ControlLogix Level 2

Data Manipulation

EthernetSupport.com

"Customized Automation Training"

13JAN08

Disclaimer:

This document is written in the hope that you can utilize for your own education to gain knowledge of PLC systems (should you decide to utilize this document).

Although I believe the information in this document to be accurate, it is YOUR responsibility to verify this information before implementing it in any way, especially when damage to personnel or equipment could result.

By continuing to read this document, you agree to hold no one who writes, modifies, or distributes this document liable in any way (even negligence).

Due to the wide variety of plant applications, some of the examples in this document may be prohibited at your location, or could cause damage to equipment, or harm personnel.

About the Author:

This document is a collection of texts and graphics I've put together over the past few years, and has been distributed under the GFDL since 1999.

I hope you get much use out of it, and I would like your feedback as to how this document can be improved.

As a supplement to this document, I would like to invite you to my website at **http://www.LearnAutomation.com**. I'm in the process of uploading documentation and videos that will further help you with problems or questions you have with Allen Bradley processors.

"Human Knowledge Belongs to Everyone"

Table of Contents

Name____Date__________________

Questionnaire for Allen Bradley Automation Systems

1)What is the primary purpose you are attending this class?

2)Are you interested in programming, troubleshooting, or both?

3)What do you find most difficult about Allen Bradley PLC's

4)How often do you access the Allen Bradley PLC? (once a day, once a week, once a month, etc?)

5)After taking this class, will you be putting your knowledge to use right away in the plant?

6)What type of equipment do you generally work with?

7)What types of networks are you using with your PLC system? Ie… Data Highway plus, controlnet, devicenet, Ethernet, etc?

8)What is your company's policy on forcing?

9)Do you generally have access to the Internet as you work?

10)Can you bring a copy of some plant programs into the classroom tomorrow?

11)Will you ever be installing new systems, or checking new systems once they have been installed?

12)Will you ever be modifying the I/O structure of existing systems?

13)Do you have any common system failures that are related to the Allen Bradley PLC? If so, what are these failures

14)Are you interested in learning features of RSLogix that are not currently in use by your plant, but, if used could reduce downtime?

Name______________________________Date_______________Score____________

ControlLogix Level 2
Pre-Test

1) If the I/O light on the processor is flashing green, what does this indicate?

2) What does a solid red OK light on the processor indicate?

3) What is the difference between the **controller tag** database, and the **program tag** database?

4) How many **Tasks** can be set up in a controller?

5) How many of these tasks can execute continuously?

6) How many programs are allowed in a task?

7) What instruction must be placed in the MainRoutines so the SubRoutines will execute?

8) What programming languages are available in the ControlLogix system?

9) What is a user-defined data type, and how can they be used to organize the tag database?

10)Diagnostic input modules can detect the difference between an open switch and a broke wire. How is this possible?

11)What is a **produced** tag?

12)What is a **consumed** tag?

13)If the processor is in program mode, is the producer/consumer model still transferring data?

14)Block transfers to an analog module using the remote I/O protocol are handled using what instruction in logic?

15)What has to be different about every device on Ethernet?

16)What protocol does the 1756-ENBT module use in order to communicate with ethernet I/O devices such as Flex I/O and drives?

17)What is the difference between an ethernet hub and an ethernet switch?

18) What software must be used to schedule connections on ControlNet?

19) What software is used to map devices on a DeviceNet network to memory locations in the processor?

20) What is 'Electronic Keying'? Describe the 3 settings for electronic keying

1. Disable Keying:

2. Compatible Module:

3. Exact Match:

21) Although there is not currently a Custom Data Monitor for RSLogix 5000, what are some options you have to monitor data in an organized manner?

22) When the 1756-ENBT module is brand new (out of the box), it has no IP address assigned. What are some methods you can use to assign the module an IP address?

23) What software is used to flash the firmware in specialty modules such as the processor, ethernet module, and devicenet module?

