Acromag®

BusWorks® 900EN Series
10/100M Industrial Ethernet I/O Modules w/ Modbus

Technical Reference – Modbus TCP/IP

INTRODUCTION TO MODBUS TCP/IP

ACROMAG INCORPORATED
30765 South Wixom Road
P.O. BOX 437
Wixom, MI 48393-7037 U.S.A.

Tel: (248) 624-1541
Fax: (248) 624-9234

Copyright 2005, Acromag, Inc., Printed in the USA.
Data and specifications are subject to change without notice.

8500-765-A05C000

TABLE OF CONTENTS

INTRODUCTION TO MODBUS TCP/IP

MODBUS AND MODBUS TCP/IP..... 3
What is Modbus?..... 3
What is Modbus TCP/IP?..... 4
Why Combine Modbus With Ethernet?..... 5
What About Determinism?..... 6
THE OSI NETWORK MODEL..... 7
TCP/IP Stack..... 9
Key Concepts & Terminology..... 12
APPLICATION LAYER..... 13
Modbus Functions & Registers..... 13
Read Coil Status (01)..... 16
Read Holding Registers (03)..... 18
Read Input Registers (04)..... 18
Force Single Coil (05)..... 19
Preset Single Register (06)..... 20
Force Multiple Coils (15)..... 20
Preset Multiple Registers (16)..... 21
Report Slave ID (17)..... 22
Modbus Exceptions..... 23
Modbus TCP/IP ADU Format..... 25
Connection Manager..... 27
TRANSPORT LAYER..... 29
TCP – Transport Control Protocol..... 29
TCP Example..... 31
NETWORK LAYER..... 33
IP – Internet Protocol..... 33
Ethernet (MAC) Address..... 35
Internet (IP) Address..... 35
ARP – Address Resolution Protocol..... 37
RARP – Reverse Address Resolution Protocol..... 38
DATA LINK (MAC) LAYER..... 40
CSMA/CD – Carrier Sense Multiple Access w/CD..... 40
MAC – Media Access Control (MAC) Protocol..... 40
Ethernet (MAC) Packet..... 41

This information is provided as a service to our customers and to others interested in learning more about Modbus TCP/IP. Acromag assumes no responsibility for any errors that may occur in this document, and makes no commitment to update, or keep this information current.

Be sure to visit Acromag on the web at **www.acromag.com.**

Windows® is a registered trademark of Microsoft Corporation.
Modbus® is a registered trademark of Modicon, Incorporated.

All trademarks are the property of their respective owners.

MODBUS AND MODBUS TCP/IP

The following information describes the operation of Modbus TCP/IP as it relates to Acromag Series 900EN-40xx I/O modules. For more detailed information on Modbus, you may also refer to the "Modicon Modbus Reference Guide", PI-MBUS-300 Rev J, available via download from www.public.modicon.com.

Acromag manufactures a line of I/O modules that support Modbus over RS485, and Modbus over Ethernet. Feel free to visit our website at www.acromag.com to obtain the latest information about these and other Acromag products.

What is Modbus?

The Modbus protocol was developed in 1979 by Modicon, Incorporated, for industrial automation systems and Modicon programmable controllers. It has since become an industry standard method for the transfer of discrete/ analog I/O information and register data between industrial control and monitoring devices. Modbus is now a widely-accepted, open, public-domain protocol that requires a license, but does not require royalty payment to its owner.

Modbus devices communicate using a master-slave (client-server) technique in which only one device (the master/client) can initiate transactions (called queries). The other devices (slaves/servers) respond by supplying the requested data to the master, or by taking the action requested in the query. A slave is any peripheral device (I/O transducer, valve, network drive, or other measuring device) which processes information and sends its output to the master using Modbus. The Acromag I/O Modules form slave/server devices, while a typical master device is a host computer running appropriate application software. Other devices may function as both clients (masters) and servers (slaves).

Masters can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a response to all queries addressed to them individually, but do not respond to broadcast queries. Slaves do not initiate messages on their own, they only respond to queries from the master.

A master's query will consist of a slave address (or broadcast address), a function code defining the requested action, any required data, and an error checking field. A slave's response consists of fields confirming the action taken, any data to be returned, and an error checking field. Note that the query and response both include a device address, a function code, plus applicable data, and an error checking field. If no error occurs, the slave's response contains the data as requested. If an error occurs in the query received, or if the slave is unable to perform the action requested, the slave will return an exception message as its response (see Modbus Exceptions). The error check field of the slave's message frame allows the master to confirm that the contents of the message are valid. Traditional Modbus messages are transmitted serially and parity checking is also applied to each transmitted character in its data frame.

At this point, It's important to make the distinction that Modbus itself is an *application protocol,* as it defines rules for organizing and interpreting data, but remains simply a messaging structure, independent of the underlying physical layer. As it happens to be easy to understand, freely available, and accessible to anyone, it is thus widely supported by many manufacturers.

What is Modbus TCP/IP?

Modbus TCP/IP (also Modbus-TCP) is simply the Modbus RTU protocol with a TCP interface that runs on Ethernet.

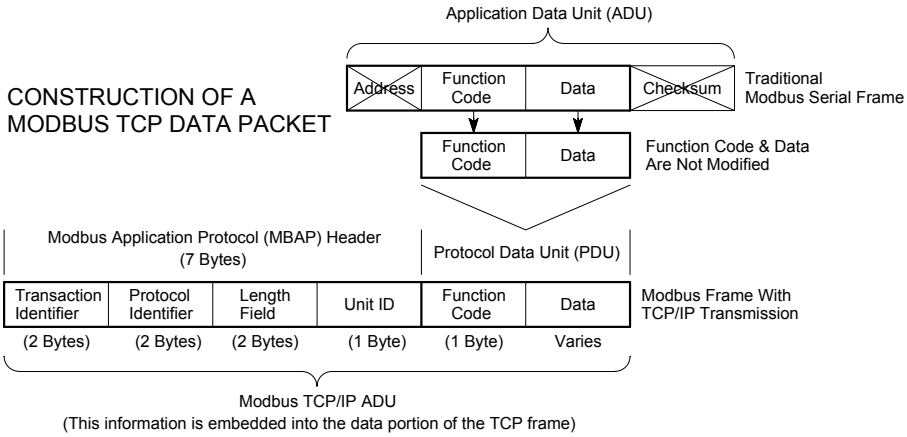
The Modbus messaging structure is the *application protocol* that defines the rules for organizing and interpreting the data independent of the data transmission medium.

TCP/IP refers to the Transmission Control Protocol and Internet Protocol, which provides the transmission medium for Modbus TCP/IP messaging.

Simply stated, TCP/IP allows blocks of binary data to be exchanged between computers. It is also a world-wide standard that serves as the foundation for the World Wide Web. The primary function of TCP is to ensure that all packets of data are received correctly, while IP makes sure that messages are correctly addressed and routed. Note that the TCP/IP combination is merely a *transport protocol, a*nd does not define what the data means or how the data is to be interpreted (this is the job of the application protocol, Modbus in this case).

So in summary, Modbus TCP/IP uses TCP/IP and Ethernet to carry the data of the Modbus message structure between compatible devices. That is, Modbus TCP/IP combines a physical network (Ethernet), with a networking standard (TCP/IP), and a standard method of representing data (Modbus as the application protocol). Essentially, the Modbus TCP/IP message is simply a Modbus communication encapsulated in an Ethernet TCP/IP wrapper.

In practice, Modbus TCP embeds a standard Modbus data frame into a TCP frame, without the Modbus checksum, as shown in the following diagram.



The Modbus commands and user data are themselves encapsulated into the data container of a TCP/IP telegram without being modified in any way. However, the Modbus error checking field (checksum) is not used, as the standard Ethernet TCP/IP link layer checksum methods are instead used to guaranty data integrity. Further, the Modbus frame address field is supplanted by the *unit identifier* in Modbus TCP/IP, and becomes part of the Modbus Application Protocol (MBAP) header (more on this later).

What is Modbus TCP/IP?

From the figure, we see that the function code and data fields are absorbed in their original form. Thus, a Modbus TCP/IP Application Data Unit (ADU) takes the form of a 7 byte header (transaction identifier + protocol identifier + length field + unit identifier), and the protocol data unit (function code + data). The MBAP header is 7 bytes long and includes the following fields:

- **Transaction/invocation Identifier (2 Bytes):** This identification field is used for transaction pairing when multiple messages are sent along the same TCP connection by a client without waiting for a prior response.
- **Protocol Identifier (2 bytes):** This field is always 0 for Modbus services and other values are reserved for future extensions.
- **Length (2 bytes):** This field is a byte count of the remaining fields and includes the unit identifier byte, function code byte, and the data fields.
- **Unit Identifier (1 byte):** This field is used to identify a remote server located on a non TCP/IP network (for serial bridging). In a typical Modbus TCP/IP server application, the unit ID is set to 00 or FF, ignored by the server, and simply echoed back in the response.

The complete Modbus TCP/IP Application Data Unit is embedded into the data field of a standard TCP frame and sent via TCP to well-known system port 502, which is specifically reserved for Modbus applications. Modbus TCP/IP clients and servers listen and receive Modbus data via port 502.

We can see that the operation of Modbus over Ethernet is nearly transparent to the Modbus register/command structure. Thus, if you are already familiar with the operation of traditional Modbus, then you are already very with the operation of Modbus TCP/IP.

Why Combine Modbus With Ethernet?

IEEE 802.3 Ethernet is a long-standing office networking protocol that has gained universal world-wide acceptance. It is also an open standard that is supported by many manufacturers and its infrastructure is widely available and largely installed. Consequently, its TCP/IP suite of protocols is used world-wide and even serves as the foundation for access to the World Wide Web. As many devices already support Ethernet, it is only natural to augment it for use in industrial applications.

Just as with Ethernet, Modbus is freely available, accessible to anyone, and widely supported by many manufacturers of industrial equipment. It is also easy to understand and a natural candidate for use in building other industrial communication standards. With so much in common, the marriage of the Modbus application protocol with traditional IEEE 802.3 Ethernet transmission forms a powerful industrial communication standard in Modbus TCP/IP. And because Modbus TCP/IP shares the same physical and data link layers of traditional IEEE 802.3 Ethernet and uses the same TCP/IP suite of protocols, it remains fully compatible with the already installed Ethernet infrastructure of cables, connectors, network interface cards, hubs, and switches.

What About Determinism?

Determinism is a term that is used here to describe the ability of the communication protocol to guaranty that a message is sent or received in a finite and predictable amount of time. We can surmise that, for critical control applications, determinism is very important.

Historically, traditional Ethernet was not considered a viable fieldbus for industrial control and I/O networks because of two major shortcomings: inherent non-determinism, and low durability. However, new technology properly applied has mostly resolved these issues.

Originally, Ethernet equipment was designed for the office environment, not harsh industrial settings. Although, many factory Ethernet installations can use this standard hardware without a problem, new industrial-rated connectors, shielded cables, and hardened switches and hubs are now available to help resolve the durability issue.

With respect to the non-deterministic behavior of Ethernet, this is largely a result of the arbitration protocol it uses for carrier transmission access on the network. That is, Carrier Sense Multiple Access with Collision Detect (CSMA/CD). Since any network device can try to send a data frame at any time, with CSMA/CD applied, each device will first sense whether the line is idle and available for use. If the line is available, the device will then begin to transmit its first frame. If another device also tries to send a frame at approximately the same time, then a collision occurs and both frames will be discarded. Each device then waits a random amount of time and retries its transmission until its frame is successfully sent. This channel-allocation method is inherently non-deterministic because a device may only transmit when the wire is free, resulting in unpredictable wait times before data may be transmitted. Additionally, because of cable signaling delay, collisions are still possible once the device begins to transmit the data, thus forcing additional retransmission/retry cycles.

As most control systems have a defined time requirement for packet transmission, typically less than 100ms, the potential for collisions and the CSMA/CD method of retransmission is not considered deterministic behavior and this is the reason that traditional Ethernet has had problems being accepted for use in critical control applications. However, CSMA/CD is naturally suppressed on a network of devices that are interconnected via Ethernet switches. Specifically, one device per switch port. This makeup is commonly referred to as switched Ethernet in an effort to distinguish itself from the non-deterministic behavior of traditional Ethernet.

Ethernet is made more deterministic via the use of fast Ethernet switches to interconnect devices. These switches increase the bandwidth of large networks by sub-dividing them into several smaller networks or separate “collision domains”. The switch also minimizes network chatter by facilitating a direct connection from a sender to a receiver in such a way that only the receiver receives the data, not the entire network.

What About Determinism?

