 0x7856, thus receiving incorrect values for holding registers 1 and 2. Note that in both examples, the DeviceNet network data is always identical, byte-for-byte, to the gateway's database. For this reason it is important to configure gateways that use a bag-of-bytes style network, such as the DNET-1000, to use the same endianness as defined for that network.

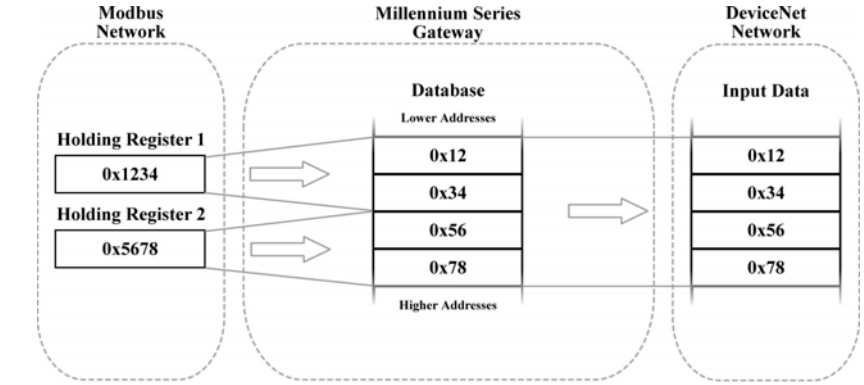


Figure 100: Modbus - DeviceNet Big Endian

13.3 Ex: BACnet - DeviceNet

This example is quite similar to the previous one as data is exchanged between an object-value style network (BACnet) and a bag-of-bytes style network (DeviceNet). The key difference is that in this example, BACnet Analog Value 0 is a 32-bit value, as apposed to two 16-bit Modbus registers. Here, the gateway reads analog value 0 from the BACnet network, stores the data into the database, and sends the input data onto the DeviceNet network. Figure 101 demonstrates the data flow from the BACnet network to the DeviceNet network through a gateway configured to use a little endian database. Because the DeviceNet specification defines multi-byte values within the byte array to be interpreted as little endian, it is recommended that the database be configured for little-endian byte order when using DeviceNet. In the example, analog value 0 has a value of 0x12345678. When the DeviceNet device receiving the input data from the gateway interprets the 4 bytes, the resulting 4-byte value will be 0x12345678, thus successfully receiving the original value of the BACnet analog value object.

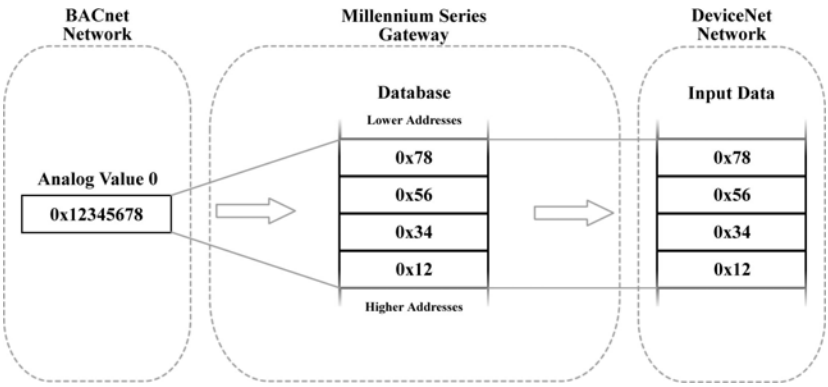


Figure 101: BACnet - DeviceNet Little Endian

Conversely, Figure 102 illustrates the consequences of configuring the database for big-endian byte order using this scenario. Once again, Analog Value 0 has a value of 0x12345678. But now, when the DeviceNet device interprets the 4 bytes of input data sent by the gateway, the resulting 4-byte value is 0x78563412, thus receiving an incorrect value for Analog Value 0. Note that in this example as well, the DeviceNet byte array is identical, byte-for-byte to the database. This example, in conjunction with the previous, demonstrates the dependence on the bag-of-bytes style networks for correct database endianness selection.

