170 ENT 110 00
Ethernet Communication Adapter
User Guide

870 USE 112 00

Version 1.0

31000485 00

July 1998

Schneider Electric

Data, Illustrations, Alterations
Data and illustrations are not binding. We reserve the right to alter products in line with our policy of continuous product development. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us using the form on one of the last pages of this publication.

Training
Schneider Electric offers suitable further training on the system.

Hotline
See addresses for the Technical Support Centers at the end of this publication.

Trademarks
All terms used in this publication to denote Schneider Electric products are trademarks of Schneider Electric.

All other terms used in this publication to denote products may be registered trademarks and/or trademarks of the corresponding corporations.
Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation, Windows is a brandname of Microsoft Corporation in the USA and other countries.
IBM® is a registered trademark of International Business Machines Corporation.
Intel® is a registered trademark of Intel Corporation.

Copyright

Contents

TSX Momentum Ethernet Communication Adapter 170 ENT 110 00 1

1.1 Product Overview 2
1.1.1 Function 2
1.1.2 Physical Structure 2
1.1.3 Operating Voltages and Error Control 2
1.1.4 Mapping Data to I/O Base Field Terminals 3
1.1.5 Managing Throughput to I/O Bases 3
1.1.6 Specifications 3
1.2 Example: Data Turnaround Time 4
1.3 Status Indicators 6
1.4 Connecting to the Network 7
1.4.1 Network Connector 7
1.4.2 Network Labels: Global Address and IP Address 7
1.5 Placing the Adapter into Service 8
1.5.1 Initialization and Self–Tests 8
1.5.2 Assigning an Ethernet IP Address 8
1.5.3 Identifying the I/O Base 9
1.5.4 Storing the IP Address in the Adapter 9
1.6 Replacing an Adapter 10
1.6.1 Erase the Stored IP Address 10
1.6.2 Remove Operating Power and Disconnect the Adapter 10
1.6.3 Install the New Adapter 10

Communicating With the Adapter 11

2.1 Communication Access Registers 12
2.1.1 Data Registers 13
2.1.2 Configuration Registers 13
2.1.3 Status Registers 15

Test Program: Source Code 19

3.1 Source: response.java 20
3.2 Source: test1.txt 28

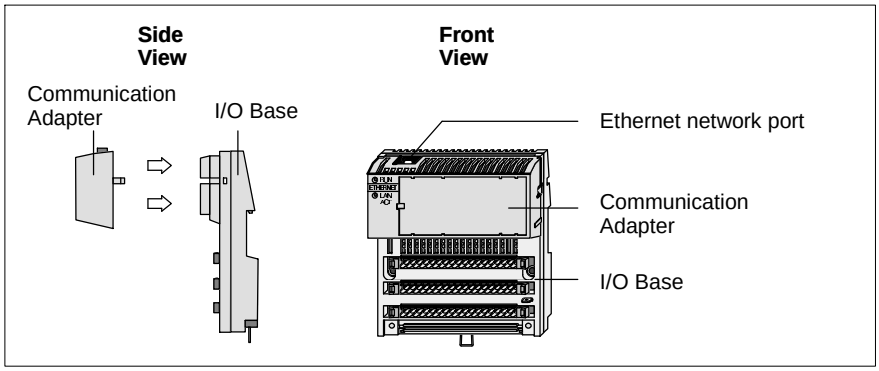
TSX Momentum
Ethernet Communication Adapter
170 ENT 110 00

1

This Communication Adapter can be connected to any TSX Momentum I/O base to create a functional I/O module. It provides direct connection to the Ethernet network, enabling an Ethernet host to communicate with field devices wired to the I/O base terminals.

Figure 1 shows the layout of a typical adapter and I/O base.

Figure 1 Communication Adapter with TSX Momentum I/O Base



This chapter describes:

- Product Overview
- Example: Data Turnaround Time
- Status Indicators
- Connecting to the Network
- Placing the Adapter into Service
- Replacing an Adapter

1.1 Product Overview

1.1.1 Function

This adapter is installed on any TSX Momentum I/O base to form a complete I/O module that communicates on an Ethernet network. A programmable controller or other host device on the network can then read from the input terminals and write to the output terminals of the I/O base.

The adapter communicates with host devices using Modbus Application Protocol with TCP/IP packets. It supports both Ethernet II and IEEE 802.3 framing.

For information about using Modbus Application Protocol with TCP/IP, refer to the *Ethernet TCP/IP Module User Guide*, part number 840 USE 107. Details of the Modbus protocol are provided in the *Modbus Protocol Reference Guide*, part number PI–MBUS–300.

For information about the application and field wiring of I/O bases, refer to the *TSX Momentum I/O Bases User Manual*, part number 870 USE 002.

1.1.2 Physical Structure

Each adapter connects to the internal communication connector of its I/O base. Clips lock the adapter in place and can be released with a common screwdriver to remove the adapter. The user can fill out the front panel wiring label (supplied with the I/O base) to identify the wiring connections at the I/O base terminals.

The adapter is considered open equipment and must be mounted in an enclosure that is approved for the site at which it is installed.