So how does a switch (or switching hub) work to increase determinism? Each port of a switch forwards data to another port based on the MAC address contained in the received data packet/frame. The switch actually learns and stores the MAC addresses of every device it is connected to, along with the associated port number. Now the port of the switch does not require its own MAC address, and during retransmission of a received packet, the switch port will instead look like the originating device by having assumed its source address. In this way, the Ethernet collision domain is said to terminate at the switch port, and the switch effectively breaks the network into separate distinct data links or collision domains, one at each switch port. The ability of the switch to target a packet to a specific port, rather than forwarding it to all switch ports, also helps to eliminate the collisions that make Ethernet non-deterministic.

So, as switches have become less expensive, the current tendency in critical industrial control applications is to connect one Ethernet device per switch port, effectively treating the switch device as the hub of a star network. Since there is only one device connected to a port, there is no chance of collisions occurring. This effectively suppresses the CSMA/CD routine. In this manner, with only one network device connected per switch port, the switch can run full-duplex, with no chance of collisions. Thus, a 10/100 Ethernet switch effectively runs at 20/200 Mbps because it can transmit and receive at 10 or 100 Mbps simultaneously in both directions (full duplex). The higher transfer speed of full-duplex coupled without the need for invoking CSMA/CD produces a more deterministic mode of operation, helping critical control applications to remain predictable and on-time.

Unfortunately, broadcast traffic on a company network cannot be completely filtered by switches, and this may cause additional collisions reducing the determinism of a network connecting more than one device to a switch port. However, if the company network and the control & I/O network are instead separated, no traffic is added to the control network and its determinism is increased. Further, if a bridge is used to separate the two networks, then the bridge can usually be configured to filter unnecessary traffic.

So we see how combining good network design with fast switches and bridges where necessary raises the determinism of a network, making Ethernet more appealing. Other advances in Ethernet switches, such as, higher speeds, broadcast storm protection, virtual LAN support, SNMP, and priority messaging further help to increase the determinism of Ethernet networks. As Gigabit (Gbit), 10Gbit, and 100Gbit Ethernet enters the market, determinism will no longer be a concern.

THE OSI NETWORK MODEL

In order to better understand how Modbus TCP/IP is structured and the meaning of the term “open standard”, we need to review the Open Systems Interconnect (OSI) Reference Model. This model was developed by the International Standards Organization and adopted in 1983 as a common reference for the development of data communication standards, like Modbus TCP/IP. It does not attempt to define an actual implementation, but rather it serves as a structural aide to understanding *“what must be done”* and *“what goes where”.*

THE OSI NETWORK MODEL

The traditional OSI model is presented below, along with the simplified 5-layer TCP/IP Standard (layers 5 & 6 suppressed). In the OSI model, the functions of communication are divided into seven (or five) layers, with every layer handling precisely defined tasks. For example, Layer 1 of this model is the physical layer and defines the physical transmission characteristics. Layer 2 is the data link layer and defines the bus access protocol. Layer 7 is the application layer and defines the application functions (this is the layer that defines how device data is to be interpreted).

The OSI Model represents the basic network architecture.

Each layer of this model uses the services provided by the layer immediately below it.

TCP/IP has no specific mappings to layers 5 and 6 of this model and these layers are often omitted when referring to the TCP/IP stack.

OSI 7-LAYER MODEL			**TCP/IP Standard**
7	Application	Used by software applications to prepare and interpret data for use by the other six OSI layers below it. Provides the application interface to the network. HTTP, FTP, email SMTP & POP3, CIP™, SNMP, are all found at this layer.	Application Layer
6	Presentation	Representation of data, coding type, and defines used characters. Performs data and protocol negotiation and conversion to ensure that data may be exchanged between hosts and transportable across the network. Also performs data compression and encryption.	
5	Session	Dialing control and synchronization of session connection. Responsible for establishing and managing sessions/ connections between applications & the network. Windows WinSock socket API is a common session layer manager.	
4	Transport	Sequencing of application data, controls start/end of transmission, provides error detection, correction, end-to-end recovery, and clearing. Provides software flow control of data between networks. TCP & UDP are found here.	Transport Layer
3	Network	Controls routing, prioritization, setup, release of connections, flow control. Establishes/maintains connections over a network & provides addressing, routing, and delivery of packets to hosts. IP, PPP, IPX, & X.25 are found here.	Internet, Network, or Internetwork Layer
2	Data Link	Responsible for ensuring reliable delivery at the lowest levels, including data frame, error detection and correction, sequence control, and flow control. Ethernet (IEEE 802.2) and MAC are defined at this level.	Network Access Layer or
1	Physical	Defines the electrical, mechanical, functional, and procedural attributes used to access and send a binary data stream over a physical medium (defines the RJ-45 connector & CAT5 cable of Ethernet).	Host-to-Network Layer

THE OSI NETWORK MODEL

By the OSI Model, we can infer that in order for two devices to be interoperable on the same network, they must have the same application-layer protocol. In the past, many network devices have used their own proprietary protocols and this has hindered their interoperability. This fact further drove the need for adoption of open network I/O solutions that would allow devices from a variety of vendors to seamlessly work together, and this drive for *interoperability* is a key reason Modbus TCP/IP was created.

Note that in the TCP/IP Standard Model, Ethernet handles the bottom 2 layers (1 & 2) of the seven layer OSI stack, while TCP/IP handles the next two layers (3 & 4). The application layer lies above TCP, IP, and Ethernet and is the layer of information that gives meaning to the transmitted data.

With Acromag 9xxEN-40xx Modbus TCP/IP modules, the application layer protocol is Modbus. That is, Modbus TCP/IP uses Ethernet media and TCP/IP to communicate using an application layer with the same register access method as Modbus RTU. Because many manufacturers happen to support Modbus RTU and TCP/IP, and since Modbus is also widely understood and freely distributed, Modbus TCP/IP is also considered an open standard.

So we see that Modbus TCP/IP is based on the TCP/IP protocol family and shares the same lower four layers of the OSI model common to all Ethernet devices. This makes it fully compatible with existing Ethernet hardware, such as cables, connectors, network interface cards, hubs, and switches.

The TCP/IP Stack

TCP/IP refers to the Transmission Control Protocol and Internet Protocol which were first introduced in 1974. TCP/IP is the foundation for the World Wide Web and forms the transport and network layer protocol of the internet that commonly links all Ethernet installations world-wide. Simply stated, TCP/IP allows blocks of binary data to be exchanged between computers. The primary function of TCP is to ensure that all packets of data are received correctly, while IP makes sure that messages are correctly addressed and routed. TCP/IP does not define what the data means or how the data is to be interpreted, it is merely a *transport protocol*.

To contrast, Modbus is an *application protocol*. It defines rules for organizing and interpreting data and is essentially a messaging structure that is independent of the underlying physical layer. It is freely available and accessible to anyone, easy to understand, and widely supported by many manufacturers.

Modbus TCP/IP uses TCP/IP and Ethernet to carry the data of the Modbus message structure between devices. That is, Modbus TCP/IP combines a physical network (Ethernet), with a networking standard (TCP/IP), and a standard method of representing data (Modbus).

The TCP/IP Stack

TCP/IP is actually formed from a "suite" of protocols upon which all internet communication is based. This suite of protocols is also referred to as a *protocol stack.* Each host or router on the internet must run a protocol stack. The use of the word stack refers to the simplified TCP/IP layered Reference Model or "stack" that is used to design network software and outlined as follows:

5	Application	Specifies how an application uses a network.
4	Transport	Specifies how to ensure reliable data transport.
3	Internet/Network	Specifies packet format and routing.
2	Host-to-Network	Specifies frame organization and transmittal.
1	Physical	Specifies the basic network hardware.

To better understand stack operation, the following table illustrates the flow of data from a sender to a receiver using the TCP/IP stack (we've renamed the Host-to-Network layer to the more commonly used Data Link Layer):

SENDER	→ *Virtual Connection* →	**RECEIVER**
↓ Application	← *Equivalent Message* →	Application ↑
↓ Transport	← *Equivalent Message* →	Transport ↑
↓ Internet/Network	← *Equivalent Message* →	Internet/Network ↑
↓ Data Link Layer	← *Equivalent Message* →	Data Link Layer ↑
→ Physical Hardware	→→→→→→→→→→→→	Physical Hardware ↑

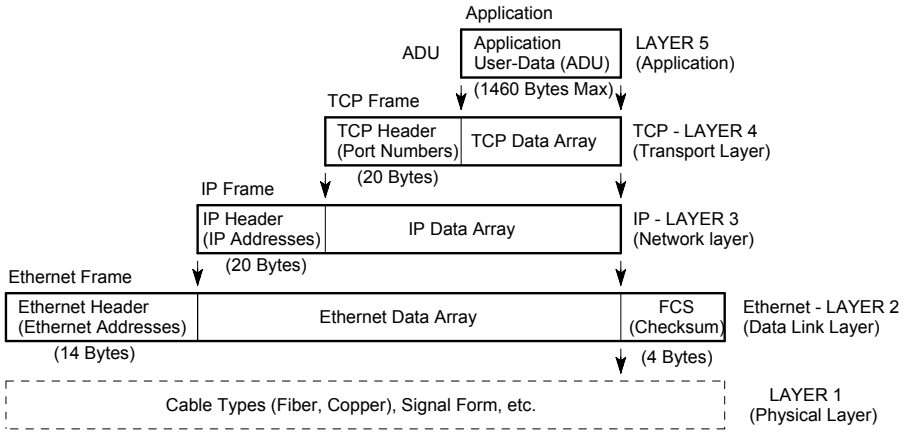
Each layer on the sending stack communicates with the corresponding layer of the receiving stack through information stored in headers. As you move the data down the stack of the sender, each stack layer adds its own header to the front of the message that it receives from the next higher layer. That is, the higher layers are encapsulated by the lower layers. Conversely, this header information is removed by the corresponding layer at the Receiver. In this way, the headers are essentially peeled off as the data packet moves up the receiving stack to the receiver application.

MODBUS TCP/IP COMMUNICATION STACK			
#	**MODEL**	**IMPORTANT PROTOCOLS**	**Reference**
7	Application	Modbus	
6	Presentation		
5	Session		
4	Transport	TCP	
3	Network	IP, ARP, RARP	
2	Data Link	Ethernet, CSMA/CD, MAC	IEEE 802.3 Ethernet
1	Physical	Ethernet Physical Layer	

TCP/IP Stack

The following figure illustrates the construction of a TCP/IP-Ethernet packet for transmission. For Modbus TCP/IP, the application layer is Modbus and the Modbus Application Data Unit is embedded into the TCP data array. When an application sends its data over the network, the data is passed down through each layer--note how the upper layer information is wrapped into the data bytes of the next lowest layer (encapsulated). Each subsequent layer has a designated function and attaches its own protocol header to the front of its packet. The lowest layer is responsible for actually sending the data. This entire wrap-into procedure is then reversed for data received (the data received is unwrapped at each level and passed up thorough to the receiver's application layer).

Figure 1: CONSTRUCTION OF A TCP/IP-ETHERNET DATA PACKET



To illustrate, with Modbus TCP/IP, the host (master/client) application forms its request, then passes its data down to the lower layers, which add their own control information to the packet in the form of protocol headers and sometimes footers. Finally the packet reaches the physical layer where it is electronically transmitted to the destination (slave/server). The packet then travels up through the different layers of its destination with each layer decoding its portion of the message and removing the header and footer that was attached by the same layer of the sending client computer. Finally the packet reaches the destination application. Although each layer only communicates with the layer just above or just below it, this process can be viewed as one layer at one end talking to its partner (peer) layer at the opposite end.

Key Concepts & Terminology

To better understand Modbus TCP/IP and the operation of a stack, please review the following key concepts and terminology:

- All network protocols are structured as a *layered model*.
- There's one or more protocols (layer entities) at every layer.
- *Peer entities* refer to two or more protocols on the same layer (including protocols at the same layer on different nodes).
- Operation rules between peer entities are called *procedures*.
- *Protocol* refers to the rules of operation followed by peer entities. The Protocol defines the format of PDU's (Protocol Data Units) and their rules of operation.
- This layering of protocol entities is referred to as the *protocol stack*.
- Layer n communicates with other layer n entities (other protocols on the same layer) using layer n *Protocol Data Units (PDU's).*
- Layer n uses the *service* of layer n-1 and offers a *service* to layer n+1.
- The interface between a layer and the layer above it is referred to as the *Service Access Point (SAP).* The interface data between the layers is the *Service Data Unit (SDU).*
- Protocols are either *connection oriented or connectionless*. A connection implies that the communication requires synchronization of all parties before application data can actually be exchanged. Modbus TCP/IP is a connection oriented protocol.
- Modbus TCP/IP follows the *Client-Server model*. Modbus masters are referred to as clients, while Modbus slaves are servers.
- A *client* (or master) is any network device that sends data requests to servers (or slaves).
- A *server* (or slave) is any program that awaits data requests to be sent to it. Servers do not initiate contact with clients, but only respond to them. Some devices operate as both clients and servers.
- A *port* is an address that is used locally at the transport layer (on one node) and identifies the source and destination of the packet inside the same node. Port numbers are divided between well-known port numbers (0-1023), registered user port numbers (1024-49151), and private/dynamic port numbers (49152-65535). Ports allow TCP/IP to multiplex and demultiplex a sequence of IP datagrams that need to go to many different (simultaneous) application processes. Modbus TCP/IP uses well-known port 502 to listen and receive Modbus messages over Ethernet.
- A *socket* is an application layer address that is formed from the combination of an IP address and port number (expressed as <Host IP Address>:<Port Number> or <Host Name>:<Port Number>) and is used as the overall identification address of an application process. Application protocols use this to keep track of the port number assigned to each instance of an application when using TCP.