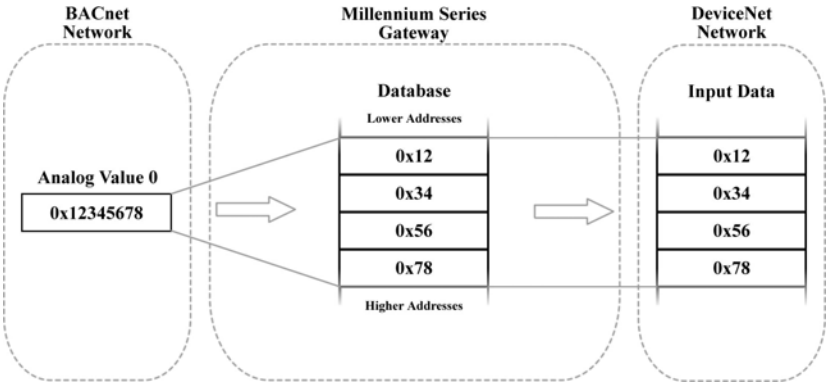


Figure 102: BACnet - DeviceNet Big Endian

13.4 Ex: BACnet - Modbus (Analog Objects-Registers)

This example exhibits two networks that both use an object value scheme to exchange data. In this scenario, the database endianness is irrelevant if the data types are the same for both networks. This example shows communication between a BACnet network and a Modbus network using two 16-bit analog value BACnet objects and two 16-bit Modbus holding registers. As shown in Figure 103, the values from the BACnet network are stored into the database with big-endian byte ordering. Figure 104 shows the values from the BACnet network being stored into the database with little-endian byte ordering. Regardless of the byte-ordering scheme used, the two holding registers on the Modbus network receive the same values. Notice that in both cases, analog values 1 and 2 have values of 0x1234 and 0x5678, respectively, while holding registers 1 and 2 also have values of 0x1234 and 0x5678, respectively. The only difference between the two cases is how the data is being stored internally on the gateway itself.

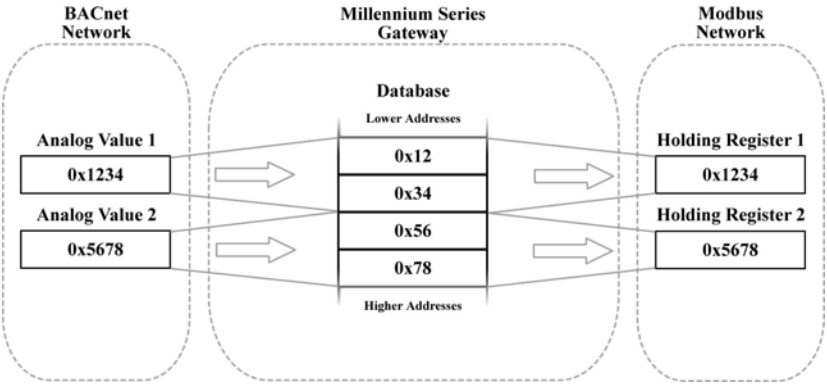


Figure 103: BACnet - Modbus (Analog Objects & Registers) Big Endian

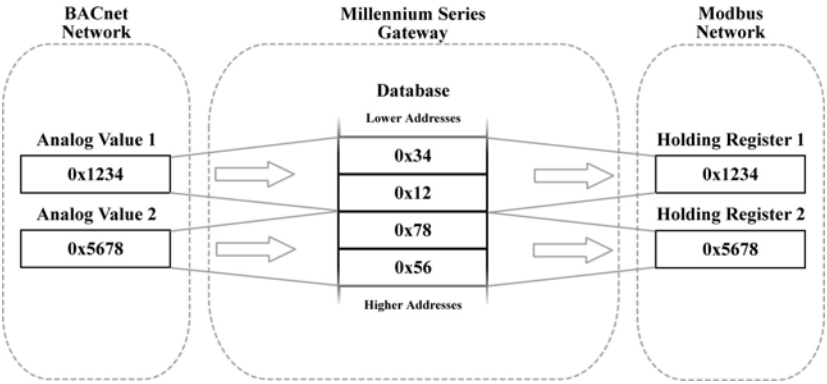


Figure 104: BACnet - Modbus (Analog Objects & Registers) Little Endian

13.5 Ex: BACnet - Modbus (Binary Objects-Discretes)

This example also contains two networks that both employ an object value method for exchanging data, but unlike the previous example, the database endianness affects the end-to-end alignment of the data. In this example, communication is taking place between a BACnet network and a Modbus network using single-bit data elements. The BACnet side is using binary values 1 through 32, while the Modbus side is using coil status 1 through 32. The byte ordering of the database is significant because of the manner in which Modbus coils are mapped in the gateway. Coils (and input statuses) are mapped to registers, not addresses (see Section 11.5.2.3 for more information). Since registers are 16-bit entities, the byte order of the registers (and by association, the coils), is affected by the endianness configured for the database. BACnet binary objects, however, are mapped on a byte-wise basis into the database.

When the database is configured for a little-endian byte order, binary value 1 – 8 corresponds to coil 1 – 8, binary value 9 – 16 corresponds to coil 9 – 16, and so on. This can be seen in Figure 105. Notice that the least significant bytes of the registers that the coils map to are placed in the lower memory addresses in the database. Because Modbus discretes are mapped to registers in a bit-wise little-endian fashion, it is recommended that the database be little endian in this scenario so that bit-wise data will align between networks.

