Chapter 16 NETWORKS and PROTOCOLS

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 devices. Other networking issues need to be discussed since the move of Ethernet from the business side of the factory to the factory floor has been around long enough for equipment to be accepted having Ethernet.

A historical network no longer in demand will be discussed briefly. This network – DeviceNet – was popular about 20 years ago as an industrial network. It had its origins in the automotive industry – CanBus – a network primarily initiated to replace wiring harnesses in automobiles. Most have moved on from it, however, and it is one of many networks in the boneyard of networks tried, overtaken by Ethernet and finally abandoned.

The chapter also includes an explanation of a protocol. The use of Modbus protocol is explained. Also introduced are techniques for communicating between devices. This includes PLC to PLC communication, PLC to RFID tag and PLC to Cognex Vision System.

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.

DeviceNet

DeviceNet is a relatively easy I/O network to install and configure. We can first discuss the initial bus structure - CanBus. Then the organization, ODVA, through which DeviceNet is governed, was created. The Allen-Bradley software RSNetworx for Devicenet explains the mapping of I/O to the processor. While this local network is on the wane today, it is an example of a recent attempt by the developers to address local area networks in the industrial environment.

CanBus

CanBus or CAN.bus is a two-wire differential serial bus. It does not need special tools to termiate. It is designed to operate in noisy electrical environments such as the automobile or the factory floor. 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? The short answer is that DeviceNet has withered and Ethernet has thrived. We will now discuss a few attributes of the DeviceNet hardware and software and then get on with the Ethernet portion.

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.

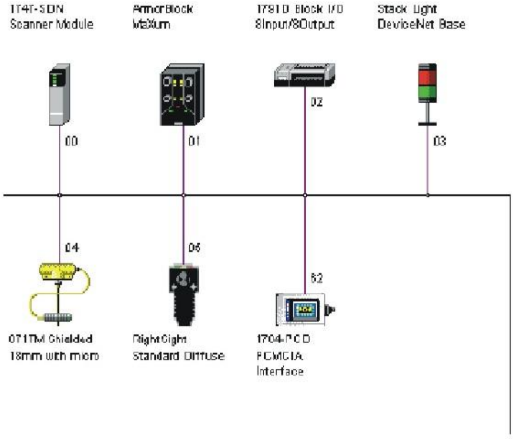


Fig. 16-1
Devicenet Network
Configured in the A-B
RSNetWorx Software

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 which allows only six nodes to be attached.

SLC 5/03

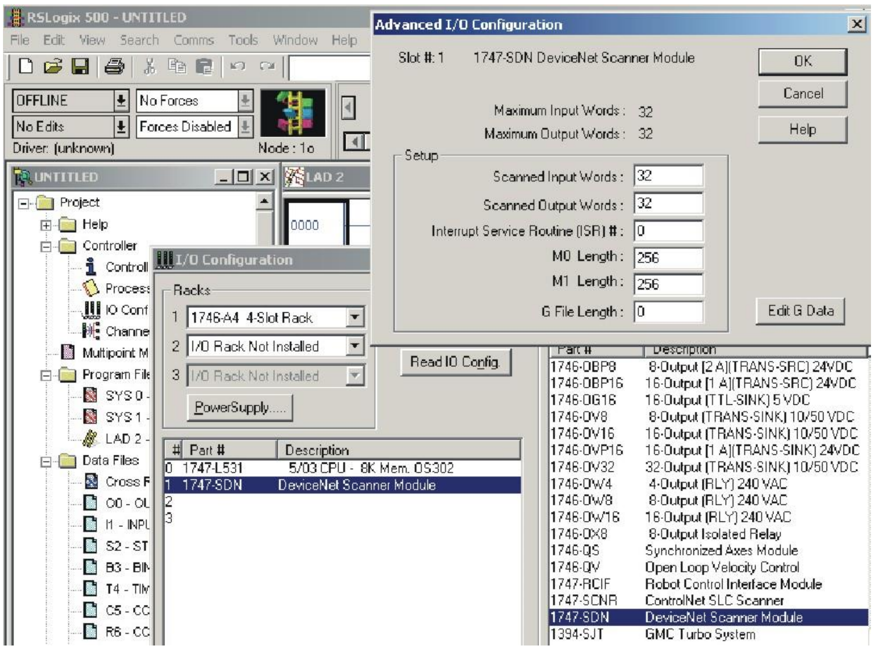


Fig. 16-2
Adding the DeviceNet
Scanner to the SLC 5/03

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

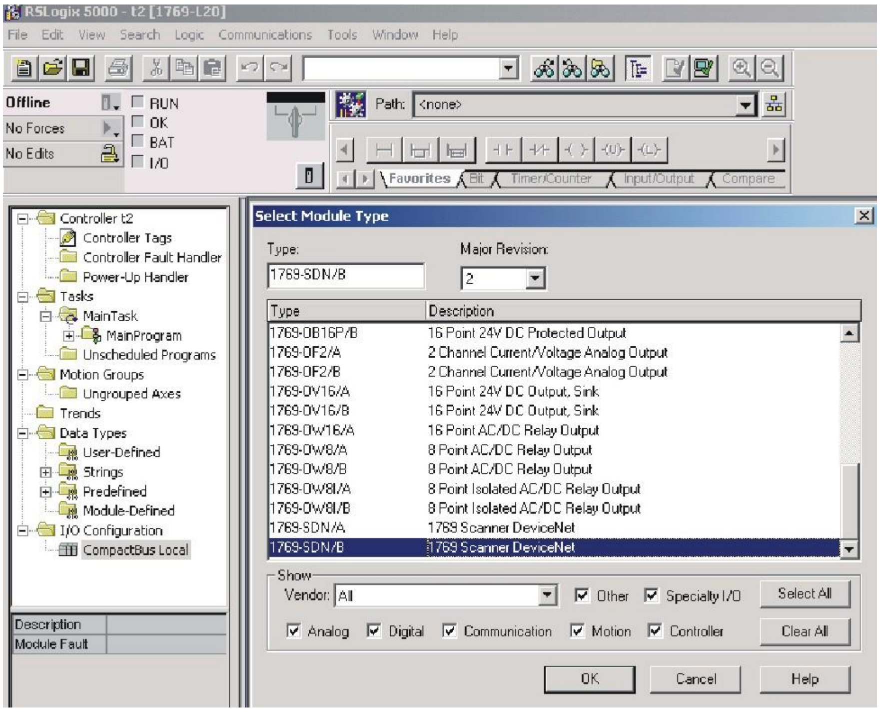


Fig. 16-3
Adding the DeviceNet Scanner to the CompactLogix Processor

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.

Output Scan-List

Device 2	Device 1
Device 4	Device 3
Device 6	Device 5

View in Scan List of RSNetworx for DeviceNet

Input Scan-List

Device 1	
Device 3	Device 2
	Device 4

Fig. 16-4

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.

Next we look at ethernet networks.

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:

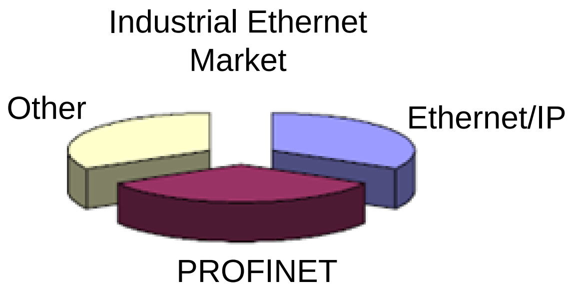


Fig. 16-5

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, its effective use in the marketplace has waned and Ethernet/IP has become 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).

Each of the networks listed provide an all-encompassing network for Industrial Automation:
Real-time I/O, Machine-to-machine Integration, Motion Control, Vertical Integration, Safety, Security, Integrates existing buses, Energy Savings.

Technology	PROFINET PROFIBUS	DeviceNet Ethernet/IP	Foundation Fieldbus	Modbus ModbusTCP	EtherCAT
Consortium	PI	ODVA	Fieldbus Foundation	Modbus IDA	ETG
Primary Backer	Siemens Phoenix Contact	Rockwell	Emerson	Schneider	Beckhoff
Founded	1989	1995	1994	2002	2004
Membership	1,400	290	350	65	1,195
Regional Organizations	26	4	4	0	3
Competence Centers	40 Certified	0	0	0	0
Test Labs	10 Certified	3	1	1	1
Training Centers	18 Certified	0	8	0	0

Fig. 16-6

A look back at the older Profibus which was similar to DeviceNet 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:	**PROFIBUS**	**PROFINET**
Physical layer	RS-485	Ethernet
Speed	12Mbits	100Mbits
Telegram	244 bytes	1440 bytes
Address space	126	2048
Technology	master/slave	provider/consumer
Connectivity	PA + others	many buses
Wireless	Possible	IEEE 802.11, 15.1
Motion	32 axes	150 axes
Mach to mach	No	Yes
Vert integration	No	Yes
No. of products	3,000	300

Fig. 16-7

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.

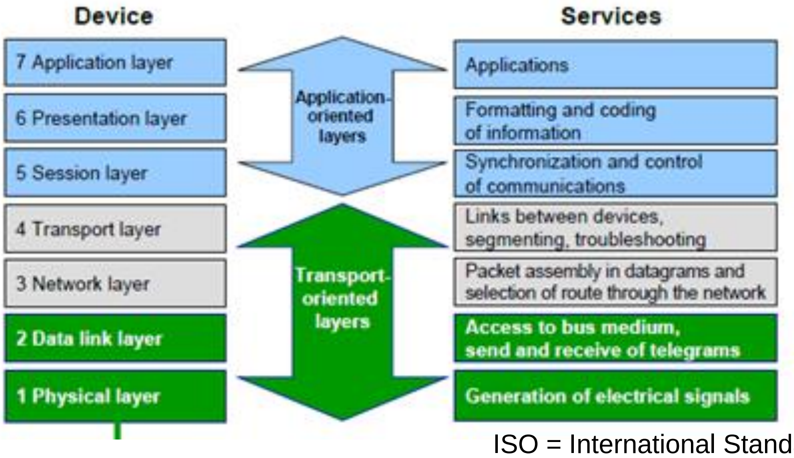
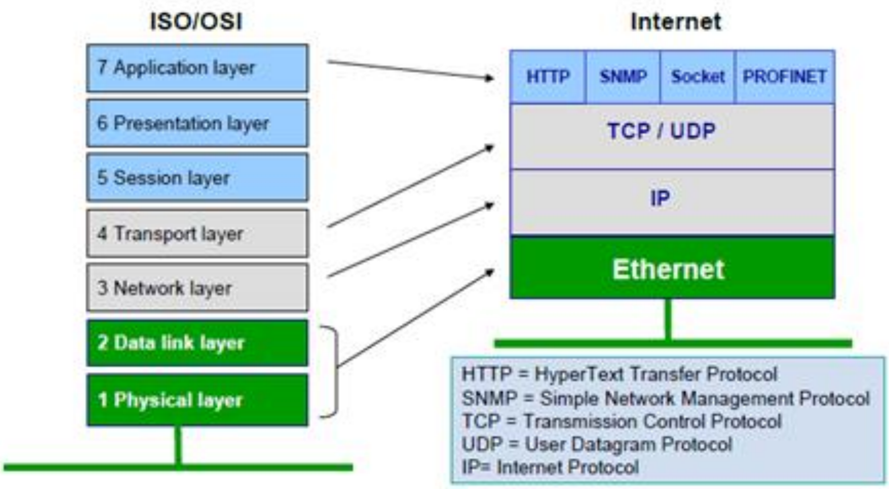


Fig. 16-8

ISO = International Standards Organization
OSI = Open Systems Interconnection



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

Communications Packet

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.

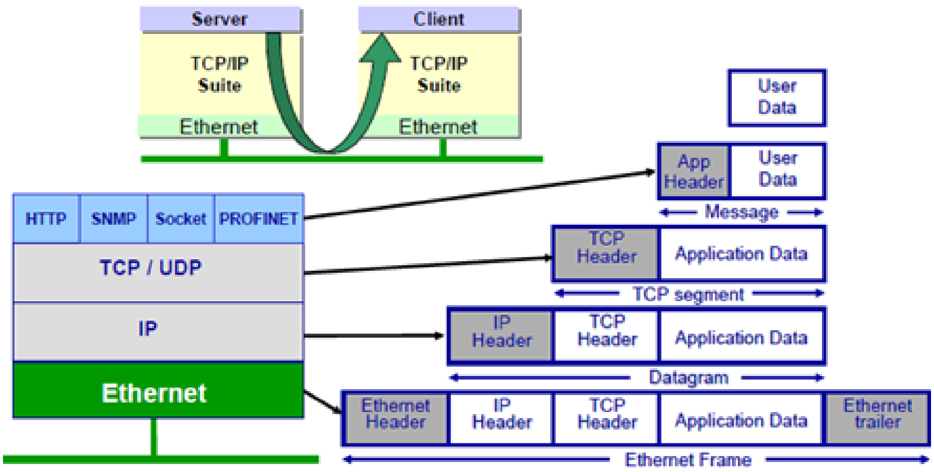


Fig. 16-9

Not all layers are needed:

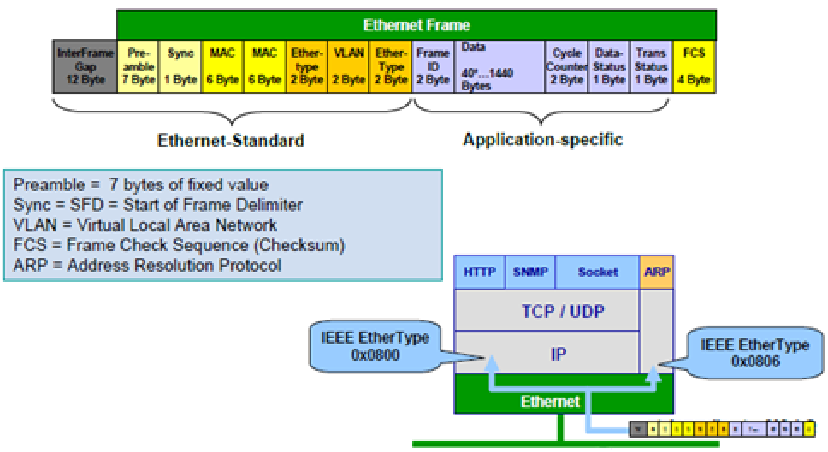


Fig. 16-10

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 PROFIBUS, use shielded cable with Ethernet as well!

Four steps to fast, deterministic communication:

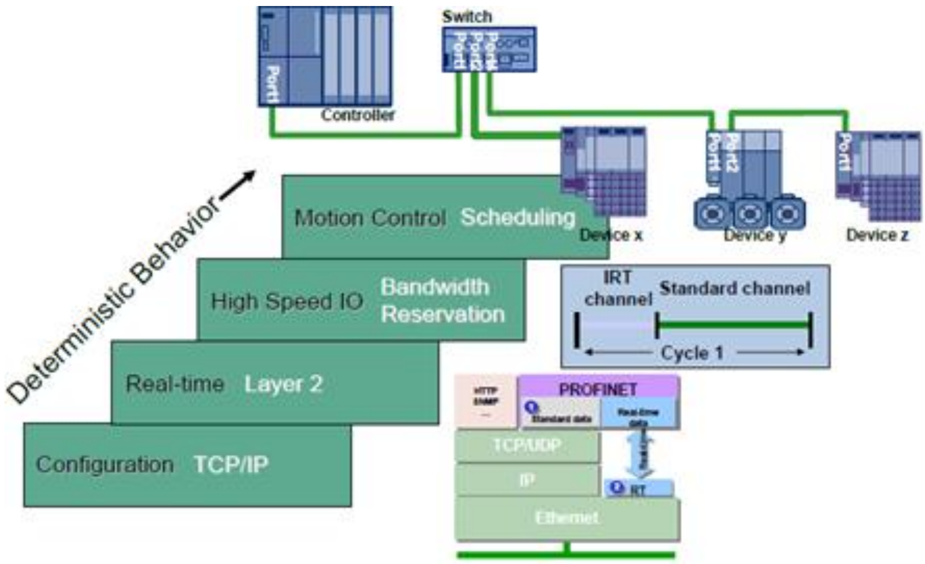


Fig. 16-11

Why not use TCP/IP for real-time?
Because it's not fast enough and it's not deterministic enough.

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.

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.