1.1.3 Operating Voltages and Error Control

Power for the adapter and I/O base is provided by the user at the field location. The adapter receives its operating voltage through its I/O base internal connection. The adapter monitors its voltage and goes offline to the network if the voltage is not within tolerance.

1.1.4 Mapping Data to I/O Base Field Terminals

Data is mapped between the application and I/O base field terminals in the IEC format. Refer to the *TSX Momentum I/O Bases User Manual*, 870 USE 002 for the mapping diagrams for the I/O bases.

1.1.5 Managing Throughput to I/O Bases

To ensure deterministic timing of I/O messages, you should design your network to include only your application host and your I/O base communication adapters. Adding other kinds of devices, such as user interfaces or programmers, can cause variables in I/O message timing when those devices access the network.

1.1.6 Specifications

Table 1 Network Specification

Description	Specification
Ethernet interface	Compliant with the STP or UTP 100 ohm connection.

Table 2 Agency Approval

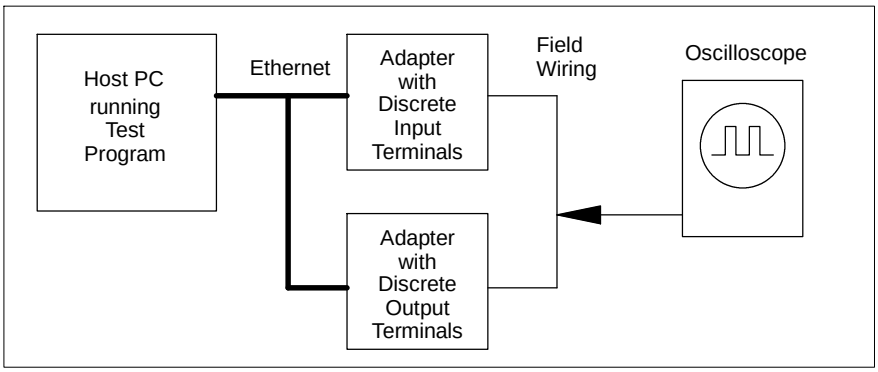
Agency	Status
UL 508	Approved
CAN/CSA C22.2NO.142	Approved
CE Mark	Approved

1.2 Example: Data Turnaround Time

Figure 2 shows an example of a control loop constructed to measure the data turnaround time at the field terminals of a pair of I/O bases.

A host PC running the test program is connected by Ethernet to two adapters with discrete I/O bases. The field output terminals of the output base are wired directly to the field input terminals of the input base. An oscilloscope is used to time the switching of the field signals.

Figure 2 Example: Data Turnaround Time

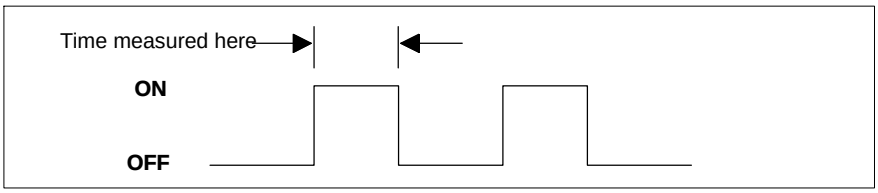


The test program is a Java loop that performs this sequence:

1. Continuously reads the input terminals of the input base module.
2. Writes an output terminal to a new (ON or OFF) condition.
3. When a changed state is received from the inputs, toggles the outputs.

The oscilloscope measures the time duration of the ON state of the outputs.

Figure 3 Data Turnaround Time Measurement



Tests were conducted on two separate NT workstations with these configurations:

- 200 MHz, 96 MB RAM
- 100 MHz, 32 MB RAM

Table 3 shows the measured data turnaround times. The results indicate that the major factor affecting data timing is the speed of the loop execution in the host.

Table 3 Results: Data Turnaround Time

Networked Devices	Network Loading	Minimum Time	Maximum Time	Average Time	Host CPU Speed and RAM
2	10%	5 ms	9 ms	6.2 ms	200 MHz 96 MB
2	40%	5 ms	9 ms	6.2 ms	200 MHz 96 MB
2	70%	6 ms	9 ms	6.3 ms	200 MHz 96 MB
64	10%	6 ms	8 ms	6.8 ms	200 MHz 96 MB
64	40%	6 ms	12 ms	8.4 ms	200 MHz 96 MB
64	70%	6 ms	13 ms	8.2 ms	200 MHz 96 MB
64	10%	25 ms	30 ms	26.7 ms	100 MHz 32 MB
64	40%	25 ms	30 ms	26.7 ms	100 MHz 32 MB
64	70%	26 ms	30 ms	27.0 ms	100 MHz 32 MB

Java source code for the test program is reproduced in Chapter 3.

1.3 Status Indicators

The adapter has two front panel indicators showing its operating status.