24) If a process is running, and is dependant on a PLC you are downloading to, what is most likely going to happen to the process when your download starts?

25) What utility can be used in RSLinx to backup the current driver configuration?

Glossary

Addressing:

Bit: Smallest unit of information the PLC can process– ON or OFF
Word: 32 Bits for Double Integer

Hardware:

Input module: Reads the STATUS of field devices

Output module: CONTROLS field devices

Discrete module: Reads or controls devices which only have 2 states: on or off

Analog Module: Reads or controls devices which have a range such as 0 to 10 volts or 4 to 20 milliamps

Power Supply: Provides control power to modules on the backplane

Chassis: The physical device that modules are plugged into.

Local Chassis: The chassis where the processor resides.

Processor: The 'Brain' of the PLC which contains the machine program.

Troubleshooting Tools:

Cross Referencing:

Allows the troubleshooter to quickly navigate through the program by listing all locations in ladder logic where a particular address is located.

Usually the troubleshooter will cross reference a false condition on a rung of logic to find the output that will turn the referenced bit on.

You can Access Cross Referencing by right clicking a particular address. (Note: You must be on the address, not the instruction)

Custom Data Monitor Utility:

Allows the troubleshooter to gather data from various memory locations onto one screen for easy troubleshooting. For example: One can create a custom data monitor for the failure of a particular motor. Next time the motor fails, the troubleshooter can simply look down the list of conditions that must be met, and see in real time which condition is causing the failure.

The custom data monitor utility has to be installed as a separate tool. Since the CDM utility is not built into RSLogix, you must have RSLinx activated.

RSLinx Lite will not work with the CDM utility.

Force:

Simulates real world jumpers. Use care while performing a force. You must understand fully how a force is going to affect your system. In most cases, only addresses starting with an I: or an O: can be forced.

To force an input or output, you can right click on the address in logic, then choose force on or force off. After the force is installed, forces can then be enabled from the on line tool bar.

Trending:

Trending acts somewhat like a 'software chart recorder', and allows you to track an analog signal over time.

Communication terminology:

RSLinx: This is the communication server. If RSLinx is not set up properly, RSLogix will not communicate to the processor.

Driver: Allows RSLinx to communicate with a particular hardware device. The most common drivers are the DF1 driver to communicate with Channel 0, and the PCMK driver for a laptop to communicate to Channel 1A for station 5. Configure drivers by clicking communication on the menu bar.

RSWho: A graphical screen which will display what devices RSLinx has established communication with. Access RSWho by clicking communication on the menu bar. Then click on the name of the driver you wish to use for communication. The right hand side of the screen will reveal devices the driver has communication with.

RSLogix: The software which allows you to troubleshoot or program a processor.

Online: Actively communicating with the processor (ladder spinning)

Download: If a program was changed offline, it must be downloaded (or sent to) the processor. When downloading the processor must be in program or remote program mode. A good way to download once RSLinx is properly set up is to click COMMS on the menu bar, and then go to Who Active. Click the driver name, highlight your processor, then click DOWNLOAD.

Memory Layout:

Tags: A section of the processor's memory that stores information. You can also think of tag elements as variables. For example: The memory location “MainTorque” could store a drive torque value.

There are two scopes of tags: Program tags which are local to the program they Reside in, and Controller Scoped tags which are global. You can access the Program Tag database by double clicking program tags just above the MainRoutine of each program. Controller tags can by accessed at the top of the Controller organizer window.

Tasks: A section of the processor's memory which holds programs. Programs hold Routines. Each controller can have multiple tasks with multiple programs in each task. Each of these programs can then have multiple routines.