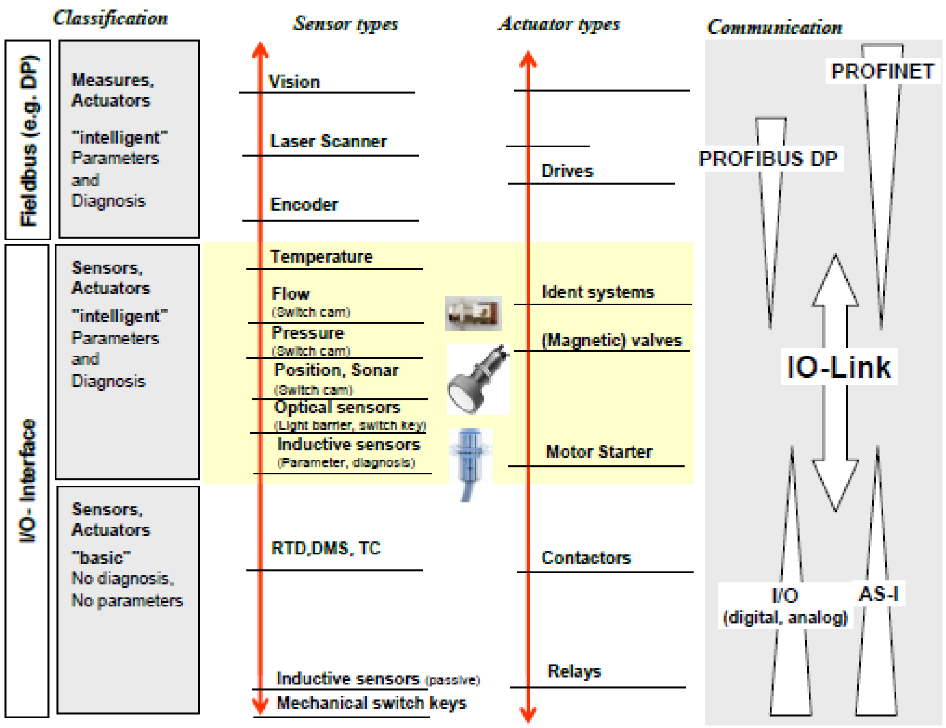
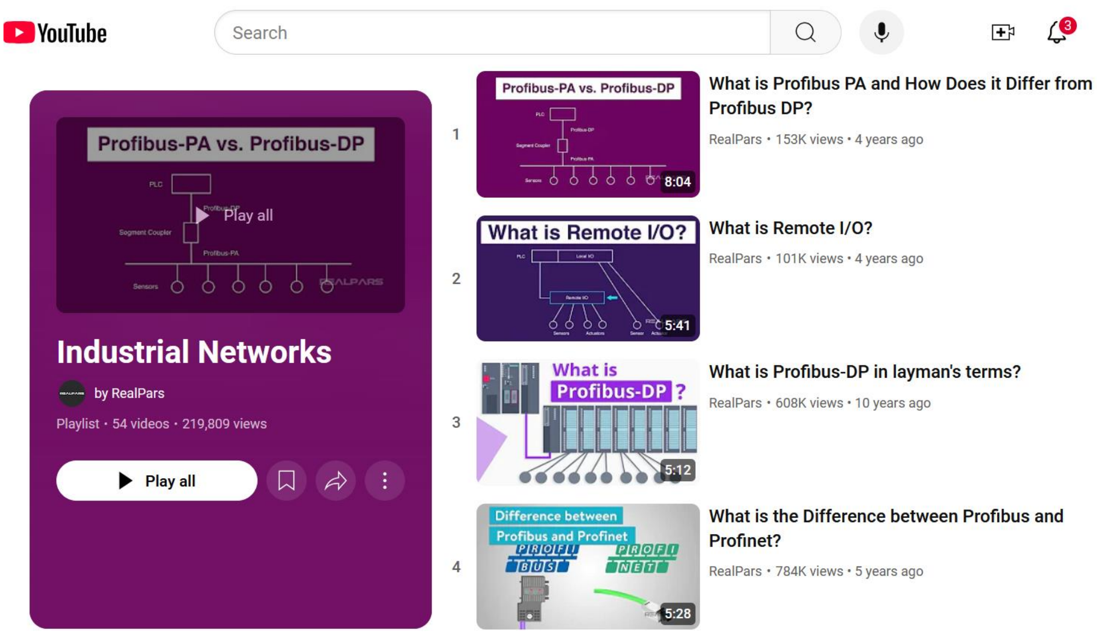
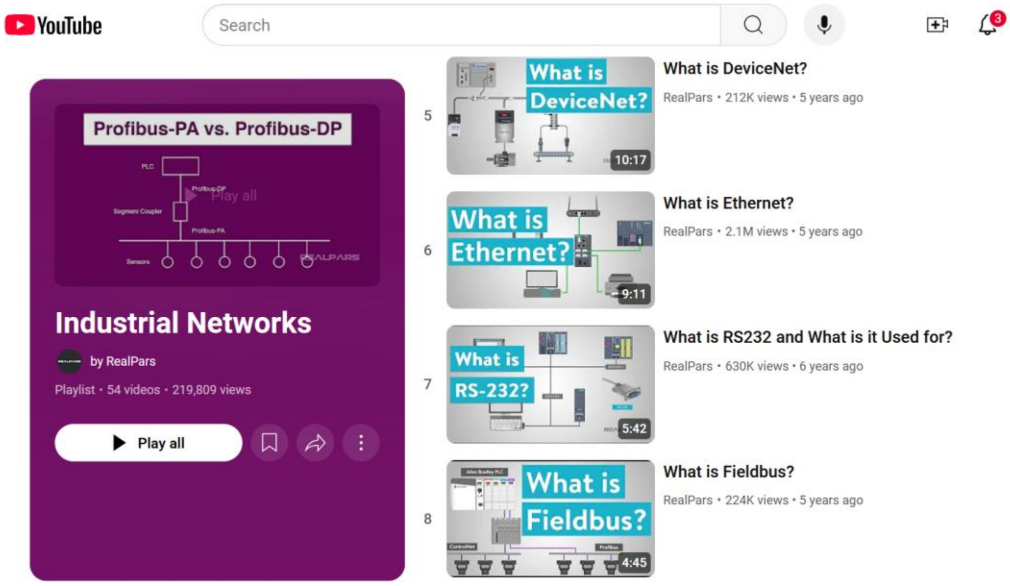
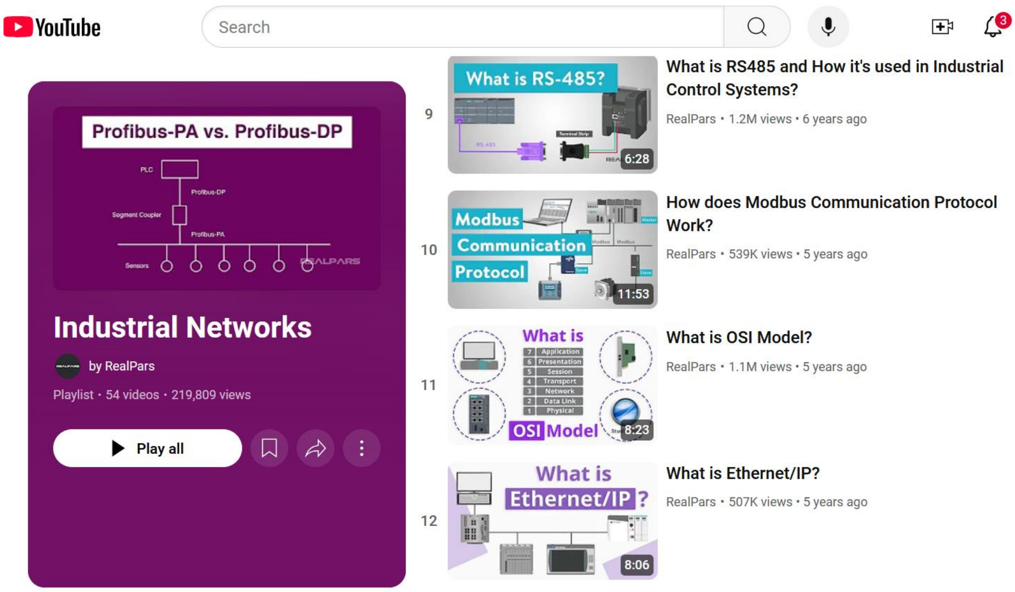


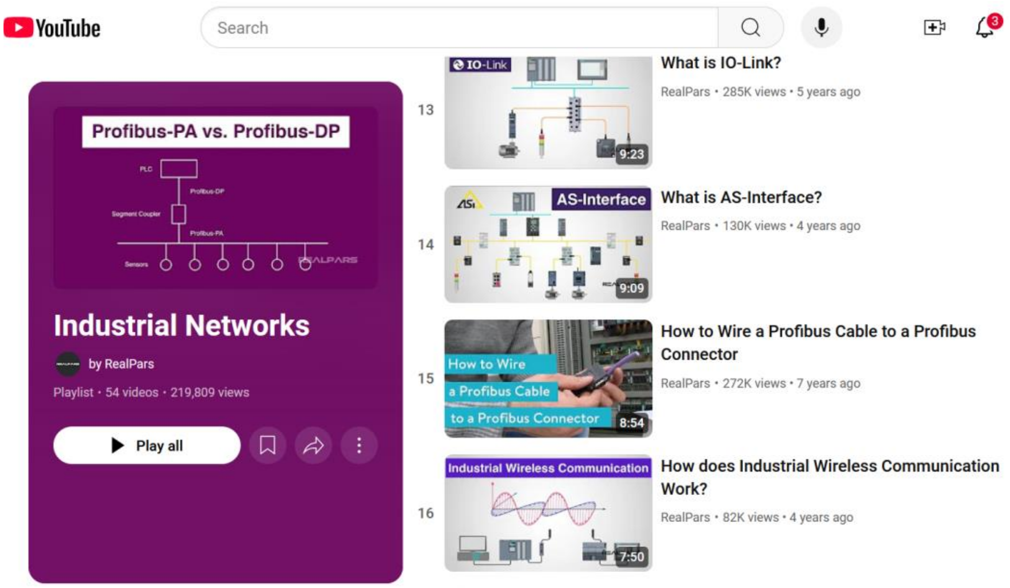
Fig. 16-12

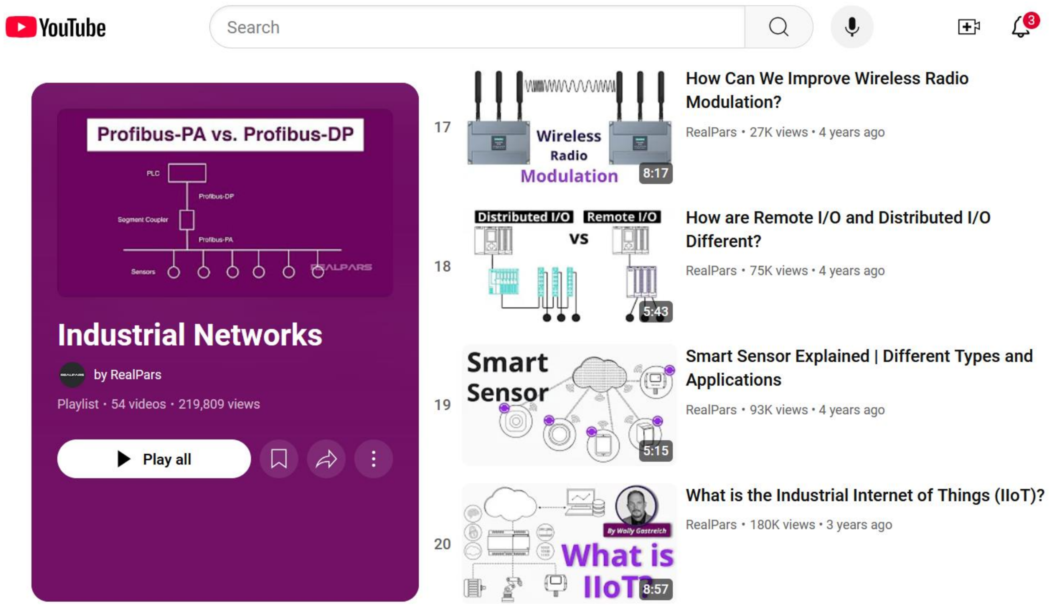
Industrial Networks from RealPar:

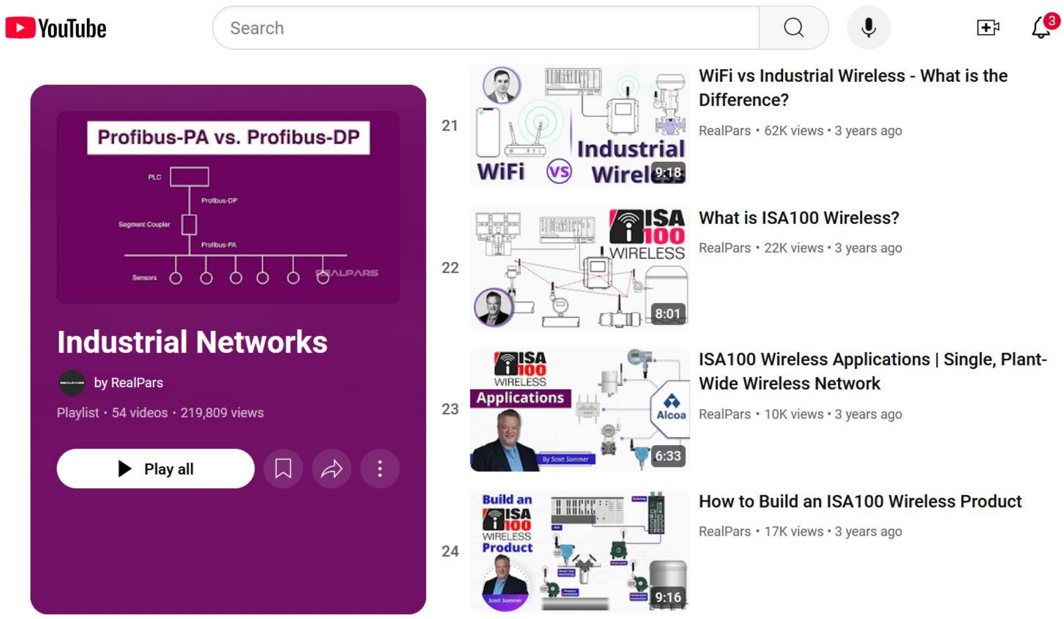


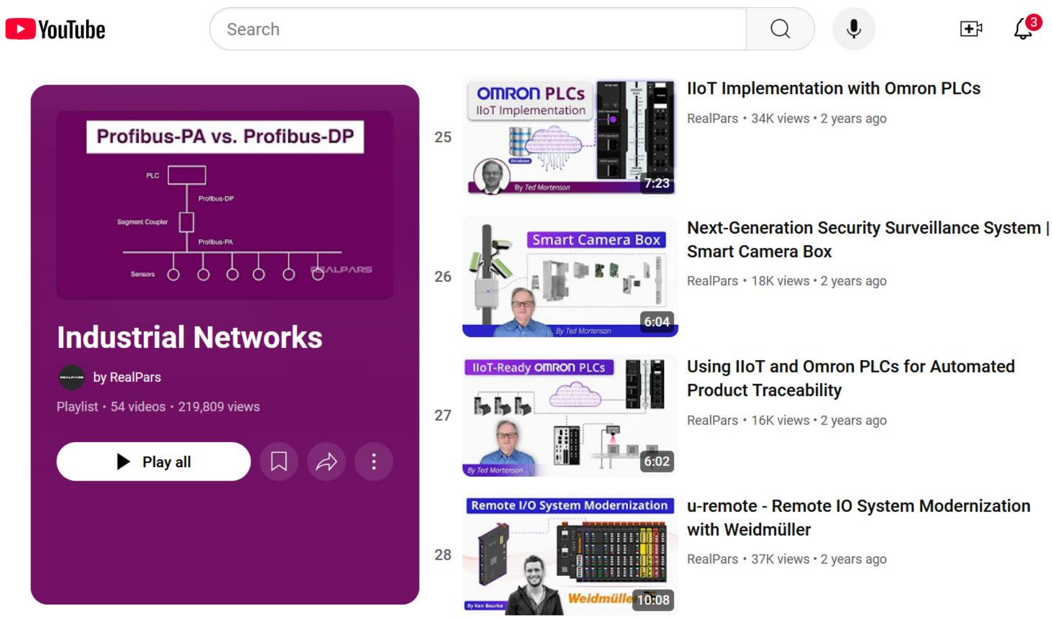

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
5
What is DeviceNet?
10:17
What is DeviceNet?
RealPars • 212K views • 5 years ago
6
What is Ethernet?
9:11
What is Ethernet?
RealPars • 2.1M views • 5 years ago
7
What is RS-232?
5:42
What is RS232 and What is it Used for?
RealPars • 630K views • 6 years ago
8
What is Fieldbus?
4:45
What is Fieldbus?
RealPars • 224K views • 5 years ago

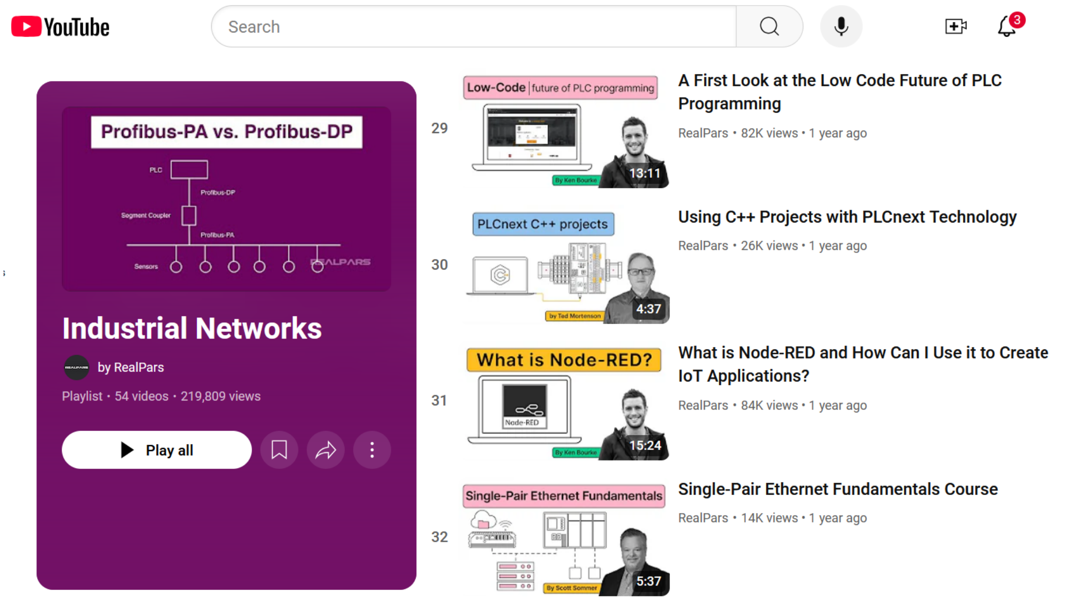

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
9
What is RS-485?
6:28
What is RS485 and How it's used in Industrial Control Systems?
RealPars • 1.2M views • 6 years ago
10
Modbus Communication Protocol
11:53
How does Modbus Communication Protocol Work?
RealPars • 539K views • 5 years ago
11
What is OSI Model
8:23
What is OSI Model?
RealPars • 1.1M views • 5 years ago
12
What is Ethernet/IP?
8:06
What is Ethernet/IP?
RealPars • 507K views • 5 years ago

