Appendix B: IMPACC and the Modbus Protocol

Overview

This appendix describes some programming aspects of using the NetPower DeviceServer with a Modbus protocol-based system, including the following topics:

- How to send commands to IMPACC devices.
- Modbus protocol functions supported by the NetPower DeviceServer.
- Interpreting 32-bit quantities from the NetPower DeviceServer.

Sending Commands to IMPACC Devices

You can send commands to IMPACC devices from either the Modbus Master or the NetPower DeviceServer. The Modbus Master writes (using Modbus Functions 6 and 16) the command to a Holding Register that is associated with the target IMPACC device. You can configure the NetPower DeviceServer to write the command to the Holding Register with Modbus Function 6 using one of the following methods:

- The NetPower DeviceServer transmits the command to the appropriate device, and returns the device response to the Modbus Master. If the transmission succeeded (no error occurred), the response is normal, and the NetPower DeviceServer stores a zero in the holding register. If the transmission failed, the response is an exception response code (1, 2, 4, or 6), and the NetPower DeviceServer stores the resulting return code in the high byte of the Holding Register. For more information on Modbus Response Codes Supported by IMPACC, see *Chapter 12: Troubleshoot*ing in the *PowerNet Software User's Guide*.
- The NetPower DeviceServer writes the device command to the Holding Register and responds to the Modbus Master with a normal response. The NetPower DeviceServer will send the command to the appropriate device at a later time. While the command is waiting to be sent, it remains in the Holding Register.

After the NetPower DeviceServer transmits the command successfully (without error), it stores a zero in the Holding Register. If the transmission failed, the NetPower DeviceServer stores the resulting error code in the high byte of the Holding Register.

With either method, the Modbus Master can read the error code by performing Modbus Function 3 (Read Holding Register).

A Holding Register may not be rewritten until the previous command is completed (exception response 6). The number of consecutive Holding Register writes (without an intervening holding or Input Register read) is limited to 10.

The NetPower DeviceServer can write device commands to Holding Registers via Modbus Function 16 using method 2 only.

A list of the available device commands for each IMPACC device type is listed in *Appendix B: Device Objects* in the *PowerNet Software User's Guide.*

Supported Modbus Protocol Functions

The following Modbus protocol functions are supported by the NetPower DeviceServer. Examples are provided to help you understand how to use the Modbus protocol to define commands and interpret data.

The functions described in this appendix are used to:

- Read Holding Registers (Function Code 3)
- Read Input Registers (Function Code 4)
- Preset a single register (Function Code 6)
- Perform a Loopback Test (Function Code 8)
- Preset multiple Holding Registers (Function Code 16)

The following sections describe each function. For more information, refer to the *Modicon Modbus Protocol Reference Guide.*

Function Code 3—Read Holding Registers

This function permits you to obtain the contents of Holding Registers at the designated slave address

Sending a Request

Holding Registers are numbered sequentially from zero through 199 (0 = register 40001, 1 = register 40002, . . . ,199 = register 40,200). The Modbus Master uses these registers to send commands to IMPACC devices. Each request can obtain the contents of up to 125 registers.

Note
The Modbus protocol uses hex format; therefore, all the values shown in the table are in hex.

For example, the following table shows the values you would use to read Holding Registers 40,108 through 40,110 from slave address 17:

		Data					
	Function	Start Register		# of Registers		Error Check	
Address	Code	High	Low	High	Low	Low	High
11	03	00	6B	00	03	76	87

Slave address. (Hex 11 equals decimal 17).

Starting high-byte register ("00" if the register contains only one byte).

Starting low-byte register. Hex 6B (decimal 107) represents holding register 40,108.

Number of high-byte registers to read, beginning from the starting high-byte register.

Number of low-byte registers to read, beginning from the starting low-byte register.

Communication transmission error check sum.

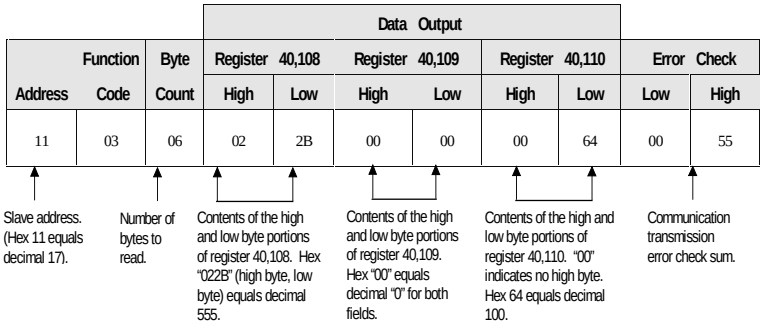
The *Starting Low-Byte Register* column value is 6B (107 decimal), which represents Holding Register 40,108. This register is the starting point for counting the registers that have been read.