Atomic Data Types:

BOOL	1-bit boolean 0 = cleared 1 = set
SINT	1-byte integer -128 to 127
INT	2-byte integer -32,768 to 32,767
DINT	4-byte integer -2,147,483,648 to 2,147,483,647
REAL	4-byte floating-point number -3.402823E38 to -1.1754944E-38

Understanding Numbering Systems

Understanding numbering systems will help you to understand various ways in which data can be monitored in the ControlLogix processor. For example, if you are reading the value of a limit switch, you would want to change the numbering system (style) to binary. If you were viewing data from an analog module, you would want to set the style for decimal. The PLC-5 modules are numbered in an Octal addressing scheme, so if you read data from a PLC-5 module, you may want to set the style to Octal. Some devices such as LED displays are wired in Hex/BCD. The two most common styles for most plants will be Binary and Decimal.

1) **Binary** – Binary is a base 2 numbering system. You only have two numbers available in Binary, 0 and 1 for any position (ones, tens, hundreds, etc...). Binary is the most common style for discrete I/O such as limit switches, pushbuttons, solenoids, and motor starters.
2) **Decimal** -- Decimal is a base 10 numbering system. Only 10 numbers exist in the Decimal numbering scheme (0 to 9) for any given position. The Decimal style is used most often when displaying analog data, such as a pressure or temperature.
3) **Octal** – Octal is a base 8 numbering system. In Octal, 8 numbers are available (0-7) for use in any position. In older PLC's such as the PLC-2, and the PLC-5, The I/O modules were numbered in the Octal addressing scheme. The Octal style can be used when connecting to one of these older modules.
4) **Hexadecimal/Binary Coded Decimal** – Hex/BCD is a base 16 numbering system. 16 numbers are available for any digital position (0 to 9 then A to F). Devices such as LED displays and thumb wheels can be BCD Devices.

On the following chart, you will write down the decimal numbers 0 to 15, and the numeric conversions for each of these numbering systems. This chart will help you understand topics covered later in the course such as masking.

Decimal	Binary	Octal	Hex/BCD

Hardware -- Discrete Input Modules

The purpose of the discrete input module is to read the status of field devices. When a voltage is detected on the terminal of an input module with respect to common, the corresponding status light is energized, and during the processor scan, the value of 1 is placed into the input data table. Examples of input devices include switches, pushbuttons, or auxiliary contacts on a motor starter. The Removable Terminal Block (RTB) can be detached from the module if the locking tab is pushed up. The enclosure of the RTB will also slide off the terminal block for easy access to the terminals.

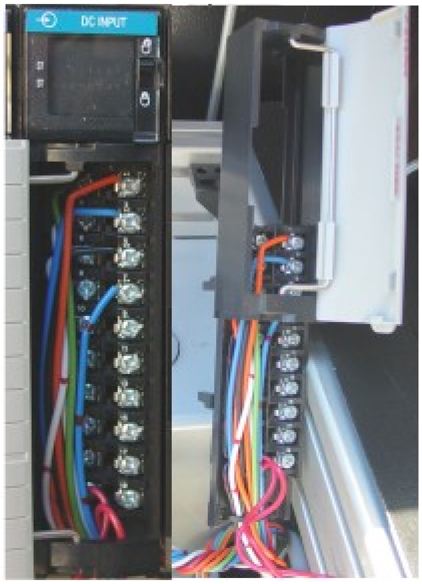
Please answer the following questions:

1) What is the catalog number of your DC Input module?

2) Name at least three field devices that can be connected to the DC Input module?

3) What do the status lights indicate on the front of the DC Input Module?

4) What is the slot number of the DC Input module at your station.



Discrete Output Modules

The purpose of the discrete output module is to control field devices. The discrete output module requires power from an external source. When a 1 is placed into the output tag of the ControlLogix (in run mode), a status light is energized on the module, and a connection is made between the source, and the corresponding output terminal. Examples of output devices include: lights, solenoids, motor starter coils, and contactors. If you have an inductive load as the output, be sure to use the proper surge suppression.

