**Grid Connect, Inc.**
**EtherNet/IP to Modbus Products**

# Getting Started Guide

## Introduction

This Getting Started Guide applies to the following part numbers:

- GC-XPORT-EIP-MB – XPort RJ45 Connector with EtherNet/IP – Modbus firmware.
- GC-NET232-EIP-MB – Grid Connect NET232 with above XPort inside
- GC-NET485-EIP-MB – Grid Connect NET485 with above XPort inside

The EtherNet/IP to Modbus products are packaged with a CD containing the following documents:

- EtherNet-IP_Modbus_Getting_Started.pdf – This document
- XPort-EIP-MB_UG.pdf – Software reference for all the above products
- XPort_IntGuide.pdf – Hardware reference only in Section 2
- NET232_UM_800232_c.pdf – Hardware reference only in Sections 2.5, 2.7, 2.8, 2.9
- NET485_UM_800240_c.pdf – Hardware reference only in Sections 1.2, 1.3, 2.2, 2.3, 2.4

## Assigning the IP Address

By default, the EIP-MB products obtain their IP address using BOOTP, therefore you need a BOOTP server on your PC. If you are using Rockwell Software, then they include a BOOTP server as a part of the RSLinx installation. Otherwise, the BOOTP server software by itself can be downloaded here:

http://www.software.rockwell.com/support/download/detail.cfm?ID=3390

## Configuring the EIP-MB Unit

The EIP-MB products are configured using EtherNet/IP Explicit Messaging which is available through any standard EtherNet/IP configuration software tool. Examples of this software are:

- Pyramid Solutions EIP Scan
- Rockwell Software RSNetworx for EtherNet/IP
- Omron Network Configurator for EtherNet/IP

To test connectivity between your configuration tool and the EIP-MB unit, you can retrieve the Vendor ID. This is done by sending an Explicit Message using the following parameters:

- Service Code: 0x0E (Get_Attribute_Single)
- Class: 0x01 (Identity Object)
- Instance: 0x01 (First instance of the Identity Object)
- Attribute: 0x01 (Vendor ID)

The EIP-MB unit should respond with 0x03AC (940 dec). If your tool displays the response in a byte-by-byte format, you may see "AC 03" which is the Little Endian representation of the 16-bit value 0x03AC.

For more details about EtherNet/IP Explicit Messaging, please refer to the ODVA EtherNet/IP Specifications.

## *Configuring the Serial Port*

Once the unit has an IP address, the Modbus Master inside must be configured to communicate with your Modbus slave. First the serial port settings must be configured. This is done by setting some of the attributes within the Bridge Configuration Object. Please refer to the EIP-MB firmware manual for details.

By default, the EIP-MB products use Modbus RTU, Slave Address 1, Baud Rate 19.2 kbps, 8 Data Bits, 1 Stop Bit, Even Parity.

For example, suppose we want to set the Baud Rate to 115.2 kbps. Referring to the baud rate table in the EIP-MB manual, we need to set the Modbus Baud Rate attribute to a value of 13 (decimal). Send the following Explicit Message:

- Service Code: 0x10 (Set_Attribute_Single)
- Class: 0x64 (Bridge Configuration Object)
- Instance: 0x01 (First instance of the Bridge Configuration Object)
- Attribute: 0x68 (Modbus Baud Rate Attribute)
- Data: 0x0D (The value 13 in hex)

As another example, suppose we want to change the parity to "None". This is done by setting the Modbus Parity Attribute to a value of 0:

- Service Code: 0x10 (Set_Attribute_Single)
- Class: 0x64 (Bridge Configuration Object)
- Instance: 0x01 (First instance of the Bridge Configuration Object)
- Attribute: 0x6B (Modbus Parity Attribute)
- Data: 0x00

Once you have set the serial port parameters to match those of your Modbus slave, cycle power to the EIP-MB unit.

## *Configuring the Modbus Mappings*

Next the Modbus Master must be configured to read and write the Modbus registers in your slave. This is done by setting mappings in the Input and Output Mapping Objects.

### Setting a Modbus Input Mapping

In this first example, we will set an input mapping to read 3 registers from Modbus address 500.