APPLICATION LAYER

The uppermost layer of the TCP/IP and OSI Reference Models is the *Application Layer*. There are many application layer protocols that may reside here, such as FTP, Telnet, HTTP, SMPT, DNS, and NNTP, among others. While each of these protocols has their own specific purpose, for Modbus TCP/IP, the primary application layer protocol of interest is Modbus.

Modbus Functions and Registers

The TCP/IP protocol suite (or stack of independent protocols) provides all the resources for two devices to communicate with each other over an Ethernet Local-Area Network (LAN), or global Wide-Area Network (WAN). But TCP/IP only guarantees that application messages will be transferred between these devices, it does not guaranty that these devices will actually understand or *interoperate* with one another. For Modbus TCP/IP, this capability is provided by the application layer protocol Modbus.

Modbus is an *application protocol* or messaging structure that defines rules for organizing and interpreting data independent of the data transmission medium. Traditional serial Modbus is a register-based protocol that defines message transactions that occur between masters and slaves. Slave devices listen for communication from the master and simply respond as instructed. The master always controls the communication and may communicate directly to one slave, or all connected slaves, but the slaves cannot communicate directly with each other.

Thus, we see that Modbus operates according to the common client/server (master/slave) model. That is, the client (master) sends a request telegram (service request) to the server (slave), and the server replies with a response telegram. If the server cannot process a request, it will instead return a error function code (exception response) that is the original function code plus 80H (i.e. with its most significant bit set to 1).

Modbus functions operate on memory registers to configure, monitor, and control device I/O. Modbus devices usually include a Register Map. You should refer to the register map for your device to gain a better understanding of its operation. You will also find it helpful to refer to the register map as you review the Modbus functions described later in this document.

The Modbus data model has a simple structure that only differentiates between four basic data types:

Discrete Inputs
Coils (Outputs)
Input Registers (Input Data)
Holding Registers (Output Data)

The service request (Modbus Protocol Data Unit) is comprised of a function code, and some number of additional data bytes, depending on the function. In most cases, the additional data is usually a variable reference, such as a register address, as most Modbus functions operate on registers.

Modbus Functions and Registers

The "x" following the leading character represents a four-digit address location in user data memory.

The leading character is generally implied by the function code and omitted from the address specifier for a given function. The leading character also identifies the I/O data type.

The Modbus registers of a device are organized around the four basic data reference types noted above and this data type is further identified by the leading number of the reference address as follows:

Reference	Description
0xxxx	Read/Write Discrete Outputs or Coils. A 0x reference address is used to drive output data to a digital output channel.
1xxxx	Read Discrete Inputs. The ON/OFF status of a 1x reference address is controlled by the corresponding digital input channel.
3xxxx	Read Input Registers. A 3x reference register contains a 16-bit number received from an external source—e.g. an analog signal.
4xxxx	Read/Write Output or Holding Registers. A 4x register is used to store 16-bits of numerical data (binary or decimal), or to send the data from the CPU to an output channel.

IMPORTANT: The reference addresses noted in the memory map are not explicit hard-coded memory addresses. Internally, all Modbus devices use a zero-based memory offset computed from the reference address. However, the system interface of Modbus systems (software) will vary in this regard and may require you to enter the actual reference address, drop the leading number, or enter an absolute memory offset from 1, or a memory address offset from 0. This is system dependent and a common source of programming errors. Be wary of this when writing higher-level application programs to access these registers.

Note that not all Modbus functions operate on register map registers. All data addresses in Modbus messages are referenced to 0, with the first occurrence of a data item addressed as item number zero. Further, a function code field already specifies which register group it operates on (i.e. 0x, 1x, 3x, or 4x reference addresses). For example, holding register 40001 is addressed as register 0000 in the data address field of the message. The function code that operates on this register specifies a "holding register" operation and the "4xxxx" reference group is implied. Thus, holding register 40108 is actually addressed as register 006BH (107 decimal).

The function code field of the message (PDU) will contain one byte that tells the slave what kind of action to take. Valid function codes are from 1-255, but not all codes will apply to a module and some codes are reserved for future use. Additionally, the Modbus specification allocates function codes 65-72 and 100-110 for user-defined services.

The following table highlights a subset of standard Modbus functions commonly supported by Acromag modules (the reference register addresses that the function operates on are also indicated). The functions below are used to access the registers outlined in the register map of the module for sending and receiving data. The Report Slave ID command does not operate on a register map register.

Modbus Functions and Registers

CODE	FUNCTION	REFERENCE
01 (01H)	Read Coil (Output) Status	0xxxx
03 (03H)	Read Holding Registers	4xxxx
04 (04H)	Read Input Registers	3xxxx
05 (05H)	Force Single Coil (Output)	0xxxx
06 (06H)	Preset Single Register	4xxxx
15 (0FH)	Force Multiple Coils (Outputs)	0xxxx
16 (10H)	Preset Multiple Registers	4xxxx
17 (11H)	Report Slave ID	*Hidden*

The client request data field provides the slave (server) with any additional information required by the slave to complete the action specified by the function code in the client's request. The data field typically includes register addresses, count values, and written data. For some messages, this field may not exist (has zero length), as not all messages will require data.

When the slave device responds to the master, it uses the function code field to indicate either a normal (error-free) response, or that some kind of error has occurred (an exception response). A normal response simply echoes the original function code of the query, while an exception response returns a code that is equivalent to the original function code with its most significant bit (msb) set to logic 1.

For example, the Read Holding Registers command has the function code 0000 0011 (03H). If the slave device takes the requested action without error, it returns the same code in its response. However, if an exception occurs, it returns 1000 0011 (83H) in the function code field and appends a unique code in the data field of the response message that tells the master device what kind of error occurred, or the reason for the exception (see Modbus Exceptions).

The client application program must handle the exception response. It may choose to post subsequent retries of the original message, it may try sending a diagnostic query, or it may simply notify the operator of the exception error.

The following paragraphs describe some of the Modbus functions commonly supported by Acromag 9xxEN modules. These examples are depicted from a Modbus TCP/IP perspective (the unit ID replaces the slave address, the traditional serial CRC/LRC error checking field is dropped). Only the first example will include the MBAP header information. You should refer to the Modbus specification for a complete description of all Modbus functions. To gain a better understanding of Modbus, please refer to your module's register map as you review this material.

When you review these examples and compare them to traditional serial Modbus commands, note that the slave address is supplanted by the unit identifier in Modbus TCP/IP (normally set to 00H or FFH). In addition, the error check field (CRC/LRC) is removed, as TCP/IP already applies its own error checking. For commands that support broadcast transmission, this applies to serial Modbus only, as Modbus TCP/IP is unicast only (except where an Ethernet-to-serial bridge is used).

The different fields of the of the Modbus TCP/IP ADU are encoded in Big-Endian format. This means that the most significant byte in the sequence is stored at the lowest storage address (i.e. it is first).

The following example will include the format of the MBAP header information, but the header information will not be repeated in the successive examples.

Read Coil Status (01)

This command will read the ON/OFF status of discrete outputs or coils (0x reference addresses) in the slave/server. For Acromag modules, its response is equivalent to reading the on/off status of solid-state output relays or switches. Broadcast transmission is not supported.

The Read Coil Status query specifies the starting coil (output channel) and quantity of coils to be read. Coils correspond to the discrete solid-state relays of this device and are addressed starting from 0 (up to 4 coils addressed as 0-3 for this model). The Read Coil Status in the response message is packed as one coil or channel per bit of the data field. For Acromag modules, the output status is indicated as 1 for ON (conducting current), and 0 for OFF (not conducting). The LSB of the first data byte corresponds to the status of the coil addressed in the query. The other coils follow sequentially, moving toward the high order end of the byte. Since this example has only 4 outputs, the remaining bits of the data byte will be set to zero toward the unused high order end of the byte.

Modbus Request ADU Example Header

MBAP Header Fields	Example Decimal (Hexadecimal)
Transaction ID High Order	0 (00) *Client sets, unique value.*
Transaction ID Low Order	1 (01) *Client sets, unique value.*
Protocol Identifier High Order	0 (00) *Specifies Modbus service.*
Protocol Identifier Low Order	0 (00) *Specifies Modbus service.*
Length High Order	0 (00) *Client calculates.*
Length Low Order	6 (06) *Client calculates.*
Unit Identifier	255 (FF) or 0 (00) *Do not bridge.*

The *transaction identifier* is used to match the response with the query when the client sends multiple queries without waiting for a prior response. It is typically a number from 1 to 16, but the maximum number of client transactions and the maximum number of server transactions will vary according to the device. The *protocol identifier* is always 0 for Modbus. The *length* is a count of the number of bytes contained in the data plus the function code (1 byte) and unit identifier (1 byte).

The *unit identifier* is 00H or FFH, as this module is Modbus TCP/IP. If this module was a traditional serial Modbus type (no Ethernet port), and it was being addressed via a bridge or gateway from an Ethernet client (an Ethernet-to-serial bridge), then the unit identifier is equivalent to the traditional serial Modbus slave address (1-247). Using 00H or FFH as shown here will cause the any serial bridge/gateway device to block the passage of this client message across the bridge. This is why some text will show the unit identifier as part of the query itself since it supplants the traditional slave address (note that the length includes the unit identifier byte), while others show it as part of the MBAP header as is done here.

Read Coil Status (01)

This example reads the output channel status of coils 0-3.

Modbus Request ADU Example - Read Coil Status Query

Field Name	Example Decimal (Hexadecimal)
Function Code	1 (01)
Starting Address High Order	0 (00)
Starting Address Low Order	0 (00)
Number Of Points High Order	0 (00)
Number Of Points Low Order	4 (04)

Note that the leading character of the 0x reference address is implied by the function code and omitted from the address specified. In this example, the first address is 00001, referenced via 0000H, and corresponding to coil 0.

Modbus Response ADU Example Header

MBAP Header Fields	Example Decimal (Hexadecimal)
Transaction ID High Order	0 (00) *Echoed back, no change*
Transaction ID Low Order	1 (01) *Echoed back, no change*
Protocol Identifier High Order	0 (00) *Echoed back, no change*
Protocol Identifier Low Order	0 (00) *Echoed back, no change*
Length High Order	0 (00) *Server calculates*
Length Low Order	4 (04) *Server calculates.*
Unit Identifier	255 (FF) or 0 (00) *No change*

Modbus Response ADU Example - Read Coil Status Response

Field Name	Example Decimal (Hexadecimal)
Function Code	1 (01)
Byte Count	1 (01)
Data (Coils 3-0)	10 (0A)

Note that the response function code is the same as the request function code. The transaction identifier is preserved by the server and returned. The protocol identifier remains 0 for Modbus. The length of the response is calculated by the server and is the size of the Modbus server's PDU, plus the unit identifier (1 byte). The unit identifier is the same as what was received from the client.

If an error had occurred, the response function code is modified and set equal to the request function code plus 80H. The transaction ID, protocol ID, and unit identifier stay the same. The length becomes 0002H (2 bytes). The PDU then becomes the exception code value itself (1 byte). Refer to Modbus Exceptions for information on exception codes.

To summarize, the status of coils 3-0 is shown as the byte value 0A hex, or 00001010 binary. Coil 3 is the fifth bit from the left of this byte, and coil 0 is the LSB. The four remaining bits (toward the high-order end) are zero. Reading left to right, the output status of coils 3..0 is ON-OFF-ON-OFF.

Bin	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
Hex	**0**				**A**			
Coil	**NA**	**NA**	**NA**	**NA**	**3**	**2**	**1**	**0**

The following examples do not repeat the information contained in the MBAP request and response headers, only the Modbus PDU information is provided. Refer to the previous example for MBAP header format.

Read Holding Registers (03)

Reads the binary contents of holding registers (4x reference addresses) in the slave device. Broadcast transmission is not supported.

The Read Holding Registers query specifies the starting register and quantity of registers to be read. Note that registers are addressed starting at 0 (registers 1-16 addressed as 0-15). The Read Holding Registers response message is packed as two bytes per register, with the binary contents right-justified in each byte. For each register, the first byte contains the high order bits and the second byte the low order bits.

This example reads holding registers 40006…40008 (Channel 0 high limit value, low limit value, & deadband value).