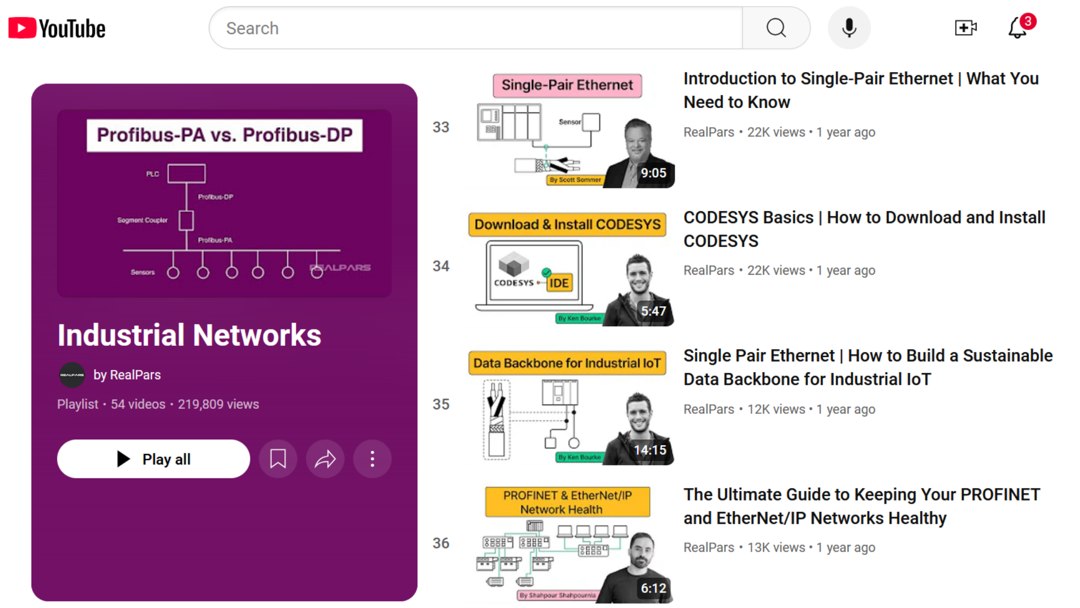

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
13
IO-Link
9:23
What is IO-Link?
RealPars • 285K views • 5 years ago
14
AS-Interface
9:09
What is AS-Interface?
RealPars • 130K views • 4 years ago
15
How to Wire a Profibus Cable to a Profibus Connector
8:54
How to Wire a Profibus Cable to a Profibus Connector
RealPars • 272K views • 7 years ago
16
Industrial Wireless Communication
7:50
How does Industrial Wireless Communication Work?
RealPars • 82K views • 4 years ago

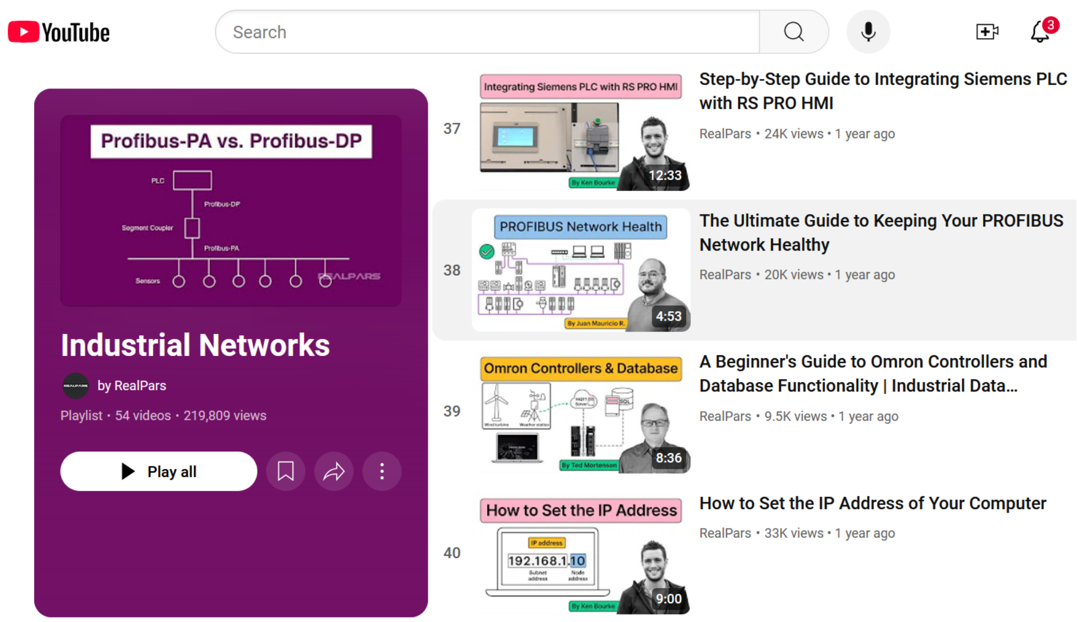

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
17
How Can We Improve Wireless Radio Modulation?
RealPars • 27K views • 4 years ago
18
How are Remote I/O and Distributed I/O Different?
RealPars • 75K views • 4 years ago
19
Smart Sensor Explained | Different Types and Applications
RealPars • 93K views • 4 years ago
20
What is the Industrial Internet of Things (IIoT)?
RealPars • 180K views • 3 years ago

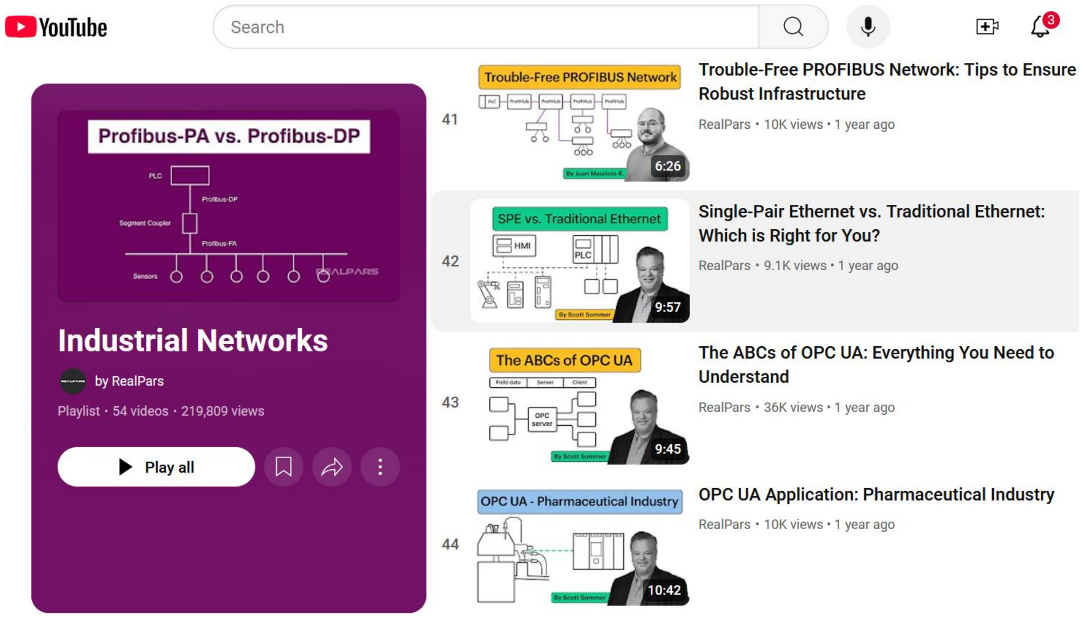

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
21
WiFi vs Industrial Wireless - What is the Difference?
RealPars • 62K views • 3 years ago
22
What is ISA100 Wireless?
RealPars • 22K views • 3 years ago
23
ISA100 Wireless Applications | Single, Plant-Wide Wireless Network
RealPars • 10K views • 3 years ago
24
How to Build an ISA100 Wireless Product
RealPars • 17K views • 3 years ago

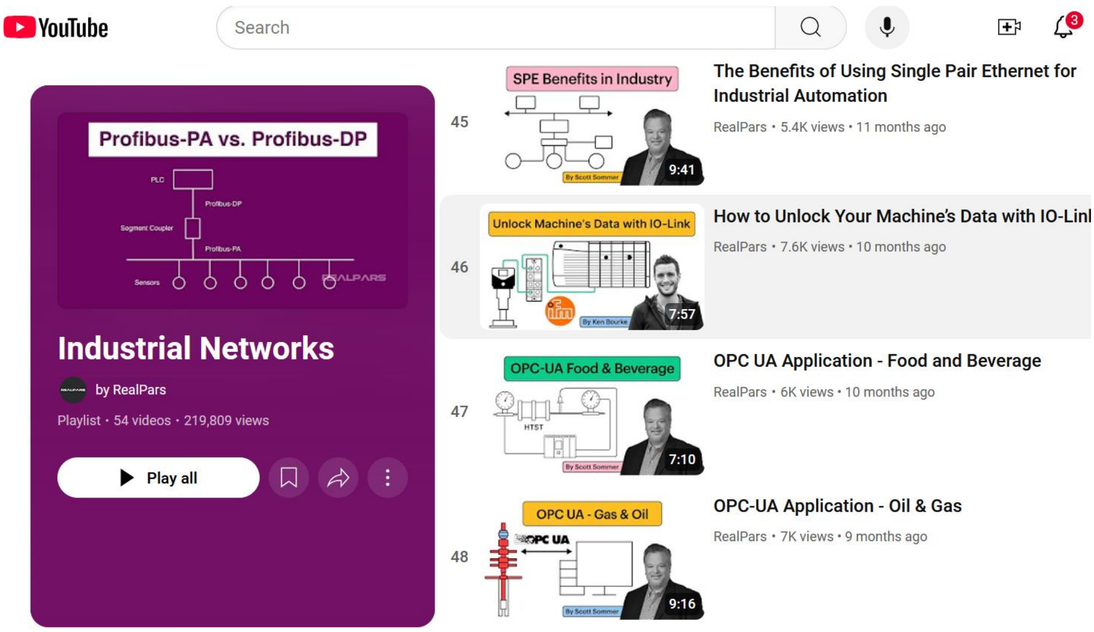

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
25
IIoT Implementation with Omron PLCs
RealPars • 34K views • 2 years ago
26
Next-Generation Security Surveillance System | Smart Camera Box
RealPars • 18K views • 2 years ago
27
Using IIoT and Omron PLCs for Automated Product Traceability
RealPars • 16K views • 2 years ago
28
u-remote - Remote IO System Modernization with Weidmüller
RealPars • 37K views • 2 years ago

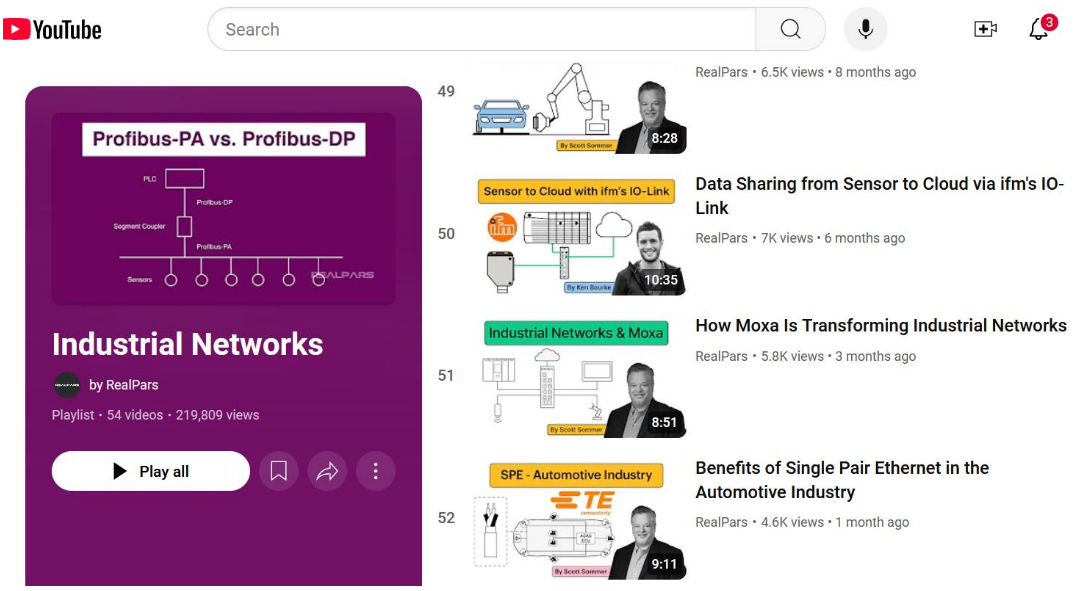

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
29
A First Look at the Low Code Future of PLC Programming
RealPars • 82K views • 1 year ago
30
Using C++ Projects with PLCnext Technology
RealPars • 26K views • 1 year ago
31
What is Node-RED and How Can I Use it to Create IoT Applications?
RealPars • 84K views • 1 year ago
32
Single-Pair Ethernet Fundamentals Course
RealPars • 14K views • 1 year ago

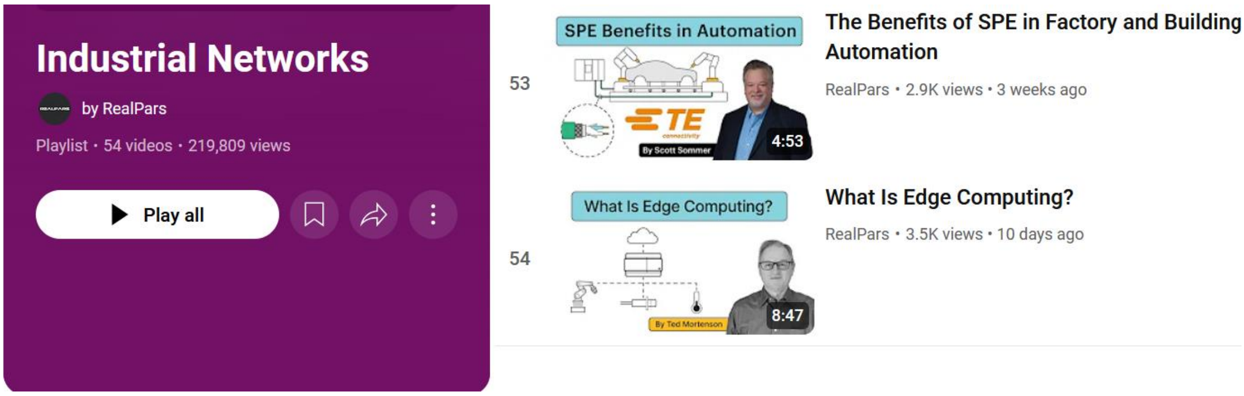

YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
33
Introduction to Single-Pair Ethernet | What You Need to Know
RealPars • 22K views • 1 year ago
34
CODESYS Basics | How to Download and Install CODESYS
RealPars • 22K views • 1 year ago
35
Single Pair Ethernet | How to Build a Sustainable Data Backbone for Industrial IoT
RealPars • 12K views • 1 year ago
36
The Ultimate Guide to Keeping Your PROFINET and EtherNet/IP Networks Healthy
RealPars • 13K views • 1 year ago


YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
37
Step-by-Step Guide to Integrating Siemens PLC with RS PRO HMI
RealPars • 24K views • 1 year ago
38
The Ultimate Guide to Keeping Your PROFIBUS Network Healthy
RealPars • 20K views • 1 year ago
39
A Beginner's Guide to Omron Controllers and Database Functionality | Industrial Data...
RealPars • 9.5K views • 1 year ago
40
How to Set the IP Address of Your Computer
RealPars • 33K views • 1 year ago


YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
41
Trouble-Free PROFIBUS Network
6:26
Trouble-Free PROFIBUS Network: Tips to Ensure Robust Infrastructure
RealPars • 10K views • 1 year ago
42
SPE vs. Traditional Ethernet
9:57
Single-Pair Ethernet vs. Traditional Ethernet: Which is Right for You?
RealPars • 9.1K views • 1 year ago
43
The ABCs of OPC UA
9:45
The ABCs of OPC UA: Everything You Need to Understand
RealPars • 36K views • 1 year ago
44
OPC UA - Pharmaceutical Industry
10:42
OPC UA Application: Pharmaceutical Industry
RealPars • 10K views • 1 year ago


YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
45
SPE Benefits in Industry
9:41
The Benefits of Using Single Pair Ethernet for Industrial Automation
RealPars • 5.4K views • 11 months ago
46
Unlock Machine's Data with IO-Link
7:57
How to Unlock Your Machine's Data with IO-Linl
RealPars • 7.6K views • 10 months ago
47
OPC-UA Food & Beverage
7:10
OPC UA Application - Food and Beverage
RealPars • 6K views • 10 months ago
48
OPC UA - Gas & Oil
9:16
OPC-UA Application - Oil & Gas
RealPars • 7K views • 9 months ago


YouTube
Search
Profibus-PA vs. Profibus-DP
Industrial Networks
by RealPars
Playlist • 54 videos • 219,809 views
Play all
49
8:28
RealPars • 6.5K views • 8 months ago
50
Sensor to Cloud with ifm's IO-Link
10:35
Data Sharing from Sensor to Cloud via ifm's IO-Link
RealPars • 7K views • 6 months ago
51
Industrial Networks & Moxa
8:51
How Moxa Is Transforming Industrial Networks
RealPars • 5.8K views • 3 months ago
52
SPE - Automotive Industry
9:11
Benefits of Single Pair Ethernet in the Automotive Industry
RealPars • 4.6K views • 1 month ago



Communication from PLC to PLC

Next outlined are various ways for PLCs to talk between each other. First Allen-Bradley:

A-B MSG Command – RSLogix 5000

The following manual describes the MSG command for the Compact Logix processors in the lab:

Rockwell Automation Publication 1756-PM012I-EN-P - September 2020

The following describe the setup and execution of the MSG command. Remember, it is only necessary to be placed in the processor desiring the information and not in both processors:
"

Introduction to Controller Messages

This section describes how to transfer (send or receive) data between controllers by executing a message (MSG) instruction. It explains cache connections and buffers so you can correctly program the controller.

Supported data types

The following data types are supported when sending CIP messages.

- SINT
- INT
- DINT
- LINT
- REAL

In addition, you can send a message with any structure type that is predefined, module-defined, or user-defined.

For more information, see "Convert between INTs and DINTs on page 15".

For complete details on programming a message instruction, see the LOGIX 5000 Controllers General Instruction Reference Manual, publication 1756-RM003.

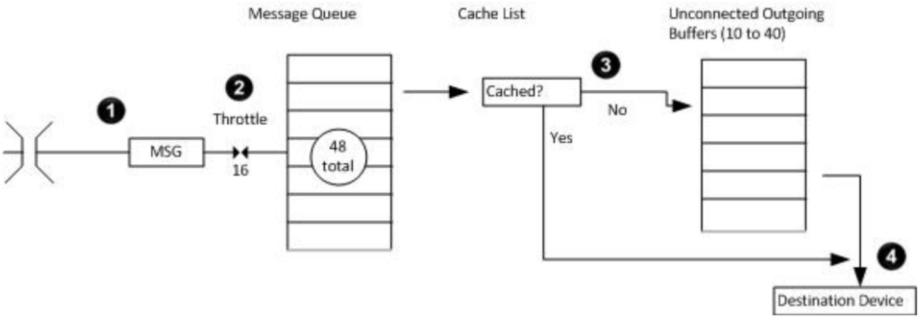
Example: **Execute a message (MSG) instruction**

If count_send = 1

and count_msg.EN = 0 (MSG instruction is not enabled)

then execute a MSG instruction that sends data to another controller.