Note: The Modbus specification defines register *addresses* and register *numbers*. A register address is the actual number that will be sent on the serial data line. A register number is offset by +1. If you want to access a Modbus register at address 100, then it is referred to as Modbus register number 101. The EIP-MB units use register *numbers* as their input. Unfortunately, many Modbus vendors use these terms interchangeably. Therefore, you may have to experiment to discover whether your Modbus slave's documentation specifies Modbus register numbers or addresses.

A Modbus Mapping consists of 3 values: Modbus Function Code (8-bit), Modbus Register Number (16-bit), and Quantity (16-bit). Note that EtherNet/IP is a Little Endian protocol, therefore the 16-bit values must be byte-swapped. To set our example mapping, the following EtherNet/IP Explicit Message must be sent:

- Service Code: 0x10 (Set_Attribute_Single)
- Class: 0x65 (Input Mapping Object)
- Instance: 0x01 (First instance of the Input Mapping Object)
- Attribute: 0x65 (First Mapping)
- Data: 0x03 0xF5 0x01 0x03 0x00

The explanation of the Data field is as follows:

- 0x03: Modbus Function Code to Read Multiple Registers. Please refer to the Modbus Specification for more information.
- 0xF5 0x01: Little Endian representation of 0x01F5 which is the 16-bit hex representation of 501, which is our starting register *number* (the address is 500).
- 0x03 0x00: Little Endian representation of 0x0003 which is the 16-bit hex representation of our desired quantity of registers, 3.

Upon receiving this message, the EIP-MB unit will attempt to send the corresponding Modbus message to your slave. The mapping will only be set if the Modbus slave responds with a success. If you receive an EtherNet/IP success response, then the mapping has been set. If you receive an error, then one of the following is most likely your problem:

- The Modbus slave is not connected, or the wiring is incorrect.
- The Modbus serial settings (Slave ID, Baud Rate, etc.) are incorrect.
- You are trying to access an invalid register range: Perhaps you didn't swap the bytes as explained above? Perhaps your Modbus slave's documentation provides you with register "numbers" but calls them "addresses"?

Once you have successfully set your mappings, cycle power to the EIP-MB unit for them to take effect.

At this point in our example, the internal Modbus Master will be reading 3 Modbus registers, therefore 3 words (6 bytes) of data have been added to the EtherNet/IP Class 1 T->O (Input) Data packet. The Input Data packet always begins with 1 word (2 bytes) of Modbus Status, so our total input data size is now 4 words (8 bytes).

## Setting a Modbus Output Mapping

This section assumes you have read and understand the previous section about setting a Modbus Input Mapping.

For this example, we will set two mappings: one to write 5 registers at address 100, and a second to write 3 registers at address 500. (Note: these are the same 3 registers we are reading in our input mapping!)

For the first Output Mapping, the Explicit Message would be as follows:

- Service Code: 0x10 (Set_Attribute_Single)
- Class: 0x66 (Output Mapping Object)
- Instance: 0x01
- Attribute: 0x65 (First Mapping)
- Data: 0x10 0x65 0x00 0x05 0x00

For the second Output Mapping, the Explicit Message would be:

- Service Code: 0x10 (Set_Attribute_Single)
- Class: 0x66 (Output Mapping Object)
- Instance: 0x01
- Attribute: 0x66 (Second Mapping)
- Data: 0x10 0xF5 0x01 0x03 0x00

Once you have successfully set your mappings, cycle power to the EIP-MB unit for them to take effect.

At this point in our example, the internal Modbus Master will be writing 8 Modbus registers, therefore 8 words (16 bytes) of data have been added to the EtherNet/IP Class 1 O->T (Output)

Data packet. The Output Data packet always begins with 1 word (2 bytes) for a Run/Idle command, so our total output data size is now 9 words (18 bytes).

## Locking the Configuration

The EIP-MB units essentially operate in two modes – one where the Configuration is "Unlocked" and one where it is "Locked".

When it is unlocked, changes to the configuration are allowed, such as setting the serial port configuration or the input/output mappings. During this time, the internal Modbus Master is not scanning, i.e. sending any Modbus messages according to the mappings. When it is locked, configuration changes are not allowed and the Modbus Master begins scanning.