Figure 4 Indicators

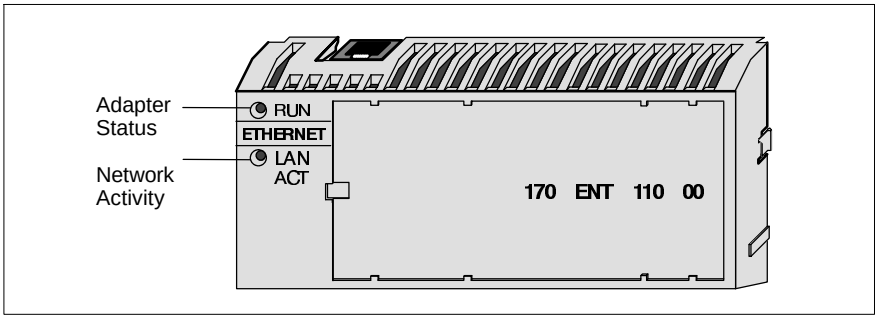


Table 4 RUN Indicator: Adapter Status

Indicator State	Status
On (steady)	Normal operation: power is present from I/O base, and the adapter is ready for network communication.
3 Flashes, long Off	No Link: The network cable is not connected or is defective.
4 Flashes, long Off	No MAC Address: The adapter's MAC address is not set. Internal hardware problem.
5 Flashes, long Off	No IP Address: The adapter is attempting to obtain an IP Address from a BOOTP server.
6 Flashes, long Off	The adapter's internal executive program has started, but cannot initialize the I/O base.
7 Flashes, long Off	The adapter has obtained an IP address, but does not have a valid executive program.
8 Flashes, long Off	The adapter's executive program has failed during execution.
Flashing constantly	Adapter is downloading its executive program.

Table 5 LAN ACT Indicator: Network Activity

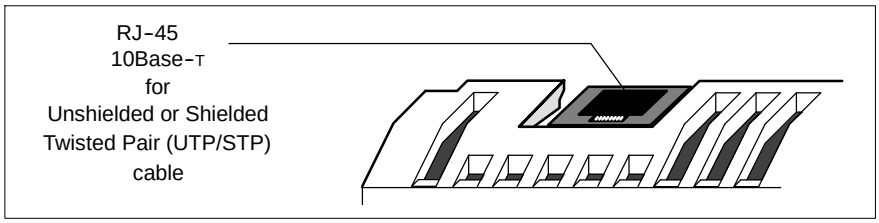
Indicator State	Status
Flashing	Normal operation: Adapter detects network activity. Flashing rate indicates the amount of activity. May appear steadily On if network activity is high.
Off	Adapter is not detecting any network activity.

1.4 Connecting to the Network

1.4.1 Network Connector

The adapter has one RJ–45 connector for a 10Base–T UTP/STP (Unshielded or Shielded Twisted Pair) cable. The adapter should be cabled directly to the Ethernet hub.

Figure 5 Network Connector

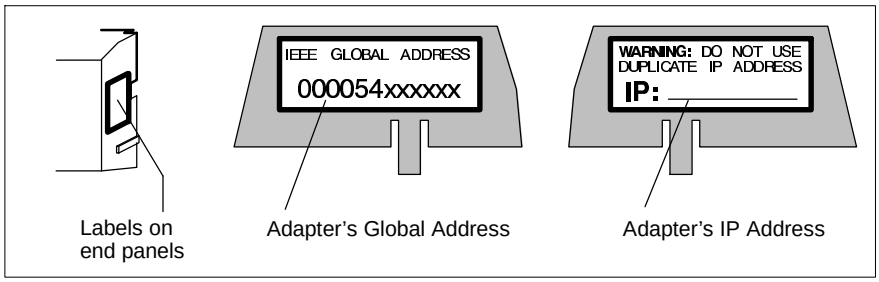


1.4.2 Network Labels: Global Address and IP Address

The adapter has two labels mounted on its end panels. One label identifies the adapter's IEEE Global Address (MAC address). The other identifies its Internet Protocol address (IP address).

The installer records the Global Address and gives it to the network administrator for use in establishing an IP address for the adapter during the BOOTP process at startup. When the IP address has been assigned, the administrator gives this address to the installer who writes it onto the adapter's IP address label.

Figure 6 Adapter Labels: Global Address and IP Address



1.5 Placing the Adapter into Service

1.5.1 Initialization and Self-Tests

When the adapter receives its initial operating power from its I/O base, it performs internal initialization and self-tests. If the tests fail, the RUN indicator flashes to indicate the failure reason, if possible, and the adapter remains offline. If the tests are successful, the adapter attempts to obtain its Ethernet IP address.

1.5.2 Assigning an Ethernet IP Address

Overview: Address Assignment
A BOOTP server is required to assign a new IP address to the adapter. After the server assigns the IP address, the server application can issue a command to the adapter to cause it to store the address internally.

If the adapter has stored its address and is re-initialized (for example, following a power loss), the adapter will again issue requests for an address from a BOOTP server. If a server responds with an address, the adapter will use it that address. If a server does not respond, the adapter will revert to its stored address.

Requesting the IP Address
After completing its initialization, the adapter requests its Ethernet IP address from a BOOTP server. The adapter uses its MAC address with the BOOTP protocol over the Ethernet network.