This diagram shows how the controller processes MSG instructions.



1	The controller scans the MSG instruction and its rung-condition-in goes true. The message passes to a throttle that has 16 positions. If the throttle is full, the message remains enabled but is held until another controller scan.	
2	The System-overhead time slice executes and the message is pulled from the throttle to the message queue.	
3	**If the MSG instruction**	**Then the MSG instruction**
	Does not use a connection or the connection was not previously cached	Uses an unconnected buffer to establish communication with the destination device.
	Uses a connection and the connection is cached	Does not use an unconnected buffer.
4	Communication occurs with the destination device.	

"

Older RSLogix 500 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.

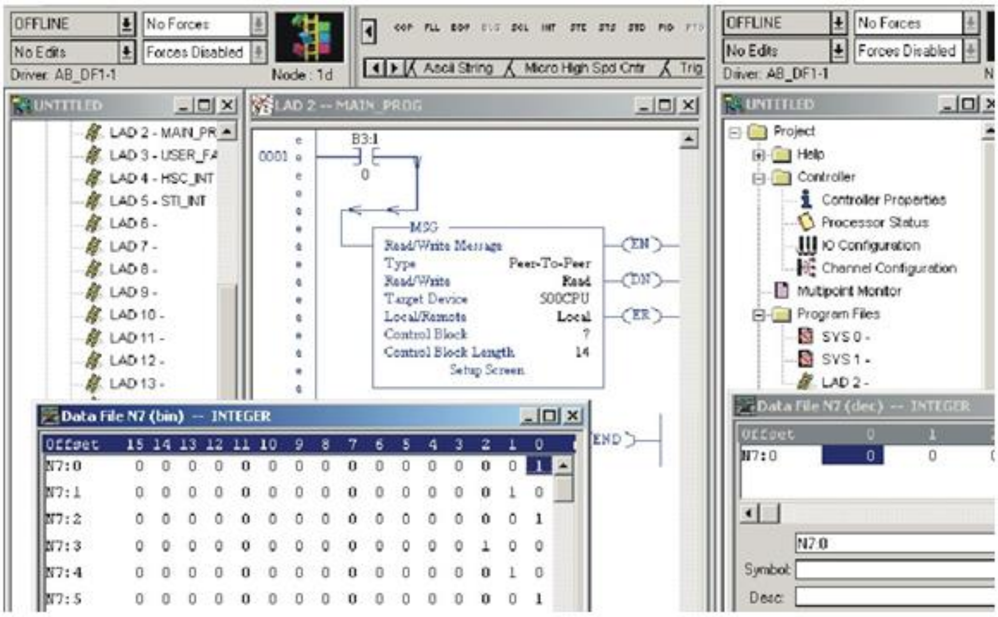


Fig. 16-13

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.

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.

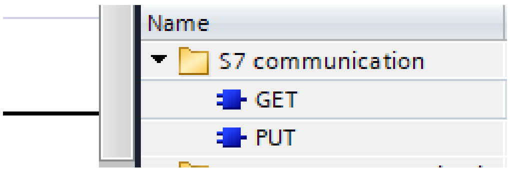
Siemens' PLC to PLC Communications

Check the Easy Book:

Easy BookManual, 01/2015, A5E02486774-AG139 - Easy to communicate between devices

and

Youtube - Siemens PLC to PLC Communication PUT GET Easy Guide



From the Help Request from TIA Program, the following descriptions for GET and PUT:

GET: Read data from a remote CPU

Description

With the instruction "GET", you can read data from a remote CPU.

The instruction is started on a positive edge at control input REQ:

- The relevant pointers to the areas to be read out (ADDR_i) are then sent to the partner CPU. The partner CPU can be in RUN or STOP mode.
- The partner CPU returns the data:

    - If the reply exceeds the maximum user data length, this is displayed with error code "2" at the STATUS parameter.
    - The received data is copied to the configured receive areas (RD_i) at the next call.

- Completion of this action is indicated by the status parameter NDR having the value "1".

Reading can only be activated again after the previous reading process has been completed. Errors and warnings are output via ERROR and STATUS if access problems occurred while the data was being read or if the data type check results in an error.

PUT: Write data to a remote CPU

Description

You can write data to a remote CPU with the instruction "PUT".

The instruction is started on a positive edge at control input REQ:

- The pointers to the areas to be written (ADDR_i) and the data (SD_i) are then sent to the partner CPU. The partner CPU can be in RUN or STOP mode.
- The data to be sent is copied from the configured send areas ((SD_i). The partner CPU saves the sent data under the addresses supplied with the data and returns an execution acknowledgment.
- If no errors occur, this is indicated at the next instruction call with status parameter DONE = "1". The writing process can only be activated again after the last job is complete.

Errors and warnings are output via ERROR and STATUS if access problems occurred while the data was being written or if the execution check results in an error.

Or, use the I-Device Method:

This method is recommended for use by most systems engineers. The fact that more than one distinct method exists to communicate between processors is interesting. The following I-Device method is the better method per Siemens personnel.

https://support.industry.siemens.com/cs/us/en/view/109478798

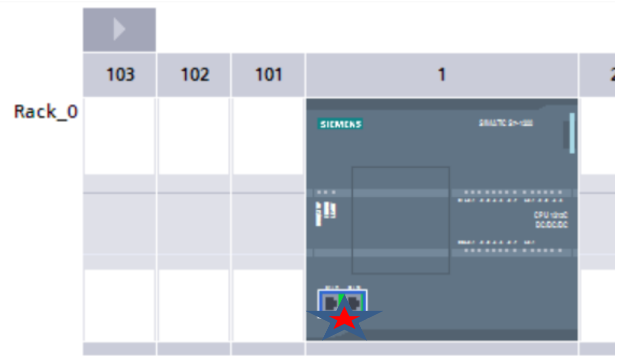
It was also used by a student in MIME 5450 – Michael Smith with the following:

"

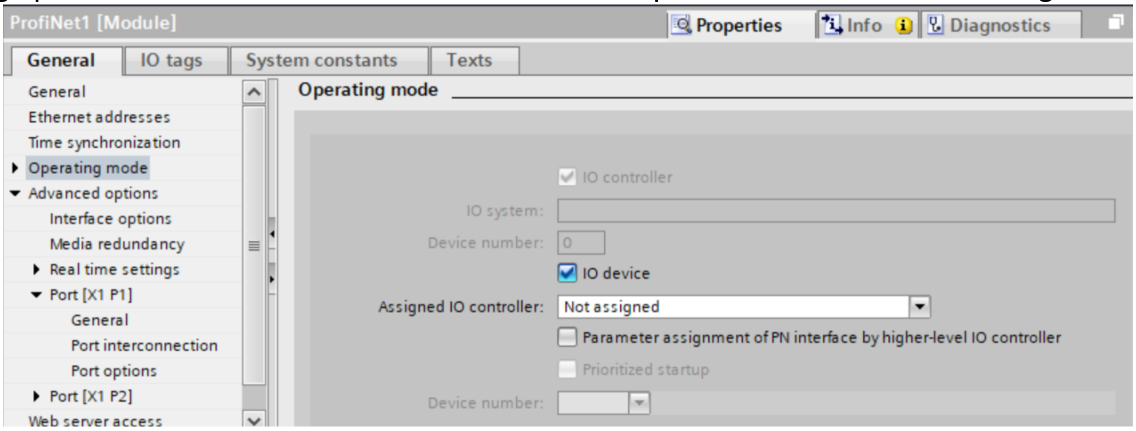
Michael Smith -MIME 5450

PLC to PLC Communication – Siemens I Device

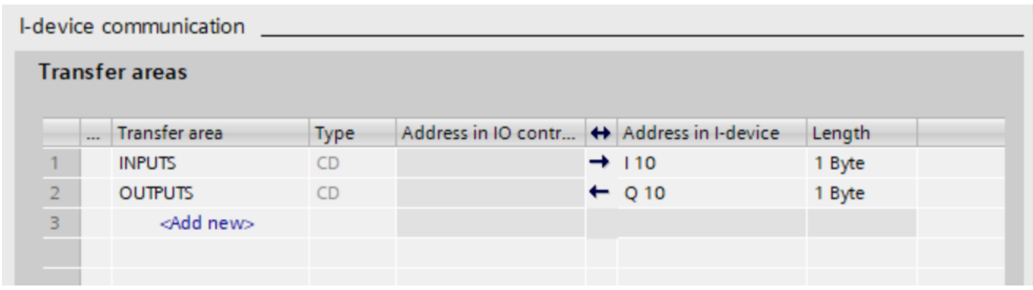
The objective of this lab was to establish communication between 2 Siemens S7-1215 PLCs using 'I Device'. I started by creating 2 projects for each of the PLCs. In each projected I inserted the S7-1215 as shown below.



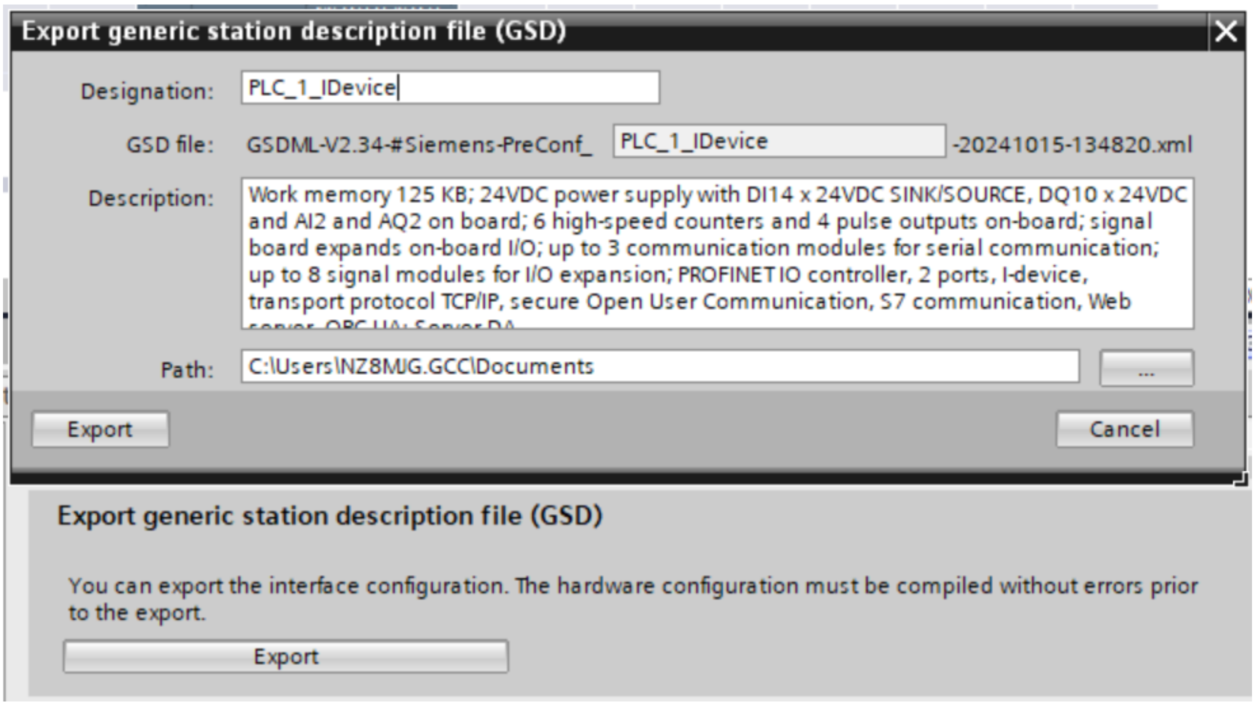
One of the PLCs had to be configured as an IO device which can be seen in the below image. To bring up the menu below I had clicked on the ProfiNet ports shown in the above image.



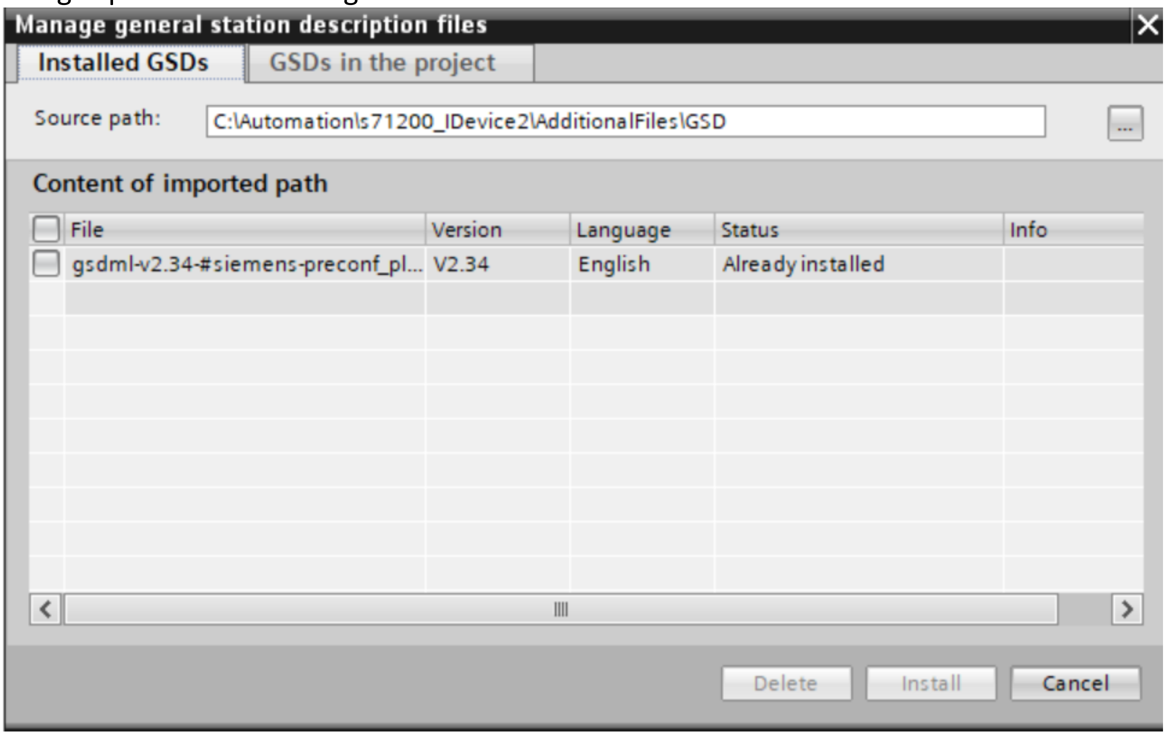
Next I had to configure the input and output sizes which can be done by just scrolling down. In the same window. Notice I have 1 byte of input and output data configured.



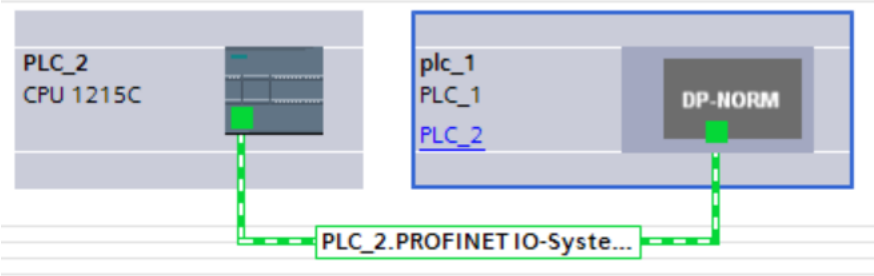
After the input and output data was configured I had to generate a GSD file which was done in the same window.

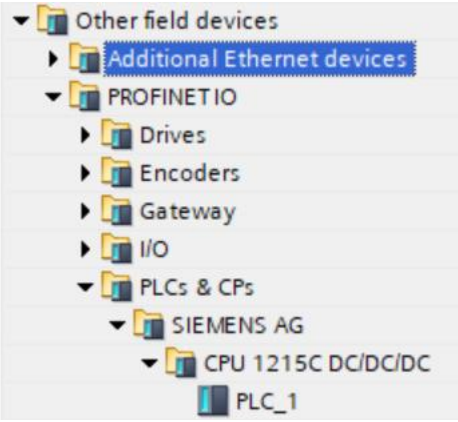


I then saved the project and downloaded the project to the PLC.
In the second PLC I had to install the GSD file generated from the first PLC. This can be done by selecting 'Options' and 'Manage GSD Files.'



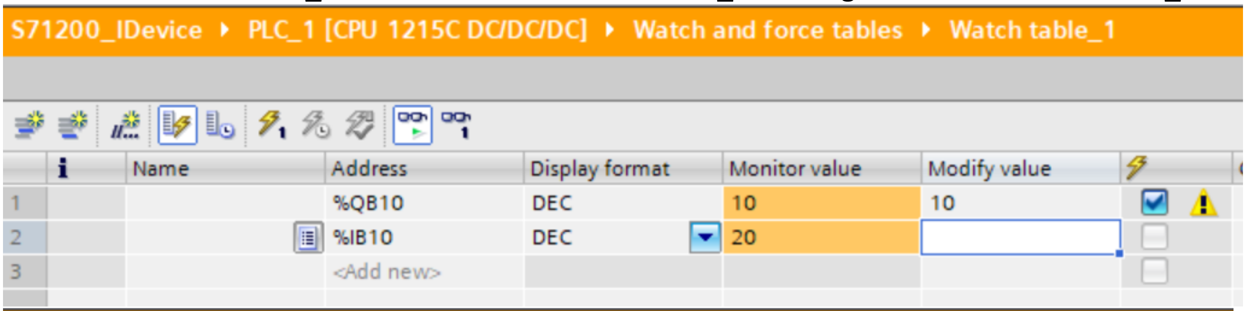
After the GSD file was installed PLC_1 can be inserted in the ProfiNet network as shown in the image below.