The configuration can be locked in one of two ways – toggling the current lock state or changing the power-up default of the lock state.

If you are still experimenting with the unit and planning to make some changes, then it would be best to just toggle the current lock state. This way, you can lock it and test it with your PLC, then by simply cycling power it will boot-up in the unlocked state again. The lock state is toggled using this "secret" message:

- Service Code: 0x45
- Class: 0x67
- Instance: 0x89
- Attribute: 0xAB
- Data: 0xCD

If you are satisfied with your configuration and would like it to be "permanent", then it would be best to change the power-up default of the lock state. You definitely want to do this once the unit is being deployed, so that when power is cycled it begins running, but also so that people cannot randomly change the configuration. Change the power-up default of the lock with this Explicit Message:

- Service Code: 0x10 (Set_Attribute_Single)
- Class: 0x64 (Bridge Configuration Object)
- Instance: 0x01
- Attribute: 0x71 (Configuration Lock Default on Power Up Attribute)
- Data: 0x01

Now the next time you cycle power, the configuration will be locked. To unlock the configuration again, simply send the "secret" message above. Then you will be free to change any of the configuration again. Note that it will re-lock after the next power cycle unless you set the power-up default back to 0!

## Establishing the Class 1 I/O Connection

Once you have set some mappings, cycled power, and locked the configuration, you can attempt to exchange I/O with your EtherNet/IP PLC. The configuration of a class 1 connection in every PLC's software is different, but there should be a place to enter the following information:

- Data Type: "INT" or "16-Bit"
- Input T->O Assembly Instance: 0x65 or 101 decimal
- Input T->O Size: 4* words (16-bit)
- Output O->T Assembly Instance: 0x66 or 102 decimal
- Output O->T Size: 9* words (16-bit)
- Configuration Assembly Instance: 0x80 or 128 decimal
- Configuration Size: 0

*Note that the above sizes follow our previous example. You should change these according to how you set your mappings.

Some EtherNet/IP Master simulators such as EIP Scan do not have a data type setting; everything is in bytes. In that case, the input and output sizes would be 8 and 18 respectively.

If the PLC is reporting a communication failure, one of the following is most likely the issue:

- The data sizes for the input and/or output are not correct.
- The GUI uses decimal values and you entered hex, or vice versa.

## Exchanging I/O Data

Once you have an I/O connection established, and if everything else was done correctly, you should immediately see input data in your PLC that matches the data in your Modbus slave's registers.

If the first word of your input data is non-zero, this indicates an error in the Modbus communication. You will see this if your Modbus slave is disconnected, powered-off, etc.

If the first word of your input data is zero, then the subsequent words would contain the Modbus register data according to your input mappings. Continuing our example, the 4 words of our input data are:

- Word 0: Modbus Status (should be 0 if everything is OK)
- Word 1: Data from Modbus Register Number 501
- Word 2: Data from Modbus Register Number 502
- Word 3: Data from Modbus Register Number 503

If the data is all zeros and you know the values in those registers is non-zero, then most likely your configuration is unlocked. The Modbus Master does not scan the mappings when it is unlocked.

Regarding the Output Data, the Modbus Master will not write any output mappings until the Run Bit in the first word of the output data is set. Change the first word to a 1 and it will begin writing registers. From our example, the 9 words of our output data are:

- Word 0: Modbus Run/Idle (set to 1 for registers to be written)
- Word 1: Data to Modbus Register Number 101
- Word 2: Data to Modbus Register Number 102
- Word 3: Data to Modbus Register Number 103
- Word 4: Data to Modbus Register Number 104
- Word 5: Data to Modbus Register Number 105
- Word 6: Data to Modbus Register Number 501
- Word 7: Data to Modbus Register Number 502
- Word 8: Data to Modbus Register Number 503

In our example, if we were to change the data in words 6, 7, or 8, we would see the input data words 1, 2, or 3 also change, because the first input mapping and second output mapping are reading/writing the same registers.

If you are not seeing your Modbus Slave's register data change according to the changes you make in the PLC data, then one of the following is most likely the issue:

- Your configuration is unlocked
- You have not set the Run/Idle bit to 1
- You have to change your PLC into Run mode (some PLCs will not send EIP output data if they are in Program mode).