Receiving the Server Response
The adapter will wait ten seconds for a BOOTP server to respond with the adapter's IP address. If the server response is received, the adapter will use that address as long as power remains applied to the adapter.

Warning
DUPLICATE ADDRESS HAZARD Having two or more devices with the same IP address can cause unpredictable operation of your network. Ensure that this device will receive a unique IP address. Failure to observe this precaution can result in injury or equipment damage.

Retries to the Server
If a BOOTP server response is not received, the adapter will retry the request six times: three times using the Ethernet II framing type, and three times using the 802.3 framing type.

Server Response Not Received (IP Address Previously Stored)
If the adapter receives no response to any of its attempts to obtain an IP address, and if an address has been previously stored by a Modbus Write command from the application, the adapter will then use that stored address.

Server Response Not Received (IP Address Not Stored)
If the adapter receives no response to any of its attempts to obtain an IP address, and if it does not have any stored address, the adapter will continue to retry the BOOTP request every 30 seconds. During this time it will flash its RUN indicator in the 'requesting' pattern (a sequence of five flashes).

1.5.3 Identifying the I/O Base

After the adapter receives its IP address, it runs an internal procedure to identify its I/O base. If the procedure fails, the adapter's RUN indicator flashes a failure pattern (six flashes) and will remain offline.

If the I/O base is successfully identified, the adapter is ready to communicate using the Modbus protocol over TCP/IP.

1.5.4 Storing the IP Address in the Adapter

The adapter has a non–volatile RAM area for storing its assigned IP address. If the application requires the adapter to retain its current IP address, the application must issue a Modbus Write command to write a boolean value into a specific register in the adapter to cause the address to be stored. The adapter's default state is to not store the address.

Section 2.1 describes how to store the IP address and how to determine if an address has been previously stored.

1.6 Replacing an Adapter

1.6.1 Erase the Stored IP Address

Before removing any adapter from service, you should clear its IP address.

The adapter has a non-volatile RAM area for storing its assigned IP parameters. The parameters are retained when power is removed from the adapter, and will remain permanently in the adapter when it is removed from service. If the adapter is subsequently returned to service it would be possible for it to cause unspecified activity on your network. You should therefore erase the current parameters before removing the adapter from service.

The adapter has an internal register which defines the boolean state (saved or not saved) of its IP parameters. The register can be read by the application, and it can be written into to cause the adapter to clear the parameters.

Warning
DUPLICATE ADDRESS HAZARD Having two or more devices with the same IP address can cause unpredictable operation of your network. Before removing any adapter from service, you should first write a logical 0 (zero) into the parameter storage register to clear the adapter's stored parameters. This will rreduce the possibility of a duplicate IP address appearing on your network if the adapter is later restored to service. Failure to observe this precaution can result in injury or equipment damage. Refer to Section 2.1 for a description of the adapter's registers, including how to clear its stored parameters.

1.6.2 Remove Operating Power and Disconnect the Adapter

Before removing the adapter, remove the operating power from the I/O base. Then disconnect the Ethernet cable, and remove the adapter from the base.

1.6.3 Install the New Adapter

Mount the new adapter onto the I/O base, following the instructions supplied with the new adapter. Record the new adapter's IEEE Global Address (MAC address), and use it to configure an Internet Protocol address (IP address) for the adapter.

Section 1.5 describes how to place the new adapter into service using the Ethernet BOOTP protocol.

Communicating With the Adapter

2

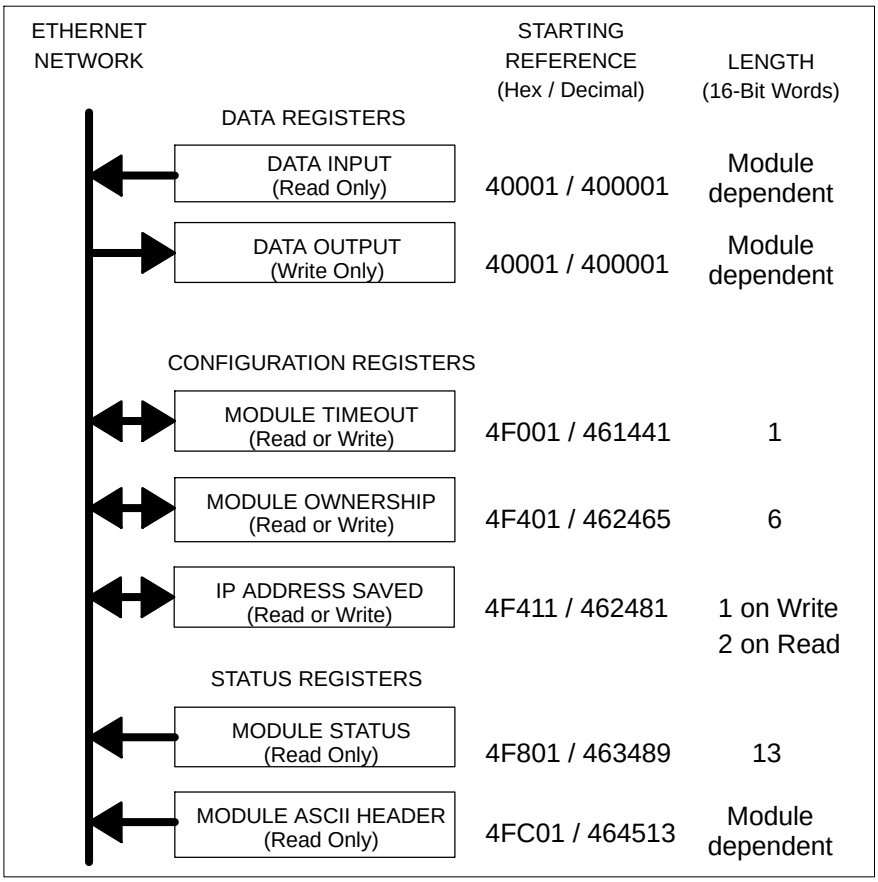
- Communication Access Registers
- Data Registers
- Configuration Registers
- Status Registers