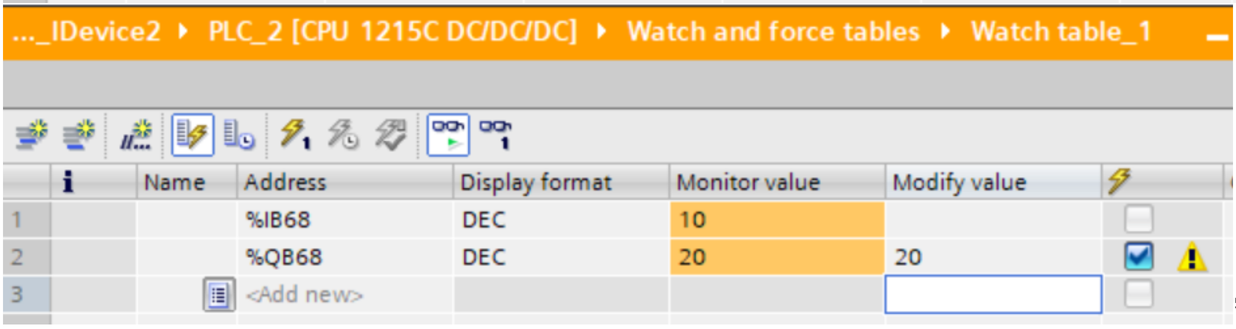
I then downloaded the project to PLC_2. With both PLCs in RUN mode and connected VIA an ethernet cable connection was established.

Below are ‘Watch Tables’ from each PLC and you can see the data from PLC_1 at QB10 is being moved into IB68 of PLC_2 and the value in QB68 in PLC_2 is being moved into IB10 of PLC_1.



S71200_IDevice ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Watch and force tables ▸ Watch table_1

	i	Name	Address	Display format	Monitor value	Modify value
1			%QB10	DEC	10	10
2			%IB10	DEC	20	
3			<Add new>			

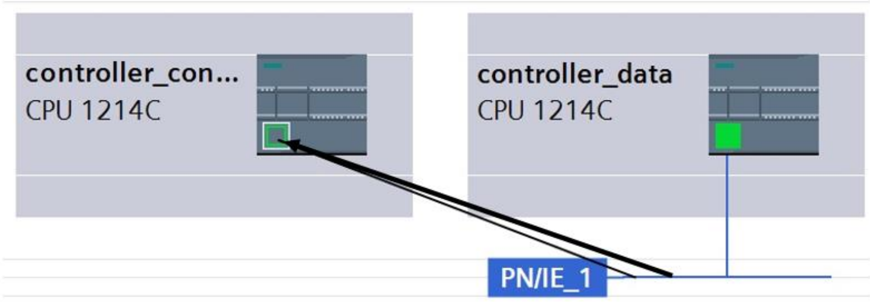


..._IDevice2 ▸ PLC_2 [CPU 1215C DC/DC/DC] ▸ Watch and force tables ▸ Watch table_1

	i	Name	Address	Display format	Monitor value	Modify value
1			%IB68	DEC	10	
2			%QB68	DEC	20	20
3			<Add new>			

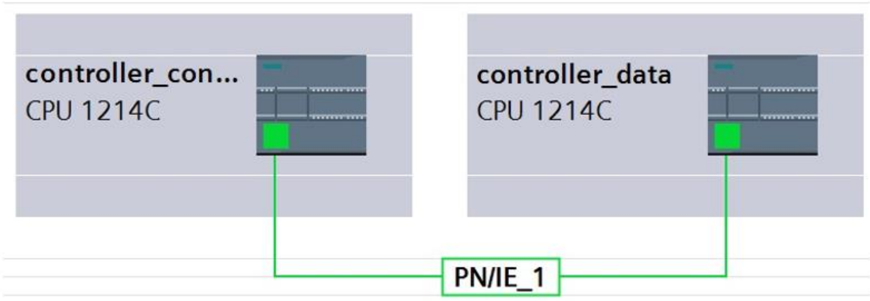
”

Also, a third method can be used. It is found in part in early literature for the S7-1200:



12.The two CPUs are now connected.

Fig. 16-15



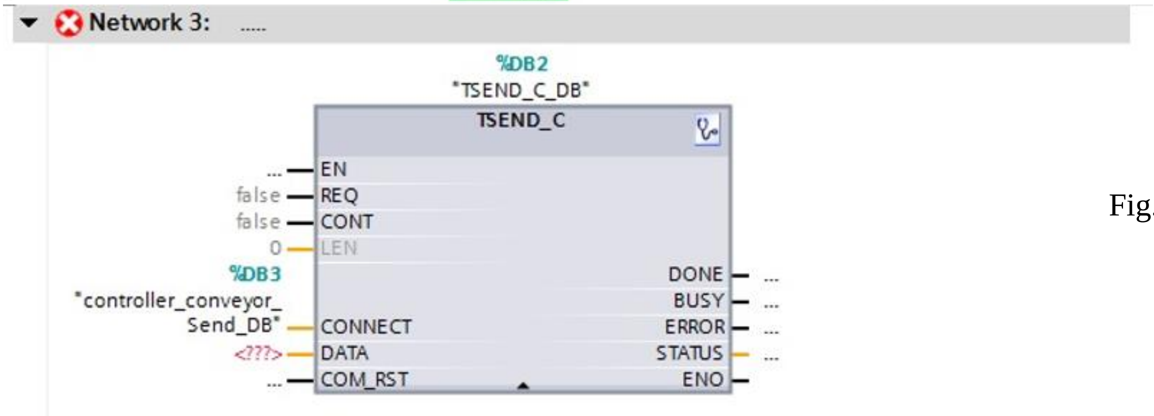
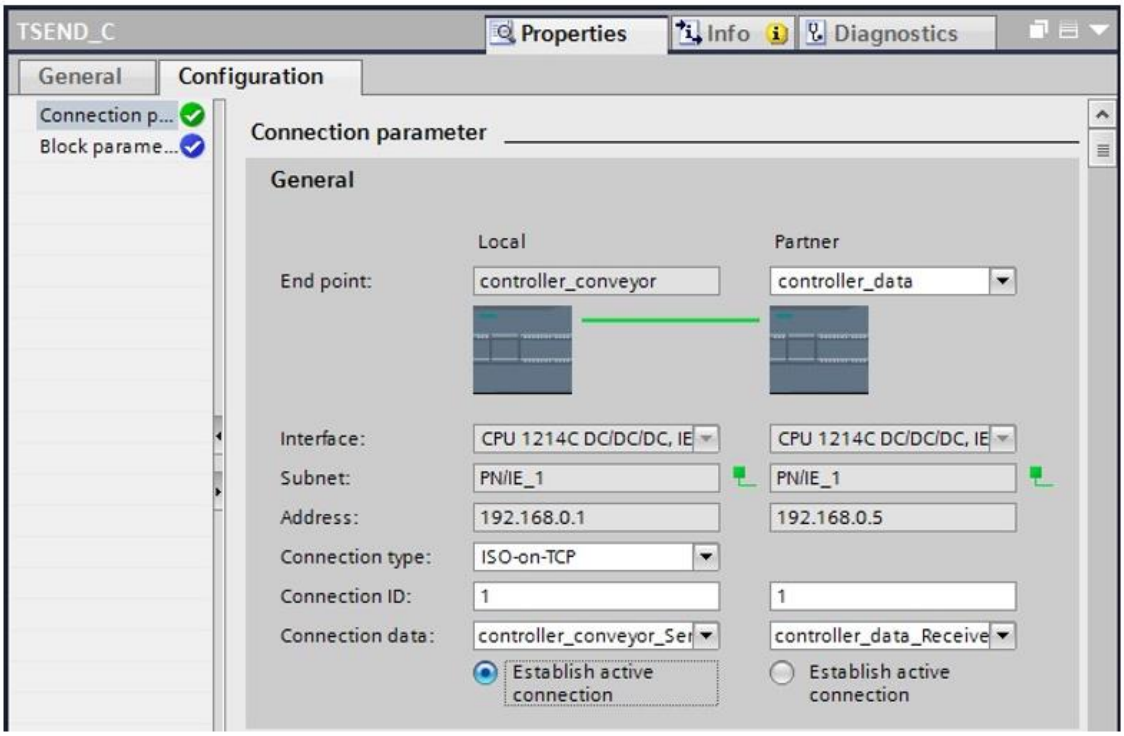


Fig. 16-16



12. The data block is generated and incorporated automatically.
Under Properties, select the connection partners.
First, at the connection data of the local controller, select the existing data block "**controller_data_connection_DB**", otherwise, a new data block is generated.
Then, select the partner controller and the associated data block.

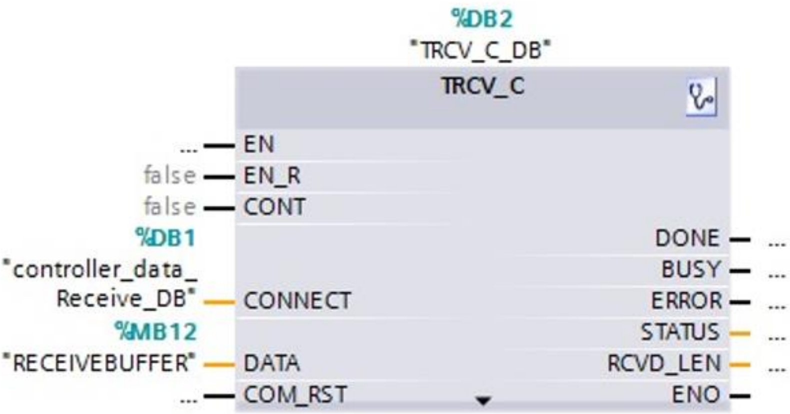


Fig. 16-17

So, there are at least three different distinct methods for the Siemens PLCs to talk one to another.

How to Interconnect Dissimilar Devices

To look at how to interconnect between PLC and other device, just look at the following ad from one of the industrial leaders in interconnecting the PLC on the factory floor:
"

"

The ad continues with a listing that gives popular interconnection types and Anybus' solution:

ANYBUS SOLUTIONS FOR INDUSTRIAL ETHERNET NETWORKS

→ BACnet/IP → CC-Link IE Field → EtherCAT → EtherNet/IP → Modbus TCP → Powerlink → PROFINET

ANYBUS SOLUTIONS FOR FIELDBUS/SERIAL/OTHERS

→ CAN/CANopen → CC-Link → ControlNet → DeviceNet → M-Bus → Modbus RTU (RS232/422/485) → PROFIBUS → Wireless

Others with similar products for interconnecting PLCs with various devices include Red Lion with their ad below:

"

COMMUNICATION CONVERTERS

- Protocol Converters
  - FlexEdge
  - Data Station
  - ICM8 Gateway
- Media Converters
- Serial Converters
- Accessories

Bridge the communication gap between disparate serial, Ethernet and fieldbus devices, orphaned by incompatible communications standards, using Red Lion's suite of protocol conversion products. Supporting more than 300 industrial protocols, communicating from vendors like ABB, Emerson, Rockwell Automation, Schneider and Siemens has never been easier than with our simple-to-use Crimson® software.

DATA STATION

The Data Station from Red Lion is a protocol conversion and data acquisition solution for a connected industry. In addition to supporting over 300 industrial drivers, it also offers data, event and security logging with cryptographic signature support; remote machine access through an advanced web server; connectivity to some of the industry's most popular IIoT platforms and embedded OPC UA server and client functionality. **Learn more.**

FLEXEDGE™ INTELLIGENT EDGE AUTOMATION PLATFORM

Regardless of the brand of PLC or other equipment specified, the FlexEdge platform – leveraging the more than 300 drivers in Red Lion's intuitive Crimson® software – enables connectivity to virtually anything in your system with point-and-click simplicity. In addition, avoid costly and complex rip-and-replace scenarios by leveraging the FlexEdge sled architecture, designed to enable organizations to leverage new communications technologies as they become available. Rugged construction, a wide operating temperature range, and industrial certifications provide a dependable solution for protocol conversion, control, edge processing and data visualization in even the harshest environments. **Learn more.**

GRAPHITE® EDGE CONTROLLER

In addition to native support of IEC 61131 programming languages, connectivity to IIoT platforms and OPC UA server and client capability, Red Lion's rugged Graphite Edge controller communicates with over 300 protocols, providing protocol conversion, data, event and security logging with cryptographic signature support, and advanced web serving and data visualization functionality in a versatile form factor ideal for a variety of applications. With its rugged, all-metal construction and wide operating temperature range, the Graphite Edge is designed to operate in extreme environments. **Learn more.**

ICM8 PANEL METER GATEWAY

The ICM8 is designed to allow Red Lion panel meters to communicate over industrial Ethernet networks. The ICM8 communicates via RS-485 to Red Lion panel meters, converting serial to Ethernet for remote monitoring and control. Up to 32 meters can be wired to the ICM8 using RS-485 serial communications. This ICM8 converter is only compatible with Red Lion panel meters; please use the DSP for all other protocols.
Learn more.

. ”

Another protocol converter is Hilscher. Their NT 151-RE-RE converter is presently planned to be used in the Lab for converting between Profibus and Ethernet/IP and Modbus/TCP.

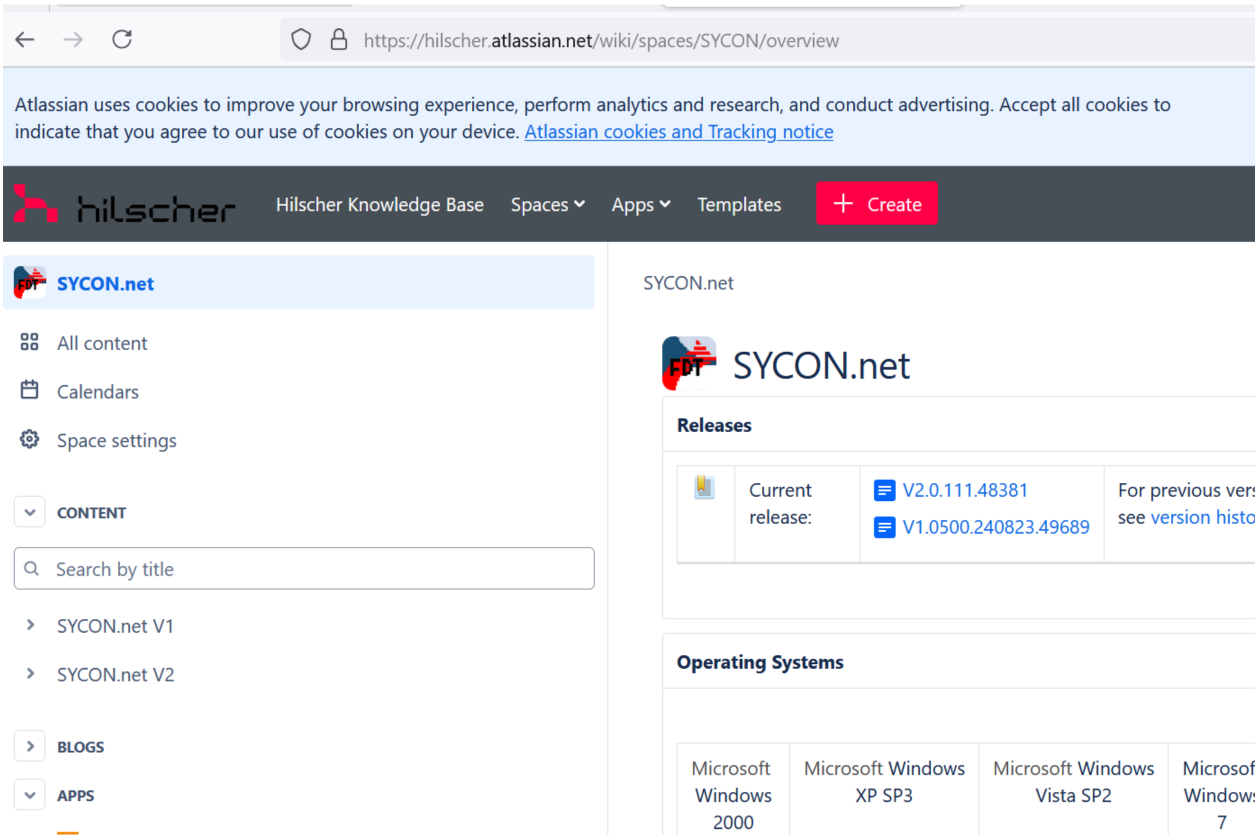
The setup of the Hilscher unit can be found at the Atlassian website under the sycon heading. The url for the site is:

https://hilscher.atlassian.net/wiki/spaces/SYCON/overview

This site gives the setup software, presently 2.0 as well as manuals for the setup of the network card for the various protocols.

This website is shown below:

(Documentation for the setup of the cards can also be found on this page.)



Setup of the card is shown in the pictures below:

Writing your own ASCII READ/WRITE BLOCKS

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 in an index to the chapter, not all codes represent letters or numbers. Many codes represent actions, special keys or control characters.

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.