Please answer the following questions:

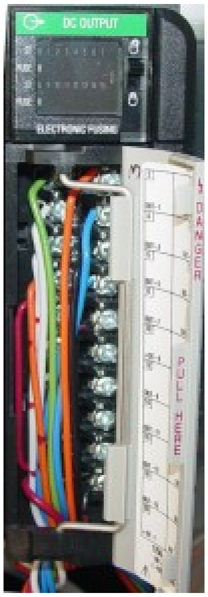
1) What is the catalog number of your DC Output module?

2) Name at least three field devices that can be connect to the DC Output module:

3) What do the status lights indicate on the DC Output module?

4) What is the slot number of the DC Output Module?

5) If the load on the DC output card is inductive, what should be done across the load to minimize the effects of inductive kick?



Analog Modules

Analog modules are used to control and read the status of analog devices. Analog devices have a range of states instead of just on/off states like discrete devices.

Some analog modules have switches which determine whether the input channels are to be set up for voltage or current. Some analog modules are configured through software.

Examples of analog inputs include: Potentiometer, Pressure Transducers, Variable speed drives, and with a thermal couple module, temperature can be read into the processor's memory.

Examples of analog outputs include: Meters, Variable Speed Drives, Valve Positioners, and chart recorders.

An analog signal cannot be expressed with a single bit, and therefore analog values will consume a word of memory. For our class, we will use the Analog module on the Flex I/O chassis.

Please answer the following questions:

1) What is the catalog number of your analog module?

2) How many channels of Input, and how many channels of Output are available on this module?

3) How do you set up the input channels to accept either a current or a voltage input?

4) What range voltage or current will the Input channels accept on your module? What range of voltage will the output channels accept?

5) Name at least three devices that are analog inputs:

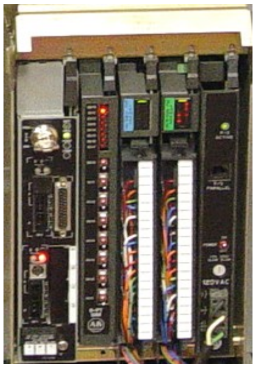
6) Name at least three field devices that are analog outputs:

The Chassis

The chassis is the device which holds modules. Allen Bradley makes the ControlLogix chassis available in 4, 7, 10, 13, and 17 slots.

Here are some chassis:

The PLC-5 Chassis (With modules): (For dip switch settings on this chassis, refer to page 4-1 and 4-2 of the PLC-5 Quick Reference Guide.)



The SLC-500 Chassis (With modules):

The ControlLogix Chassis (With Modules):

The Flex Chassis:

The Power Supply

The power supply supplies power to the modules on the backplane. Generally power from field devices DOES NOT come from the power supply. The power supply only provides control power to modules on the backplane. Power for field devices come from a separate source which is connected to the output module. The power supply merely provides the power needed to shut a contact, or fire a triac or transistor to pass power from this external source to the field device. On the back of the power supply, a jumper is used to set the voltage range.

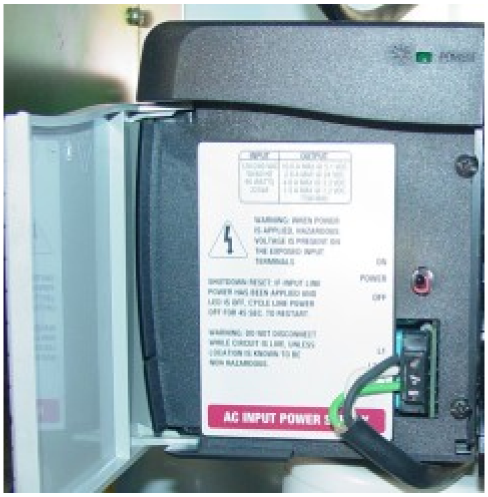
Please answer the following questions:

1) Where does power come from to power field devices such as solenoids, lights, and motor starters?

2) What must be set up on a new power supply before it can be placed into service?

3) How many amps will your power supply provide to the backplane?

4) What is the catalog number of your power supply?