Modbus PDU Example - Read Holding Register Query

Field Name	Example Decimal (Hexadecimal)
Function Code	3 (03)
Starting Address High Order	0 (00)
Starting Address Low Order	5 (05)
Number Of Points High Order	0 (00)
Number Of Points Low Order	3 (03)

Modbus PDU Example - Read Holding Register Response

Field Name	Example Decimal (Hexadecimal)
Function Code	3 (03)
Byte Count	6 (06)
Data High (Register 40006)	(3A)
Data Low (Register 40006)	75%=15000 (98)
Data High (Register 40007)	(13)
Data Low (Register 40007)	25%=5000 (88)
Data High (Register 40008)	(00)
Data Low (Register 40008)	1%=200 (C8)

To summarize our example, the contents of register 40006 (2 bytes) is the channel 0 high limit of 75% (15000=3A98H). The contents of register 40007 (2 bytes) is the channel 0 low limit of 25% (5000=1388H). The contents of register 40008 is the channel 0 deadband value (2 bytes) of 1% (200=00C8H).

Read Input Registers (04)

This command will read the binary contents of input registers (3x reference addresses) in the slave device. Broadcast transmission is not supported.

The Read Input Registers query specifies the starting register and quantity of registers to be read. Note that registers are addressed starting at 0. That is, registers 1-16 are addressed as 0-15. The Read Input Registers response message is packed as two bytes per register, with the binary contents right-justified in each byte. For each register, the first byte contains the high order bits and the second byte the low order bits.

Modbus PDU Example - Read Input Registers Query

Field Name	Example Decimal (Hexadecimal)
Function Code	4 (04)
Starting Address High Order	0 (00)
Starting Address Low Order	2 (02)
Number Of Points High Order	0 (00)
Number Of Points Low Order	2 (02)

Read Input Registers (04)

This example reads input registers 30003 & 30004 (Channel 0 input value and status).

Modbus PDU Example - Read Input Registers Response

Field Name	Example Decimal (Hexadecimal)
Function Code	4 (04)
Byte Count	4 (04)
Data High (Register 30003)	(3E)
Data Low (Register 30003)	80%=16000 (80)
Data High (Register 30004)	(00)
Data Low (Register 30004)	136 (88)

To summarize our example, the contents of register 30003 (2 bytes) is the channel 1 input value of 80% (16000=3E80H). The contents of register 30004 (2 bytes) is the channel 0 status flags of 136 (0088H)—i.e. flagging high limit exceeded.

Force Single Coil (05)

Forces a single coil/output (0x reference address) ON or OFF. With broadcast transmission (address 0), it forces the same coil in all networked slaves (serial Modbus only).

The Force Single Coil query specifies the coil reference address to be forced, and the state to force it to. The ON/OFF state is indicated via a constant in the query data field. A value of FF00H forces the coil to be turned ON (i.e. the corresponding solid-state relay is turned ON or closed), and 0000H forces the coil to be turned OFF (i.e. the solid-state output relay is turned OFF or opened). All other values are invalid and will not affect the coil.

Coils are referenced starting at 0—up to 4 coils are addressed as 0-3 for our example and this corresponds to the discrete output channel number.

Modbus PDU Example - Force Single Coil Query and Response

Field Name	Example Decimal (Hexadecimal)
Function Code	5 (05)
Coil Address High Order	0 (00)
Coil Address Low Order	3 (03)
Force Data High Order	255 (FF)
Force Data Low Order	0 (00)

The following example forces discrete output 3 ON.

The Force Single Coil response message is simply an echo (copy) of the query as shown above, but returned after executing the force coil command. No response is returned to broadcast queries from a master device (serial Modbus only).

Preset Single Register (06)

This command will preset a single holding register (4x reference address) to a specific value. Broadcast transmission is supported by this command (serial Modbus only) and will act to preset the same register in all networked slaves.

The Preset Single Register query specifies the register reference address to be preset, and the preset value. Note that registers are addressed starting at 0--registers 1-16 are addressed as 0-15. The Preset Single Registers response message is an echo of the query, returned after the register contents have been preset.

This example writes a baud rate of 9600bps to holding register 40002 (Baud Rate).

Modbus PDU Example - Preset Holding Register Query and Response

Field Name	Example Decimal (Hexadecimal)
Function Code	6 (06)
Register Address High Order	0 (00)
Register Address Low Order	1 (01)
Preset Data High Order	0 (00)
Preset Data Low Order	2 (02)

The response message is simply an echo (copy) of the query as shown above, but returned after the register contents have been preset. No response is returned to broadcast queries from a master (serial Modbus only).

Force Multiple Coils (15)

Simultaneously forces a series of coils (0x reference address) either ON or OFF. Broadcast transmission is supported by this command (serial Modbus only) and will act to force the same block of coils in all networked slaves.

The Force Multiple Coils query specifies the starting coil reference address to be forced, the number of coils, and the force data to be written in ascending order. The ON/OFF states are specified by the contents in the query data field. A logic 1 in a bit position of this field requests that the coil turn ON, while a logic 0 requests that the corresponding coil be turned OFF. Unused bits in a data byte should be set to zero. Note that coils are referenced starting at 0—up to 4 coils are addressed as 0-3 for this example and this also corresponds to the discrete output channel number.

This example forces outputs 1 & 3 OFF, and 0 & 2 ON for coils 0-3.

Modbus PDU Example - Force Multiple Coils Query

Field Name	Example Decimal (Hexadecimal)
Function Code	15 (0F)
Coil Address High Order	0 (00)
Coil Address Low Order	0 (00)
Number Of Coils High Order	0 (00)
Number Of Coils Low Order	4 (04)
Byte Count	01
Force Data High (First Byte)	5 (05)

Note that the leading character of the 0x reference address is implied by the function code and omitted from the address specified. In this example, the first address is 00001 corresponding to coil 0 and referenced via 0000H. Thus, in this example the data byte transmitted will address coils 3...0, with the least significant bit addressing the lowest coil in this set as follows (note that the four unused upper bits of the data byte are set to zero):

Bin	0	0	0	0	0	1	0	1
Hex	0				5			
Coil	NA	NA	NA	NA	3	2	1	0

Force Multiple Coils (15)

Modbus PDU Example - Force Multiple Coils Response

Field Name	Example Decimal (Hexadecimal)
Function Code	15 (0F)
Coil Address High Order	0 (00)
Coil Address Low Order	0 (00)
Number Of Coils High Order	0 (00)
Number Of Coils Low Order	4 (04)

The Force Multiple Coils normal response message returns the slave address, function code, starting address, and the number of coils forced, after executing the force instruction. Note that it does not return the byte count or force value. No response is returned to broadcast queries from a master device (serial Modbus).

Preset Multiple Registers (16)

Presets a block of holding registers (4x reference addresses) to specific values. Broadcast transmission is supported by this command and will act to preset the same block of registers in all networked slaves (serial Modbus only).

The Preset Multiple Registers query specifies the starting register reference address, the number of registers, and the data to be written in ascending order. Note that registers are addressed starting at 0--registers 1-16 are addressed as 0-15.

Modbus PDU Example - Preset Multiple Registers Query

Field Name	Example Decimal (Hexadecimal)
Function Code	16 (10)
Starting Register High Order	0 (00)
Starting Register Low Order	0 (00)
Number Of Registers High Order	0 (00)
Number Of Registers Low Order	3 (03)
Preset Data High (First Register)	0 (00)
Preset Data Low (First Register)	200 (C8)
Preset Data High (Second Reg)	0 (00)
Preset Data Low (Second Reg)	5 (05)
Preset Data High (Third Reg)	0 (00)
Preset Data Low (Third Reg)	2 (02)

This example writes a new slave address of 200, a baud rate of 28800bps, and sets parity to even, by writing to holding registers 40001 through 40003 (changes to slave address, baud rate, and parity will take effect following the next software or power-on reset).

Modbus PDU Example - Preset Multiple Registers Response

Field Name	Example Decimal (Hexadecimal)
Function Code	16 (10)
Starting Register High Order	0 (00)
Starting Register Low Order	0 (00)
Number Of Registers High Order	0 (00)
Number Of Registers Low Order	3 (03)

Preset Multiple Registers (16)

The Preset Multiple Registers normal response message returns the slave address, function code, starting register reference, and the number of registers preset, after the register contents have been preset. Note that it does not echo the preset values. No response is returned to broadcast queries from a master device (serial Modbus only).

Report Slave ID (17)

This command returns the model, serial, and firmware number for an Acromag slave/server device (97xEN for this example), the status of the Run indicator, and any other information specific to the device. This command does not address Register Map registers and broadcast transmission is not supported (serial Modbus).

The command query simply sends the unit identifier and function code.

Modbus PDU Example - Report Slave ID Query

Field Name	Example Decimal (Hexadecimal)
Function Code	17 (11)

Modbus PDU Example - Report Slave ID Response (Acromag 97xEN)

FIELD	DESCRIPTION
Unit ID	Echo Unit ID Sent In Query
Function Code	11
Byte Count	42
Slave ID (Model No.)	08=972EN-4004 (4 Current Outputs) 09=972EN-4006 (6 Current Outputs) 0A=973EN-4004 (4 Voltage Outputs) 0B=973EN-4006 (6 Voltage Outputs)
Run Indicator Status	FFH (ON)
Firmware Number String (Additional Data Field)	41 43 52 4F 4D 41 47 2C 39 33 30 30 2D **31 32 37** 2C 39 **37 32** 45 4E 2D **34 30 30 36** 2C *30 31 32 33 34 35 41* 2C *30 31 32 33 34 35* (“ACROMAG,9300-**127**,9**72**EN-**4006**,*serial number&rev*,*six-byteMACID*”)

Supported Data Types (Acromag Modules)

All I/O values are accessed via 16-bit Input Registers or 16-bit Holding Registers (see Register Map). Input registers contain information that is read-only. For example, the current input value read from a channel, or the states of a group of digital inputs. Holding registers contain read/write information that may be configuration data or output data. For example, the high limit value of an alarm function operating at an input, or an output value for an output channel.

I/O values of Acromag modules take the following common forms of data to represent temperature, percentage, and discrete on/off, as required. This is not a Modbus standard and will vary between devices.

Supported Data Types (Acromag Modules)

Summary Of Data Types Used By Acromag 900MB/900EN Modules

Data Types	Description
Count Value	A 16-bit signed integer value representing an A/D count, a DAC count, time value, or frequency with a range of –32768 to +32767.
Count Value	A 16-bit unsigned integer value representing an A/D count, a DAC count, time value, or frequency with a range of 0 to 65535.
Percentage	A 16-bit signed integer value with resolution of 0.005%/lsb. ±20000 is used to represent ±100%. For example, -100%, 0% and +100% are represented by decimal values –20000, 0, and 20000, respectively. The full range is –163.84% (-32768 decimal) to +163.835% (+32767 decimal).
Temperature	A 16-bit signed integer value with resolution of 0.1°C/lsb. For example, a value of 12059 is equivalent to 1205.9°C, a value of –187 equals –18.7°C. The maximum possible temperature range is –3276.8°C to +3276.7°C.
Discrete	A discrete value is generally indicated by a single bit of a 16-bit word. The bit number/position typically corresponds to the discrete channel number for this model. Unless otherwise defined for outputs, a 1 bit means the corresponding output is closed or ON, a 0 bit means the output is open or OFF. For inputs, a value of 1 means the input is in its high state (usually >> 0V), while a value of 0 specifies the input is in its low state (near 0V).

With Modbus TCP/IP, error checking of the data is handled by the underlying TCP protocol and traditional serial Modbus error checking will not be reviewed here.

Modbus Exceptions

Recall that a server may generate an exception response to a client request and this is normally flagged by returning the original function code plus 80H (the original code with its most significant bit set). Additionally, it may also return an exception code in the data field of the response that can be used to trouble-shoot the problem.

For example, if a client requests an unsupported service (specifies an invalid function code), then the server may return exception code 01 (Illegal Function) in the data field of the response message. Likewise, if a holding register is written with an invalid value, then exception code 03 (Illegal Data Value) will be returned in the data field of the response message. The following table gives some common error codes:

Modbus Exceptions

Modbus Exception Codes

Code	Exception	Description
01	Illegal Function	The function code received in the query is not allowed or invalid.
02	Illegal Data Address	The data address received in the query is not an allowable address for the slave or is invalid.
03	Illegal Data Value	A value contained in the query data field is not an allowable value for the slave or is invalid.
04	Slave/Server Device Failure	The server failed during execution. An unrecoverable error occurred while the slave/server was attempting to perform the requested action.
05	Acknowledge	The slave/server has accepted the request and is processing it, but a long duration of time is required to do so. This response is returned to prevent a timeout error from occurring in the master.
06	Slave/Server Device Busy	The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free.
07	Negative Acknowledge	The slave cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 (codes not supported by this model). The master should request diagnostic information from the slave.
08	Memory Parity Error	The slave attempted to read extended memory, but detected a parity error in memory. The master can retry the request, but service may be required at the slave device.
0A	Gateway Problem	Gateway path(s) not available.
0B	Gateway Problem	The target device failed to respond (the gateway generates this exception).
FF	Extended Exception Response	The exception response PDU contains extended exception information. A subsequent 2 byte length field indicates the size in bytes of this function-code specific exception information.