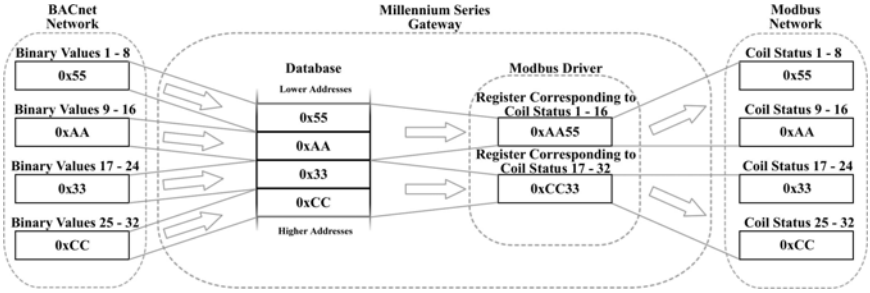


Figure 105: BACnet - Modbus (Binary Objects & Discretes) Little Endian

However, when the database is configured for a big-endian byte order, binary values 1 – 8 correspond to coils 9 – 16, binary values 9 – 16 correspond to coils 1 – 8, and so on. This can be seen in Figure 106. Since the most significant bytes of the Modbus registers that the coils map to are now mapped to lower addresses, the alignment between the two networks' bit-wise data is byte swapped. While this alignment can still be used, it is much more intuitive when the database is configured to be little endian.

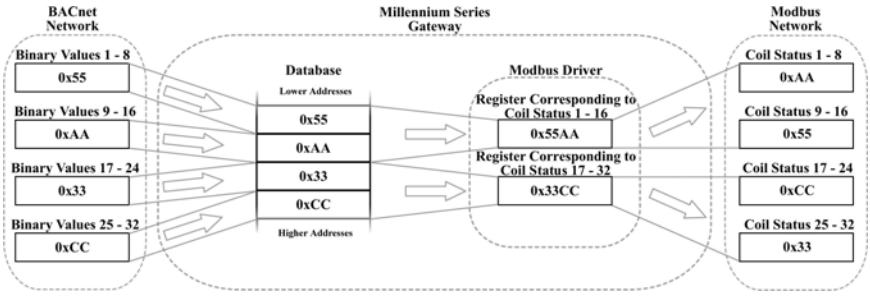


Figure 106: BACnet - Modbus (Binary Objects & Discretes) Big Endian

14. Appendix B: Status Information

This section details the information that is enabled by checking the **Reflect Status** checkbox while configuring a service object. Figure 107 diagrams the structure of this status information. Because this 16-byte structure resides in the database at a user-designated location, it can be accessed from the opposite port in order to continuously determine the performance of the corresponding service object.

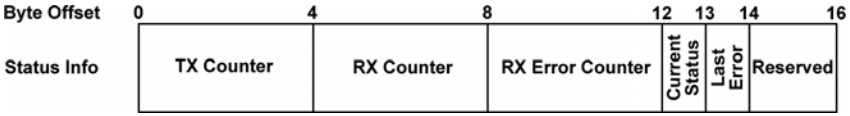


Figure 107: Service Object Status Format

TX Counter

This is a 32-bit counter that increments when a packet is transmitted from the gateway.

RX Counter

This is a 32-bit counter that increments when the gateway successfully receives a valid packet.

RX Error Counter

This is a 32-bit counter that increments when the gateway receives an error response packet or when an error occurs upon reception of a packet.

Current Status

This byte indicates the status of the most recently received packet. The status is updated each time the RX Counter or RX Error Counter increments. Refer to Table 12 for a list of currently used codes.

Last Error

This byte indicates the last reception error that occurred. The last error is updated each time the RX Error Counter increments. Refer to Table 12 for a list of currently used codes.

Reserved

These two bytes are currently unused but are reserved for future use.

Table 12: Status / Error Codes

Status / Error Code (Hex)	Description
0x00	No Error
0xF0	Invalid Data Address
0xF1	Data Error
0xF2	Write To Read-Only
0xF3	Read From Write-Only
0xF4	Target Busy
0xF5	Target Error
0xF6	Cannot Execute
0xF7	Mode Error
0xF8	Other Error
0xF9	Memory Error
0xFA	Receive Error
0xFB	Invalid Function
0xFC	Invalid Packet
0xFD	Security Error
0xFE	Checksum Error
0xFF	Timeout Error

ICC

INDUSTRIAL CONTROL COMMUNICATIONS, INC.

1600 Aspen Commons, Suite 210
Middleton, WI USA 53562-4720
Tel: [608] 831-1255 Fax: [608] 831-2045

http://www.iccdesigns.com

Printed in U.S.A