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

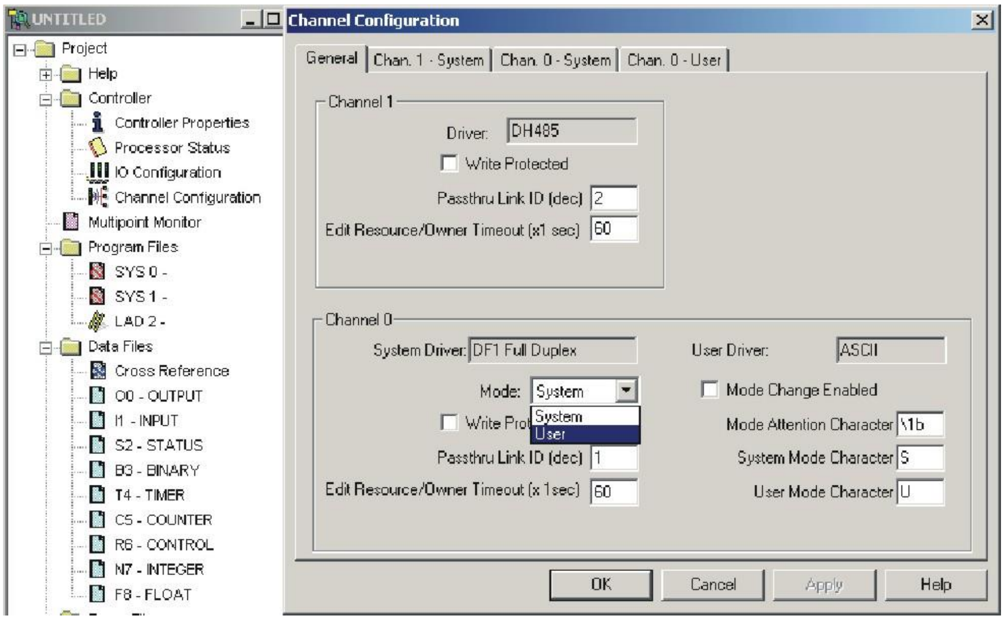


Fig. 16-18 Using the SLC Processor as ASCII Device

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:

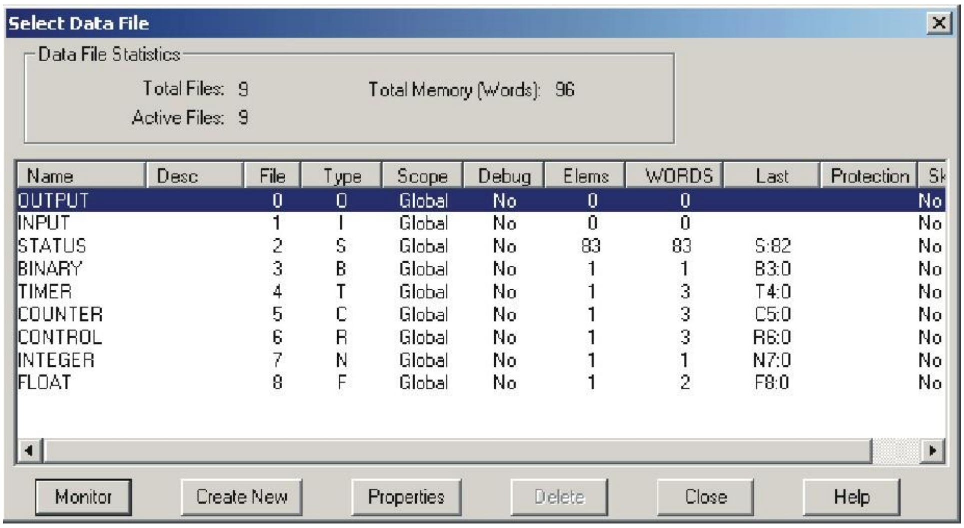


Fig. 16-19 Configure the String or ASCII file in the SLC

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.

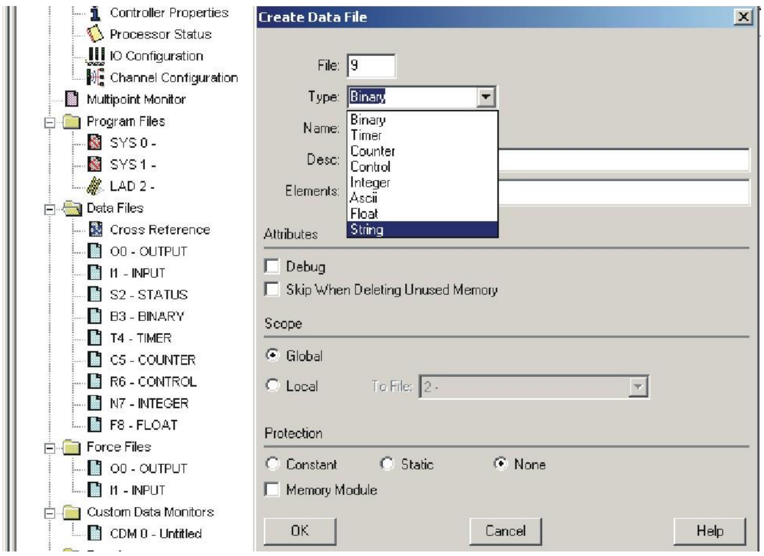


Fig. 16-20 Configure the String or ASCII file in the SLC

ASCII Transmission

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:

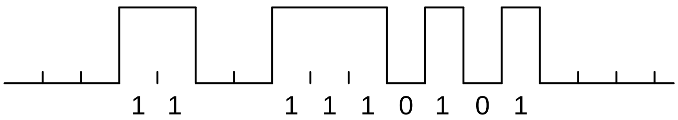


Fig. 16-21 Oscilloscope Representation of ASCII Character

Transmission Modes and ASCII Tables

The Modbus protocol discussed later in this chapter is set up to communicate in one of two types of transmission: ASCII or RTU. The choice of ASCII or RTU is made by the system designer and must be kept the same throughout. This communication represents 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. 16-22 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.

Example of a Simple ASCII Protocol

The protocol described here is a simple protocol using ASCII characters. It is less sophisticated than the Modbus protocol discussed later. 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>
Response: A<Status><DB><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>
Response: w<Status><DB><CHCK><ETX>

Fig. 16-23 Simple ASCII Communication

Read bytes, Auto mode
Command: W<HdNo><StAdrH><BytesH>CHCK><ETX>
Response: W<Status><DB><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>
Response: K<Status><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 <CHCK>, the following addition is performed:

```
K            4Bh     Ascii char "K"
2            32h     Ascii char "2"
0            30h     Ascii char "0"
A            41h     Ascii char "A"
0            30h     Ascii char "0"
3            33h     Ascii char "3"
50h          50h     hex char 50
2Bh          2Bh     hex char 2B
46h     +    46h     hex char 46
----------------
          (2)12h
```

This gives a check sum <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.

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.

From 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 addressing of all Modicon PLCs is as follows:

00001 – 0xxxx – Discrete Outputs and Internal Coils
10001 – 1xxxx – Discrete Inputs
30001 – 3xxxx – 16 bit word Input
40001 – 4xxxx – 16 bit word Output and Internal Storage

What is common about each of these is that the displacement **to any address is from 1, not from 0**. For instance, if the address for location 500 of the discrete output table, the displacement would not be 500 but 499 and this number would be converted to hex. The same would be for any of the addressing types.

The Modbus Transaction

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 Peprl-Fuchs protocol in that if an error occurs, the slave sends the appropriate error message.

* Modicon: Modbus Protocol Reference Guide
PI–MBUS–300 Rev. J

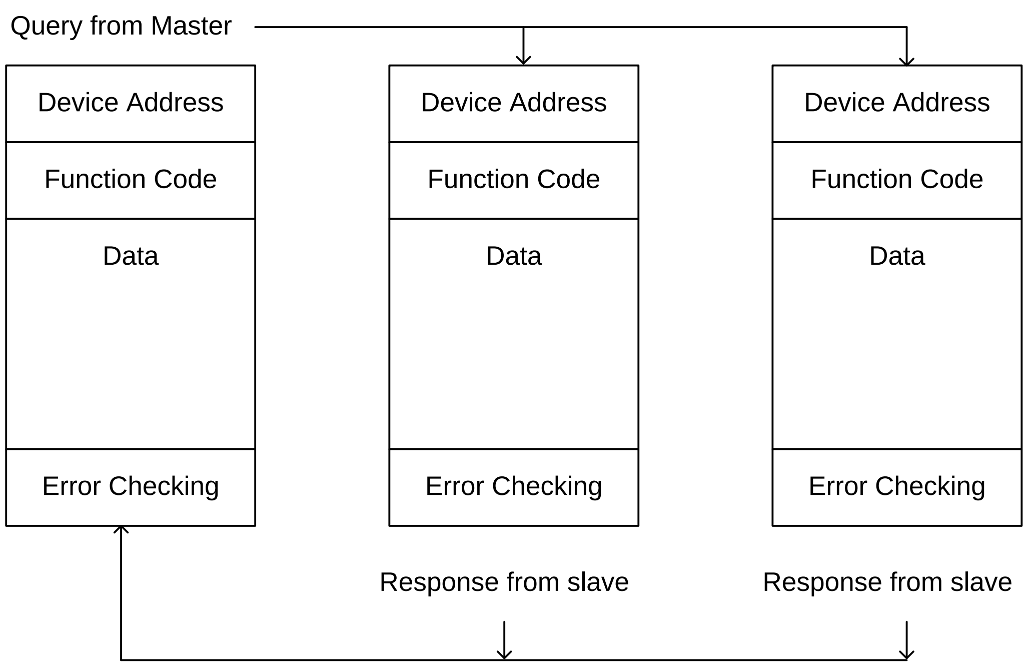


Fig. 16-25

The Query:

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

Code	Name	Comments *
01	Read Coil Status	specifies 16*n discrete slave PLC outputs
02	Read Input Status	specifies 16*n discrete slave PLC inputs
03	Read Holding Registers	specifies n 16 bit words from slave PLC output tbl
04	Read Input Registers	specifies n 16 bit words from slave PLC input table
05	Force Single Coil	
06	Preset Single Register	
07	Read Exception Status	
08	Diagnostics	
09	Program 484	84 specifies a type of Modicon PLC
10	Poll 484	
11	Fetch Comm. Event Ctr.	
12	Fetch Comm. Event Log	
13	Program Controller	
14	Poll Controller	
15	Force Multiple Coils	
16	Preset Multiple Registers	
17	Report Slave ID	
18	Program 884/M84	884/M84specifies a type of Modicon PLC
19	Reset Comm. Link	
20	Read General Reference	
21	Write General Reference	
22	Mask Write 4X Register	
23	Read/Write 4X Registers	
24	Read FIFO Queue	

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

Every part of the data field in the Modbus protocol must be addressed correctly. For example, the data address must be referenced to one. 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:

Start	Address	Function	Data	LRC Check	End
1 Char :	2 Chars	2 Chars	n Chars	2 Chars	2 Chars CRLF

Fig. 16-26 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:

Start	Address	Function	Data	CRC Check	End
T1-T2-T3-T4	8 Bits	8 Bits	n * 8 Bits	16 Bits	T1-T2-T3-T4

Fig. 16-27 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

Field Name	
Slave Address	11
Function	02
Starting Address Hi	00
Starting Address Lo	C4
No. of Points Hi	00
No. of Points Lo	16
Error Check (LRC or CRC)	----

Hex 11 = Decimal 17 or the 17th device
Function 02 = Data Read from Digital Input Table starting at 10001
High Address = 00
Low Address = C4 or 12*16 + 4 or 192 + 4 = 196 which is address 10197 or 196 offset from 10001
No of points = 16 or 1*16 + 6 = 22 data points

Sample Modbus Read Input Query

Response

Field Name	
Slave Address	11
Function	02
Byte Count	03
Data (Inputs 10204 - 10197)	AC
Data (Inputs 10212 - 10205)	DB
Data (Inputs 10218 - 10213)	35
Error Check (LRC or CRC)	----

Hex 11 = Decimal 17 or the 17th device
Function 02 = Data Read from Digital Input Table starting at 10001
Byte Count = number of 8 bit bytes to be sent. There are 3 based on bit count of 22
Data includes the first byte (AC), second byte (DB) and third byte (35). Notice that the third byte is not complete due to the fact that only 22, not 24 bits are desired. The extra two bits are not included and recorded as 0's.

Fig. 16-28

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 10212 are found in the second entry and values of 10213 to 10218 are found in the third.

10204	10203	10202	10201	10200	10199	10198	10197
1	0	1	0	1	1	0	0

A C

10212	10211	10210	10209	10208	10207	10206	10205
1	1	0	1	1	0	1	1

D B

		10218	10217	10216	10215	10214	10213
0	0	1	1	0	1	0	1

3 5

Fig. 16-28 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

Field Name	
Slave Address	11
Function	04
Starting Address Hi	00
Starting Address Lo	08
No. of Points Hi	00
No. of Points Lo	01
Error Check (LRC or CRC)	----

Sample Modbus Read Registers Query

Response

Field Name	
Slave Address	11
Function	04
Byte Count	02
Data Hi (Register 30009)	00
Data Lo (Register 30009)	0A
Error Check (LRC or CRC)	----

Fig. 16-29

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.

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.

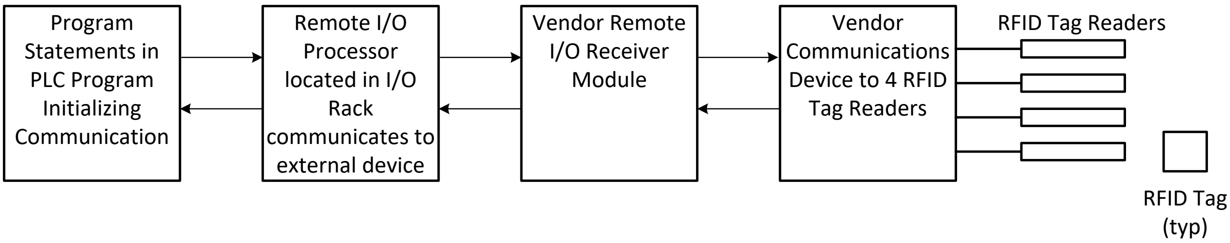


Fig. 16-30

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.

And, we would be negligent not to introduce Factory 4.0 or Industry 4.0. The concepts introduced in this chapter are an integral part of the concepts of both. It is necessary to remain engaged in the need to constantly improve our factory control systems to implement the concepts of Factory 4.0.

Nor would we want you to be discouraged by the number of topics found in the Siemens TIA portal programming software concerning either hardware or software for communication between processors or other devices. The following is a sampling of the Siemens offerings.

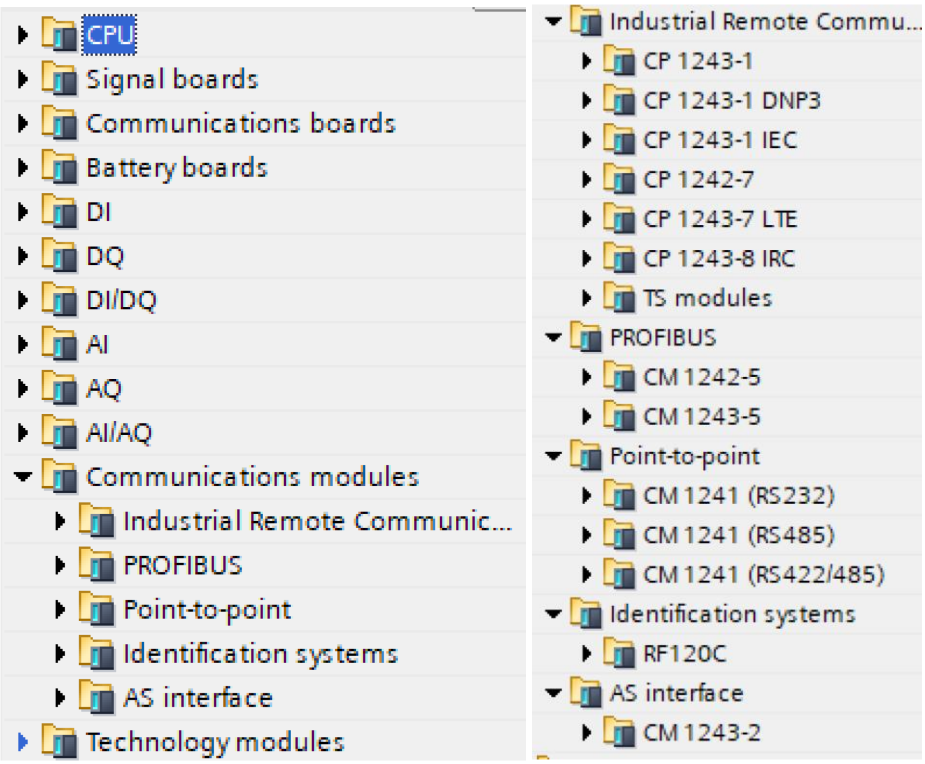
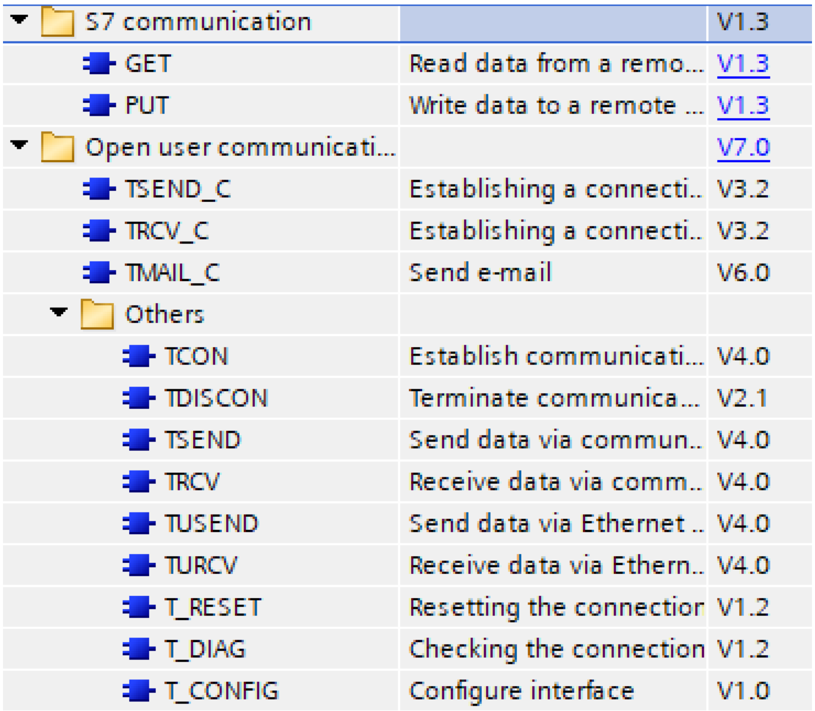


Fig. 16-31
Siemens' 1215 processor describes the following Siemens hardware for industrial remote communication with the



When programming in OB1 or other program space, the following S7 Communication instructions are available for use. They are extensive.