POWER
WARNING: WHEN POWER IS APPLIED, HAZARDOUS VOLTAGE IS PRESENT ON THE EXPOSED INPUT TERMINALS
SHUTDOWN RESET: IF INPUT LINE POWER HAS BEEN APPLIED AND LED IS OFF, CYCLE LINE POWER OFF FOR 45 SEC. TO RESTART
WARNING: DO NOT DISCONNECT WHILE CIRCUIT IS LIVE, UNLESS LOCATION IS KNOWN TO BE NON HAZARDOUS
ON
POWER
OFF
AC INPUT POWER S

The Processor

The processor is the main part of your ControlLogix system. The processor is where the program is stored that reads the status of your equipment, and based on certain status, makes a decision on what to control. For example: The processor is reading the status of a switch. When the operator energizes the switch, the processor might call for solenoid to energize that extends a cylinder. When the cylinder reaches the end of it's travel, it might close a limit switch. The processor will see that a limit switch has been closed, and shut off the solenoid. Although traditionally the processor usually is placed in slot 0, it can be placed anywhere in the chassis, as long as the program is setup for the processor to be in that slot. You can also use as many processors as you like in a chassis (not to exceed the limitation of the power supply)

The processor consists of several components:

1) The battery: The battery retains the processor's program when the PLC is powered down. Certain AB documentation states that the shelf life of the battery is up to 2.5 years. When the battery is low, you will see a BATT light on the front of the processor. A minor fault bit is also set in the memory of the processor when the battery is low or missing.

2) On the front of the processor, you will find several status lights:
   1. RUN – Indicates when when processor is in RUN Mode
   2. OK – If flashing red, usually indicates a software problem. Go on line to get a description of the fault. You will find the description in the Controller Properties on the FAULT tab. If the fault light is solid red, this could indicate a hardware problem. You can try the following: re-seat the processor, clear memory and reload program, or replace processor.
   3. BATT-- Indicates the battery is low or missing
   4. IO – If flashing indicates the Processor lost it's connection with at least one I/O device.
   5. FORCE – If flashing indicates forces are installed but not enabled... If solid indicates forces are installed, and enabled in the processor. This indicator is not available on all ControlLogix processors.
   6. RS232 – This light will flicker as data is transferred over the RS232 port (channel 0).

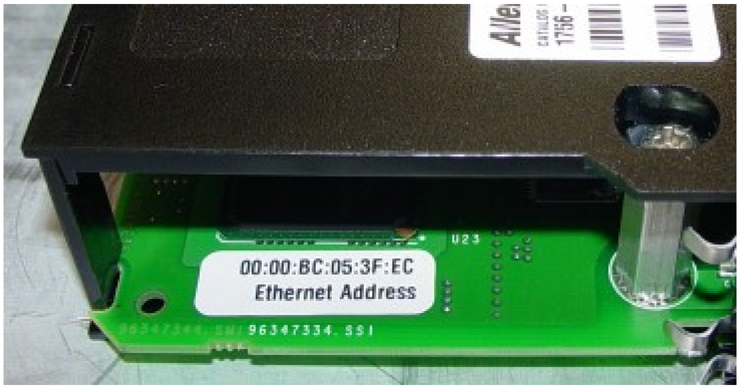
3) The Key Switch:
   1. Run Mode: In this position, certain tag values can be modified, but ladder logic cannot. The mode of the processor cannot be changed to program mode from the On line tool bar in RSLogix. When the switch is in run mode, a program cannot be downloaded to the processor.
   2. Program Mode: In this position, the ladder is not executing. Changes can be made to the ladder diagram or to data files. The processor cannot be changed to run mode from the on line tool bar in RSLogix while the switch is in this position.
   3. Remote Mode: When the key switch is in Remote Mode, the mode of the processor can be changed from RSLogix (Program or Run). On line editing is allowed.