In a normal response, the slave simply echoes the function code of the original query in the function field of the response. All function codes have their most-significant bit (msb) set to 0 (their values are below 80H). In an exception response, the slave sets the msb of the function code to 1 in the returned response (i.e. exactly 80H higher than normal) and returns the exception code in the data field. This is used by the client/master application to actually recognize an exception response and to direct an examination of the data field for the applicable exception code.

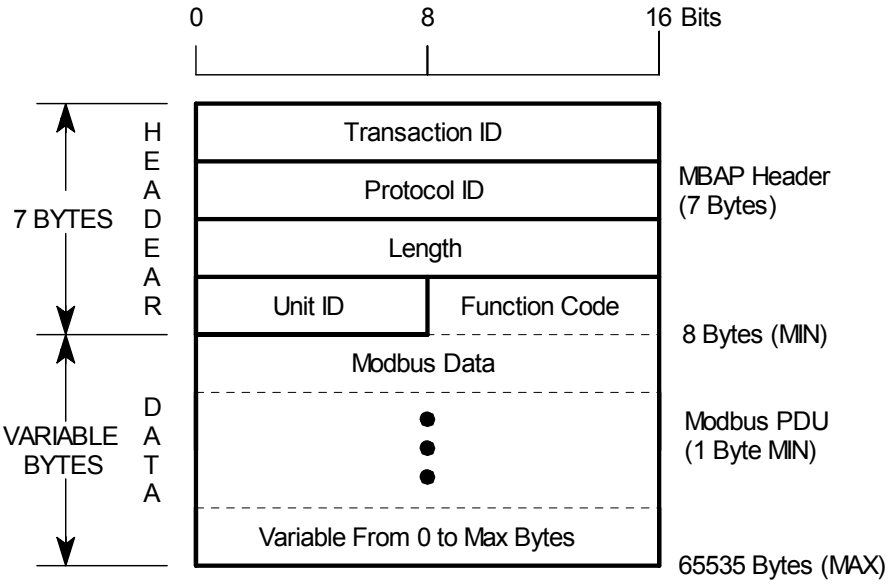
TCP is a data-stream based protocol, it may send almost any length IP packet it chooses, and it can parse this message as required. For example, it may encapsulate two back-to-back encapsulation messages in a single TCP/IP/MAC packet, or it may divide an encapsulation message across two separate TCP/IP/MAC packets.

Modbus TCP/IP ADU Format

In the introduction, we talked about how a traditional Modbus message (Modbus Application Data Unit) was stripped of its checksum and device address field, then combined with an MBAP header (ModBus Application Protocol), to build a Modbus TCP/IP Application Data Unit. This information is then nested into the data/payload field of a standard TCP frame, the total of which is then nested into the IP frame, which is then nested into the Ethernet/MAC frame for transmission over Ethernet. This nesting is the message *encapsulation* process that is commonly referred to. The following sections will attempt to describe the encapsulation that occurs at each layer as we move down the stack to the connection media, starting from the application layer, Modbus.

We know that the application layer is said to ride on top of TCP. Prior to passing the application message via TCP, a Modbus TCP/IP Application Data Unit is formed from a 7-byte Modbus Application Protocol (MBAP) header and the Protocol Data Unit (Modbus function code and data). This packet takes the following form:

Modbus TCP/IP Application Data Unit (ADU)



Modbus TCP/IP ADU Format

The 7-byte MBAP header includes the following fields:

- **Transaction/Invocation Identifier (2 Bytes):** This identification field is used for transaction pairing when several Modbus transactions are sent along the same TCP connection without waiting for completion of the prior transaction.
- **Protocol Identifier (2 bytes):** This field is always 0 for Modbus services and other values are reserved for future extensions.
- **Length (2 bytes):** This field is a byte count of the remaining fields and includes the destination identification and data fields.
- **Unit Identifier (1 byte):** This field is used to identify a remote server located on a non TCP/IP network (for bridging Ethernet to a serial sub-network). In a typical slave application, the unit ID is ignored and just echoed back in the response. It is recommended that a unit ID of FF be used to keep this value insignificant to a serial bridge or gateway (see below).

The Protocol Data Unit (PDU) is the Modbus function code and data field in their original form. The original Modbus error checking field (checksum) is not used, as the standard ethernet TCP/IP link layer checksum methods are instead used to guaranty data integrity. Further, the original Modbus device address field is supplanted by the *unit identifier* in Modbus TCP/IP and becomes part of the Modbus Application Protocol (MBAP) header. The original device address is not needed because Ethernet devices already contain their own unique MAC addresses. However, it is used if a serial bridge or gateway is being used to bridge Ethernet to a serial sub-network of Modbus devices.

With traditional serial Modbus, a client can only send one request at a time and must wait for an answer before sending a second request. However, Modbus TCP/IP devices may send several requests to the same server without waiting for the prior response. In this instance, the transaction identifier is use to match a future response with its originating request and must be unique per transaction. It is commonly a TCP sequence number driven by a counter that is incremented by each request. The maximum number of client transactions will vary from device to device, but is generally a number from 1 to 16. Likewise the maximum number of server transactions also varies. For Acromag 9xxEN-4 modules, this number is 10.

The unit identifier field was intended to facilitate communication between Modbus TCP/IP Ethernet devices and traditional Modbus serial devices by using a bridge or gateway to route traffic from one network to a serial line sub-network. In this case, the destination IP address identifies the bridge or gateway device to send the message to, while the bridge device itself uses the Modbus Unit Identifier to forward the request to the specific slave device of the sub network. Recall that serial Modbus uses addresses 1 to 247 decimal, and reserves 0 as a broadcast address. Thus, the unit identifier assumes the same assignment for these applications. Further, this is the only way that broadcast messages are supported with Modbus TCP/IP, as TCP alone only sends unicast (point-to-point) messages.

Modbus TCP/IP Application Data Unit

With TCP/IP devices, a Modbus server is addressed using its IP address, rendering the unit identifier non-functional and FFH is used in its place. The hex address FF remains non-significant to a gateway or bridge and will continue to be ignored if the network is later expanded or augmented with serial bridge or gateway devices.

The Modbus TCP/IP ADU is then inserted into the data field of a standard TCP frame and sent via TCP on well-known system port 502, which is specifically reserved for Modbus applications. Thus, this packet is encapsulated by the data frames imposed by the TCP/IP stack of protocols (TCP/IP/MAC) before being transmitted onto the network. The term encapsulation refers to the action of packing (embedding) this message into the TCP container, the IP container, and the MAC container. This lower level encapsulation is illustrated as follows:

TCP/IP/MAC Encapsulation (Explicit Message)

Ethernet Header (14 Bytes)	IP Header (20 Bytes)	TCP Header (20 Bytes)	Encapsulation Message	CRC

Connection Manager

Because TCP is a connection-oriented protocol, a TCP connection must first be established before a message can be sent via Modbus TCP/IP. Following the client-server principle, this connection is established by the client (master). This connection can be handled explicitly by the client user-application software, or automatically by the client TCP connection manager. More commonly, this is handled automatically by the client protocol software via the TCP socket interface and this operation remains transparent to the application.

All Modbus TCP/IP message connections are point-to-point communication paths between two devices, which require a source address, a destination address, and a connection ID in each direction. Thus Modbus TCP/IP communication is restricted to unicast messages only.

Well-known port 502 has been specifically reserved for Modbus applications. A Modbus server will listen for communication on port 502. When a Modbus client wants to send a message to a remote Modbus server, it opens a connection with remote port 502. As soon as a connection is established, the same connection can be used to transfer user data in either direction between a client and server. A client and server may also establish several TCP/IP connections simultaneously. When a connection is established, all the transmissions that are part of that connection are associated with a Connection ID (CID). If this connection involves transmission in both directions, then two Connection ID's are assigned. The maximum number of connections allowed is dependent on the specifications of the particular TCP/IP interface (for Acromag 9xxEN modules, this number is 10). In the case of cyclic transmission between a client and server, a permanent connection can also be employed. If it is only necessary to transfer parameter or diagnostic information when a special event occurs, then this connection can be closed after each data transmission and reopened as needed.

Connection Manager

With regard to Modbus TCP/IP, the following is true:

- A TCP connection is established by the client. Servers cannot initiate TCP transactions. It is good practice to keep a TCP connection open with a remote server and not open and close it for each Modbus message. The client can close a connection as required, but can also process a request-for-close message from the server and then close the connection.
- Some Modbus devices may operate as both clients and servers. The Acromag 900EN-40xx modules are strictly servers.
- For devices that operate as both clients and servers, two-way communication is possible with separate connections opened for the client data flow and the server data flow.
- A Modbus client may have many simultaneous TCP connections open at any given time. It uses a local port to send its message (different than 502 and different for each connection), while the remote server receives this message on well-known port 502.
- The Modbus server always listens for messages on well-known port 502 (in some applications, it may also have other TCP ports for Modbus service). A Modbus client does not send messages on local port 502. Port 502 is a reserved listening/receiving port.
- A client can initiate several Modbus messages with a remote server without waiting for the end of a previous one. Thus, several Modbus messages can be sent on the same TCP connection (but not the same PDU). In this case, the Modbus transaction identifier (of the MBAP header) is used to match requests to corresponding responses.
- A TCP frame must transport only one Modbus Application Data Unit (ADU) at a time. Although it's possible, it is not recommended to send multiple Modbus requests on the same PDU.
- A Modbus client will open a minimum of connections with a remote Modbus server of the same IP address, usually one per application.
- A Modbus server may have multiple TCP connections to it at any time. It uses port 502 to receive these messages and the connection ID to match the message to the response. The maximum number of TCP connections will vary from device to device. For Acromag 9XXEN-40xx Modbus TCP/IP modules, this number is 10.

Of course, a process must also exist to establish connections between devices that are not connected yet. This is the purpose of the Unconnected Message Manager (UCMM), which serves to process these connection requests. Once the UCMM has established a connection, then all the connection resources needed between the two devices (including bridges/routers) are reserved for that connection.

TRANSPORT LAYER

The Transport Layer resides just below the Application Layer and is responsible for the transmission, reception, and error checking of the data. There are a number of Transport Layer protocols that may operate at this layer, but the primary one of interest for Modbus TCP/IP is the Transport Control Protocol (TCP).

TCP- Transport Control Protocol

The Transport Control Protocol (TCP) resides one layer above the Internet Protocol (IP) and is responsible for transporting the application data and making it secure, while IP is responsible for the actual addressing and delivery of the data. The TCP packet is inserted into the data portion of the IP packet below it. IP itself is an unsecured, connectionless protocol and must work together with the overlaying TCP in order to operate. In this way, TCP is generally considered the upper layer of the IP platform that serves to guaranty secure data transfer. The use of the label Modbus-TCP (versus Modbus TCP/IP) does not imply that IP is not used or not important.

If data is lost, it must be retransmitted. This type of data exchange refers to explicit messaging and is commonly used for exchanging information that is not time-critical, but still necessary. TCP uses explicit messaging and will work to ensure that a message is received, but not necessarily on time.

TCP is a connection-oriented protocol. TCP establishes a connection between two network stations for the duration of the data transmission. While establishing this connection, conditions such as the size of the data packets are specified (which apply to the entire connection session).

TCP also follows the Client-Server communication model. That is, whichever network station takes the initiative and establishes the connection is referred to as the *TCP Client*. The station to whom the connection is made is called the *TCP Server.* In Modbus TCP/IP, the communication is always controlled by the master (client) and the master/client will establish the connection. The server (slave) cannot initiate communication on its own, but just waits for the client (master) to make contact with it. The client then makes use of the service offered by the server (note that depending on the service, one server may accommodate several clients at one time).

TCP verifies the sent user data with a checksum and assigns a sequential number to each packet sent. The receiver of a TCP packet uses the checksum to verify having received the data correctly. Once the TCP server has correctly received the packet, it uses a predetermined algorithm to calculate an acknowledgement number from the sequential number. The acknowledgement number is returned to the client with the next packet it sends as an acknowledgement. The server also assigns a sequential number to the packet it sends, which is then subsequently acknowledged by the client with an acknowledgement number. This process helps to ensure that any loss of TCP packets will be noticed, and that if needed, they can then be re-sent in the correct sequence.

