Chapter 16  NETWORKS and PROTOCOLS

Definition of Protocol

A protocol is a specification for standardized packets of data. Moving packets of data is the method that allows networks to share information. Packets of information move through the protocol stack and then move across the transmission media.

Introduction

In their web-based documentation of Windows 2000, Microsoft details how TCP/IP came about:

“TCP/IP is an industry-standard suite of protocols designed for large internetworks spanning wide area network (WAN) links. TCP/IP was developed in 1969 by the U.S. Department of Defense Advanced Research Projects Agency (DARPA), the result of a resource-sharing experiment called ARPANET (Advanced Research Projects Agency Network). The purpose of TCP/IP was to provide high-speed communication network links. Since 1969, ARPANET has grown into a worldwide community of networks known as the Internet.”

Ethernet, TCP/IP and other networking principles are subjects too vast to be expanded here. It is important for the student to master the principles of networking, however, as they pertain to the factory floor. The use of addresses is discussed when setting up some devices. Other networking issues will need to be discussed since the move of Ethernet from the business side of the factory to the factory floor has been underway long enough for equipment to be available with Ethernet connectivity.

DeviceNet

DeviceNet is a relatively easy I/O network to install and configure. First will be an introduction to CanBus. Then the organization, ODVA, through which DeviceNet is governed, will be discussed. The software RSNetworx for Devicenet will explain the mapping of I/O to the processor. While this local network is on the wane today, it will be explored since it was a recent attempt by the user to address local area network problems in the industrial environment.

CanBus

CanBus or CAN.bus is a two-wire differential serial bus. It is designed to operate in noisy electrical environment. The CanBus system also guarantees a high degree of data integrity between components. It is also an open architecture which means that many companies are encouraged to design components and systems for its use. CanBus is capable of high-speed data transmission (up to 1 Mbits/s) in short distance applications. It can also operate at longer range but with lower speeds. CanBus is multi-master with a high fault tolerance and error detection capability.

CanBus was originally developed in Germany by Bosch and was designed to replace electrical sensor components in the automobile. The design allowed a smaller wiring harness than was the design standard prior to CanBus. This bus was also recognized as an ideal bus for the industrial market but the automotive market has remained as its primary focus.

CanBus is especially well adapted for working with intelligent devices in a system or sub-system. CanBus has become the standard for IVN or in-vehicle network applications. Applications
include power-train applications for automobiles as well as between the truck and trailer in truck applications. One author described CanBus in the following way: “Many American and European truck and bus manufacturers have implemented CAN-based IVNs, and more and more truck-based superstructure control systems (e.g. fire fighting equipment and concrete mixers) also use CAN as their embedded control network.”

The description continues: “CAN is often used as the embedded network to run functions such as the power-train, body electronics, super-structure control and trailer communications. Likewise, CAN is also used to run add-on sub-systems such as harvesters, cranes, winches, drums, etc. In cases where several CAN-based IVNs are used to run multiple functions, these IVNs may be interconnected via gateways. This keeps the systems separate to avoid interferences and disturbances that may be caused if all operations run simultaneously in one physical layer.”

An article in the October, 2003 issue of Control Engineering identifies a question concerning the use of CanBus or Device Net and its competitor, Ethernet. In the article, Colin MacDonald writes:

“In choosing the appropriate network bus to support, a designer should ask several questions: Will both CAN and Ethernet continue to be widely adopted? If so, how will they co-exist? And finally, how will the choice of buses affect the design of network processors?

**New Software to Learn**

When starting a new DeviceNet application, it is necessary to become familiar with a new software product from Allen-Bradley called RSNetWorx for DeviceNet. A DeviceNet Scanner Card will be installed in both a SLC 5/03 and in a CompactLogix processor. With CompactLogix, the newer RSLogix 5000 software programming package will be necessary. This may be the first exposure for students with this software package as well as RSNetWorx for DeviceNet.

Notice that in the above figure, the network screen for DeviceNet shows no devices on the graph portion of the screen. This shows a new network with no devices attached. Also notice the note in the description section in the message box. Copy protection was not installed in this instance on the software and the software will run in this instance in demo mode only which allows only six nodes to be attached.
In the figure above, the DeviceNet scanner is being added to the rack’s I/O configuration. Notice the maximum input and output word count for the SDN module listed under Advanced I/O Configuration. This is the maximum number of data words available for the scanner module to share with this SLC 5/03 PLC. The scanner is located in slot 1.

**CompactLogix Processor**

The figure above shows the method of adding the scanner card to the CompactLogix I/O configuration. Notice the type is 1769-SDN/B. Cards must be added to the I/O list in RSLogix 5000 and not read from the I/O as was possible in RSLogix 500. This scanner is also located in slot 1.
Use the following scan-list information for a scanner in slot 2 of both a SLC and Compact processor. Both input and output scan lists appear below as they would in the scan-list shown in RSNetworx for Device-Net. Devices are shown stacked from first to last in 16 bit word format. If a device uses only 1 bit for communication to a PLC, 8 data bits are reserved. Many times, the other bits are used to transmit other information. A manual for the device will define the use of all bits and how they are used.

<table>
<thead>
<tr>
<th>Device 2</th>
<th>Device 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 4</td>
<td>Device 3</td>
</tr>
<tr>
<td>Device 6</td>
<td>Device 5</td>
</tr>
</tbody>
</table>

Output Scan-List

<table>
<thead>
<tr>
<th>Device 1</th>
<th>Device 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 2</td>
<td>Device 4</td>
</tr>
</tbody>
</table>

Input Scan-List

View in Scan List of RSNetworx for DeviceNet

For the SLC 5/03, addresses follow 16 bit words. If the scanner were in slot 2, the addressing would proceed with word 0 of slot 2 assigned a control word status. Starting with word 1, the first device would occupy bits 0 to 7 and device 2 would occupy word 1, bits 8-15. Other words would proceed after the first. For example, if an input were assigned from bit 1 of device 1, the address would be I:2.1/1. For the CompactLogix processor, addresses follow 32 bit word length and two 16 bit words would occupy each 32-bit word. For instance, if the bit were from input device 1, input bit 1, the address would be Local:I:2.0/1. For device 2, input bit 0 would be Local:I:2.0/16.

**Flash for CompactLogix Processor**

Before the CompactLogix processor will work, the processor must have its flash memory loaded. This process must be done before the processor can be attached or programmed in ladder logic. Out-of-the-box processors will not allow an up-load or download without the flash memory task. With a change in version of the processor, the memory must also be re-flashed. The program shown above, Control Flash, must be run. Control Flash is accessible from the Allen-Bradley website and can be downloaded from the internet.
ASCII READ/WRITE BLOCKS and PROTOCOLS: Introduction

Computers use numeric codes to transfer information to a PLC and other control devices. Various sets of numeric codes have been implemented to transfer text information between computers. One of these codes is ASCII. ASCII stands for American Standard Code for Information Interchange. As can be seen from the table of ASCII characters in the ASCII Table later in the chapter, not all codes represent letters or numbers. Many codes represent actions, special keys or control characters.

The chapter explains the use of ASCII codes to transmit information between computers. One computer capable of transmitting and accepting ASCII coded data is the PLC. The SLC 5/03 as well as a number of other PLCs is capable of reading and writing ASCII string data through its 9 pin D-shell connector on the CPU front panel. Instructions for transfer of data are found in Allen-Bradley’s SLC 500 Instruction Set Reference Manual, Chapter 10.

Writing a program to communicate between computer controllers involves handling of ASCII codes or other numeric data in order to move information or request an action. Many of these programs use ASCII data to accomplish the task. A mixture of ASCII data and other numeric data is the norm for most tasks. A second data transmission type is RTU (Remote Terminal Unit) with a more dense data packet. Use of RTU will be discussed as well in the chapter.

Definition of the communication between two controllers is included in a protocol. Many protocols are very simple and require only a simple description of the query and any expected response.

Then a more complete protocol is explored. How one controller requests information from another device and how that device responds is the basis for an advanced protocol. Contents of various fields in the protocol are described so the protocol can function properly.

Configuring a PLC for ASCII Read/Write

Fig. 16-6  
Using the SLC Processor as ASCII Device

The figure above shows the Channel Configuration setting change that must take place prior to using the port as an ASCII port. Channel 0 must be changed from System to User.
Many devices use ASCII code to transmit information. Scales send weights from the scale computer to a batching computer. Bar code readers send bar code data to a sorting computer. RFID tag readers send tag information to and from the tag and communicate to a computer system controlling the process. PLCs have the ability to read and write ASCII strings of data from these devices and are used to gather data and control processes using the string data.

A new file type is needed to hold ASCII string information, the String file type. It is added to the Data File list as follows:

File, Data Files, Select Data File and from below, Create New:

The process of creating a new file may also be done by right clicking Data Files and then choosing Add New. The example below shows the adding of File 9 as a String file. Files do not necessarily need to be added sequentially although most applications tend to reserve File 9 as a String or ASCII file type.
Table of ASCII Characters

ASCII characters are transmitted sequentially a bit at a time through a serial data cable. Each bit is transmitted sequentially starting with the left-most bit. Two start bits and 1, 1.5, or 2 stops may be specified. The channel may be configured from RSLogix 500 using the project tree – channel configuration, channel 0, and then user. From this tab, choose the baud rate, parity, stop bits and data bits to be used. Typical choices are 9600 baud, no parity, 1 stop bit and 8 data bits. For protocol control, chose Control Line – no handshaking – and for Delete Mode – ignore. Echo and XON/XOFF are usually left unchecked. The important point about these settings is that they must match the settings with the other device.

Looking at a transmission on the oscilloscope may yield good information. Storage oscilloscopes are definitely superior to non-storage oscilloscopes for this job. Transmissions should be visible per character with start and stop bits present as well as data bits of the 7 or 8 bit character string. For instance, the character “:” from the ASCII Table is 00111010 binary. With two start bits and one stop bit, it would resemble the following on the oscilloscope:

Fig. 16-8 Oscilloscope Representation of ASCII Character

Transmission Modes and ASCII Tables

The advanced protocol discussed later in this chapter is set up to communicate in one of two types of transmission: ASCII or RTU. This protocol will be discussed more completely later in the chapter. The choice of ASCII or RTU is made by the system designer and must be kept the same throughout. While this example shows only the transmission for a standard Modbus network, it shows a typical transmission for a computer to computer data exchange.