2.1 Communication Access Registers

Each adapter contains three groups of registers that enable the application to communicate with the I/O base module. The application accesses the registers to transfer input or output data at the I/O base module's field terminals, to set or retrieve the module's configuration, or to monitor its status.

All of the registers can be accessed as 4XXXX references by MSTR function blocks in the application program.

Figure 7 Communication Adapter Access Registers



2.1.1 Data Registers

40001 hex -- Data Input or Output
Starting reference 40001 is used to address input data from field inputs and output data to field outputs. Data is transferred in the IEC format. Mapping between the controller's data registers and I/O base field terminals is unique to each base, and is described in the *TSX Momentum I/O Bases User Manual,* 870 USE 002 00.

2.1.2 Configuration Registers

4F001 hex -- Outputs Holdup Timeout Value
Reference 4F001 specifies the amount of time that outputs will be held in their current state, if they are not updated by a new Modbus Write command. If the module's holdup time expires before a new write command is received, all outputs are set to logical 0 (zero).

The field length is one word. The timeout value is expressed in units of 10 milliseconds, with a minimum register value of 30 (300 milliseconds) and maximum value of 6000 (60 seconds). The default value is 100 (1 second).

The register's contents can be read using a Modbus Read command.

4F401 hex -- Ownership of Write Privilege
When the adapter first receives power, it will give sole write privilege to the first node that writes to it using the Modbus Write command. The adapter maintains an internal 60-second timer for handling the write privilege, and will reserve sole privilege to that node as long at the node continues to write within 60-second intervals to the adapter.

Starting reference 4F401 specifies the IP addresses of up to three more nodes which may concurrently own write privilege to the adapter. A node which currently owns the write privilege may write up to three IP addresses (2 words per address) to the adapter starting at 4F401. With those addresses stored, any of those three nodes may then write to the adapter in addition to the original privileged node. This allows up to four nodes to concurrently own write privilege to the adapter.

If writes continue to occur within the 60-second interval from any of the three privileged nodes, no other node may write to the adapter. If the timer is allowed to expire, any node may write to the adapter.

Note that this 60-second Write Privilege timer is separate from the Outputs Holdup timer, and applies only to the write privilege. The 60-second time is a fixed value and is not accessible to the application.

Any node may read the input data or status information from the adapter.

4F411 hex -- IP Address Saved
This reference serves a dual purpose, depending on whether the application issues a Modbus Write command or a Modbus Read command.

Modbus Write Command: Save or Clear IP Address For a Modbus Write command the reference is treated as a one-word register, with the application writing one word of data. The Modbus Write data may consist of a 1 or 0 (zero), which causes the adapter to save or clear its current IP address.

If a data 1 is written to the reference, the adapter will save its currently assigned IP address in its non-volatile RAM. If a new initialization occurs and the adapter cannot find a BOOTP server, the adapter will use this saved address.

If a data 0 is written to the reference, the current IP address will be erased.

Warning
DUPLICATE ADDRESS HAZARD Having two or more devices with the same IP address can cause unpredictable operation of your network. Before removing any adapter from service, you must first write a logical 0 (zero) into register 4F411 to clear the adapter's stored address. This will reduce the possibility of a duplicate IP address appearing on your network if the adapter is later restored to service. Failure to observe this precaution can result in injury or equipment damage.

Warning
THE ADAPTER INITIALIZES WHEN THESE CONTENTS CHANGE Any change of state of this reference's contents will cause the adapter to re-initialize.

Modbus Read Command: Get Current IP Address For a Modbus Read command the reference is treated as a two-word register, with the application reading two words of data. If the adapter has IP parameters saved in its non-volatile RAM, it will return its current IP address to the Modbus Read command, indicating that it has stored parameters. If IP parameters are not currently saved, the adapter returns all ones (FFFFFFFF hex) to the Read.

2.1.3 Status Registers

4F801 hex -- Module Status Block
These registers provide information about the module's revision level and current operating parameters.

The block's length is 13 words. The registers can be read by the application, but cannot be written into.