WEB Server		V1.1
WWW	Synchronizing user-def...	V1.1
Others		
MODBUS TCP		V5.2
MB_CLIENT	Communicate via PROF...	V5.2
MB_SERVER	Communicate via PROF...	V5.2
MODBUS TCP Redund...		V5.2
MB_RED_CLIENT	Redundant communic...	V1.2
MB_RED_SERVER	Redundant communic...	V1.2

Fig. 16-32

Communication processor		
PtP Communication		V3.2
Port_Config	Configure PtP commun...	V1.2
Send_Config	Configure PtP sender	V1.2
Receive_Config	Configure PtP recipient	V1.3
P3964_Config	Configure protocol	V1.3
Send_P2P	Send data	V3.1
Receive_P2P	Receive data	V2.6
Receive_Reset	Delete receive buffer	V1.2
Signal_Get	Read status	V1.4
Signal_Set	Set accompanying sign...	V1.2
Get_Features	Get extended functions	V2.1
Set_Features	Set extended functions	V2.1

USS communication		V4.3
USS_Port_Scan	Communication via US...	V3.3
USS_Drive_Control	Data exchange with th...	V2.0
USS_Read_Param	Read data from drive	V1.5
USS_Write_Param	Change data in drive	V1.6
MODBUS (RTU)		V4.4
Modbus_Comm_...	Configure port for Mod...	V3.1
Modbus_Master	Communicate as Modb...	V3.3
Modbus_Slave	Communicate as Modb...	V4.3

TeleService		V1.9
TM_MAIL	Send email	V1.4

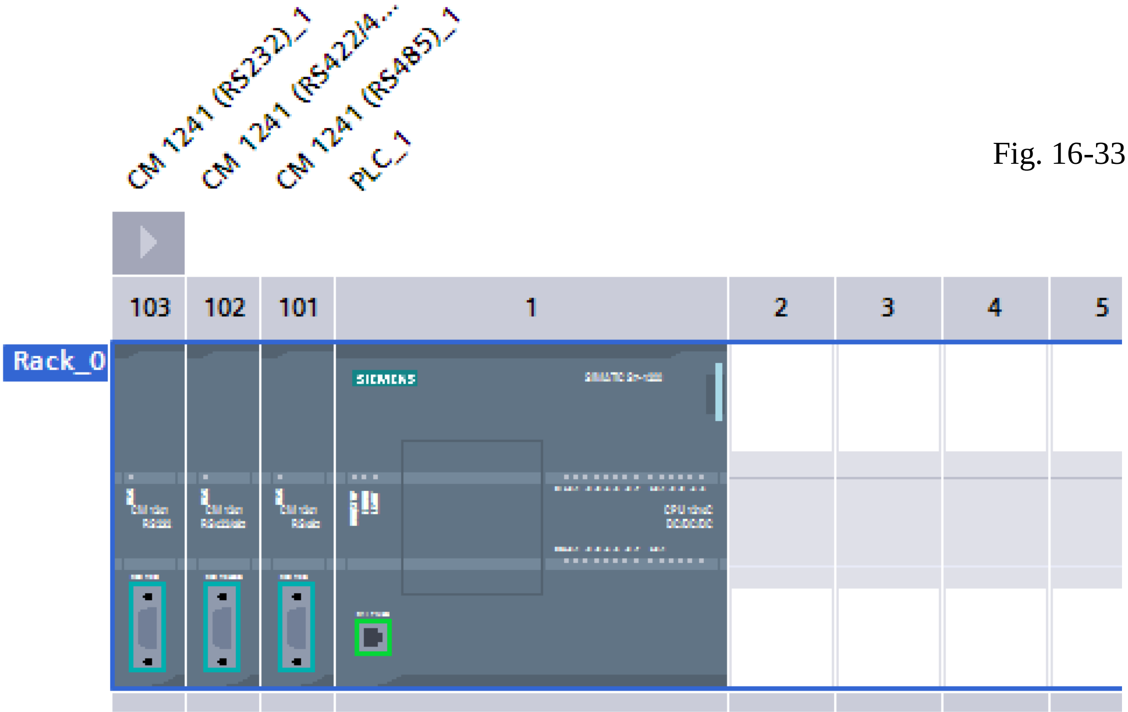


Fig. 16-33

The three cards attached to the left of the CPU in the picture above represent three different configurations of the serial communications port available with the S7-1200 processors. These are the RS-232, RS-422 and RS-485 protocols. Each has a different voltage and current requirement for completion of a data transmission.

Some of the devices these cards can be attached to include bar code readers and RFID tag readers. Siemens has a standard set of instructions for communication to the Siemens RFID tag system. Lab 16.1 outlines the requirements for implementing this RFID tag reader.

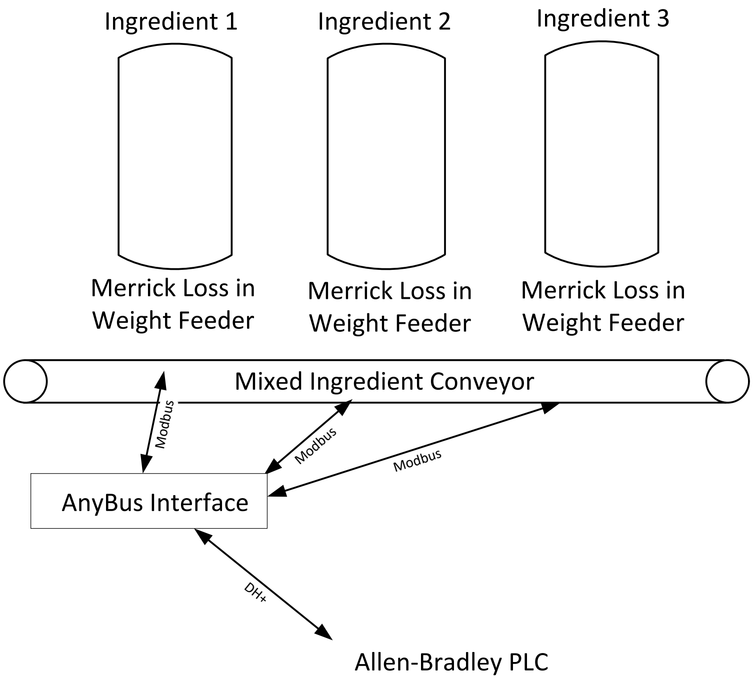


Fig. 16-34

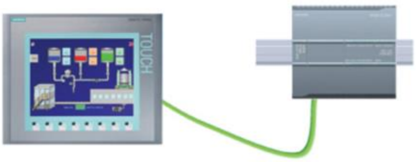
The process described here was installed by the instructor in a plant. The three ingredient tanks had material that was mixed on the conveyor belt at the base. The three Merrick loss in weight feeders metered the material onto the belt. Setpoint data was fed to the three feeders from the A-B PLC through the AnyBus interface and then through the Modbus protocol to the Merrick controllers. Setpoint data as well as actual feed rates were part of the data shared over the Modbus connection.

Again, we turn to the Easy Book from Siemens to get a better idea of some of the flexibility of this chapter on communications. Their Chapter 7 starts as follows: “

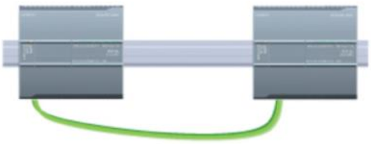
Easy to communicate between devices 7

For a direct connection between the programming device and a CPU:

- The project must include the CPU.
- The programming device is not part of the project, but must be running STEP 7.



For a direct connection between an HMI panel and a CPU, the project must include both the CPU and the HMI.



For a direct connection between two CPUs:

- The project must include both CPUs.
- You must configure a network connection between the two CPUs.

”

Or Chapter 9: “

Web server for easy Internet connectivity 9

The Web server provides Web page access to data about your CPU and to the process data within the CPU. With these Web pages, you access the CPU (or Web-enabled CP) with the Web browser of your PC or mobile device. The standard web pages allow authorized users to perform these functions and more:

- Changing the operating mode (STOP and RUN) of the CPU
- Monitoring and modifying PLC tags, data block tags, and I/O values
- Viewing and downloading data logs
- Viewing the diagnostic buffer of the CPU.
- Updating the firmware of the CPU.

The Web server also allows you to create user-defined Web pages that can access CPU data. You can develop these pages with the HTML authoring software of your choice. You insert pre-defined "AWP" (Automation Web Programming) commands in your HTML code to access the data in the CPU.

You set up users and privilege levels for the Web server in the device configuration for the CPU in STEP 7.

”

Exercises

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

Output Scan-List

Device 2	Device 1
Device 4	Device 3
Device 6	Device 5

Fig. 16-35

Input Scan-List

<table>
<tr><td colspan="2">Device 1</td></tr>
<tr><td>Device 3</td><td>Device 2</td></tr>
<tr><td></td><td>Device 4</td></tr>
</table>

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:

Output Scan-List

<table>
<tr><td>Device 2</td><td>Device 1</td></tr>
<tr><td>empty</td><td>Device 3</td></tr>
<tr><td colspan="2">Device 4</td></tr>
<tr><td>Device 6</td><td>Device 5</td></tr>
</table>

Fig. 16-36

Input Scan-List

<table>
<tr><td colspan="2">Device 1</td></tr>
<tr><td>empty</td><td>Device 2</td></tr>
<tr><td>Device 4</td><td>Device 3</td></tr>
</table>

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

Field Name	
Slave Address	11
Function	02

Fig. 16-37

Starting Address Hi	00
Starting Address Lo	C4
No. of Points Hi	00
No. of Points Lo	16
Error Check (LRC or CRC)	----

Response

Field Name	
Slave Address	11
Function	02
Byte Count	03
Data (Inputs 10204 - 10197)	AC
Data (Inputs 10212 - 10205)	DB
Data (Inputs 10218 - 10213)	35
Error Check (LRC or CRC)	----

10204	10203	10202	10201	10200	10199	10198	10197
1	0	1	0	1	1	0	0
10212	10211	10210	10209	10208	10207	10206	10205
1	1	0	1	1	0	1	1
		10218	10217	10216	10215	10214	10213
0	0	1	1	0	1	0	1

Description of "04" Read Registers

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

Query

Fig. 16-38

Field Name	
Slave Address	11
Function	04
Starting Address Hi	00
Starting Address Lo	08
No. of Points Hi	00
No. of Points Lo	01
Error Check (LRC or CRC)	----

Response

Field Name	
Slave Address	11
Function	04
Byte Count	02
Data Hi (Register 30009)	FF
Data Lo (Register 30009)	F1
Error Check (LRC or CRC)	----
Input Register	Value
30001	A123
30002	0102

30003	BBE3
30004	30E3
30005	0001
30006	4E23
30007	8989
30008	234F
30009	FFF1

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
;

CkSum =

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.

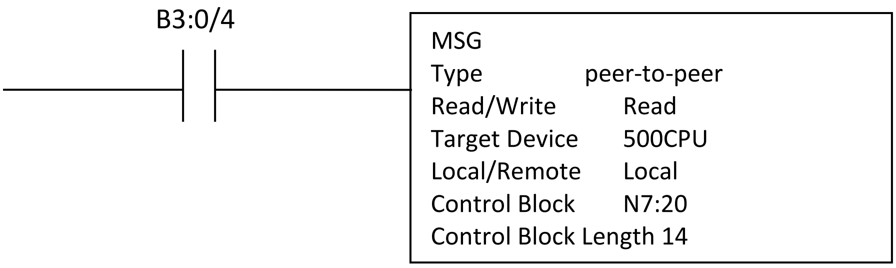


Fig. 16-39

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

Lab 16.1

This lab is a transitional lab requiring students to build the set-up with the equipment supplied and provide a resulting system capable of reading and writing to multiple tags and being able to demonstrate the operation to the instructor:

This is the manual for the RF200 system. It covers transponders and readers but no the configuration / setup.

https://support.industry.siemens.com/cs/us/en/view/109766065

This is an application example specific to the equipment you have:

https://support.industry.siemens.com/cs/us/en/view/109483367

This is a full list of application examples.

https://support.industry.siemens.com/cs/us/en/view/109483416

At this time, there is no directly applicable Allen-Bradley equipment available in the lab to incorporate RFID although this equipment does exist and works in a manner similar to the Siemens.

Lab 16.2

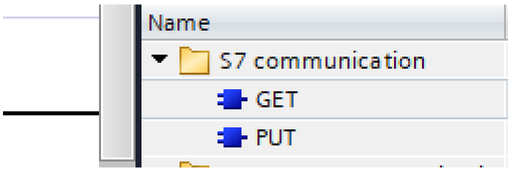
Demonstrate a communication between two PLCs using Ethernet.

This lab is a transitional lab requiring students to build the set-up with the equipment supplied and provide a resulting system capable of reading and writing.

Easy BookManual, 01/2015, A5E02486774-AG139 - Easy to communicate between devices

and

Youtube - Siemens PLC to PLC Communication PUT GET Easy Guide



From the Help Request from TIA Program, the following descriptions for GET and PUT:

GET: Read data from a remote CPU

Description

With the instruction "GET", you can read data from a remote CPU.

The instruction is started on a positive edge at control input REQ:

- The relevant pointers to the areas to be read out (ADDR_i) are then sent to the partner CPU. The partner CPU can be in RUN or STOP mode.
- The partner CPU returns the data:
  - If the reply exceeds the maximum user data length, this is displayed with error code "2" at the STATUS parameter.
  - The received data is copied to the configured receive areas (RD_i) at the next call.

- Completion of this action is indicated by the status parameter NDR having the value "1".

Reading can only be activated again after the previous reading process has been completed. Errors and warnings are output via ERROR and STATUS if access problems occurred while the data was being read or if the data type check results in an error.

PUT: Write data to a remote CPU

Description

You can write data to a remote CPU with the instruction "PUT".

The instruction is started on a positive edge at control input REQ:

- The pointers to the areas to be written (ADDR_i) and the data (SD_i) are then sent to the partner CPU. The partner CPU can be in RUN or STOP mode.
- The data to be sent is copied from the configured send areas ((SD_i). The partner CPU saves the sent data under the addresses supplied with the data and returns an execution acknowledgment.
- If no errors occur, this is indicated at the next instruction call with status parameter DONE = "1". The writing process can only be activated again after the last job is complete.

Errors and warnings are output via ERROR and STATUS if access problems occurred while the data was being written or if the execution check results in an error.

Or, use the I-Device Method:

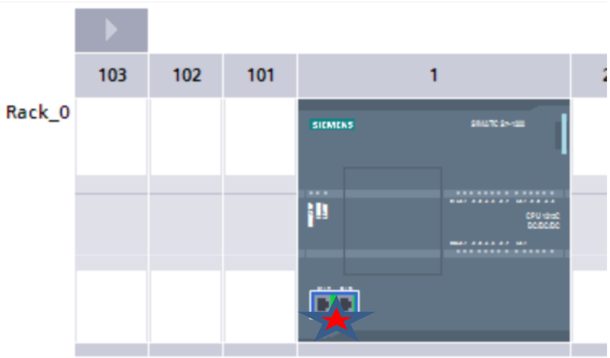
https://support.industry.siemens.com/cs/us/en/view/109478798

Or use the T-Send method discussed above in the text. Yes, Siemens gives the choice of any of three different methods for communicating between PLCs.

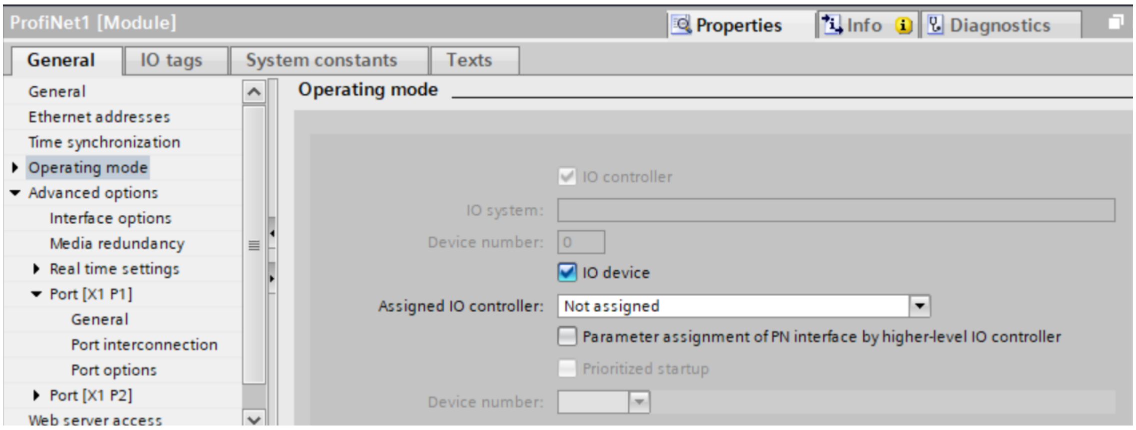
Michael Smith - MIME 5450

PLC to PLC Communication – Siemens I Device

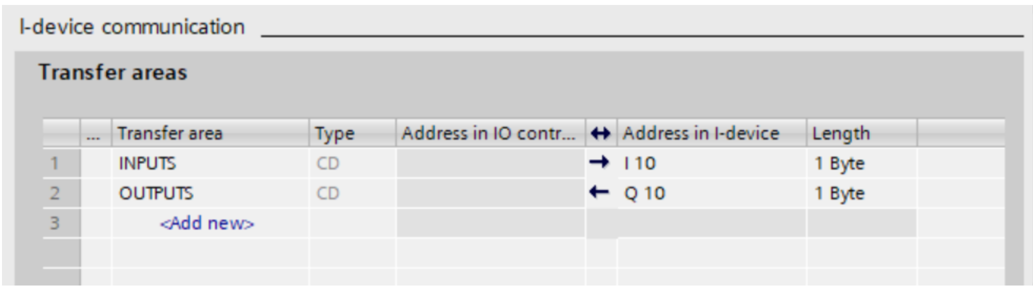
The objective of this lab was to establish communication between 2 Siemens S7-1215 PLCs using ‘I Device’. I started by creating 2 projects for each of the PLCs. In each projected I inserted the S7-1215 as shown below.