From the manual defining the Modbus protocol, one finds the two following serial transmission diagrams, one for ASCII and the other for RTU data transmission. The data is sent in the order defined by the diagram from left to right:

ASCII with Parity Checking

```
START 1 2 3 4 5 6 7 PRTY STOP
```

ASCII without Parity Checking

```
START 1 2 3 4 5 6 7 STOP STOP
```

RTU with Parity Checking

```
START 1 2 3 4 5 6 7 8 PRTY STOP
```

RTU without Parity Checking

```
START 1 2 3 4 5 6 7 8 STOP STOP
```

Fig. 16-9 Modbus Protocol Frames
ASCII Mode in Modbus Protocol

In the Modbus protocol, each 8-bit byte is set up to be sent in a message as two separate ASCII characters. This protocol gives the following rules for coding data in a message as: (from the Modbus manual)

"Coding System:

- Hexadecimal, ASCII characters 0–9, A–F
- One hexadecimal character contained in each ASCII character of the message

Bits per Byte:

- 1 start bit
- 7 data bits, least significant bit sent first
- 1 bit for even/odd parity; no bit for no parity
- 1 stop bit if parity is used; 2 bits if no parity

Error Check Field:

- Longitudinal Redundancy Check (LRC)

RTU Mode in Modbus Protocol

In the Modbus protocol, each 8-bit byte transmits two hexadecimal characters in the RTU mode. This protocol gives the following rules for coding data in a message as: (from the Modbus manual)

"Coding System:

- 8-bit binary, hexadecimal 0–9, A–F
- Two hexadecimal characters contained in each 8-bit field of the message

Bits per Byte:

- 1 start bit
- 8 data bits, least significant bit sent first
- 1 bit for even/odd parity; no bit for no parity
- 1 stop bit if parity is used; 2 bits if no parity

Error Check Field:

- Cyclical Redundancy Check (CRC)

In general, while ASCII may be configured as either 7 or 8 bit, the standard ASCII table identifies only 128 characters. With the Modbus protocol defined above, 7 bit ASCII is sufficient and is required per the protocol. ASCII protocol is less efficient in the Modbus protocol than RTU, in that for each transmission, only 4 bits of data is transmitted. With RTU, 8 bit must be selected since 8 bits define 8 bits or one byte of data to be transmitted.

The table that follows is the standard set of ASCII characters:
Char  Dec  Oct  Hex | Char  Dec  Oct  Hex | Char  Dec  Oct  Hex | Char Dec  Oct   Hex
-------------------------------------------------------------
(nul)   0 0000 0x00 | (sp)   32 0040 0x20 | @      64 0100 0x40 | `      96 0140 0x60
(soh)   1 0001 0x01 | !      33 0041 0x21 | A      65 0101 0x41 | a      97 0141 0x61
(stx)   2 0002 0x02 | "      34 0042 0x22 | B      66 0102 0x42 | b      98 0142 0x62
(etx)   3 0003 0x03 | #      35 0043 0x23 | C      67 0103 0x43 | c      99 0143 0x63
(eot)   4 0004 0x04 | $      36 0044 0x24 | D      68 0104 0x44 | d     100 0144 0x64
(enq)   5 0005 0x05 | %      37 0045 0x25 | E      69 0105 0x45 | e     101 0145 0x65
(ack)   6 0006 0x06 | &      38 0046 0x26 | F      70 0106 0x46 | f     102 0146 0x66
(bel)   7 0007 0x07 | '      39 0047 0x27 | G      71 0107 0x47 | g     103 0147 0x67
(bs)    8 0010 0x08 | (      40 0050 0x28 | H      72 0110 0x48 | h     104 0150 0x68
(ht)    9 0011 0x09 | )      41 0051 0x29 | I      73 0111 0x49 | i     105 0151 0x69
(nl)   10 0012 0x0a | *      42 0052 0x2a | J      74 0112 0x4a | j     106 0152 0x6a
(vt)   11 0013 0x0b | +      43 0053 0x2b | K      75 0113 0x4b | k     107 0153 0x6b
(np)   12 0014 0x0c | ,      44 0054 0x2c | L      76 0114 0x4c | l     108 0154 0x6c
(cr)   13 0015 0x0d | -      45 0055 0x2d | M      77 0115 0x4d | m     109 0155 0x6d
(so)   14 0016 0x0e | .      46 0056 0x2e | N      78 0116 0x4e | n     110 0156 0x6e
(si)   15 0017 0x0f | /      47 0057 0x2f | O      79 0117 0x4f | o     111 0157 0x6f
(dle)  16 0020 0x10 | 0      48 0060 0x30 | P      80 0120 0x50 | p     112 0160 0x70
(dc1)  17 0021 0x11 | 1      49 0061 0x31 | Q      81 0121 0x51 | q     113 0161 0x71
(dc2)  18 0022 0x12 | 2      50 0062 0x32 | R      82 0122 0x52 | r     114 0162 0x72
(dc3)  19 0023 0x13 | 3      51 0063 0x33 | S      83 0123 0x53 | s     115 0163 0x73
(nak)  20 0025 0x18 | 5      53 0065 0x35 | T      84 0125 0x55 | t     116 0165 0x75
(syn)  21 0026 0x19 | 6      54 0066 0x36 | U      85 0126 0x56 | u     117 0166 0x76
(ets)  22 0027 0x1a | 7      55 0067 0x37 | V      86 0127 0x57 | v     118 0167 0x77
(can)  23 0030 0x1d | 8      56 0068 0x38 | W      87 0128 0x58 | w     119 0168 0x78
(em)   24 0031 0x1f | 9      57 0069 0x39 | X      88 0129 0x59 | x     120 0169 0x79
(sub)  25 0032 0x20 | :      58 0070 0x3a | Y      89 0130 0x5a | y     121 0170 0x7a
_esc)  26 0033 0x21 | ;      59 0071 0x3b | Z      90 0131 0x5b | z     122 0171 0x7b
(fs)   27 0034 0x22 | <      60 0072 0x3c | [      91 0132 0x5c | [     123 0172 0x7c
(gs)   28 0035 0x23 | >      61 0073 0x3d | \      92 0133 0x5d | \     124 0173 0x7d
(us)   29 0036 0x24 | ?      62 0074 0x3f | _      93 0134 0x5e | _     125 0174 0x7e
(del)  30 0037 0x25 | ^      63 0075 0x3f | (del)  30 0037 0x25

Example of a Simple ASCII Protocol

Many times a two-way communication between devices requires a set sequence of characters that define a proper communication. The protocol for the device below is a letter, a number (Head Number), a check sum followed by an end of text character or <ETX>. Each communication follows roughly the same simple pattern. A computer receives the request from the device and responds with the appropriate information. The device initiates a request and gathers the results. The examples below are between a computer and a radio-frequency identification system from Peprl and Fuchs. Peprl and Fuchs literature defines each specific data type. For instance <HdNo> refers to a specific head number in the range 1 to 4.

Command: A<HdNo><CHCK><ETX>
Example: Read all data in Auto mode with read head 1:
Command: A 1 72h 03h

Read bytes, Single mode
Command: w<HdNo><STAdrH><BytesH><CHCK><ETX>
Example: Read bytes 7 to 11 in Auto mode with read head 1:
Command: W 1 07 05 54h 03h

Write bytes, Auto mode
Command: K<HdNo><STAdrH><BytesH><DB><CHCK><ETX>
Example: Write "P & F" to data carrier, at start address 10, in Auto mode with read head 2:
Command: K 2 0A 03 50h 2Bh 46h 12h 03h
To calculate a check sum \(<\text{CHCK}>\), the following addition is performed:

\[
\begin{align*}
K & \quad 4Bh \quad \text{Ascii char "K"} \\
2 & \quad 32h \quad \text{Ascii char "2"} \\
0 & \quad 30h \quad \text{Ascii char "0"} \\
A & \quad 41h \quad \text{Ascii char "A"} \\
0 & \quad 30h \quad \text{Ascii char "0"} \\
3 & \quad 33h \quad \text{Ascii char "3"} \\
50h & \quad 50h \quad \text{hex char 50} \\
2Bh & \quad 2Bh \quad \text{hex char 2B} \\
46h & \quad 46h \quad \text{hex char 46} \\
\hline
& \quad (2) 12h
\end{align*}
\]

This gives a check sum \(<\text{CHCK}>\) of 12h.

The check sum is used in many applications for error-checking. If the check sum does not equal the calculated checksum, the data is discarded as bad. Check sum is also referenced as LRC or Longitudinal Redundancy Check. It is a simple procedure giving a good check on validity of the characters sent.

**ASCII Instructions in the PLC**

The SLC Instruction Set includes several ASCII instructions for reading and writing data from the PLC.

Different applications require some or all of these instructions to accurately find information in the string of data and use the information in the control of the process. AWA is ASCII Write with append and AWT is an ASCII string write with no append. While the student may be at first excited about the use of serial data transmission and writing a protocol, these programs are among the most difficult to keep running in a factory environment. Noise may interfere with a proper transmission and add a random character. A computer may not respond when asked. Error recovery programs, time-outs, re-trys all become an integral part of any program to implement one of these programs in a factory. Testing a procedure on the lab bench is usually not enough to ensure success with this type of program in a field application.

**Building A Test System**

ASCII devices may need to be tested to prove data transmission. One of the best devices for testing purposes is the personal computer. Use of Hyperterminal found on Windows-based products is possible. A second terminal suggested is Br@y’s Terminal, a program downloadable on your computer. The PLC and terminal must agree with all variables in the screen below:
The cable between the PLC and PC is an RS-232 cable with the transmit signal from one device attached to the receive signal of the other device. Pin numbers are crossed with pin 2 of one device connected to pin 3 of the other connector.

Be sure to use the PIC module to attach the SLC 5/03 to RSLogix500. Use Channel 1 for this link. Channel 0 is now to be used as a simple ASCII port. It must be set for baud rate and parity to match the Com1 port on the pc.

To send from the terminal, start typing. The character string will be transmitted a character at a time. Many times the transmission occurs when the ‘enter’ key is pressed. To receive characters, initiate communications from the PLC and watch for the character string to appear on the pc terminal.
Example of ASCII Write in PLC

Fig. 16-12 Example of ASCII Write Instruction in SLC Processor

Notice above that the String file ST9:0 has text as follows: A1\72\03. This signifies that the AWT command is used to write the A1 command followed by the CHCK of 72h followed by the ETX or 03h.

Use of the oscilloscope to diagnose problems in serial transmissions is not common but may be useful. With serial transmissions, use the oscilloscope as an arbitrator to find whether the device really did transmit the string or the second device saw the character string and did not recognize it. Storage scopes are best but with repeated character transmission, a non-storage scope will also enable students to verify the data transmitted.