The ControlLogix Processor:

The Ethernet Module

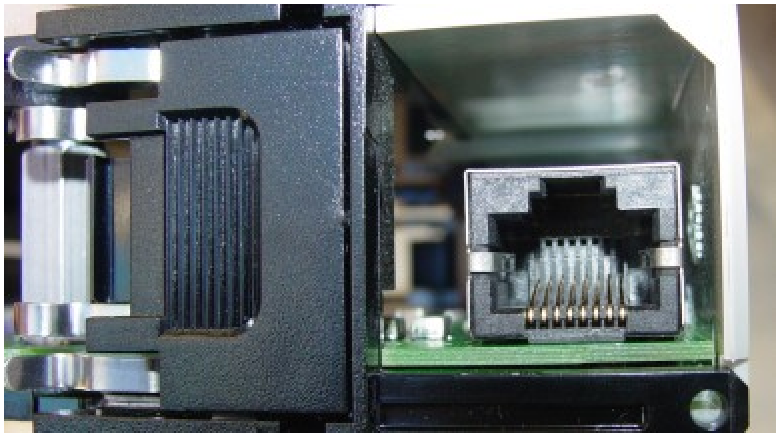
Ethernet is a protocol that has been widely used for many years. Before Ethernet was used with PLC systems, it was used in Office environments for sharing files, printers, data from databases, etc... Ethernet is the fastest communication protocol available for the ControlLogix system with speeds up to 100 Mbps (Million bits per second). Ethernet can be used to communicate with the ControlLogix system from a computer, for communication between controllers, to allow the controller to communicate with I/O, Or MMI (man-machine interface) devices to communicate with processors.

Every device on Ethernet has a Unique hardware address which can usually be found on the device itself, or on a configuration screen for the device. This hardware address can be used to issue the Ethernet module an address using a bootP utility. The hardware address will also scroll across the alphanumeric display if no IP address has been assigned.



Once the IP address is assigned, the IP will scroll across the alphanumeric display. This IP address can then be used in the Ethernet Driver for RSLinx, or if type the IP address into the address bar of a web browser, such as Mozilla, the modules on board web server will show the module's status, and the status of every other module in the chassis.

On the bottom of the 1756-ENBT module, you will find an RJ-45 port. Using a standard patch cable, you can connect the module to an Ethernet switch or a hub. To connect to your computer directly, you will need a crossover cable (switches transmit and receive).

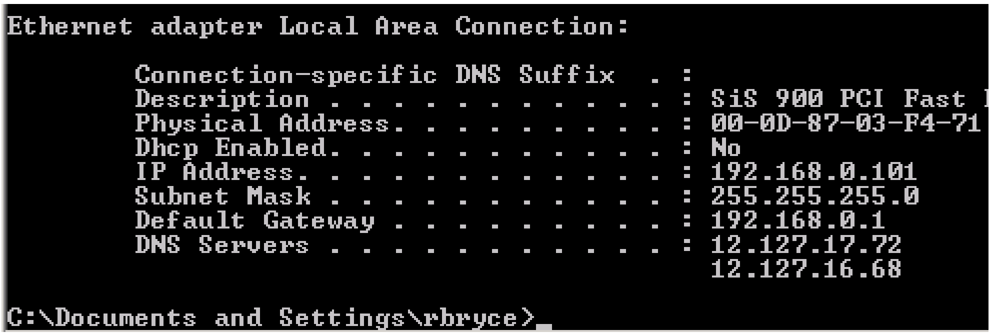


Ethernet Addressing

Every device on the same Ethernet network must have a unique address. Allen Bradley PLC's currently use the IP (Internet protocol) addressing scheme (version 4). This is the addressing scheme discussed in this document.

Examples of Ethernet devices might be a Personal Computer (PC), a 1756-ENBT module, a 1794-AENT flex adapter, a printer with built-in print server capabilities, and plant servers for data storage and processing.