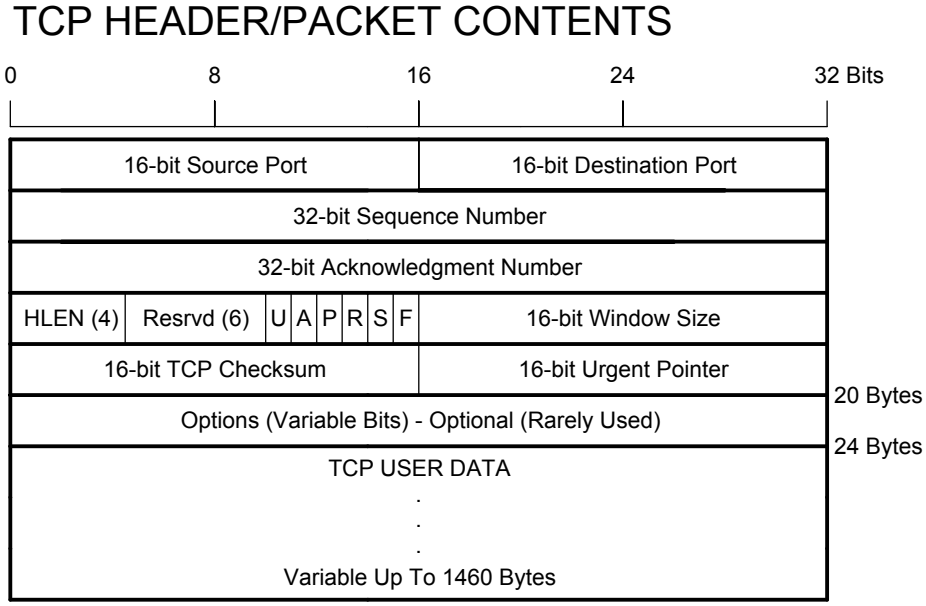
TCP also directs the user data on the destination computer to the correct application program by accessing various application services using various well-known port numbers. For example, Telnet can be reached through Port 23, FTP through port 21, and Modbus through port 502. In this way, the port number is analogous to the room number in a large office building—if you address a letter to the public relations office in room 312, you are indicating that you wish to utilize the services of the public relations office.

TCP- Transport Control Protocol

A *port* is the address that is used locally at the transport layer (on one node) and identifies the source and destination of the packet inside the same node. Port numbers are divided between well-known port numbers (0-1023), registered user port numbers (1024-49151), and private/dynamic port numbers (49152-65535). Ports allow TCP/IP to multiplex and demultiplex a sequence of IP datagrams that need to go to many different (simultaneous) application processes.

To reiterate, with TCP, the transmitter expects the receiver to acknowledge receipt of the data packets. Failure to acknowledge receipt of the packet will cause the transmitter to send the packet again, or the communication link to be broken. Because each packet is numbered, the receiver can also determine if a data packet is missing, or it can reorder packets not received into the correct order. If any data is detected as missing, all subsequent received data will be buffered. The data will then be passed up the protocol stack to the application, but only when it is complete and in the correct order. Of course this error checking mechanism of the connection-oriented TCP protocol takes time and will operate more slowly than a connection-less protocol like UDP (UDP is not used in Modbus TCP/IP). Thus, sending a message via TCP makes the most sense where continuous data streams or large quantities of data must be exchanged, or where a high degree of data integrity is required (with the emphasis being on *secure* data).

The following figure illustrates the construction of a TCP Packet:



TCP- Transport Control Protocol

TCP Header Field Definitions (Left-to-Right and Top-to-Bottom):

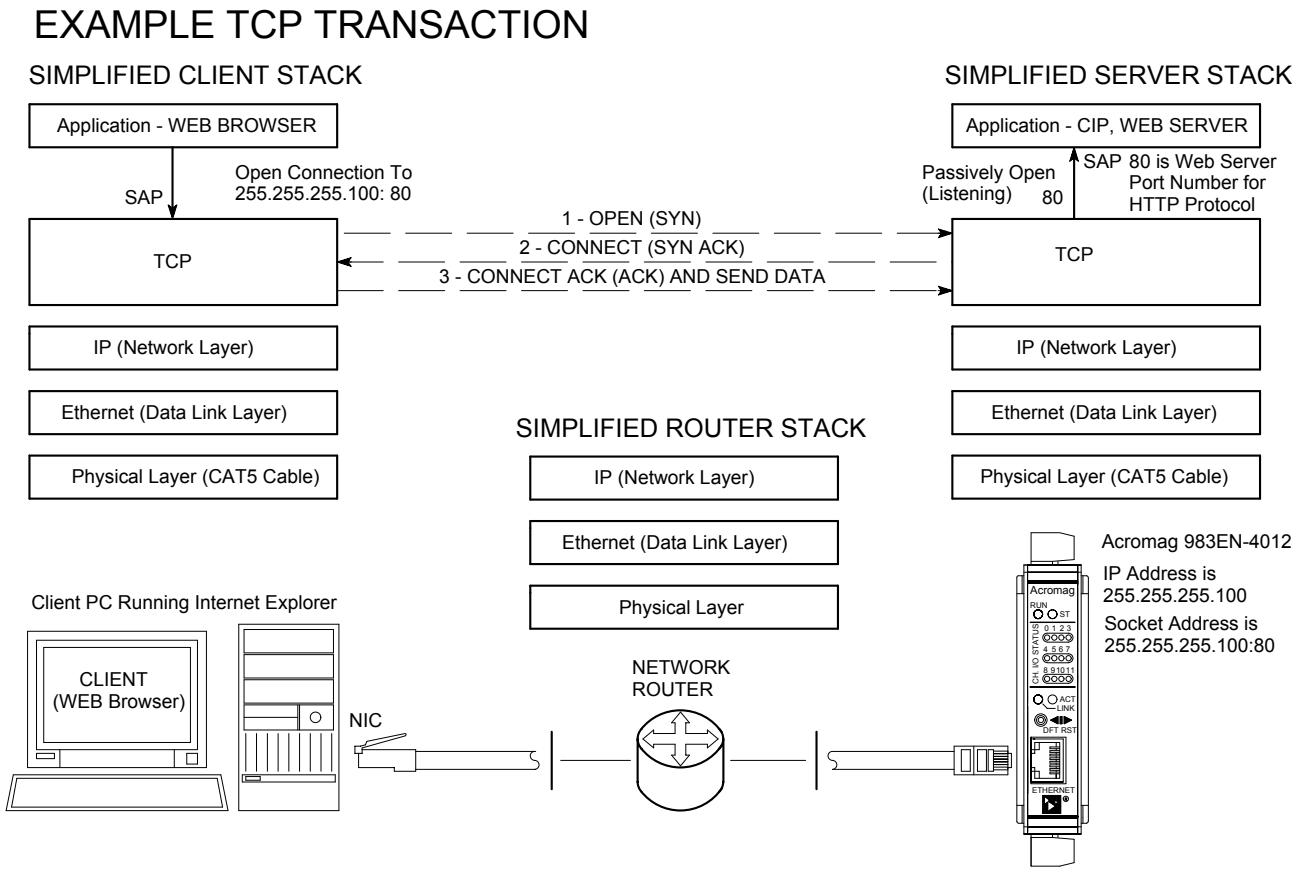
Source Port (SP) – Port of sender's application (the port the sender is waiting to listen for a response from the destination machine).
Destination Port (DP) – Port number of the receiver's application (the port of the remote machine the sent packet will be received at).
Sequence Number (SN) – Offset from the first data byte relative to the start of the TCP flow which is used to guaranty that a sequence is maintained when a large message requires more than one transmission.
Acknowledgment Number (AN) – This is the sequence number expected in the next TCP packet to be sent and works by acknowledging the sequence number as sent by the remote host. That is, the local host's AN is a reference to the remote machine's SN, and the local machine's SN is related to the remote machine's AN.
Header Length (HLEN) – A measure of the length of the header in increments of 32-bit sized words.
Reserved – These 6 bits are reserved for possible future use.
UARPSF Flags (URG, ACK, PSH, RST, SYN, FIN) – U=Urgent flag which specifies that the urgent point included in this packet is valid; A=Acknowledgement flag specifies that the portion of the header that has the acknowledgement number is valid; P=Push flag which tells the TCP/IP stack that this should be pushed up to the application layer program that needs it or requires it as soon as time allows; R=Reset flag used to reset the connection; S=Synthesis flag used to synchronize sequence numbers with acknowledgement numbers for both hosts (synthesis of the connection); F=Finish flag used to specify that a connection is finished according to the side that sent the packet with the F flag set.
Window Size (WS) – This indicates how many bytes may be received on the receiving side before being halted from sliding any further and receiving more bytes as a result of a packet at the beginning of the sliding window not having been acknowledged or received.
TCP Checksum (TCPCS) – This is a checksum that covers the header and data portion of a TCP packet to allow the receiving host to verify the integrity of an incoming TCP packet.
Urgent Pointer (UP) – This allows for a section of data as specified by the urgent pointer to be passed up by the receiving host quickly.
IP Options – These bits are optional and rarely used.
TCP User Data – This portion of the packet may contain any number of application layer protocols (CIP™, HTTP, SSH, FTP, Telnet, etc.).

TCP Example

The following simplified example illustrates a typical TCP transaction. In this example a network client (web browser) initiates data transfer with a web server (such as the web server embedded within the Acromag 9xxEN modules). The client is a PC running Internet Explorer and connected to the network via a Network Interface Card (NIC).

Earlier we talked about how ports are used to send and receive messages via TCP. For example, local port 502 was reserved for listening/receiving Modbus messages. Likewise, port 80 is another well-known port that is reserved for web applications. The Acromag 9xxEN-40xx modules actually use both well-known port 502 (Modbus) and port 80 (web server).

TCP Example



Note that a web browser always uses TCP for communication with a web server. The web browser (client application) starts by making a service request to TCP of its transport layer for the opening of a connection for reliable data transport. It uses the IP address of the remote server combined with the well-known port number 80 (HTTP Protocol) as its socket address. TCP opens the connection to its peer entity at the web server by initiating a three-way handshake. If this handshake can complete and the connection successfully open, then data can flow between the web browser (client) and the web server (Acromag module).

Once the connection is made, the web browser and remote server assume that a reliable open data pipe has formed between them and they begin transporting their data in sequence, and without errors, as long as TCP does not close the connection. TCP will monitor the transaction for missing packets and retransmit them as necessary to ensure reliability.

Note that in the figure above, an observer in the data paths at either side of the router would actually see the beginning of the message from the client to the web server begin only in the third data frame exchanged (the client's request message is combined with the connection acknowledge of the third exchange).

NETWORK LAYER

The Network Layer or Internet Layer resides just below the Transport Layer and is responsible for routing the packets to the network. Although there are many network layer protocols such as ICMP, IGMP, and others, the most important of these for our purpose are IP, ARP, and RARP.

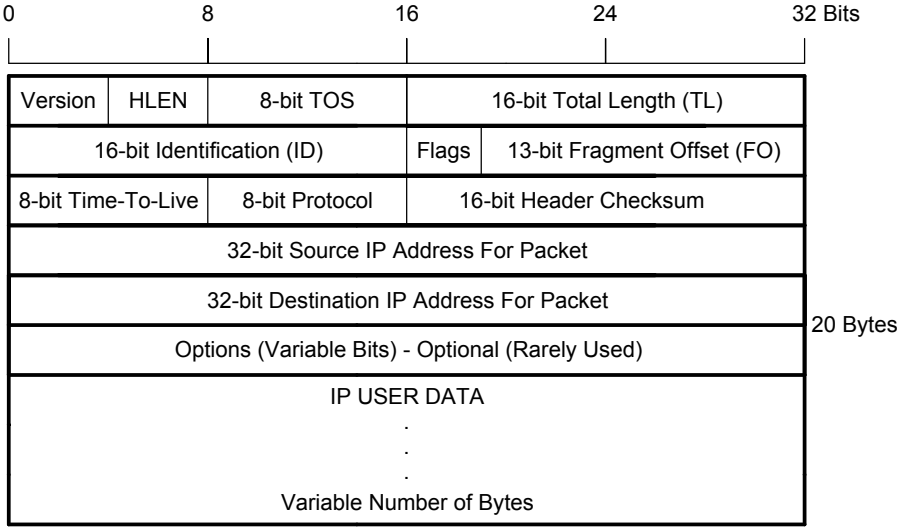
IP – Internet Protocol

Although Modbus TCP/IP are named together, they are really complimentary protocols. The Internet Protocol (IP) manages the actual addressing and delivery of the data packets. IP provides a connectionless and unacknowledged method for sending data packets between two devices on a network. IP does not guaranty delivery of the data packet, it relies on a transport layer protocol (like TCP) or application layer protocol (like Modbus) to do that. IP also makes it possible to assemble an indefinite number of individual networks into a larger overall network, without regard to physical implementation of the sub networks. That is, the data is sent from one network station to another, transparent to these differences.