Full duplex communication allows the devices to respond when a transmission is received. To test whether a communication has occurred, be sure to use the oscilloscope with the transmit lead to neutral. The other lead is the receive lead.

A break-out box is also useful for testing transmissions. While a professionally made break-out box is preferred, a simple connector to pins 2 and 3 of the cable and to pin 5 for the neutral will work when used with the oscilloscope. This should be tried to verify that a character string is being transmitted. If a storage oscilloscope is not available, continuous repeating of a character transmitted from the terminal emulator should eventually be seen on the oscilloscope if set up properly. Use of the oscilloscope in this kind of application may be the only way to verify that the transmission of characters is actually occurring.

Data Transmissions

Devices transmit to or from Channel 0 User Port of the SLC 5/03 on an event. The port can either send or collect characters in a buffer. If serial data is being collected at the port, the ASCII read instruction sends the characters to a buffer when the expected number of characters in the buffer is achieved. This type of transmission is prone to error and frustration. Noise in the data transmission may be received as an extra character. Several of these noise characters may be present and waiting in the print queue when a transmission occurs. As a result, what may work well in the lab or on a test bench may not work at all or only marginally in a noisy factory environment.
Many transmissions start with a special character — STX — to identify an authentic start-of-text message. Other special characters to identify text include ETX or end-of-text.

Error-checking methods include check sum commonly referred to as LRC or the more sophisticated CRC to verify data integrity. Check sum is a simple calculation of the addition of all data bytes before the check sum. The data is added and any carry from the low byte is ignored. The resulting check sum is transmitted and data is verified as good or not good based on the check sum transmitted from the sending device and calculated from the receiving device. Check sum calculations include the STX character sent with the transmission.

**A More Complete Protocol**

A protocol is a defined method by which computers communicate. A very early PLC protocol first used by Modicon and the Modicon PLC family is the Modbus protocol. This protocol has provided a standard for communication between various controllers since the late 1970s and remains active today in a number of PLC and other industrial products. The protocol defines a message stream that various controllers can recognize. Its inclusion here gives an example of a more complete protocol with a more robust set of functions available. This protocol has survived over the years due to its early acceptance, its flexibility, and its adaptability to a wide range of control devices. A rigorous explanation of the protocol is found in Modicon's Modbus Protocol Reference Guide.


> "It describes the process a controller uses to request access to another device, how it will respond to requests from the other devices, and how errors will be detected and reported. It establishes a common format for the layout and contents of message fields. The Modbus protocol provides the internal standard that the Modicon controllers use for parsing messages. During communications on a Modbus network, the protocol determines how each controller will know its device address, recognize a message addressed to it, determine the kind of action to be taken, and extract any data or other information contained in the message. If a reply is required, the controller will construct the reply message and send it using Modbus protocol."

**The Modbus Transaction**

The Modbus transaction is most typically used with an RS-232C serial interface. The RS-232C designation defines connector pin-outs as well as other electrical and timing constraints on signals between the communicating devices.

Modbus defines a master-slave protocol in which the master device queries the slave device which then responds with its own transmission. A slave response is typically a table of data but may include an action as requested by the master. A master may include computers or HMI terminals but may also include other PLCs. A typical slave is a PLC or other control device. Devices used as slave devices include any device from which control information is desired or needs to be changed. A wide range of devices other than Modicon PLCs have adopted the Modbus protocol and are programmed as Modbus slaves. Responses from slave devices range from a single device response to a general broadcast query message to all slave devices on the network.

Modbus provides a format for both master and slave protocols. From the figure on the next page, a typical master query followed by a slave response is shown. In the query are information such
as the device address being communicated to, a function code, any data being sent or requested, and an error checking field. This protocol is more sophisticated than the simple Peppl-Fuchs protocol in that if an error occurs, the slave sends the appropriate error message.

PI-MBUS–300 Rev. J

The Query:

A query provides a function code requesting an action. The query may include any of the following allowable function codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Read Coil Status</td>
<td>specifies 16*n discrete slave PLC outputs</td>
</tr>
<tr>
<td>02</td>
<td>Read Input Status</td>
<td>specifies 16*n discrete slave PLC inputs</td>
</tr>
<tr>
<td>03</td>
<td>Read Holding Registers</td>
<td>specifies n 16 bit words from slave PLC output tbl</td>
</tr>
<tr>
<td>04</td>
<td>Read Input Registers</td>
<td>specifies n 16 bit words from slave PLC input table</td>
</tr>
<tr>
<td>05</td>
<td>Force Single Coil</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Preset Single Register</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Read Exception Status</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>Diagnostics</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Program 484</td>
<td>84 specifies a type of Modicon PLC</td>
</tr>
<tr>
<td>10</td>
<td>Poll 484</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Fetch Comm. Event Ctr.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Fetch Comm. Event Log</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Program Controller</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Poll Controller</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Force Multiple Coils</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Preset Multiple Registers</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Report Slave ID</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Program 884/M84</td>
<td>884/M84 specifies a type of Modicon PLC</td>
</tr>
<tr>
<td>19</td>
<td>Reset Comm. Link</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Read General Reference</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Write General Reference</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Mask Write 4X Register</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 16-13
The function code field is followed by information in the master query telling the slave at which word to begin and the number of words to read or write. Error checking provides a validation of the message.

**The Response:**

The response is an echo of the function code found in the query plus any data requested followed by the error checking byte or bytes. The new error checking provides the master with a validation of the message and its contents.

**Data Addresses Referenced to Zero**

Every part of the data field in the Modbus protocol must be addressed correctly. For example, the data address must be referenced to zero. From the Modbus Protocol manual, the following examples show calculations of offsets for data addresses:

"The coil known as ‘coil 1’ in a programmable controller is addressed as coil 0000 in the data address field of a Modbus message. Coil 127 decimal is addressed as coil 007E hex (126 decimal).

Holding register 40001 is addressed as register 0000 in the data address field of the message. The function code field already specifies a ‘holding register’ operation. Therefore the ‘4XXXX’ reference is implicit.

Holding register 40108 is addressed as register 006B hex (107 decimal)."

**Framing - ASCII**

From the Modbus manual, the following:

"In ASCII mode, messages start with a ‘colon’ ( : ) character (ASCII 3A hex), and end with a ‘carriage return – line feed’ (CRLF) pair (ASCII 0D and 0A hex). The allowable characters transmitted for all other fields are hexadecimal 0–9, A–F."

The typical transmission for a Modbus ASCII transmission resembles:

<table>
<thead>
<tr>
<th>Start</th>
<th>Address</th>
<th>Function</th>
<th>Data</th>
<th>LRC Check</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Char :</td>
<td>2 Chars</td>
<td>2 Chars</td>
<td>n Chars</td>
<td>2 Chars</td>
<td>2 Chars CRLF</td>
</tr>
</tbody>
</table>

Fig. 16-14  ASCII Transmission Frame

**Framing - RTU**

From the Modbus manual, the following:

"In RTU mode, messages start with a silent interval of at least 3.5 character times. This is most easily implemented as a multiple of character times at the baud rate that is being used on the network (shown as T1–T2–T3–T4 in the figure below). The first field then transmitted is the device address. The allowable characters transmitted for all fields are hexadecimal 0–9, A–F."
Networked devices monitor the network bus continuously, including during the ‘silent’ intervals. When the first field (the address field) is received, each device decodes it to find out if it is the addressed device. Following the last transmitted character, a similar interval of at least 3.5 character times marks the end of the message.

The typical transmission for a Modbus RTU transmission resembles:

<table>
<thead>
<tr>
<th>Start</th>
<th>Address</th>
<th>Function</th>
<th>Data</th>
<th>CRC Check</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1-T2-T3-T4</td>
<td>8 Bits</td>
<td>8 Bits</td>
<td>n * 8 Bits</td>
<td>16 Bits</td>
<td>T1-T2-T3-T4</td>
</tr>
</tbody>
</table>

Fig. 16-15 RTU Transmission Frame

Example Modbus Transmissions

Description of "02" Read Input Status

The "02" request reads the status of discrete input points. The request is for inputs 197 to 218 from slave device 17.

Query

<table>
<thead>
<tr>
<th>Field Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave Address</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>02</td>
</tr>
<tr>
<td>Starting Address Hi</td>
<td>00</td>
</tr>
<tr>
<td>Starting Address Lo</td>
<td>C4</td>
</tr>
<tr>
<td>No. of Points Hi</td>
<td>00</td>
</tr>
<tr>
<td>No. of Points Lo</td>
<td>16</td>
</tr>
<tr>
<td>Error Check (LRC or CRC)</td>
<td>----</td>
</tr>
</tbody>
</table>

Sample Modbus Read Input Query

Response

<table>
<thead>
<tr>
<th>Field Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave Address</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>02</td>
</tr>
<tr>
<td>Byte Count</td>
<td>03</td>
</tr>
<tr>
<td>Data (Inputs 10204 - 10197)</td>
<td>AC</td>
</tr>
<tr>
<td>Data (Inputs 10212 - 10205)</td>
<td>DB</td>
</tr>
<tr>
<td>Data (Inputs 10218 - 10213)</td>
<td>35</td>
</tr>
<tr>
<td>Error Check (LRC or CRC)</td>
<td>----</td>
</tr>
</tbody>
</table>

Sample Modbus Read Input Response
**Explanation of Response:**

The Slave Address is repeated from the query as 11. The function is also repeated from the query as 02. Byte count is calculated from the number of bytes to be sent. The number of bytes sent can be calculated by establishing the number of bytes necessary to send 16 hex data points. The number 16 hex is equal to 22 decimal. If each input represents one data point, three 8-bit bytes are needed for 22 data points. The data is sent from input addresses starting at offset C4 hex from the first data point 10001. The hex number C4 is equal to 12*16 or 192 plus 4 or total 196. The first data point is displaced 196 from 10001 or 10197. The first 8 data points reside in addresses 10197 to 10204. The next eight reside in addresses 10205 to 10212. The final 6 reside in addresses 10213 to 10218. Two bits are not used (10219, 10220). The data from these bits is sent in three consecutive bytes. The value of 10197 through 10204 is shown in the first entry of the figure below. Values of 10205 through 10218 are found in the second entry and values of 10213 to 10218 are found in the third.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10204</td>
<td>1 0 1 0 1 1 0 0</td>
</tr>
<tr>
<td>10203</td>
<td></td>
</tr>
<tr>
<td>10202</td>
<td></td>
</tr>
<tr>
<td>10201</td>
<td></td>
</tr>
<tr>
<td>10200</td>
<td></td>
</tr>
<tr>
<td>10199</td>
<td></td>
</tr>
<tr>
<td>10198</td>
<td></td>
</tr>
<tr>
<td>10197</td>
<td></td>
</tr>
</tbody>
</table>

A

<table>
<thead>
<tr>
<th>Address</th>
<th>Data (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10212</td>
<td>1 1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>10211</td>
<td></td>
</tr>
<tr>
<td>10210</td>
<td></td>
</tr>
<tr>
<td>10209</td>
<td></td>
</tr>
<tr>
<td>10208</td>
<td></td>
</tr>
<tr>
<td>10207</td>
<td></td>
</tr>
<tr>
<td>10206</td>
<td></td>
</tr>
<tr>
<td>10205</td>
<td></td>
</tr>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>Address</th>
<th>Data (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10218</td>
<td>0 0 1 1 0 1 0 1</td>
</tr>
<tr>
<td>10217</td>
<td></td>
</tr>
<tr>
<td>10216</td>
<td></td>
</tr>
<tr>
<td>10215</td>
<td></td>
</tr>
<tr>
<td>10214</td>
<td></td>
</tr>
<tr>
<td>10213</td>
<td></td>
</tr>
</tbody>
</table>

C

D

Fig. 16-17 Calculating the Slave Response
Description of "04" Read Registers

This is a request to read input register 30009 from the slave device at location 17.