Table 6 Module Status Block Layout

Reference (hex)	Purpose	Contents
4F801	Length of status block (words)	13 decimal
4F802	I/O module quantity of input words	Module dependent
4F803	I/O module quantity of output words	Module dependent
4F804	I/O module ID number	Module dependent
4F805	Communication Adapter revision number	Format: XR where: X = upper 4 bits, always 0000 R = lower 12 bits, defining the revision as 3 hex characters. Example: 100 hex = Rev. 1..00 200 hex = Rev. 2.00
4F806	ASCII header block length (words)	Module dependent
4F807	Last IP address to communicate with this adapter in most recent Modbus transaction (low word of 2 words – see 4F80D)	Node address dependent
4F808	Remaining ownership reservation time	milliseconds
4F809	Remaining outputs holdup time	milliseconds
4F80A	I/O module health	8000 hex = healthy 0000 hex = not healthy
4F80B	I/O module last error value	Module dependent
4F80C	I/O module error counter	Error count 0000 ... FFFF hex
4F80D	Last IP address to communicate with this adapter in most recent Modbus transaction (high word of 2 words – see 4F807)	Node address dependent

4FC01 hex -- Module ASCII Header Block

These registers contain an ASCII text description of the module. The registers can be read by the application, but cannot be written into.

The block length depends upon the type of I/O base to which the adapter is connected. The maximum length is 64 bytes of ASCII characters, corresponding to a length of 8 ... 32 words as specified in word 6 of the Module Status Block (at reference 4F806).

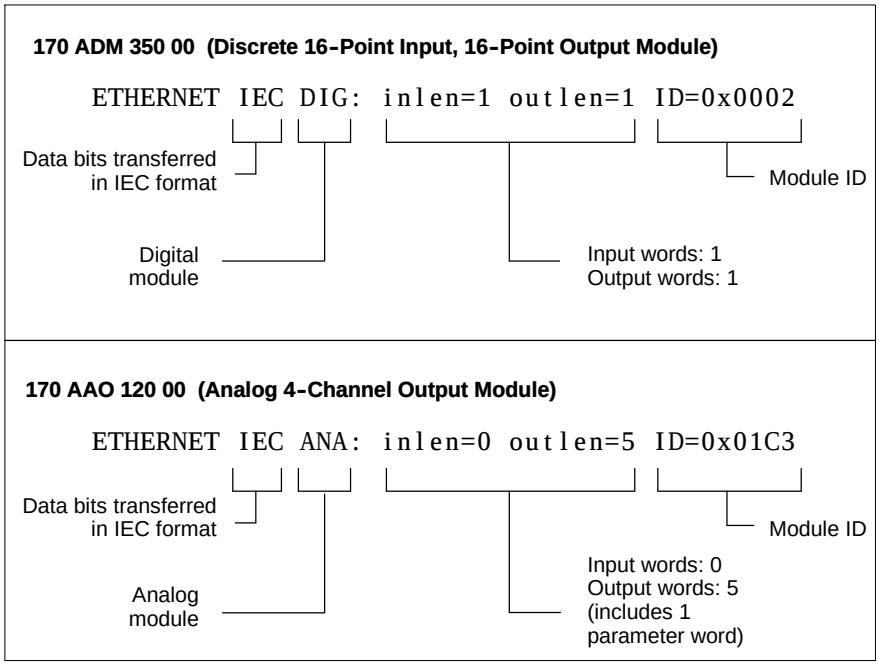
The block contains labels to identify the quantities of input and output words, and the ID code of the I/O base. You can parse the block contents to extract this information into your application.

Table 7 ASCII Header Block Layout

ASCII Characters	Meaning
ETHERNET	Identifies Ethernet Communication Adapter
20 hex (32 decimal)	space
IEC	Data is transferred with I/O base in IEC format
20 hex (32 decimal)	space
DIG: EXP: ANA:	Digital module (ID range: XX00 ... XX7F hex) Expert module (ID range: XX80 ... XXBF hex) Analog module (ID range: XXC0 ... XXFE hex)
20 hex (32 decimal)	space
inlen=*n*	Input words (*n* = quantity of words, decimal)
20 hex (32 decimal)	space
outlen=*n*	Output words (*n* = quantity of words, decimal)
20 hex (32 decimal)	space
ID=0x*nnnn*	Module ID code (*nnnn* = ID code, hex)

Figure 8 shows examples of the ASCII Header Block contents for two I/O bases.

Figure 8 Examples: ASCII Header Block



Test Program: Source Code

3

- Source: response.java
- Source: test1.txt

3.1 Source: response.java

This Java program furnishes the looping for data throughput testing as described in Chapter 1.

It reads the text file `test1.txt` to get the target addresses and Modbus commands for communicating with the input and output communication adapters.