An IP packet is a chunk of data transferred over the Internet using standard Internet Protocol (IP). Each packet begins with a header containing addressing and system control information. Unlike uniform ATM "cells", IP packets vary in length depending on the data being transmitted.

The following illustrates the contents of the IP header. The first 5 rows are commonly used (20 Bytes), while the 6th (or more) rows will depend on how many 32-bit option words are required by special options. The data is the encapsulated packet of the upper Transport Layer, a TCP or UDP packet (Modbus TCP/IP does not utilize UDP).

IP HEADER/PACKET CONTENTS



IP – Internet Protocol

IP Header Field Definitions (Left-to-Right and Top-to-Bottom):

Version – A 4-bit field that specifies what version of the Internet Protocol is being used (currently IP version 4). IP version 6 has many advantages over IP version 4, but is not in widespread use yet.

Header Length (HLEN) – A 4-bit number that specifies the increments of 32-bit words that tell the machine decoding the IP packet where the IP header is supposed to end (this dictates the beginning of the data). For example, "0101" (5) would specify an IP packet having only the first 5 rows as header information and its data thus beginning with the 6th row.

Type-of-Service (TOS) – Used for special IP packet information that may be accessed by routers passing along the packet, or by receiving interfaces. The first 3 bits are reserved, the fourth bit is set to 0, and the remaining 4 bits are used to flag the following (respectively): minimize delay for this packet, maximize throughput for this packet, maximize reliability for this packet, and minimize monetary costs. Many application layer protocols have recommended values for each of these bits based on the kind of service they are using. For example, NNTP (Net News Transfer Protocol) is not a very time critical operation. NNTP is used for USENET posts, and group synchronization between servers, or to a client from a server. If it happens to take a long time to transfer all this data, that's OK. Since it's not time sensitive, the "minimize monetary costs" bit may be set for a server synchronizing itself with another server under these conditions. Thus, it is left to the router to determine the paths which are the cheapest and then route a packet based on the flags that are set. If a router has only two routes (one to/from the internet, a second to/from a Local Area Network), then these 4 bits are often ignored since there are not multiple routes to its destination. These bits may be useful where a router may have four routes to a distant network, each route using a medium that has specific costs related to bandwidth, reliability, or latency (fiber, satellite, LAN line, or VPN for example). With each of these links up through a router, an incoming packet may be routed via any of these paths, but a properly configured router may be able to take advantage of how these packet bits are set by the sender in determining which route to take. These bits are sometimes known as the Differentiated Services Code Point (DSCP) which defines one of a set of classes of service. This value is usually set to 0, but may be used to indicate a particular Quality of Service request from the network.

Total Length (TL) – The total length of the IP packet in 8-bit (byte) increments. By subtracting header length (HL) from total length (TL), you can determine how many bytes long the data portion of the IP packet is. As a 16-bit value, valid ranges would be from 20 (minimum size of IP header) to 65535 bytes. A TL of only 20 is unlikely (no data), but could happen if something was broken. Very large IP packets (greater than 1500 bytes) are also uncommon, since they must typically be fragmented onto some networks.

Identification (ID) – A 16-bit number used to distinguish one sent IP packet from another by having each IP packet sent increment the ID by 1 over the previous IP packet sent.

Flags – A sequence of 3 fragmentation flags used to control where routers are allowed to fragment a packet (via Don't Fragment flag) and to indicate the parts of a packet to the receiver: 001=More, 010=Don't Fragment, 100= Unused.

Fragmentation Offset (FO) – A byte count from the start of the original packet sent and set by any router that performs IP router fragmentation.
Time-to-Live (TTL) – An 8-bit value that is used to limit the number of routers through which a packet may travel before reaching its destination (the number of hops/links which the packet may be routed over). This number is decremented by 1 by most routers and is used to prevent accidental routing loops. If the TTL drops to zero, the packet is discarded by either the server that has last decremented it, or the next server that receives it.
Protocol – An 8-bit value that is used to allow the networking layer to know what kind of transport layer protocol is in the data segment of the IP packet. For example, 1=ICMP, 2=IGMP, 6=TCP, 17=UDP.
Header Checksum – A 16-bit checksum (1's complement value) for the header data that allows a packet to offer verification data to help ensure that the IP header is valid. This checksum is originally inserted by the sender and then updated whenever the packet header is modified by a router. This is used to detect processing errors on the part of the router or bridge where the packet is not already protected by a link layer Cyclic Redundancy Check (CRC). Packets with an invalid checksum are discarded by all nodes in an IP network.
Source IP Address (32 bits) – The IP address of the source machine sending the data onto the network. This address is commonly represented by 4 octets representing decimal values and separated by periods (255.255.255.10 for example).
Destination IP Address (32 bits) – The IP address of the destination machine to which the packet is being routed for delivery. This address is commonly represented by 4 octets representing decimal values and separated by periods (255.255.255.10 for example).
Options (Variable Number of Bits/Words) – These bits are reserved for special features and are rarely used, but when they are used, the IP header length will be greater than 5 (five 32-bit words) to indicate the relative size of the option field.
IP Data (Variable Number of Bits/Words) - This portion of the packet may contain any number of nested protocols (TCP, UDP, ICMP, etc.).

IP – Internet Protocol

A Network's Infrastructure includes the physical hardware used to transmit data electronically such as routers, switches, gateways, bridges, and hubs.

A router is located at the gateway where it directs the flow of network traffic and determines the route of packets as they travel from one network to another network(s). A router can be either a hardware device or a software application.

The **Ethernet Address** or **MAC Address** refers to the Media Access Control Address that uniquely identifies the hardware of any network device. This is a unique, 48-bit, fixed address assigned and hard-coded into an Ethernet device at the factory. This is usually expressed in hexadecimal form as 12 hex characters (6 bytes), with the first 3 bytes (6 leftmost hex characters) representing the device manufacturer, and the last 3 bytes (6 rightmost hex characters) uniquely assigned by the manufacturer. All six bytes taken together uniquely identify the network device.

Do not confuse the Ethernet Address (MAC address) with the Internet Protocol (IP) Address, which is a 32-bit number assigned to your computer (see below) that can change each time you connect to a network.

Ethernet (MAC) Address

TIP: If you want to determine the Ethernet address of the NIC card installed in your PC, at the DOS command prompt, type WINIPCFG <Enter> (Windows 98), or IPCONFIG /ALL <Enter> (Windows XP).

IP addresses are 32-bit numbers that are administered by an independent authority (InterNIC) and are unique for any device on the network. The IP address is a 32-bit value made up of four octets (8 bits), with each octet having a value between 0-255 (00H-FFH). It is commonly expressed as four decimal numbers (8-bit values) separated by a decimal point. This provides about 4.3 billion possible combinations.

Internet (IP) Address

Internet (IP) Address

Large networks of corporations, communications companies, and research institutions will obtain large blocks of IP addresses, then divide them into subnetworks within their own organization and distribute these addresses as they see fit. The smaller networks of universities and companies will acquire smaller blocks of IP addresses to distribute among their users. Because these numbers are ultimately assigned by the Internet Assigned Number's Authority, an IP address can be used to approximate a machine's location.

Similar to the Ethernet Address, the IP address is comprised of two parts: the network address or Net ID (first part), and the host address or Host ID (last part). This last part refers to a specific machine on the given sub-network identified by the first part. The number of octets of the four total that belong to the network address depend on the Class definition (Class A, B, or C) and this refers to the *size* of the network.

*A **Subnet Mask** is used to subdivide the host portion of the IP address into two or more subnets. The subnet mask will flag the bits of the IP address that belong to the network address, and the remaining bits correspond to the host portion of the address. The unique subnet to which an IP address refers to is recovered by performing a bitwise AND operation between the IP address and the mask itself, with the result being the sub-network address.*

A *Subnet* is a contiguous string of IP addresses. The first IP address in a subnet is used to identify the subnet and usually addresses the server for the subnet. The last IP address in a subnet is always used as a broadcast address and anything sent to the last IP address of a subnet is sent to every host on that subnet.

Subnets are further broken down into three size classes based on the 4 octets that make up the IP address. A Class A subnet is any subnet that shares the first octet of the IP address. The remaining 3 octets of a Class A subnet will define up to 16,777,214 possible IP addresses ($2^{24} – 2$). A Class B subnet shares the first two octets of an IP address (providing $2^{16} – 2$, or 65534 possible IP addresses). Class C subnets share the first 3 octets of an IP address, giving 254 possible IP addresses. Recall that the first and last IP addresses are always used as a network number and broadcast address respectively, and this is why we subtract 2 from the total possible unique addresses that are defined via the remaining octet(s).

A *Subnet Mask* is used to determine which subnet an IP address belongs to. The use of a subnet mask allows the network administrator to further divide the host part of this address into two or more subnets. The subnet mask flags the network address part of the IP address, plus the bits of the host part, that are used for identifying the sub-network. By mask convention, the bits of the mask that correspond to the sub-network address are all set to 1's (it would also work if the bits were set exactly as in the network address). It's called a mask because it can be used to identify the unique subnet to which an IP address belongs to by performing a bitwise AND operation between the mask itself, and the IP address, with the result being the sub-network address, and the remaining bits the host or node address.

For our example, the default IP address of this module is 128.1.1.100. If we assume that this is a Class C network address (based on the default Class C subnet mask of 255.255.255.0), then the first three numbers represent this Class C network at address 128.1.1.0, the last number identifies a unique host/node on this network (node 100) at address 128.1.1.100.

Internet (IP) Address

For our example, if we wish to further divide this network into 14 subnets, then the first 4 bits of the host address will be required to identify the subnetwork (0110), then we would use "11111111.11111111.11111111. 11110000" as our subnet mask. This would effectively subdivide our Class C network into 14 subnetworks of up to 14 possible nodes each.

Note that the first node address (0) is typically reserved for the network server and should not be used. The last node (255) is a broadcast address. Use of these node addresses for any other purpose may yield poor performance.

With respect to the default settings of Acromag 9xxEN modules:

Subnet Mask 255.255.255.0 (11111111.11111111.11111111.00000000)
IP Address: 128.1.1.100 (10000000.00000001.00000001.01100100)
Subnet Address: 128.1.1.0 (1000000.00000001.00000001.00000000)

Subnetwork address 128.1.1.0 has 254 possible unique node addresses. We are using node 100 of 254 possible for our module.

At this point, we see that each layer (application, transport, network, and data link layer) uses its own address method. The application layer uses socket numbers, which combine the IP address with the port number. The transport layer uses port numbers to differentiate simultaneous applications. The network layer uses the IP address, and the Data Link layer uses the MAC address.

ARP –Address Resolution Protocol

The Address Resolution Protocol (ARP) is a TCP/IP function that resides at the network layer (layer 3) with the Internet Protocol (IP), and its function is to map Ethernet addresses (the MAC ID) to IP addresses, and maintain a mapping table within the network device itself. This protocol allows a sending station to gather address information used to form a layer 2 frame complete with the IP address and hardware (MAC) address. Every TCP/IP-based device contains an ARP Table (or ARP cache) that is referred to by a router when it is looking up the hardware address of a device for which it knows the IP address and needs to forward a datagram to. If this device wants to transmit an IP packet to another device, it first attempts to look-up the Ethernet address of that device in its ARP table. If it finds a match, it will pass the IP packet and Ethernet address to the Ethernet driver (physical layer). If no hardware address is found in the ARP table, then an ARP broadcast is sent onto the network. The ARP protocol will query the network via a local broadcast message to ask the device with the corresponding IP address to return its Ethernet address. This broadcast is read by every connected station, including the destination station. The destination station sends back an ARP reply with its hardware address attached so that the IP datagram can now be forwarded to it by the router. The hardware address of the ARP response is then placed in an internal table and used for subsequent communication.

The ARP maps TCP/IP addresses to physical MAC addresses.

Even though ARP is a layer 3 protocol, it does not use an IP header and has its own packet format that it broadcasts on the local LAN within the data field of a layer 2 frame, without needing to be routed (the Ethernet Type field of the layer 2 frame has the value 0806H in it to indicate an ARP request).

It is important to note that when we used the term "local broadcast message" above, that Ethernet broadcast messages will pass through hubs, switches, and bridges, but will not pass through routers. As such, broadcast messages are confined to the subnet on which they originate and will not propagate out onto the worldwide web.

ARP is used to search for another station's MAC address knowing only its IP address.

RARP is used to search for a local station's IP address knowing only its MAC address.

ARP –Address Resolution Protocol

To make sure that the broadcast message is recognized by all connected network stations, the IP driver uses “FF FF FF FF FF FF” as the Ethernet address. The station that recognizes its own IP address in the ARP request will confirm this with an ARP reply. The ARP reply is a data packet addressed to the ARP request sender with an ARP identifier indicated in the protocol field of the IP header.