Query

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave Address</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>04</td>
</tr>
<tr>
<td>Starting Address Hi</td>
<td>00</td>
</tr>
<tr>
<td>Starting Address Lo</td>
<td>08</td>
</tr>
<tr>
<td>No. of Points Hi</td>
<td>00</td>
</tr>
<tr>
<td>No. of Points Lo</td>
<td>01</td>
</tr>
<tr>
<td>Error Check (LRC or CRC)</td>
<td>----</td>
</tr>
</tbody>
</table>

Sample Modbus Read Registers Query

Response

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave Address</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>04</td>
</tr>
<tr>
<td>Byte Count</td>
<td>02</td>
</tr>
<tr>
<td>Data Hi (Register 30009)</td>
<td>00</td>
</tr>
<tr>
<td>Data Lo (Register 30009)</td>
<td>0A</td>
</tr>
<tr>
<td>Error Check (LRC or CRC)</td>
<td>----</td>
</tr>
</tbody>
</table>

Sample Modbus Read Registers Response

Explanation of Response:

The Slave Address is repeated from the query as 11. The function is also repeated from the query as 04. Byte count is calculated from the number of bytes to be sent. The number of bytes sent can be calculated by establishing the number of bytes necessary to send 01 16-bit words. This number is 2 bytes. Contents of input register 30009 are sent in the following two bytes. First is sent the high byte (00). Second is sent the low byte (0A). Input register 30009 is displacement 8 from the first input register 30001. If more than 1 register is requested, the number of bytes added would equal 2 bytes for each register requested.
Possible Responses to Request

As a more sophisticated protocol, the programmer must be aware of possible outcomes other than a normal response as shown above. Any of four different responses might occur when a request is initiated and the PLC or computer program must be capable of handling any of them.

1. Normal Response
2. Slave doesn't Receive - No response (master program should process a timeout and retry)
3. Communication Error (parity, LRC, or CRC error) - No Response
4. Unable to handle request - Returns an exception response

Error Checking

Either the ASCII or RTU mode may be scheduled for the Modbus protocol. Error checking for these two modes varies in that a different algorithm is used to calculate the error checking field. ASCII uses the Longitudinal Redundancy Check (LRC) and RTU uses the Cyclical Redundancy Check (CRC). ASCII mode begins the transmission with a 'colon' and ends the transmission with CRLF characters. In either mode, if the LRC or CRC does not match the calculated value in the computer program, an error is present and the communication is terminated. Calculating these two checking types may be obtained from various websites with the programs written and examples provided. A more thorough discussion of these methods is also found in the Modbus Reference Manual. LRC mode is similar to the check sum example of the Simple ASCII Protocol.

The user rarely needs to fully understand the protocol or the various codes for the transmission. He should be able to configure the transmission and the expected response, however. The use of this protocol shows up in the most unusual situations with equipment not at all related to the Modicon organization. You may not believe it but a capstone group in Fall 2013 was required to implement this protocol to succeed with their project.

How a PLC implements a simple ASCII transmission is next reviewed. There is no present lab experience associated with this section although you may read through it and conclude that there once was a lab and you would be right.

Message Instruction Overview

The Message Block is an output instruction that allows data to be read or written from one processor to another via the communication channel(s). The SLC 5/-2 processor can service one message instruction at any given time. The SLC 5/03 and higher processors can service up to four message instructions per channel at a time, for a maximum of eight message instructions at any given time. To invoke the MSG instruction, toggle the MSG instruction rung from false to true or set the instruction to run continuously. Do not toggle the rung again until the MSG instruction has successfully or unsuccessfully completed the previous message, indicated by the processor setting either the DN or EN bit.

SLC 5/03 and higher – If a MSG instruction has entered one of the four “channel dependent” transmission buffers and is waiting to be transmitted, its control block will have status bits EN and EW set. If more than four MSG instructions for that channel are enabled at one time, a “channel dependent” overflow queue is used to store the MSG instruction header blocks (not the data for a MSG write) from the fifth instruction to the fourteenth. These instructions, queued in a FIFO order, will only have control block status bit EN set.
If more than 14 MSG instructions are enabled at one time for any one channel, only control block status bit WQ is set, as there is no room available to currently queue the instruction. This instruction must be re-scanned with true rung conditions until space exists in the overflow queue.

Tip: If you consistently enable more MSG instructions than the buffers and queues can accommodate, the order in which MSG instructions enter the queue is determined by the order in which they are scanned. This means MSG instructions closest to the beginning of the program enter the queue regularly and MSG instructions later in the program may never enter the queue.

Message blocks may be set to read/write on a trigger or read/write continuously. Examples of both are found in the Reference Manual in the MSG chapter.

It is advised to run multiple applications of RSLogix 500 when debugging the MSG command. The command to write a word or multiple words from one processor to a second processor may be triggered by toggling the input contact to the MSG block. Verification of the data move may be seen immediately after a toggle operation if the data was successfully transmitted in the second processor. In the example above, data is to be read from the processor on the right to the processor on the left. Note that the instruction for the MSG command is only present in the processor initiating the command to read or write.

The setup screen brings the user to the screen for setting up the MSG block. When set up properly, and the bit B3:1/0 toggled on, the MSG block should execute. First the EN coil turns on. Then either DN or ER will turn on. If DN turns on, the operation was executed. Click to see if the operation occurred successfully (if the data moved). If so, the command was successful. If the command was not executed successfully or if the ER bit turned on, more work is necessary to configure the MSG command correctly.

MSG commands are set up to work automatically in most programs. The use of timers to allow a sufficient time for the operation to occur followed by a check for DN or ER is appropriate. A count of 3 or 5 can be set for allowable retries of the communication before the command is alarmed as not working properly.
MSG Instruction Parameters:

Enter the following parameters when programming this instruction:

- **Read/Write** – read indicates that the local processor (processor in which the instruction is located) is receiving data; write indicates that it is sending data.

Local or Remote identifies if the message is sent to a device on a local network, or to a remote device on another network through a bridge. Valid options are:

- local, if the target device is on the local network
- remote, if the target device is on the remote network

Control Block is an integer file address that you select. It is a block of words, containing the status bits, target file address, and other data associated with the message instruction.

Control Block Length is a display-only field that indicates how many integer file words are being used by the control block.

Use the MSG Setup Screen to set up the values in the control block.

To troubleshoot a MSG operation, open multiple copies of RSLogix 500 to examine the state of the data as the read or write block is being executed. This leads to a quick verification that the data (one or multiple words) is being transmitted successfully.

**PROFINET and Industrial Networks**

Profinet is an industrial Ethernet network. It is capable of connecting computers with PLCs with I/O. It is also capable of wireless communication as well as operating in safe environments.

Comparison of Profinet with Ethernet/IP as the leading network choices for industrial networks shows the following:

<table>
<thead>
<tr>
<th>Industrial Ethernet Market</th>
<th>Ethernet/IP</th>
<th>PROFINET</th>
<th>Other</th>
</tr>
</thead>
</table>

Ethernet/IP is the choice of Allen-Bradley controllers and their vendor partners while Profinet is the choice of Siemens and their vendor partners. As with all industrial equipment, a choice must be made which network (or both) to have in a facility.

While DeviceNet was discussed earlier in the chapter, its effective use on the marketplace has waned and Ethernet/IP is the defacto standard for all A-B products. Profinet has a similar relationship with Profibus. On the next page is a list of various industrial network technologies...
and their relative strength. The two listed first are Profinet (from Siemens) and Ethernet/IP (from Allen-Bradley).


<table>
<thead>
<tr>
<th>Technology</th>
<th>PROFIBUS</th>
<th>DeviceNet</th>
<th>Foundation Fieldbus</th>
<th>Modbus ModbusTCP</th>
<th>EtherCAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consortium</td>
<td>PI</td>
<td>ODVA</td>
<td>Fieldbus Foundation</td>
<td>Modbus IDA</td>
<td>ETG</td>
</tr>
<tr>
<td>Primary Backer</td>
<td>Siemens</td>
<td>Rockwell</td>
<td>Emerson</td>
<td>Schneider</td>
<td>Beckhoff</td>
</tr>
<tr>
<td>Membership</td>
<td>1,400</td>
<td>290</td>
<td>350</td>
<td>65</td>
<td>1,195</td>
</tr>
<tr>
<td>Regional Organizations</td>
<td>26</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Competence Centers</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test Labs</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Training Centers</td>
<td>18</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 16-21

A look back at the older Profibus and comparing it to the newer Profinet follows. This chart shows some of the needed changes that took place as the Ethernet-based networks have taken over.

Differences:

<table>
<thead>
<tr>
<th></th>
<th>PROFIBUS</th>
<th>PROFINET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical layer</td>
<td>RS-485</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Speed</td>
<td>12Mbits</td>
<td>100Mbits</td>
</tr>
<tr>
<td>Telegram</td>
<td>244 bytes</td>
<td>1440 bytes</td>
</tr>
<tr>
<td>Address space</td>
<td>126</td>
<td>2048</td>
</tr>
<tr>
<td>Technology</td>
<td>master/slave</td>
<td>provider/consumer</td>
</tr>
<tr>
<td>Connectivity</td>
<td>PA + others</td>
<td>many buses</td>
</tr>
<tr>
<td>Wireless</td>
<td>Possible</td>
<td>IEEE 802.11, 15.1</td>
</tr>
<tr>
<td>Motion</td>
<td>32 axes</td>
<td>150 axes</td>
</tr>
<tr>
<td>Mach to mach</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Vert integration</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No. of products</td>
<td>3,000</td>
<td>300</td>
</tr>
</tbody>
</table>

Fig. 16-22

What do we mean when we say Ethernet?