The *# of Registers Low* column indicates that three low-byte registers will be read; thus, Holding Registers 40108, 40109, and 40110 are read.

Interpreting the Response

The addressed slave responds with its address and the function code, followed by the information field. The information field contains two bytes describing the quantity of data bytes to be returned. The contents of the requested registers are two bytes each, with the binary content right-justified within each pair of characters. The first byte includes the high order bits. The second byte contains the low order bits.

In the following example, registers 40,108 through 40,110 have the decimal contents 555, 0, and 100, respectively:



Function Code 4—Read Input Registers

This function permits you to obtain the contents of Input Registers

Sending a Request

Input registers are numbered sequentially from zero through 7999 as follows:

0 = register 30001, 1 = register 30002, . . . ,7999 = register 38,000

These registers receive their values from IMPACC devices and can only be referenced, but not altered, from within the controller or via the Modbus. The contents of up to 125 registers can be obtained at each request.

Note
The Modbus protocol uses hex format; therefore, all the values shown in the table are in hex.

For example, the following table shows the values you would use to read Input Register 30,009 from slave address 17:

		Data					
	Function	Start Register		# of Registers		Error Check	
Address	Code	High	Low	High	Low	Low	High
11	04	00	08	00	01	B2	98

Slave address. (Hex 11 equals decimal 17).	Starting high-byte register ("00" if the register contains only one byte).	Starting low-byte register. Hex 08 (decimal 8) represents input register 30,009.	Number of high-byte registers to read, beginning from the starting high-byte register.	Number of low-byte registers to read, beginning from the starting low-byte register.	Communication transmission error check sum.

The *Starting Low-Byte Register* value is 08 hex (8 decimal), which represents register 30,009. This register is the starting point for counting which registers are read.

The *# of Registers Low* column indicates that one low-byte register will be read; thus, Input Register 30,009 is read.

Interpreting the Response

The addressed slave responds with its address and the function code, followed by the information field. The information field contains two bytes describing the quantity of data bytes to be returned. The contents of the requested registers are two bytes each, with the binary content right-justified within each pair of characters. The first byte includes the high order bits. The second byte contains the low order bits.

In the following example, register 30,008 contains the decimal value 0:

			Data Input			
	Function	Byte	Register 30,009		Error Check	
Address	Code	Count	High	Low	Low	High
11	04	02	00	00	78	F3

Slave address. Hex 11 equals decimal 17).

Number of bytes to read.

Contents of the high and low byte portions of register 30,009. Hex "0000" (high byte, low byte) equals decimal 0.

Communication transmission error check sum.

Function Code 6—Preset Single Register

This function permits you to modify the contents of a Holding Register

Important
This function overrides the controller memory protection.

Sending a Command

Holding Registers are numbered sequentially from zero through 199 (0 = register 40001, 1 = register 40002, . . . ,199 = register 40,200).

The values are provided in binary up to the maximum capacity of the controller. Unused high bits must be set to zero.

When used with slave address 0 (Broadcast mode), all slave controllers load the specified register with the designated contents.

For example, the following table shows the values you would use to read Holding Registers 40,108 through 40,110 from slave address 17:

		Data					
	Function	Register #		Data Value		Error Check	
Address	Code	High	Low	High	Low	Low	High
11	06	00	87	03	9E	BA	2B

Slave address. (Hex 11 equals decimal 17).

Starting high-byte register ("00" if the register contains only one byte).

Starting low-byte register. Hex 87 (decimal 135) represents holding register 40,136.

Value to which the register will be set ("039E" hex equals 926 decimal).

Communication transmission error check sum.

The *Starting Low-Byte Register* value is 87 (135 decimal), which represents Holding Register 40,136. This register will be preset to hex value 039E (decimal 926).

Interpreting the Response

The normal response to a request to preset a single register is to re-transmit the query message after the register has been altered. Based on the above example, the response would be as follows:

		Data					
Address	Function Code	Register High	40,136 Low	Data	Data	Error Check	
11	06	00	87	03	9E	C1	CRC

Function Code 8—Loopback Test

The Loopback Test permits you to test the communications system. It does not affect the content of the NetPower DeviceServer. Variations in the response might indicate a fault in Modbus communications.

Sending a Command (Query)

The information field contains two bytes used to designate a diagnostic code followed by two bytes used to designate the action to be taken.

Using the Loopback Test also permits you to determine the contents of the Diagnostic Register and to read the slave's bus message, bus CRC error, and bus exception error counters.

For a detailed description of how the Loopback test can be used and a list of the diagnostic codes, refer to the *Modicon Modbus Protocol Reference Guide*.

In the following table, the Loopback Test requests a simple return of the query message (diagnostic code 0) sent to slave address 17:

	Function	Diagnostic	Code	Data	Data	Error	Check
Address	Code	High	Low			Low	High
11	08	00	00	A5	37	00	0B

Slave address. (Hex 11 equals decimal 17).
Starting high-byte.
Starting low-byte.
Data to be sent. Hex A5 equals decimal 165.
Data to be sent. Hex 37 equals decimal 55.
Communication transmission error check sum.