The IP driver then extracts the Ethernet address obtained from the ARP reply and enters it into the ARP table. Normally, these dynamic entries do not remain in the ARP table and are aged out, if the network station is not subsequently contacted within a few minutes (typically 2 minutes under Windows).

The ARP table may also support static address entries, which are fixed addresses manually written into the ARP table and not subject to aging. Static entries are sometimes used for passing the desired IP address to new network devices which do not yet have an IP address.

Recall that ARP allowed a station to recover the destination hardware (MAC) address when it knows the IP address. The Reverse Address Resolution Protocol (RARP) is a complimentary protocol used to resolve an IP address from a given hardware address (such as an Ethernet address). A station will use RARP to determine its own IP address (it already knows its MAC address).

RARP – Reverse Address Resolution Protocol

RARP is the complement of ARP and is used to translate a hardware interface address to its protocol (IP) address, while ARP translates a protocol address to a hardware (MAC) address.

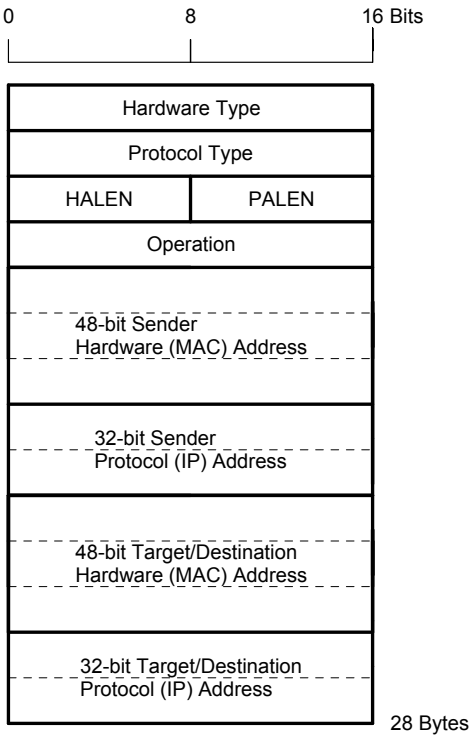
Like ARP, RARP is a layer 3 protocol that does not use an IP header and has its own packet format that it broadcasts on the local LAN within the data field of a layer 2 frame, without needing to be routed (the Ethernet Type field of the layer 2 frame has the value 0835H in it to indicate an RARP request).

RARP essentially allows a node in a local area network to request its IP address from a gateway server’s ARP table. The ARP table normally resides in the LAN’s gateway router and maps physical machine addresses (MAC addresses) to their corresponding Internet Protocol (IP) addresses. When a new machine is added to a LAN, its RARP client program requests its IP address from the RARP server on the router. Assuming that an entry has been set up in the router table, the RARP server will return the IP address to the machine which will then store it for future use.

ARP and its variant RARP are needed because IP uses logical host addresses (the IP address), while media access control protocols (Ethernet, Token-Ring, FDDI, etc.) need MAC addresses. The IP addresses are assigned by network managers to IP hosts and this is usually accomplished by configuration file options and driver software. That is why these are sometimes referred to as software addresses. LAN topologies cannot use these software addresses and they require that the IP addresses be mapped to their corresponding MAC addresses.

For example, a diskless workstation cannot read its own IP address from configuration files. They will send an RARP request (or BOOTP request) to a RARP server (or BOOTP server). The RARP server will find the corresponding IP address in its configuration files using the requesting station’s MAC address as a lookup, and then send this IP address back in a RARP reply packet.

ARP/RARP HEADER STRUCTURE



RARP – Reverse Address Resolution Protocol

ARP/RARP Header Field Definitions (Left-to-Right, Top-to-Bottom):

Hardware Type – Specifies a hardware interface type from which the sender requires a response. This is "1" for Ethernet.
Protocol Type – This is the protocol type used at the network layer and is used to specify the type of high-level protocol address the sender has supplied.
Hardware Address Length (HALEN) – This is the hardware address length in bytes which is "6" for an Ethernet (MAC Address).
Protocol Address Length (PALEN) – This is the protocol address length in bytes which is "4" for a TCP/IP (IP Address).
Operation Code – This code is commonly used to indicate whether the packet is an ARP request (1), or an ARP response (2). Valid codes are as follows: 1 = ARP Request; 2 = ARP Response; 3 = RARP Request; 4 = RARP Response; 5 = Dynamic RARP Request; 6 = Dynamic RARP Reply; 7 = Dynamic RARP Error; 8 = InARP Request; 9 = InARP Reply.
Sender Hardware (MAC) Address – This is the 48-bit hardware (MAC) address of the source node.
Sender Protocol (Software) Address – This is the 32-bit senders protocol address (the layer 3/network layer address—the IP address).
Target/Destination Hardware (MAC) Address – Used in an RARP request. The RARP request response carries both the target hardware address (MAC address) and layer 3 address (IP address).
Target/Destination Protocol (Software) Address – Used in an ARP request. The ARP response carries both the target hardware address (MAC address) and layer 3 address (IP address).

RARP – Reverse Address Resolution Protocol

Most network stations will send out a gratuitous ARP request when they are initializing their own IP stack. This is really an ARP request for their own IP address and is used to check for a duplicate IP address. If there is a duplicate IP address, then the stack does not complete its initialization.

DATA LINK (MAC) LAYER

The Data Link Layer or Host-to-Network Layer provides the protocol for connecting the host to the physical network. Specifically, this layer interfaces the TCP/IP protocol stack to the physical network for transmitting IP packets.

Recall from Figure 1 of the TCP/IP Stack section, how the various protocols at each of the different layers are encapsulated (nested) into the data frame of the next lowest layer. That is, packets generally carry other packet types inside them and their function is to often contain or encapsulate other packets. In this section, we will look at the lowest encapsulation layer (often referred to as the data link layer or the MAC layer) where Ethernet resides.

Note that bits are transmitted serially with the least significant bit of each byte transmitted first at the physical layer. However, when the frame is stored on most computers, the bits are ordered with the least significant bit of each byte stored in the rightmost position (bits are generally transmitted right-to-left within the octets, and the octets are transmitted left-to-right).

CSMA/CD

The data link layer also uses the CSMA/CD protocol (Carrier Sense Multiple Access w/ Collision Detection) to arbitrate access to the shared Ethernet medium.

Recall that with CSMA/CD, any network device can try to send a data frame at any time, but each device will first try to sense whether the line is idle and available for use. If the line is available, the device will begin to transmit its first frame. If another device also tries to send a frame at approximately the same time (perhaps because of cable signaling delay), then a collision occurs and both frames are subsequently discarded. Each device then waits a random amount of time and retries its transmission until it is successfully sent.

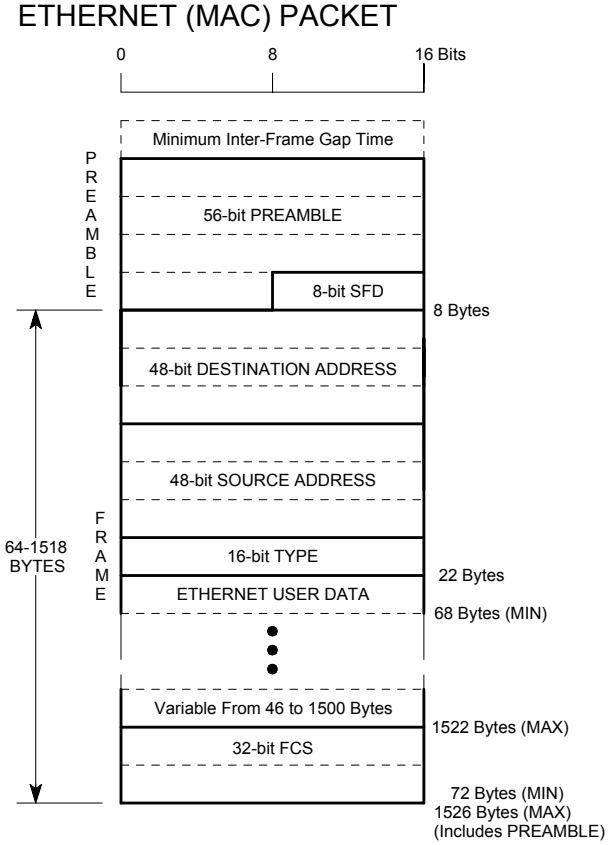
With switched Ethernet, a deliberate effort has been made to suppress CSMA/CD in order to increase determinism. This is done by using Ethernet switches to interconnect devices, connecting only one device per switch port. With only one device per switch port, there is no chance of collisions and devices will communicate full-duplex, at effectively double the base data rate.

Medium Access Control (MAC) Protocol

The Medium Access Control (MAC) protocol provides the services required to form the data link layer of Ethernet. This protocol encapsulates its data by adding a 14-byte header containing the protocol control information before the data, and appending a 4 byte CRC value after the data. The entire frame is preceded by a short idle period (the minimum inter-frame gap), and an 8-byte preamble.

Ethernet (MAC) Packet

The figure at right shows the construction of the Ethernet packet along with its preamble (via the MAC protocol). Note that the frame preamble is normally preceded by a short idle period that corresponds to a minimum inter-frame gap of 9.6 microseconds (at 10Mbps). This idle time before transmission is to allow the receiver electronics at each station to settle after completion of the prior frame.



Ethernet (MAC) Frame Definitions (Left-to-Right & Top-to-Bottom):

Preamble & SFD (56-bits+8-bits SFD) – Technically, the preamble is not part of the Ethernet frame, but is used to synchronize signals between stations. It is comprised of a pattern of 62 alternating 1's & 0's, followed by two set bits "11" (the last 8-bits of this pattern are actually referred to as the Start-of-Frame Delimiter or SFD byte). This 8-byte preamble/SFD is also preceded by a small idle period that corresponds to the minimum inter-frame gap period of 9.6us (at 10Mbps). After transmitting a frame, the transmitter must wait for this period to allow the signal to propagate through the receiver of its destination. The preamble allows the receiver at each station to lock into the Digital Phase Lock Loop used to synchronize the receive data clock to the transmit data clock. When the first bit of the preamble arrives, the receiver may be in an arbitrary state (out of phase) with respect to its local clock. It corrects this phase during the preamble and may miss/gain a number of bits in the process. That is why the special pattern of two set bits (11) is used to mark the last two bits of the preamble. When it receives these two bits, it starts assembling the bits into bytes for processing by the MAC layer. When using Manchester encoding at 10Mbps, the 62 alternating 1/0 bits of the preamble will resemble a 5MHz square wave.

RARP – Reverse Address Resolution Protocol

Destination Address (48-bits) – This 6-byte address is the destination Ethernet address (MAC Address). It may address a single receiver node (unicast), a group of nodes (multicast), or all receiving nodes (broadcast).
Source Address (48-bits) – This 6-byte address is the sender's unique node address and is used by the network layer protocol to identify the sender and also to permit switches and bridges to perform address learning.
Type (16-bits) – This 2-byte field provides a Service Access Point (SAP) and is used to identify the type of network layer protocol being carried. The value 0800H would be used to indicate an IP network protocol, other values indicate other network layer protocols. For example, 0806H would indicate an ARP request, 0835H would indicate a RARP request. For IEEE 802.3 LLC (Logical Link Control), this field may alternately be used to indicate the length of the data portion of the packet.
CRC Cyclic Redundancy Check (32-bits) – The CRC is added at the end of a frame to support error detection for cases where line errors or transmission collisions result in a corrupted MAC frame. Any frame with an invalid CRC is discarded by a MAC receiver without further processing and the MAC protocol does not provide any other indication that a frame has been discarded due to an invalid CRC.

The Ethernet standard dictates a minimum frame size which requires at least 46 data bytes in a MAC frame. If a network layer tries to send less than 46 bytes of data, the MAC protocol adds the requisite number of 0 bytes (null padding characters) to satisfy this requirement. The maximum data size which may be carried in a MAC frame over Ethernet is 1500 bytes.

Any received frame less than 64 bytes is illegal and referred to as a "runt". Runts may result from a collision and a receiver will discard all runt frames.

Any received frame which does not contain an integral multiple of octets (bytes) is also illegal (misaligned frame), as the receiver cannot compute the CRC for the frame and these will also be discarded by the receiver.

Any received frame greater than the maximum frame size is referred to as a "giant" and these frames are also discarded by an Ethernet receiver.

If you made it this far and are still awake, you must be a glutton for punishment. Please refer to www.modbus.org for more information on Modbus to help satisfy your apetite.

Acronym Reference

ADU	Application Data Unit
ARP	Address resolution Protocol
CSMA/CD	Carrier Sense Multiple Access w/ Collision Detection
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
MAC	Medium Access Control or Media Access Control
MBAP	Modbus Application Protocol
PDU	Protocol Data Unit
PHY	Physical Interface
TCP	Transport Control Protocol or Transmission Control Protocol
UDP	Universal Datagram Protocol