- The IEEE 802.3 Specification defines:
  1. The physical media
2. The media access rules
3. The structure of an Ethernet frame

It is important to see that Ethernet is not the entire network solution. The specification does not specify any physical or environmental operating requirements. The figures on the next page are an attempt to show the relative position of Ethernet to TCP/IP and the application oriented layers. It is not the intent of this text to train the student on the principles of network configuration. This entails a separate course or two and much work on the intricacies of the various network layers.

The ISO/OSI communications model
Each communication procedure is divided into logical components which are linked via defined interfaces.

ISO = International Standards Organization
OSI = Open Systems Interconnection

Multiple application protocols can run at the same time on the physical medium (Ethernet)

Communications

Flow of data in a communication packet is shown in this section. Again, the text in no means is trying to establish itself as a text on the intricacies of network communication.
Not all layers are needed:

Industry's Demand for Speed

TCP/IP has methods in place to resend telegrams when lost
- But the timing is not acceptable for industrial use!
There is no such thing as a protocol protection against noise
- The need for shielding is independent of the protocol used
Grounding at both ends is best – but not always applicable due to ground loops
If you used shielded cable with DeviceNet or PROFINET, use shielded cable with Ethernet as well!

Four steps to fast, deterministic communication:
Why not use TCP/IP for real-time?
Because it’s not fast enough and it’s not deterministic enough. Here’s why.

Ethernet doesn’t cause delays.

PROFINET RT (real time)
The figure below shows the flow of data in Profinet RT, the real-time ethernet packet from Profinet.
The data packet is created to ensure scheduling traffic for even motion control. While the use of one of the ethernet networks can be used for I/O, it is not always best. The following picture shows the positioning of the ethernet network with some of the Siemens’ answers for I/O network. Two answers are IO Link and ASi.

We do not want to define the positioning of the various network types, either ethernet (pick your type) or I/O network (pick your type). You will have to make this decision as you work with the various types of PLC and their network topology. I wish you well.

**Real-time I/O Ethernet Networks Comparison**

<table>
<thead>
<tr>
<th>System</th>
<th>PROFINET</th>
<th>Ethernet/IP</th>
<th>EtherCAT</th>
<th>Modbus TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Model</td>
<td>Provider-Consumer</td>
<td>Provider-Consumer</td>
<td>Master-Slave</td>
<td>Client-Server</td>
</tr>
<tr>
<td>Real-time Protocol Layer</td>
<td>IEEE 802.3 (Layer 2)</td>
<td>UDP/IP (Layer 4)</td>
<td>IEEE 802.3 (Layer 2)</td>
<td>TCP/IP (Layer 4)</td>
</tr>
<tr>
<td>Application Protocol</td>
<td>IEC 61158</td>
<td>IEC 61158</td>
<td>IEC 61158</td>
<td>IEC 61158</td>
</tr>
<tr>
<td>Transmission Type</td>
<td>Unicast</td>
<td>Multicast</td>
<td>Shift Register</td>
<td>Unicast</td>
</tr>
<tr>
<td>Safety</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>Wireless</td>
<td>Yes</td>
<td>Yes (with caution)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Fast Start Up</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>Simple Device replacement</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>Energy Management</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
IO Link Functional Positioning: The figure below shows the positioning of various bus types in the factory with the Ethernet at the top and various I/O networks at the bottom.

![Diagram of bus types and positioning](image)

A Failure to Communicate

Some of the most serious problems encountered as a PLC programmer involve problems with communications. Whether it is that they can’t be seen or that they can be easily over-loaded, the stories about failures in communications problems are many and varied. The following is just one not to be duplicated.

The following engineered system was to be implemented in an automotive parts plant. The PLC program was to send a command string to the Remote I/O module located in a rack in the PLC housing. From there, the command was to be sent to the vendor’s remote I/O receiver module and from there to the vendor’s communications device. Any communication that was to be successful would need to pass through four CPUs and successfully return to the initiating program. While the test program worked successfully on the bench, when it entered the field, problems developed. The problem was that if a communication was not successfully received back prior to a second communication starting, the communications became confused and stopped. The only procedure for re-starting the communications was to power down and back up both vendor devices, a task that would be painful to do even if given permission by the end user. The fact that proximity switches initiated the communication and they are noisy devices that many times have multiple inputs when a device is sensed was also part of the problem.
This system obviously was in serious trouble before the programmer entered the picture. Anyone designing a system with so many computer cpu’s involved should be questioned as to their sanity. The problem was that at the time of this installation, there were few alternatives to this design and it looked, at least on paper, to be one of the best. In retrospect, this type of design is always problematic and should be avoided. The fewer devices placed in a critical path, the better the system.

Summary

The subject of PLC communications could encompass an entire book on its own.

The subject could be depressing but hopefully we leave it with a smile. Thank you, Paul!

You do not want to be associated with either of these situations, the one at left, or especially, the one below.

If you are not familiar with these people, look up the movie “Cool Hand Luke”. At left is Strother Martin, the captain of the prison camp with his famous statement “what we have here is a failure to communicate”. Below is Paul Newman, the object of the comment.
IoT is alphabet soup. IIoT, IoE, HTTP, REST, JSON, MQTT, OPC UA, DDS, and the list goes on. Conceptually, we’ve discussed IoT for a long time and understand the basic idea and technical feasibility. Now we’re moving forward, identifying use cases and building prototypes. So it’s about time to work on that alphabet. A big challenge in IoT is interoperability. In a recent Nexus survey, 77% of respondents stated that interoperability was their biggest challenge in IoT. Connecting industrial devices to IT and IoT platforms is big business, and it’s where a lot of the abbreviations come from. There are many protocols to accomplish this: some that are proprietary and others that are open standards. All are jockeying to be the one and only IoT protocol, but it’s clear this will never be the case. These protocols will co-exist—each with their own strengths and weaknesses—and it’s our job to understand where and when to use them. This whitepaper focuses on the open standards for connecting industry to IT and provides use cases for each.

Client/Server vs. Publish/Subscribe For the purpose of this discussion, it’s important to group protocols into two categories: client/server and publish/subscribe. Client/server protocols require the client to connect to the server and make requests. In this model, the server holds the data and responds to requests from the client. For example, the client may read a temperature. This requires the client to know about the server in advance and be able to connect. Publish/subscribe protocols require the devices to connect and publish data to a topic on an intermediary broker. Consumers can connect to the broker and subscribe to data from the topic. For example, a device can sample the temperature every minute and publish once an hour. An application subscribed to the data stream receives an hour’s worth of one-minute samples every hour. This model decouples the producer of the data from the consumer of the data. Client/server protocols are best used when you understand your infrastructure. For example, you know that your server in the field has an IP address of 55.55.55.55 and is listening on port 1234. The client can connect and make requests. Publish/subscribe protocols are a better fit when your infrastructure is unknown. For example, if a remote device changes networks or has intermittent connectivity, it’s easier for the device to call home when it’s online and publish its data. In terms of pros and cons, client/server protocols are more interoperable and secure because they are based on point-to-point connections. They are, however, less scalable, because point-to-point connections are harder to manage and more resource intensive. In contrast, publish/subscribe protocols are more scalable because the decoupling of producers and consumers allows each to be added and removed independently. That said, securing these protocols is more complex because there are more pieces involved. They can also have interoperability issues given the lose coupling of the producer and consumer. For example, changing the message format that a producer sends requires all consumers to adapt to the new message type. Now that we understand the basic categories, let’s look at client/server and publish/subscribe protocols in more detail.

Protocols

The protocols we’ll discuss have the potential to connect together industrial devices with IoT platforms. It may go without saying, but if you’re trying to connect two applications and both support HTTP, try HTTP first. If that doesn’t work or if your environment doesn’t support it, keep reading. We’ll describe each protocol and when to use it. Here is the short list that we’ll cover:

- OPC UA
- HTTP (REST/JSON)
- MQTT
- CoAP
- DDS
- AMQP
OPC UA

OPC Unified Architecture (OPC UA) is the next generation standard from the OPC Foundation. Classic OPC is well known in the industrial space and provides a standard interface to communicate with PLCs. OPC UA aims to expand OPC’s interoperability to the device and enterprise levels. OPC UA is a client/server protocol. Clients connect, browse, read, and write to industrial equipment. UA defines communications from the application to the transport layer, making it very interoperable between vendors. It’s also highly secure, and uses two-way message signing and transport encryption. OPC UA has a wide install base in the industrial space. It is a good solution for tying PLC and sensor data into existing industrial applications like SCADA and MES systems, where OPC and OPC UA connectivity are already available. OPC UA is new to the IT space, however. Some people in IT are intimidated by the complexity of UA compared to other IT protocols. A lot of this complexity is a result of OPC UA being an industrial protocol, but this perception has led to slow adoption by IoT platforms and the open source community. Things are changing, however: recently, the OPC Foundation open sourced the OPC UA standard to make it more accessible and help increase adoption. For now, use OPC UA when you need to get PLC and sensor data into existing SCADA and MES solutions, and keep an eye out for OPC UA adoption by IoT platform providers and the open source community.

HTTP (REST/JSON)

Hypertext Transfer Protocol (HTTP) is a connectionless client/server protocol ubiquitous in IT and the web. Because there are countless open source tools that use HTTP, and every coding language has HTTP libraries, it is very accessible. The focus on HTTP in IoT is around Representational State Transfer (REST), which is a stateless model where clients can access resources on the server via requests. In most cases, a resource is a device and the data that a device contains. HTTP provides a transport, but doesn’t define the presentation of the data. As such, HTTP requests can contain HTML, JavaScript, JavaScript Object Notation (JSON), XML, and so forth. In most cases, IoT is standardizing around JSON over HTTP. JSON is similar to XML—without all the overhead and schema validation—making it more lightweight and flexible. JSON is also supported by most tools and programming languages.