The file's contents follow:

```
// response.java, 6/17/98. This is a variation of mbtest.java. This code
// will send a (write) command to one address and a separate command (read)
// to the second address. It will loop performing the read each time, but
// only doing the write when the read command detects a change in the input
// that was read. The loop counter is set as a constant. An improvement
// would be to pass it in the command line. The purpose of this program
// is to use an oscilloscope to measure the time elapsed between the input
// going high, and the output responding to it. The IP addresses of the IO
// modules, and the write and read commands are read from a text file.
//-------------------------------------------------------------------------

// mbtest.java 4/26/96

// minimal application to communicate and do performance measurement
// to the Quantum Ethernet module using ASA registered TCP port and
// MODBUS_ENCODING PDU format

// variant 5/14/96 to allow comparison of responses to the same request from 2 targets
// variant 5/24/96 to allow specification of independent addresses on the 2 targets
//                 and to allow time delay in script
//         11/11/97 generate 'usage:' and handle blank lines as comment

import java.io.* ;
import java.net.* ;

class mbtest {
   public static void main(String argv[]) {
      if (argv.length<1) {
         System.out.println("usage: java mbtest scriptfile > reportfile\n"+
                            "eg.  java mbtest script.txt > result.txt");
      } else
      try {
```

```
int dolog = 1;
int keybuf;
long skipcnt;
int ioCount;
byte expect;
int first;
DataInputStream di = new DataInputStream(new FileInputStream(argv[0]));
String cmd;

// analyse 'target' lines. Note must be followed by 'address'
// note this version of the program requires that the 'target'
// lines be the first 1 or 2 lines of the script

String target;
Socket es=null;
OutputStream os=null;
FilterInputStream is=null;
cmd = di.readLine().trim();
if (cmd.startsWith("target")) {
   target = cmd.substring(6).trim();
   System.out.println("mbtest: connecting to "+target);
   es = new Socket(target,502);
   os= es.getOutputStream();
   is = new BufferedInputStream(es.getInputStream());
   cmd = di.readLine().trim();
}
String target2;
Socket es2=null;
OutputStream os2=null;
FilterInputStream is2=null;
if (cmd.startsWith("target")) {
   target2 = cmd.substring(6).trim();
   System.out.println("mbtest: connecting to "+target2);
   es2 = new Socket(target2,502);
   os2= es2.getOutputStream();
   is2 = new BufferedInputStream(es2.getInputStream());
   cmd = di.readLine().trim();
}
int address = 1;
int address2 = 1;
byte obuf[] = new byte[261];
byte obuf2[] = new byte[261];
byte ibuf[] = new byte[261];
```

```
            byte ibuf2[] = new byte[261];
            obuf[0] = 0;
            obuf[1] = 0;
            obuf[2] = 0;
            obuf[3] = 0;
            obuf[4] = 0;
            obuf2[0] = 0;
            obuf2[1] = 0;
            obuf2[2] = 0;
            obuf2[3] = 0;
            obuf2[4] = 0;
            for (;;) {
               if (cmd.startsWith(";")) {
                  System.out.println(cmd);
                  cmd = di.readLine().trim();
                  continue;
               }
               if (cmd.startsWith("address2")) {
                  address2 = Integer.parseInt(cmd.substring(9));
                  cmd = di.readLine().trim();
                  continue;
               }
               if (cmd.startsWith("address")) {
                  address = Integer.parseInt(cmd.substring(8));
                  address2 = address;
                  cmd = di.readLine().trim();
                  continue;
               }
               if (cmd.startsWith("quit")) break;
               // handle script delay
               if (cmd.startsWith("wait")) {
                  int delayTime = Integer.parseInt(cmd.substring(5));
                  if (delayTime < 0 || delayTime > 30000) {
                     System.out.println("mbtest: warning: invalid delay time - "+delayTime+"
- ignoring");
                  }
                  else {
                     Thread.sleep(delayTime);
                  }
                  cmd = di.readLine().trim();
                  continue;
               }
```

```
if (os == null || os2 == null ) {
   System.out.println("mbtestt: abort: no connection established to target");
   break;
}

// now convert to a byte string
// assume format is hex separated by whitespace
int ix = 0; // output index = number of bytes found
int sx = 0; // input index = substring position
int l = cmd.length();
while (sx < l) {
   int ex;
   ex = cmd.indexOf(' ',sx);
   if (ex<0) ex = l;
   int bval = Integer.parseInt(cmd.substring(sx,ex), 16);
   obuf[7+ix++] = (byte) bval;
   sx = ex+1;
}

if (ix == 0) {
   // handle blank line as comment
   System.out.println(cmd);
   cmd = di.readLine().trim();
   continue;
}

obuf[5] = (byte) (ix + 1);
obuf[6] = (byte) address;
//-----------------------------------------------------------------
// read the command for the second target
//-----------------------------------------------------------------
cmd = di.readLine().trim();
ix = 0; // output index = number of bytes found
sx = 0; // input index = substring position
l = cmd.length();
while (sx < l) {
    int ex;
    ex = cmd.indexOf(' ',sx);