Interpreting the Response

Using the previous example, a successful Loopback Test would return the following results (the same information that was sent).

	Function	Diagnostic	Code			Error	Check
Address	Code	High	Low	Data	Data	Low	High
11	08	00	00	A5	37	00	0B

Slave address.
Starting high-byte.
Starting low-byte.
Data received.
Data received.
Check sum.

Function Code 16—Preset Multiple Holding Registers

This function permits you to modify the contents of multiple Holding Registers .

Important
60 Registers maximum.

This function overrides the controller memory protection.

Writing Data to Holding Registers

Holding Registers associated with devices are numbered sequentially from zero through 199 (0 = register 40001, 1 = register 40002, ..., 199 = register 40,200).

Unmapped Holding Registers are numbered sequentially from 1000 through 1149 (1000 = register 41,001, 1001 = register 41,002, ...,1149 = register 41150).

The values are provided in binary up to the maximum capacity of the controller. Unused high bits must be set to zero.

When used with slave address 0 (Broadcast mode), all slave controllers load the specified registers with the designated contents.

For example, the following table shows the values you would use to write Holding Registers 41,001 to 41,002 (inclusive) at slave address 17:

		Data									
	Function	Register#		Quantity		Byte	DataValue		DataValue		
Address	Code	High	Low	High	Low	Count	High	Low	High	Low	CRC
11	10	03	E8	00	02	04	01	02	03	04	

Slave address. (Hex 11 equals decimal 17).

Starting holding register. Hex 3E8 (decimal 1000) represents holding register 41,001.

Quantity of holding registers to be written.

Number of data bytes to follow.

Values to which the registers will be set (102 hex equals 258 decimal, 304 hex equals 772 decimal).

Checksum.

The *Starting Low-Byte Register* value is 3E8 (1000 decimal), which represents Holding Register 41,001; the quantity of registers to write is 2. Holding Registers 41,001 and 41,002 will be preset to hex value 102 (decimal 258) and 304 (decimal 772) respectively.

Interpreting the Response

The normal response to a preset multiple register command is to retransmit the address function code, starting register, and quantity of registers to be loaded. Based on the above example, the response would be as follows:

		Data					
	Function	Register #		Quantity		Byte	
Address	Code	High	Low	High	Low	Count	CRC
11	10	03	E8	00	02	04	

Interpreting 32-bit Quantities from the NetPower DeviceServer

The NetPower DeviceServer displays certain fields in two 16-bit registers (Most Significant, Least Significant). For example, the IQ Analyzer IA field (Ph a current x 100). This is an unsigned 32-bit number within the IMPACC system and should be interpreted as such.

Unsigned 32-bit Values

Using the IQ Analyzer IA field example, assume the value is 821.26 amps and the IA is sent via Input Registers 30017 and 30018. Register

30017 contains the Least Significant 16 bits and 30018 contains the Most Significant 16 bits.

Register Address	Contents (hex)	Contents (dec)
30017	40CE16	1659010
30018	000116	110

These two registers must be interpreted as the unsigned 32-bit number:

000140CE16 = 8212610.

This may mean that the Modbus Master will have to manipulate the contents of two 16-bit registers as follows:

((Register 30018*6553610)+ Register 30017)
=((1*6553610)+1659010)
=821610

And then applying the IA scale factor (0.01) and display IA as:

= 8212610 /10010
= 821.26

Signed 32-bit Values

Using the IQ Analyzer WATTSDEMAND (Average Real Power) field example, assume the value is -9625423 watts and the WATTSDEMAND is sent via Input Registers 30102 and 30103. Register 30102 contains the Least Significant 16 bits, 30103 contains the Most Significant 16 bits.

Register Address	Contents(hex)	Contents(dec)
30102	20B116	836910
30103	FF6D16	-14710

These two registers must be interpreted as a signed 32-bit number:

FF6D20B116 = -962542310 .

This may mean that the Modbus Master will have to display WATTSDEMAND using logic similar to the following:

```
if (Register 30103 LESS THAN 0)
{
    //Take 2's Complement of Register Pair.
    Register 30103 = NOT (Register 30103)
    Register 30102 = NOT (Register 30102)
    if (Register 30102 EQUALS 65535)
    {
        Register 30102 = 0
        Register 30103 = Register 30103 + 1
    }
    else
    {
```

Register 30102 = Register 30102 + 1

}

{(Register 30103 * 6553610) + Register 30102)

= ((14610 * 6553610) + 5716710

= 96542310

//Display as unsigned 32-bit value (above but with minus sign to denote negative

=96542310

}

else

(Register 30203* 6553610) + Register 30102)

//Display as unsigned 32-bit value (above but with plus sign to denote positive.