Industry has some experience using HTTP for device and product configuration, but not for data access. As such, many IoT and IT platforms support consuming and providing data in HTTP form, but few industrial platforms do. This is changing as more gateways and PLCs begin to add native HTTP support. Use HTTP for sending chunks of data, like one-minute temperature readings every hour. Don’t use HTTP for streaming high-velocity data. HTTP can do sub-second data, but 100 ms updates over HTTP are difficult. It has a lot of overhead per message, so streaming small messages is inefficient. And always secure communications with HTTPS. The overhead is minimal. Be aware of interoperability issues with HTTP products. Just because two products support HTTP/REST/JSON doesn’t mean they’ll work out of the box. Often the JSON formats are different and require minimal integration to get things working.

MQTT

Message Queuing Telemetry Transport (MQTT) is a publish/subscribe protocol designed for SCADA and remote networks. It focuses on minimal overhead (2 byte header) and reliable communications. It’s also very simple. Like HTTP, MQTT’s payload is application specific, and most implementations use a custom JSON or binary format. MQTT isn’t as widely used as HTTP, but it still has a large market share in IT. There are many open source clients/producers, brokers, projects, and examples in every language. Many IoT platforms support HTTP and MQTT as their first two inbound protocols for data. Use MQTT when bandwidth is at a premium...
and you don’t know your infrastructure. Make sure you or your vendor has an MQTT broker you can publish data to—and always secure communication via Transport Layer Security (TLS).

Does the end application not support MQTT? If so, there are a lot of open source tools for getting MQTT data into databases and other formats like HTTP. Beware of interoperability issues similar to HTTP. Just because two applications support MQTT doesn’t mean they are interoperable. The topic and JSON formats may need to be adjusted to make the two products interoperable.

CoAP

The Constrained Application Protocol (CoAP) was created by the Internet Engineering Task Force (IETF) to provide the interoperability of HTTP with minimal overhead. CoAP is similar to HTTP, but uses UDP/multicast instead of TCP. It also simplifies the HTTP header and reduces the size of each request. CoAP is used in edge-based devices where HTTP would be too resource intensive, and is often the third protocol supported by IoT platforms after HTTP and MQTT. Similar to HTTPS, CoAP uses Datagram Transport Layer Security (DTLS) to secure communications. Use CoAP when HTTP is too bandwidth intensive. Keep in mind that CoAP’s market adoption is not as large as HTTP, so it may limit your software and hardware options. There are solutions for converting CoAP messages to and from HTTP that make CoAP solutions more interoperable.

DDS

Data Distribution Service (DDS) is a publish/subscribe protocol that’s focused on communication at the edge of the network. DDS is an open standard managed by the Object Management Group (OMG). Unlike MQTT which requires a centralized broker, DDS is decentralized. DDS nodes communicate directly in peer-to-peer fashion using UDP multicast. This removes the need for centralized network management and also makes DDS a faster protocol, reaching sub-millisecond resolution. DDS is a good solution for reliable, real-time data delivery at the edge. Use it for fast M2M communications. DDS supports brokers to integrate DDS networks with the enterprise, but in practice it is not well positioned as the integration point between industry and IT as brokers are often secondary to the DDS network.

AMQP

Advanced Message Queuing Protocol (AMQP) is another publish/subscribe protocol that comes out of the financial services sector. It has a presence in IT, but a limited presence in industry. AMQP’s biggest benefit is its robust communications model that supports transactions. Unlike MQTT, AMQP can guarantee transactions complete—which, though useful, is not always required by IoT applications. AMQP often gets grouped with IoT protocols and it is one—but its biggest con is that it’s a heavy protocol. It was meant for backend IT systems, and not the edge of the network.

Conclusion

OPC UA, HTTP, MQTT, CoAP, DDS, and AMQP all have a place in IoT. Which protocols take majority market share is unclear, but each has its pros and cons. It’s important to pick the protocol that best fits your needs, and select technology partners that can adapt to these protocols. This will ensure the success of your IoT applications and protect you from the protocol wars.
Exercises

1. Use the following scan-list information for a scanner in slot 2 of both a SLC and Compact processor:

<table>
<thead>
<tr>
<th>Device 2</th>
<th>Device 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 4</td>
<td>Device 3</td>
</tr>
<tr>
<td>Device 6</td>
<td>Device 5</td>
</tr>
</tbody>
</table>

Output Scan-List

<table>
<thead>
<tr>
<th>Device 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 3</td>
</tr>
<tr>
<td>Device 2</td>
</tr>
<tr>
<td>Device 4</td>
</tr>
</tbody>
</table>

Input Scan-List

Show the address of bit 3 of output device 3 for the SLC processor, for the CompactLogix processor.

Show the address of bit 5 of input device 4 for the SLC processor, for the CompactLogix processor.

2. Use the following scan-list information for a scanner in slot 6 of both a SLC and Compact processor:

<table>
<thead>
<tr>
<th>Device 2</th>
<th>Device 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>Device 3</td>
</tr>
<tr>
<td>Device 4</td>
<td>Device 5</td>
</tr>
</tbody>
</table>

Output Scan-List

<table>
<thead>
<tr>
<th>Device 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
</tr>
<tr>
<td>Device 2</td>
</tr>
<tr>
<td>Device 3</td>
</tr>
</tbody>
</table>

Input Scan-List

Show the address of bit 5 of output device 6 for the SLC processor, for the CompactLogix processor.

Show the address of bit 5 of input device 4 for the SLC processor, for the CompactLogix processor.

Use the following information for the Modbus Protocol Problems below:

**Description of "02" Read Input Status**

The "02" request reads the status of discrete input points. The request is for inputs 197 to 218 from slave device 17.

**Query**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Fig. 16-35</th>
</tr>
</thead>
</table>

Ch. 16 Networks and Protocols 33
### Description of "04" Read Registers

This is a request to read input register 30009 from the slave device at location 17.

### Query

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave Address</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>04</td>
</tr>
<tr>
<td>Starting Address Hi</td>
<td>00</td>
</tr>
<tr>
<td>Starting Address Lo</td>
<td>08</td>
</tr>
<tr>
<td>No. of Points Hi</td>
<td>00</td>
</tr>
<tr>
<td>No. of Points Lo</td>
<td>01</td>
</tr>
<tr>
<td>Error Check (LRC or CRC)</td>
<td>----</td>
</tr>
</tbody>
</table>

### Response

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave Address</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>04</td>
</tr>
<tr>
<td>Byte Count</td>
<td>02</td>
</tr>
<tr>
<td>Data Hi (Register 30009)</td>
<td>FF</td>
</tr>
<tr>
<td>Data Lo (Register 30009)</td>
<td>F1</td>
</tr>
<tr>
<td>Error Check (LRC or CRC)</td>
<td>----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>
### For the following Modbus query, describe the slave’s response:

<table>
<thead>
<tr>
<th>30001</th>
<th>A123</th>
</tr>
</thead>
<tbody>
<tr>
<td>30002</td>
<td>0102</td>
</tr>
<tr>
<td>30003</td>
<td>BBE3</td>
</tr>
<tr>
<td>30004</td>
<td>30E3</td>
</tr>
<tr>
<td>30005</td>
<td>0001</td>
</tr>
<tr>
<td>30006</td>
<td>4E23</td>
</tr>
<tr>
<td>30007</td>
<td>8989</td>
</tr>
<tr>
<td>30008</td>
<td>234F</td>
</tr>
<tr>
<td>30009</td>
<td>FFF1</td>
</tr>
</tbody>
</table>

3. For the following Modbus query, describe the slave’s response:

03
02
00
C4
00
0C
CRC

4. For the following Modbus query, describe the slave’s response:

05
04
00
03
00
04
CRC

5. For the following Modbus query, describe the slave’s response:

05
04
00
05
00
03
CRC

6. Find a device other than a Modicon device that uses the Modicon protocol to communicate with it.

7. Use the ASCII tables to find the checksum for the following:

A
2
3
;
8. For the following, an operator at an HMI station may push a button B3:0/4 to request a MSG Read block be executed. B3:0/4 is a multistate toggle button programmed to be either on or off from the HMI. Add logic to allow three automatic retries with a delay between retry of 4 seconds if an error occurs. Assume B3:0/4 stays on while the retries occur.

<table>
<thead>
<tr>
<th>MSG Type</th>
<th>peer-to-peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>Read</td>
</tr>
<tr>
<td>Target Device</td>
<td>500CPU</td>
</tr>
<tr>
<td>Local/Remote</td>
<td>Local</td>
</tr>
<tr>
<td>Control Block</td>
<td>N7:20</td>
</tr>
<tr>
<td>Control Block Length</td>
<td>14</td>
</tr>
</tbody>
</table>

9. Of PROFINET, Ethernet/IP, EtherCAT and Modbus TCP, which is/are unicast, multicast, shift register? What do the terms unicast and multicast mean?

10. In the following Automation Weekly magazine, circle the articles concerned primarily with the topic of PLC communication:
Non-Proprietary Controllers-to-Controller Communications

By Bill Lydon for Automation.com

The improvement of manufacturing productivity and quality requires sharing data and synchronizing controllers. This can now be accomplished by using new standards that were jointly developed by PCDopen and the OPC Foundation.

Otto Mation Comic Caption Contest

Submit a funny caption for this comic. If your caption is chosen as the winner, we will send you an Automation.com mug.

Learn more about Otto and view the comic archive.

INDUSTRY NEWS

VIEW MORE>>

Manufacturers moving to Mexico

Mexican manufacturers are the second-largest importers of U.S. packaging and processing machinery, because companies like Nestlé and PepsiCo are expanding manufacturing operations there.

Endress+Hauser sales reach $2.43 billion

Endress+Hauser Group increased its net sales by approximately 7 percent to over 1.8 billion euros ($2.43 billion) in the 2013 financial year.

Relden supplies EtherCAT® firewall to Schneider Electric

Schneider Electric added EtherCAT® Deep Packet Inspection (DPI) to the Connexum Tofro Firewall to further harden industrial control systems against network incidents and cyberattacks.

Fieldbus Foundation 2014 General Assembly Nixed


9TH Sensor Technology Joins Ballard Group

As development and technology partners in the area of specialty photovoltaic sensors, the companies are connected by a long-standing cooperation venture.

North American Robotics Shipments Grow in 2013

Following a strong year in 2012, the North American robotics market recorded its best year ever in 2013 in terms of robot shipments, according to the Robotic Industries Association.

ISA to Conduct Four Training Events

ISA will conduct four highly intensive technical training events—comprised of technician, automation engineering and safety courses—in Denver, Houston, and Carson, California during 2014.

ISA Food and Pharmaceuticals Symposium March 5-6

The 2014 ISA FFID Symposium will be held 5-6 March 2014 at ISA headquarters in Research Triangle Park, North Carolina, USA.