    if (ex<0) ex = l;
    int bval = Integer.parseInt(cmd.substring(sx,ex), 16);
    obuf2[7+ix++] = (byte) bval;
    sx = ex+1;
}
```

```
            if (ix == 0) {
               // handle blank line as comment
                System.out.println(cmd);
                cmd = di.readLine().trim();
                continue;
            }

            obuf2[5] = (byte) (ix + 1);
            obuf2[6] = (byte) address;
            cmd = di.readLine().trim();
         }  // end of FOR loop reading the input text
         // purge the key buffer
         //for( ; ; ) {
             //keybuf = System.in.available();
             //if( keybuf == 0 )
                 //break;
             //keybuf = System.in.read();
         //}
         keybuf = System.in.available();
         if( keybuf > 1 )  {
//             skipcnt = keybuf;
//             System.out.println("skip: "+keybuf );
             do  {
                 System.in.read();
                 keybuf--;
             } while( keybuf > 1 );
         }

         System.out.println("Press any key to stop");
         expect = 0x40;
         first = 1;
         // Now loop performing the IO
         for( ; ; ) { // loop doing IO
         //for( ioCount=0; ioCount < 500; ioCount++ ) { // loop doing IO
            int c = 0;
            int c2 = 0;
            int ix;
            keybuf = System.in.available();
            if( dolog == 1 )
              System.out.println("keybuf = "+keybuf );
            if( keybuf >= 1 )  {
                keybuf = System.in.read();
```

```
    break;
}
ix = (int)obuf[5] - 1;
os.write(obuf,0,ix+7);
c = is.read(ibuf,0,261);
if (c<=0) {
   System.out.println("mbtest: abort: detected unexpected close of channel");
   break;
}
while (c<7 || c<(6 + (0xff & (int)(ibuf[5])))) {
   System.out.println("mbtest: warning: response appears fragmented");
   int cx = is.read(ibuf,c,261-c);
   if (cx<=0) {
      System.out.println("mbtest: warning: response incomplete");
      break;
   }
   c += cx;
}


// verify input length against fragmentation
if (((int)(ibuf[5])&0xff) != (c-6)) {
   System.out.println("mbtest: warning: response length mismatch");
}
if( dolog == 1 )  {
  log_buf(obuf, ix+7, ">");
  log_buf(ibuf, c, "<");
}
// write to the second target, using the 2nd buffer
// wait until the input has changed
if ( first == 1 || ( ibuf[12] & 0x40 ) == expect ) {
    first = 0;
   ix = (int)obuf2[5] - 1;
   // if input bit 0x40 is on, turn on output bit 0x01
   if( expect == 0x40 )  {
       expect = 0;
       obuf2[16] = (byte) (obuf2[16] & 0xbf); // bit 40 off
       obuf2[16] = (byte) (obuf2[16] | 0x01); // bit 01 on
   }
   else  {
       expect = 0x40;
       obuf2[16] = (byte) (obuf2[16] | 0x40); // bit 40 on
       obuf2[16] = (byte) (obuf2[16] & 0xfe); // bit 01 off
```

```
                }

                os2.write(obuf2,0,ix+7);

                c2 = is2.read(ibuf2,0,261);
                if (c2<=0) {
                   System.out.println("mbtest: abort: detected unexpected close of
channel");
                   break;
                }
                while (c2<7 || c2<(6 + (0xff & (int)(ibuf2[5])))) {
                   System.out.println("mbtest: warning: response appears fragmented");
                   int cx2 = is2.read(ibuf2,c2,261-c2);
                   if (cx2<=0) {
                      System.out.println("mbtest: warning: response incomplete");
                      break;
                   }
                   c2 += cx2;
                }
                if (((int)(ibuf2[5])&0xff) != (c2-6)) {
                   System.out.println("mbtest: warning: response length mismatch");
                }
                if( dolog == 1 )  {
                  log_buf(obuf2, ix+7, ">");
                  log_buf(ibuf2, c2, "<");
                }
                // check only the modbus response for equality (ignore address)
                //if (buf_diff(c, ibuf, c2, ibuf2)) {
                   //System.out.println("************* different");
                //}
             }
             if( dolog == 1 )
               System.out.println();
          } // end of FOR loop doing IO
       } catch (Exception e) { System.out.println("mbtest: unexpected exception:"+e); }
    }

    static boolean buf_diff(int c1, byte [] b1, int c2, byte [] b2) {
       // check only the modbus response for equality (ignore address)
       if (c1 != c2) return true;
       int i;
       for (i=7; i<c1; i++) {
          if (b1[i] != b2[i]) return true;
```

```
        }
        return false;
    }

    public static void log_buf(byte buf[], int len, String prefix) {
        System.out.print(prefix);
        int i;
        for (i=0; i<len; i++) {
            System.out.print(" "+Integer.toString((buf[i]>>4)&0xf, 16)+
                             Integer.toString(buf[i]&0xf, 16));
        }
    System.out.println();
    }
}
```

3.2 Source: test1.txt

This file contains the target addresses of the input and output communication adapters used in the test loop. The host server must resolve these to the IP addresses of the adapters.

The file also contains the input and output Modbus commands to the adapters for reading the base module inputs and writing the base module outputs.

The file's contents follow:

```
target eio4
target eio2
; input command followed by output command
3 0 0 0 2
10 0 0 0 2 4 00 00 00 40
quit
```