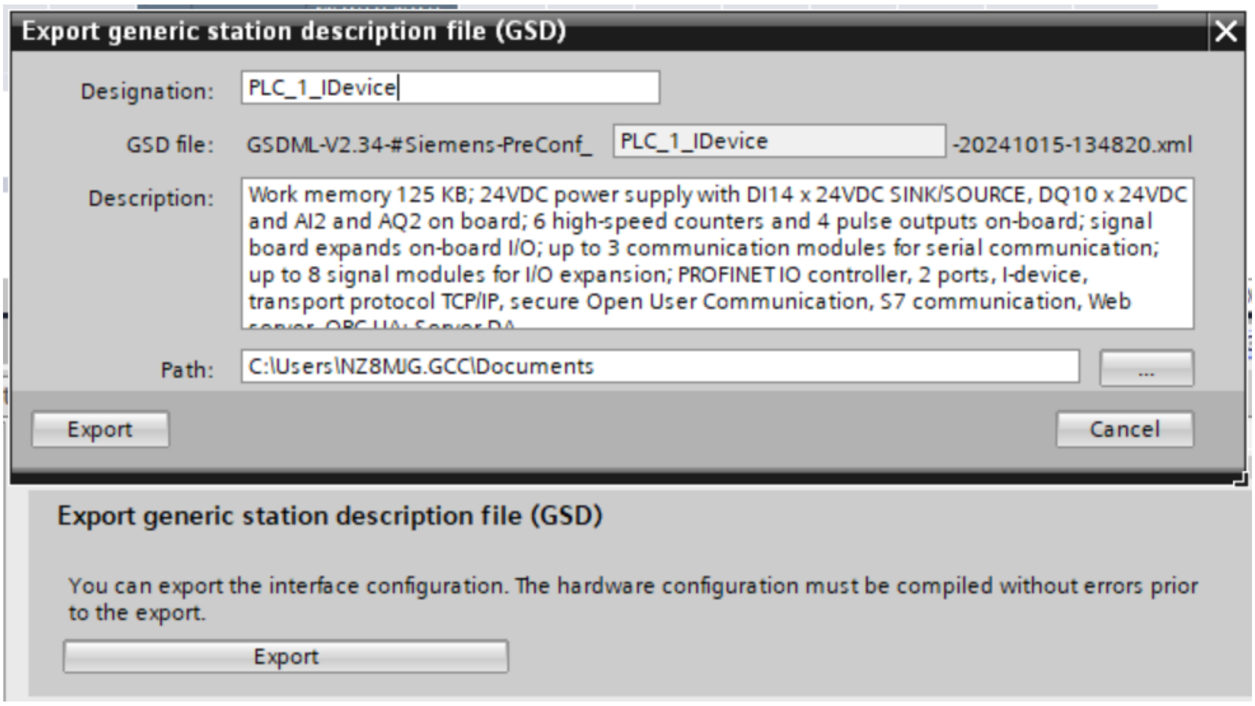
One of the PLCs had to be configured as an IO device which can be seen in the below image. To bring up the menu below I had clicked on the ProfiNet ports shown in the above image.



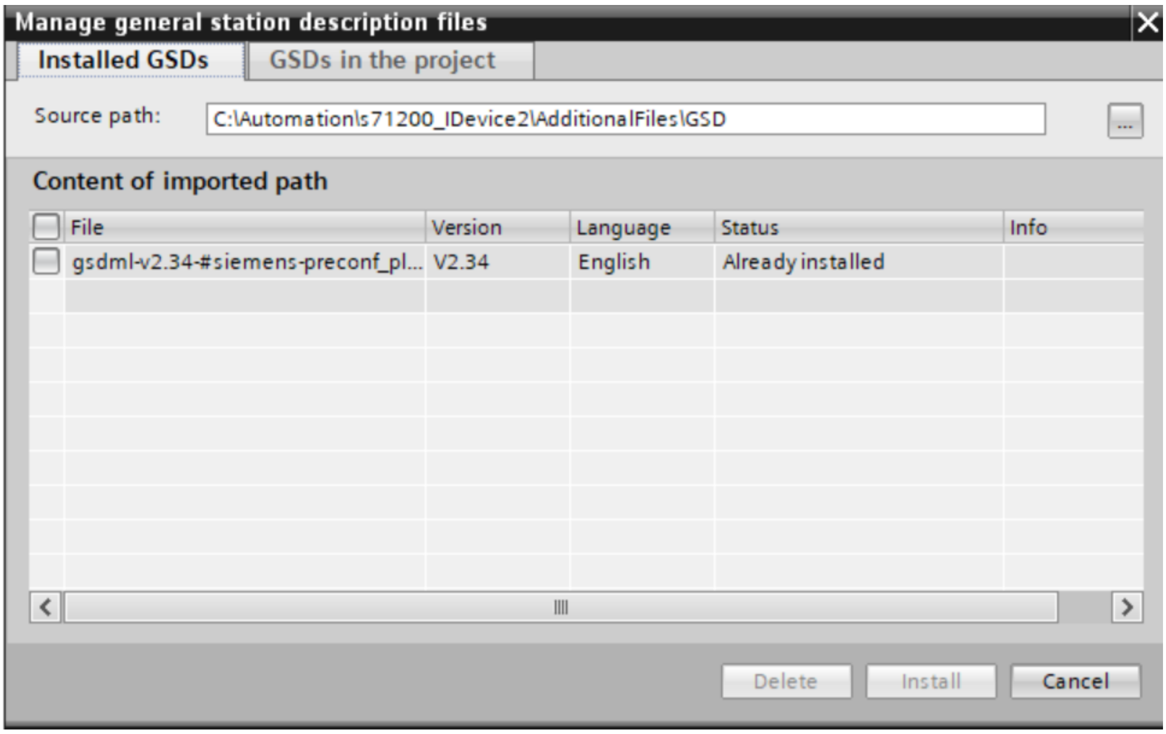
Next I had to configure the input and output sizes which can be done by just scrolling down. In the same window. Notice I have 1 byte of input and output data configured.



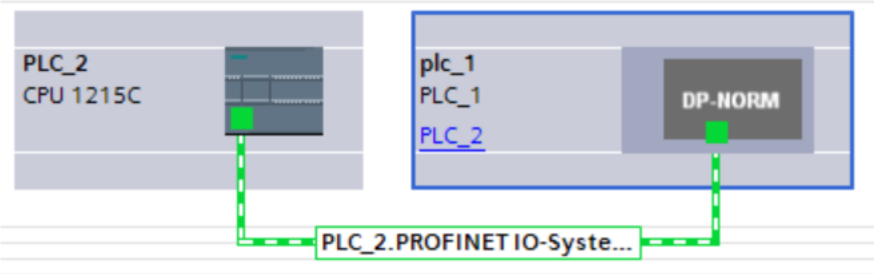
After the input and output data was configured I had to generate a GSD file which was done in the same window.

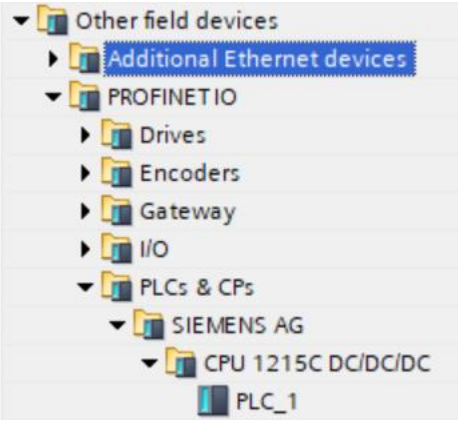


I then saved the project and downloaded the project to the PLC.
In the second PLC I had to install the GSD file generated from the first PLC. This can be done by selecting 'Options' and 'Manage GSD Files.'



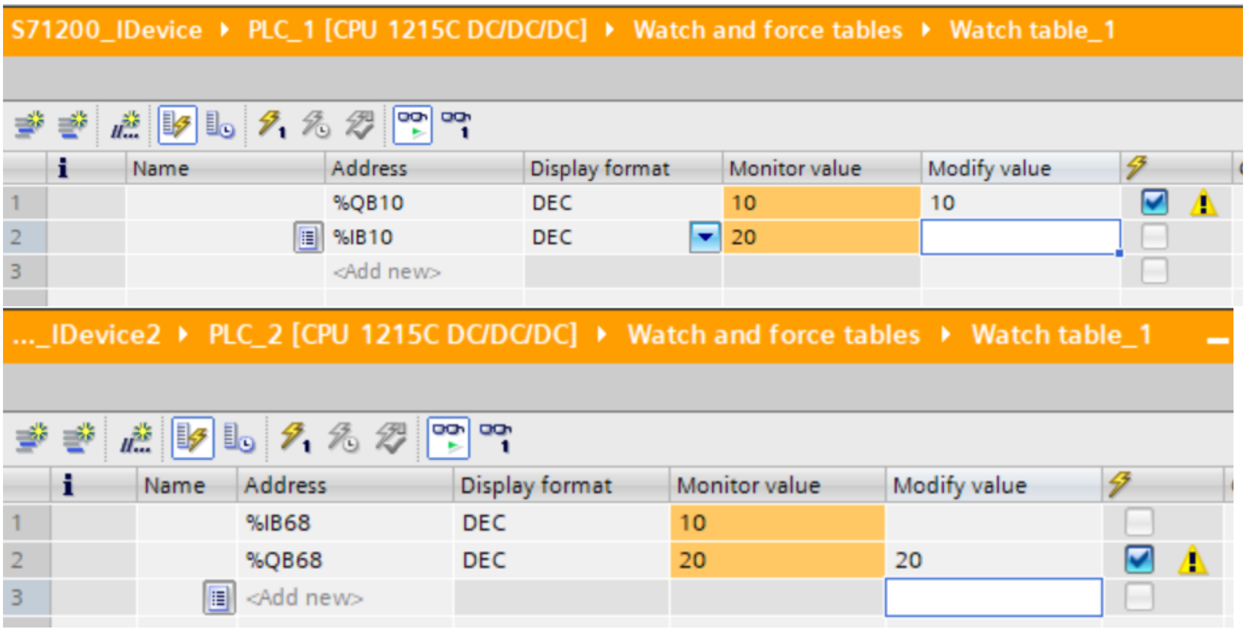
After the GSD file was installed PLC_1 can be inserted in the ProfiNet network as shown in the image below.





I then downloaded the project to PLC_2. With both PLCs in RUN mode and connected VIA an ethernet cable connection was established.

Below are 'Watch Tables' from each PLC and you can see the data from PLC_1 at QB10 is being moved into IB68 of PLC_2 and the value in QB68 in PLC_2 is being moved into IB10 of PLC_1.


S71200_IDevice ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Watch and force tables ▸ Watch table_1

	i	Name	Address	Display format	Monitor value	Modify value	⚡
1			%QB10	DEC	10	10	☑ ⚠
2			%IB10	DEC	20		☐
3			<Add new>				☐

..._IDevice2 ▸ PLC_2 [CPU 1215C DC/DC/DC] ▸ Watch and force tables ▸ Watch table_1

	i	Name	Address	Display format	Monitor value	Modify value	⚡
1			%IB68	DEC	10		☐
2			%QB68	DEC	20	20	☑ ⚠
3			<Add new>				☐

Or for A-B:

For Allen-Bradley, set up communications with MSG commands similar to the commands described in the text.

Lab 16.3

Demonstrate a communication between two PLCs using an ASCII communication protocol (RS422, 232, etc.)

This lab is a transitional lab requiring students to build the set-up with the equipment supplied and provide a resulting system capable of reading and writing.

You could set up two Siemens PLCS using the COMS add-on port and use their modbus instructions to set up two PLCs communicating using Modbus. One would be the master and the other the slave.

Use the following website as needed:

modbus tools - https://www.modbustools.com/contact.html

then choose 'MODBUS' and then 'PROTOCOL'.

Lab 16.4

Write a review at least 2 pgs on the following report (use 12 font single space):

https://www.boozallen.com/content/dam/boozallen/documents/2016/09/ukraine-report-when-the-lights-went-out.pdf

or about a hack of a safety PLC system at a huge chemical plant.

https://darknetdiaries.com/transcript/68/

Lab 16.5 – Communication between PLC and Cognex

The Siemens instructions for connecting the S7-1200 PLC to the Cognex cameras can be found in the instructions at the following:

MIME_4450_Notes>

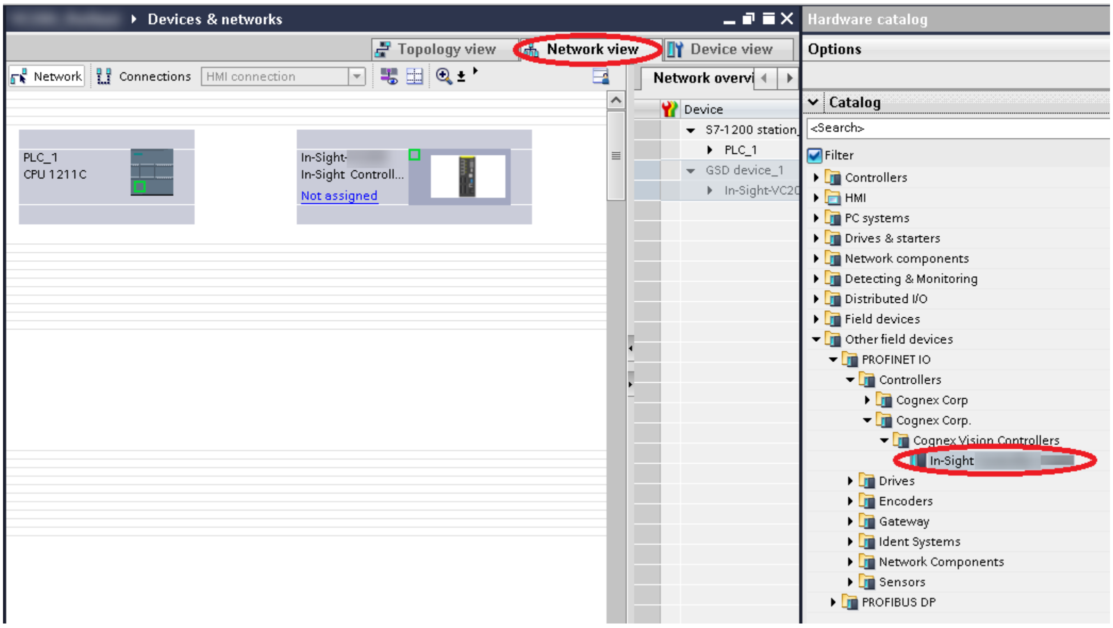
Cognex_Siemens> (this is for Siemens P

InSight Explorer IS7802>

Cognex Siemens Quick Start Guide ISE 2023 05 04_7802.pdf

InSight Vision Suite_IS3805>

Cognex Siemens Quick Start Guide ISVS 2023 05 04_3805.pdf



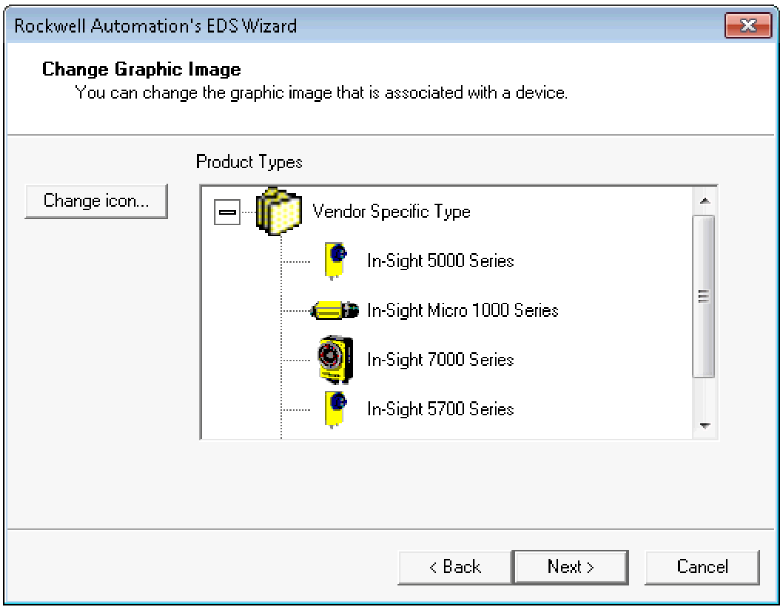
The Rockwell instructions for connecting the 16ER-BB1B PLC to the Cognex cameras can be found in the instructions at the following:

MIME_4450_Notes>

Cognex_Rockwell_IS7802_ISE>

Cognex Rockwell Quick Start Guide ISE 2023 11 01_7802.pdf

Cognex Rockwell Quick Start Guide ISVS 09012023_3805.pdf



Appendix 1

Flash for CompactLogix Processor

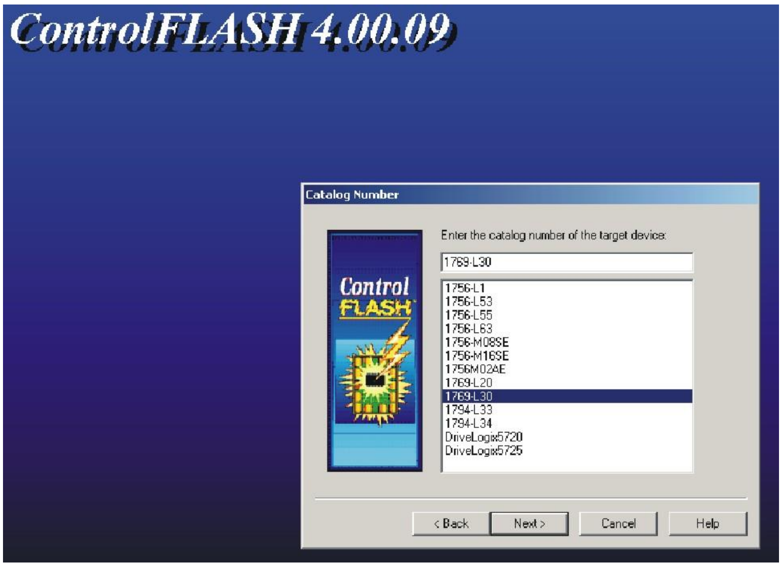


Fig. 16-40 Flash Software for A-B Processors

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.

Appendix 2

Table of 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
(dc4)  20 0024 0x14 | 4      52 0064 0x34 | T      84 0124 0x54 | t     116 0164 0x74
(nak)  21 0025 0x15 | 5      53 0065 0x35 | U      85 0125 0x55 | u     117 0165 0x75
(syn)  22 0026 0x16 | 6      54 0066 0x36 | V      86 0126 0x56 | v     118 0166 0x76
(etb)  23 0027 0x17 | 7      55 0067 0x37 | W      87 0127 0x57 | w     119 0167 0x77
(can)  24 0030 0x18 | 8      56 0070 0x38 | X      88 0130 0x58 | x     120 0170 0x78
(em)   25 0031 0x19 | 9      57 0071 0x39 | Y      89 0131 0x59 | y     121 0171 0x79
(sub)  26 0032 0x1a | :      58 0072 0x3a | Z      90 0132 0x5a | z     122 0172 0x7a
(esc)  27 0033 0x1b | ;      59 0073 0x3b | [      91 0133 0x5b | {     123 0173 0x7b
(fs)   28 0034 0x1c | <      60 0074 0x3c | \      92 0134 0x5c | |     124 0174 0x7c
(gs)   29 0035 0x1d | =      61 0075 0x3d | ]      93 0135 0x5d | }     125 0175 0x7d
(rs)   30 0036 0x1e | >      62 0076 0x3e | ^      94 0136 0x5e | ~     126 0176 0x7e
(us)   31 0037 0x1f | ?      63 0077 0x3f | _      95 0137 0x5f | (del) 127 0177 0x7f
```