Endress+Hauser donates instrumentation to Polk State College

Through the donations of Endress+Hauser, Rockwell Automation and TriNova, Polk State College will be the new home to a $1 million, state-of-the-art PTU (Process Training Unit).

Codenomicon joins ISA Security Compliance Institute

Codenomicon, a Finland-based cybersecurity company, specializes in identifying unknown software bugs.

Hinreck Suite to Supply Fire & Gas Protection System for Oil Platform

The HImax Programmable Electronic System fire & gas detection system will be installed on the Killkade Oil Platform in UK North Sea.

Rootstock to supply ERP to Northeast Lantern

Northeast Lantern selected Rootstock’s Cloud Manufacturing ERP to track inventory and have vision on its production flow on the floor.

Sponsored by: Software Toolbox

New Release - CSV Import/Export Support and More

CSV/Import/Export of devices, topics, items and more for easier edits on large protocols. Import protocol/configuration files. FREE WEBINAR.

Ch. 16 Networks and Protocols 37
Strategic advice for investing in data analytics, the cloud, and the Internet of Things (IoT), and how a holistic strategy can even improve your PLC programming

What is digital manufacturing, and how can manufacturers implement the technology to meet their objectives? During a recent engineering forum discussion, Jeff Hall, a Siemens global account manager, defined digital manufacturing as follows: “It is using new technologies such as data analytics, the cloud, and the Internet of Things (IoT) to merge the virtual and real worlds. It enables manufacturers to increase productivity across their entire value chain, from design and engineering to sales, production, and service. In concrete terms, this means faster time-to-market, greater flexibility, and enhanced availability of systems on the plant floor.”

Imagine these scenarios:

- A technician from a system integrator sits on an overturned bucket next to an open controller cabinet with a laptop balanced on his knees. He’s been sitting there sweating for the last six hours trying to fix programming problems on an assembly line that his company built. He’ll be there again tomorrow and maybe the day after to solve the issue, costing his company valuable time and money.

- A plant manager just finished a project to rearrange a production line in his plant in an effort to create more manufacturing flexibility. On paper the floor plan looked good, but it was all done from old, two-dimensional specifications. In the real-world application, some of the machinery doesn’t fit together very well.

Digital manufacturing offers solutions that can mitigate or even eliminate these types of concerns—and many others. For manufacturers, it means the ability to perform a wide variety of manufacturing operations, from product design through production, including production equipment design, programming, and plant layouts. With a digital manufacturing strategy, engineers can perform a wide variety of tasks, from virtual prototyping to machine design and PLC programming on a workstation, with much of the tedious work automated.

Digital Manufacturing in the Real World

Those involved in different manufacturing positions and sectors are likely to emphasize specific areas of digital manufacturing based on their experience and specific job responsibilities. While digital manufacturing includes a range of technologies that have been evolving during the last 20 to 30 years, each individual begins with a different starting point. For the most part, all the
capabilities contained under this umbrella developed in silos. Only recently have manufacturers realized the benefits of connecting and integrating the various parts.

A consensus is emerging within manufacturing that supports Hall’s idea: digital manufacturing means a series of processes capable of encompassing the entire manufacturing lifecycle, from the earliest product design work (using virtual modeling, prototyping and simulation) to automated manufacturing and assembly, and beyond to field service.

All aspects of the process are connected and taken into consideration at each stage. For example, those involved in service can have input into the product’s initial design to ensure that repairs in the field are not hampered by the configuration. In fact, the digital manufacturing concept can extend to designing the manufacturing process and facility itself. Even controller programming can be built and simulated in a virtual environment. The product is designed for manufacturing efficiency in a given facility, and placement of the machinery in a plant can be analyzed using computer modeling.

**Programming Flexibility**

One aspect of the discussion that came heavily from the system integrator community related to writing programs for controllers and PLCs driving manufacturing machinery. To them, the ability to simulate how a new piece of equipment will perform when installed at the customer’s site is a huge advantage. Creating the models necessary to perform such a simulation can be time consuming and load more cost into the early stages of a project, but the savings at the back end can be huge.

The first scenario above was described by an integrator. He talked of many hours spent in customers’ facilities sitting on a bucket, laptop on his knees, connected to a PLC that his company had programmed, trying to sort through problems with the ladder logic. He shared how much better it was to write programs in the comfort of his office, sleeping in his own bed at night, rather than being sent to the field to fix things. By and large, his experience is not unique, as it is getting increasingly difficult to get young engineers to spend days or weeks at a time in the field. Digital manufacturing provides opportunities to run that machine while it’s still on the computer screen, making sure all the code works as expected. Virtual commissioning has become far more practical, rapidly lessening the number of changes that must be performed on site, and reducing the time to startup.

Of course, accurate simulations require reliable data and analysis; the ability to do good simulations is directly related to the ability to generate good models.

Recall the second scenario above, where a company struggled with creating a virtual plant layout because designers only had two-dimensional information on the equipment involved. While the advantages of exploring a variety of layout iterations on a screen rather than trying to move conveyors and robots is apparent, it’s clear that accuracy of the models is paramount.

**The Function of Data in Legacy Integration**

For most manufacturers, interfacing with legacy equipment is a major concern. New machines may use all the advantages of digital design and simulation, but they have to work in an environment where the next manufacturing section upstream or downstream may be 20 years old and was designed using traditional, two-dimensional techniques. Digital
manufacturing integrates legacy systems with advanced technologies by using the available data, and extending the new capabilities to the older equipment.

The solution begins by looking at the available data, comparing it with the data that is available, and then delivering the data. As long as the data flow is seamless, the rest is easy to resolve. Also, programming code from older machines can be brought into a simulator and connected virtually to the new machinery. Such an approach makes integration far easier.

But using data for machine-to-machine integration is only part of the picture. Process improvements depend on having data to indicate how things are working and to identify weaknesses and bottlenecks. The number of connections between machinery on the plant floor and enterprise-level networks is also growing, which increases the need for connectivity using IT techniques. This is far easier with the current generation of controllers. In addition, manufacturing management wants cost information, and the demands for more granular data are increasing dramatically.

**Making the Case for Investment**

Making the case to invest in digital manufacturing is a universal challenge. Finance people in large manufacturing companies can be hard to convince when it comes to implementing new technologies. These decision makers expect a proven track record before approving purchases. Since digital manufacturing is new for so many companies, it is difficult to show where and how the technology has saved money in ways the company will find compelling. Yet most engaged in the discussion agreed that this situation is changing: the advantages of digital manufacturing are beginning to filter through to even the most hard-nosed financial managers. The potential for significant savings, even without many years of cost history, makes for a compelling argument. One of the participants (from one of the “Big Three” automotive manufacturers) described making the case for investment in digital manufacturing as “building on a three-legged stool.” The first leg is cost reduction and cost avoidance, the second is product quality improvement, and the third is flexibility.

Multiple product platforms are being built on the same production lines to keep up with customer demand, and digital manufacturing is a key driver of that flexibility. Changes can be made quickly and with minimal disruption because all aspects of the process, from design to customer service, are connected and integrated. When a component is redesigned and prototyped virtually, it can be produced and moved into the larger assembly seamlessly. All documentation can be updated automatically and related manufacturing operations, even down to specific PLC programming, can be adjusted to make the implementation essentially invisible.

**Closing the Loop**

The head of a system integration company offered a good summary of digital manufacturing’s capabilities: it provides data able to support improvements as a closed-loop system. Every time there is an advance in one area, it can loop back to the beginning of the process and launch the next improvement. When products can be designed, prototyped, and tested virtually, the possibilities are endless. When manufacturing processes are changed on a screen, new approaches can be tested and verified without disrupting any existing production. These changes can be documented automatically, eliminating traditional steps.
John Billings, vice president of Siemens Digital Factory U.S., has a similar view: “The hallmark of advanced manufacturing is taking a holistic approach from product design through the rest of the phases of the manufacturing process. This is how we’ll get to a point of autonomous, self-correcting production processes. The linchpin of this approach is the data. Globally, you can make better decisions because you’re no longer guessing about what’s going on in your operations.”

Such improvements are being realized today in an expanding range of manufacturing sectors.
A Q&A with the author of the ISA book:


ISA recently published the second edition of *Industrial Automation and Control System Security Principles* by Ronald L. Krutz, Ph.D., P. E., CISSP, ISSEP, Chief Scientist at Security Risk Solutions, Inc. The title of the second edition—*Industrial Automation and Control System Security Principles: Protecting Critical Infrastructure*—was expanded and updated to reflect the latest advances in industrial automation and control system (IACS) security. IACS serve as the operational underpinnings of critical infrastructure, such as power generation, water treatment, petroleum and chemical processing, and other vital operations.

In this Q&A feature, Dr. Krutz highlights the importance of the book’s new and enhanced content.

*Note: A brief author biography is included at the bottom of this page.*

Ronald L. Krutz

Q. Why were you compelled to publish an updated edition? What differentiates the second edition from the initial version?

A. I wanted to cover the latest thinking and approaches to industrial automation and control system (IACS) security. This new edition addresses the most recent, formal methods and their practical applications to IACS security. The book is able to describe the latest advances in cybersecurity and critical infrastructure protection from industrial, governmental, and commercial sources, and show how they can be practically applied to protect IACS.

Q. Could you outline, in specifics, the new and enhanced areas of content in the second edition?

A. The second edition of my book contains a significant amount of new and enhanced content. This was needed to cover and describe all the significant technologies and methodologies that have been developed since the publication of the first edition.

There is an entirely new chapter, Chapter 9, on emerging approaches to industrial automation and control system security. The new content includes such topics as the Internet of Things (IoT), the Industrial Internet of Things (IIoT), the Open Platform Communications Unified Architecture (OPC UA) (IEC 62541), Industry 4.0, the OWASP “Internet of Things Top Ten” security categories, Big Data Analytics, the NIST Big Data Interoperability Framework, the NIST Framework for Cyber-Physical Systems, the NIST Framework for Improving Critical Infrastructure Cybersecurity, and Software-Defined Elements.


Chapter 5 has been updated to include coverage of the latest attacks on critical infrastructure systems. In addition to Stuxnet, the overview of malware includes the Shamoon Trojan Horse, Flame modular computer malware, the Norway cyberattack, and Havex.

Chapter 8 includes updated coverage of NIST SP 800-1371, “Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations;” in applications to Industrial Automation and Control Systems, The Smart Grid Maturity Model (SGMM); and the Introduction to NISTIR 7628, “Guidelines for Smart Grid Cybersecurity.”
I also have added a new appendix, Appendix B to the second edition. This new appendix comprises ICS Supplemental Guidance for NIST SP 800-53 Security Controls.

The new and updated chapters also include revised end-of-chapter review questions.

Q. What areas of new and enhanced content would you particularly want to highlight and encourage readers to focus on?

I point out the following sections and topic areas as being particularly valuable and informative.

- Industrial Internet of Things (IIoT)
- The Open Platform Communications Unified Architecture (OPC UA) (IEC 62541)
- Industry 4.0
- Big Data Analytics
- The NIST Big Data Interoperability Framework
- NIST Framework for Cyber-Physical Systems
- NIST Framework for Improving Critical Infrastructure Cybersecurity
- NIST Special Publication 800-82, Revision 2 “Guide to Industrial Control Systems Security”
- NIST Special Publication (SP) 800-53 Revision 4, “Recommended Security Controls for Federal Information Systems”
- Coverage of latest IACS malware
OPC UA and IEC 61131-3

Unprecedented integration of control and HMI

By Gary L. Pratt, PE, and Timothy L. Triplett, PE

In the beginning…the world was flat. Or at least the industrial control system (ICS) programming namespace portion of the world was flat. In the 1970s when systems consisted of only a small number of tags, tag names could be simple (like T2). However, as systems grew in the 1980s, tag naming quickly became unwieldy. Engineers first began to add pseudohierarchy to names by embedding underscores (like M123_T2). Then, in the 1990s, data structures (i.e., user-defined data structures [UDTs]) were introduced to the ICS programming world and became very popular over the next decade. With data structures, tags could now be structured, and multiple instances could be differentiated with the “dot” convention (M123.T2). However, this still required creating and instantiating structures and copying values into and out of these structures. In this decade, new standards allow direct access to function block hierarchical I/O, eliminating the need for UDTs, tags, and copying data.

Similarly, in the beginning, there was ladder logic. It was great for representing electrical equipment and simple discrete logic. However, as programming size and complexity increased, the choice of industrial control languages offered by controller vendors did not keep pace. As a result, ladder logic was recruited into purposes for which it was never intended and was poorly suited. Fortunately, the latest standards have programming languages and techniques that fill this gap and give 21st century industrial control system (ICS) programmers the tools they need to produce large, scalable, and maintainable programs—and allows ladder logic to return to the purpose for which it is best suited.

Just as UDTs transformed the 1990s, new features in OPC UA released in 2008 and IEC 61131-3 released in 2013 are transforming application programming in this decade. The new capabilities provided by these standards deliver unprecedented integration of control and the human-machine interface (HMI).

One of the most powerful features of IEC 61131-3 is its ability to nest function blocks (FBs) to any arbitrary width and depth using any of the IEC 61131-3 languages, and then to easily navigate the hierarchy by simply double clicking on any block to pop into its underlying code. This feature allows the ICS engineer to create a precise hierarchical representation of the plant and build each function within the plant in the best language for the task. For instance, engineers can use Continuous Function Chart (CFC) for high-level block diagrams, Sequential Function Chart (SFC) for state-based control, Ladder Diagram (LD) for discrete logic, and Structured Text (ST) for complex math, conditionals, looping, and bit manipulation.

The IEC 61131-3 CFC graphical language is a great tool for building a representation of the plant hierarchy. Typically, this begins with a single top-level block diagram of the plant called the plant view (PV), which instantiates additional subsystem PV block diagrams as necessary and ends with the control-and-equipment (C&E) view diagrams. The
C&E view shows the complete control of a subsection of a plant with input equipment on the left, the control in the middle, and the output equipment on the right.

Within the C&E view, equipment models can be written in LD or ST, and typically deal with scaling, alarming, signal quality, latching, and manual override. The exact nature of the control block will depend on the type of control required. For instance, a process plant may use a CFC containing a startup sequence in SFC; proportional, integral, derivative (PID) from libraries; and other low-level control code written in ST. Control in a batch or discrete plant usually consists of an SFC describing the process sequence.
A typical multilevel hierarchical view is illustrated in figures 2 and 3. The plant consists of two levels of PVs, one level of C&E view, and several additional levels, each implemented in the language that is the best fit for the purpose. In this example, the PID and low-pass filters are from the OSCAT open-source industrial control library, and the block to integrate the incoming flow rate and compare it to the summation of the auger shaft-encoder pulses is implemented in ST. Imagine how simple this hierarchical multilanguage approach is for a plant technician to understand: drill down in the hierarchy to find the appropriate C&E view, examine the state of the control signals to determine if the problem is in the control or in the equipment, then push into that to diagnose the issue.

As alluded to earlier, a powerful benefit of the multiple languages of IEC 61131-3 is the ability to use the same tools for discrete, batch, and continuous process programming. In all types of programming, the plant-level views are similar, as are the input and output equipment in the C&E view. The only significant difference is the control block, which in a batch process is typically an SFC. Figure 4 shows a typical C&E view for a batch process with the control implemented in SFC, the temperature switch in ST, and the auger motor in traditional LD.
Obviously, an integrated control system is not complete without a seamless connection to its human-machine interface. Fortunately, the new OPC UA standard makes this seamless connection possible with its platform independence, encryption, full hierarchical browsing, and meta-tags. Platform independence allows the OPC server to be placed directly in the industrial controller hardware (eliminating the expense and security vulnerability of an OPC server PC), and encryption ensures the security of the data and control. Hardware vendors can use true random number generators, crypto-coprocessors, and deeply embedded root of trust to further secure the connections to both the programming software and the HMI. Programming and HMI connections can be made through the open Internet while remaining protected from cyberattack or mischief.
Figure 5 shows how OPC UA makes the entire hierarchy within the process plant available within an OPC UA tag browser (without explicitly connecting tags to objects or data structures within the ICS design or exporting tag lists). Within the control development environment, programmers can expose the entire namespace tree or select only certain branches. Tags can also be exposed directly in the code where their corresponding variable is declared (figure 6). The latter is especially handy for library parts with inputs and outputs that are intended to be used by the HMI.

```
FUNCTION_BLOCK LevelSensor
  VAR_INPUT
    FieldInput : REAL; //From field instrument.
    {attribute 'symbol' := 'readwrite'}
    OverRide : BOOL; //When True, input is "ManualInput"
    {attribute 'symbol' := 'readwrite'}
    ManualInput : REAL; //From HMI. Active when OverRide = TRUE
  END_VAR
  VAR_OUTPUT
    {attribute 'symbol' := 'read'}
    Level : REAL; //Output to program
  END_VAR
```

Although figure 5 illustrates how all the necessary data is available throughout the design hierarchy, we would never want to deal with that complexity manually. Fortunately, with OPC UA, the HMI can browse the server and create matching complex tags with drag-and-drop simplicity. And if an HMI project is defined with a library of the same base objects as the control design, OPC UA provides enough information for the HMI to automatically create all the complex tags and their structures based on those.

To carry out this automation, the HMI begins by examining the tree from the top. Where it encounters objects in the OPC UA tree that have a corresponding item in the coordinated library, it instantiates that library object. Where it encounters objects that do not, it creates a new folder. It then continues down the tree, either finding and instantiating library objects, or creating further new folders until the complex tag is fully defined and instantiated. All the tags are automatically connected as this process proceeds. At the end, all that is left for the HMI is to organize the visual presentation.

In addition, meta-tags can be added to the control function blocks to provide additional information to the HMI system for it to automatically perform much of the visual presentation adjustment. For example, meta-tags can differentiate the type of process equipment associated with a complex tag structure, determining the default image presented by the HMI.

Figure 7 shows how the project hierarchy in the HMI system matches the project hierarchy in the control design. Figures 8 and 9 show the HMI screens corresponding to the continuous process control design in figures 2 and 3. Notice how the connectivity between the entire ICS and entire HMI designs is made with just the top-level tag name. Thousands of tags below may be automatically connected based on the hierarchy of the design.
Figure 10 shows the corresponding screens for the batch control. Notice that the “ReactorSequence” block in the ICS library has a corresponding object in the HMI library that represents the current state of the process and allows the operator to manually override the process and select new active steps if an unusual situation occurs. Also notice that the HMI has pop-up screens for the motors in the process plant, and that these are all automatically created and connected based on the OPC UA hierarchy and associated library object templates.
Figure 11 shows how the same IEC 61131-3 modeling can be used to create a complete plant simulation, which allows control system designs to be error-free the first time. With development systems that include a complete run-time PC with embedded OPC UA server, ICS engineers can create their control project and HMI screens, and completely test the entire system on a laptop. This results in the confidence that the design is complete and correct before commissioning begins.
The features of IEC 61131-3, OPC UA, and the latest ICS and HMI systems greatly streamline the process of creating ICS and HMI designs. The process is simply:

1. Create an ICS design by instantiating items from the coordinated ICS/HMI library and user-created function blocks created from coordinated library objects.
2. Connect the HMI system to the OPC UA server and read in the design hierarchy.
3. Have the HMI system build a corresponding design using parts from the coordinated library and new subobjects.
4. Organize the visual aspects of the HMI screens.
5. Deploy the project.

The features in the IEC 61131-3 and OPC UA standards implemented in the latest ICS and HMI systems give automation system designers unprecedented integration capabilities. More than ever before, they can leverage best-in-class hardware and software to create larger, more scalable, more reliable, more maintainable, and more secure...
control systems. This stands as an example of how those who create and advance standards are paving the way for development of the tools that ICS programmers need for 21st century industrial control systems.

Merl,

The following must be set up to get the PID program to run properly with the HMI simulate mode:

Click on the SetPG/PC Interface box above:

We are going to use the Siemens program TIA Portal V14 in order to run the program given for the ball-in-tube lab. This program will be used to download the PLC program but not the HMI program.

Choose the third of the Broadcom choices. Click OK.
This allows the Siemens program to run the HMI program in simulate mode. Then download the program to the PLC. Do not download the HMI program since we do not have the HMI to download to.

Then click on the HMI’s Screen, Screen_1. Notice the Start Simulation button turn blue. It now allows the student to run the HMI via simulation mode from the screen of the pc.

We need to find the variables that need to be written to the historical data. Click on Laser Input and get the tag Laser_Percent. This tag is one to be written to the historical data logger.
Next, go to the Historical data tag under HMI tags. Set the variables from the HMI screen above that are to be saved. Fill in the appropriate fields and start the historical logger. This circular file will contain the data from the analog data saved. Then run the program and run the historical data logger. It is a circular file and will wrap around after the table fills up.