To see what the IP address is of your Windows NT, 2000, or XP machine type:
ipconfig /all
at the command prompt. Here is the result:



```
Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        Description . . . . . . . . . . . : SiS 900 PCI Fast
        Physical Address. . . . . . . . . : 00-0D-87-03-F4-71
        Dhcp Enabled. . . . . . . . . . . : No
        IP Address. . . . . . . . . . . . : 192.168.0.101
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.0.1
        DNS Servers . . . . . . . . . . . : 12.127.17.72
                                            12.127.16.68

C:\Documents and Settings\rbryce>_
```

For Unix/Linux systems, the command **/sbin/ifconfig** will product similar results.

In this example, my PC has the IP address of **192.168.0.101**. No other machine on the same network will have this same IP address, nor should you attempt to assign the same address to another device on the network. If this happens, one of the devices will not be seen.

Each segment of the IP address is called an **OCTET**. All IP addresses in the IPv4 addressing scheme are made up of 4 octets. Each octet is an 8 bit unsigned integer.

Subnet Mask

Notice the Subnet Mask: **255.255.255.0**. The purpose of the subnet mask is to identify which part of the IP address is the network address, and which part of the IP address is the host or terminal on the network. To understand how the subnet mask works, it has to be broken down into a binary format... Look at the IP address in Binary:

11000000 . 10101000 . 00000000 . 01100101 = 192.168.0.101

Now look at the subnet mask:

11111111 . 11111111 . 11111111 . 00000000 = 255.255.255.0

Anywhere a 1 exists in the subnet mask, that bit of the IP address is viewed as the NETWORK part of the address.... Lets see what bits are passed:

```
11000000  .  10101000  .  00000000 . 01100101  =  192.168.0.101
11111111  .  11111111  .  11111111 . 00000000  =  255.255.255.0
-----------------------------------------------------------------------
11000000  .  10101000  .  00000000 . don't pass =  192.168.0.X
```

Wherever there was a 1 in the subnet, we passed that bit of the IP address as part of the network address. Therefore, we would say the network address for this machine is 192.168.0.X. Every device on this network must have an IP address that starts with 192.168.0. and X is the terminal address on the network.

For example: Devices with these two IP addresses can communicate with each other directly without going through a router:

192.168.0.3 and 192.168.0.200 with subnet 255.255.255.0

These two devices cannot communicate with each other directly:

192.168.1.3 and 192.168.3.200 with subnet 255.255.255.0

However if the subnet mask was changed:

192.168.1.3 and 192.168.3.200 with subnet 255.255.0.0

These two devices will communicate with each other because the network address is only made up of the first two octets, 192.168.X.X... since the network address is the same for the two devices, they will communicate directly.

Subnet Mask Exercise:

Indicate whether or not the following devices can communicate with each other directly:

1. 192.168.0.5 and 192.168.0.6 with subnet mask of 255.255.255.0 YES NO

2. 192.168.1.5 and 192.168.0.6 with subnet mask of 255.255.255.0 YES NO

3. 192.168.1.5 and 192.168.0.6 with subnet mask of 255.255.0.0 YES NO

4. 10.1.1.5 and 10.1.1.6 with subnet mask of 255.255.255.0 YES NO

5. 10.1.1.5 and 10.1.1.6 with subnet mask of 255.0.0.0 YES NO

6. 10.2.5.5 and 10.1.1.6 with subnet mask of 255.255.255.0 YES NO

7. 10.2.5.5 and 10.1.1.6 with subnet mask of 255.0.0.0 YES NO

8. 10.2.5.5 and 10.1.1.6 with subnet mask of 255.255.0.0 YES NO

9. 192.168.0.1 and 10.1.1.6 with subnet mask of 255.0.0.0 YES NO

10. 10.2.1.5 and 10.1.1.6 with subnet mask of 255.0.0.0 YES NO

Other Terms:

GATEWAY
The gateway address is the IP address of a server or hardware router that connects you to other networks such as the Internet.

DNS (Primary and Secondary)
The DNS server (Domain Name Server) resolves host names into IP addresses. When you enter an address such as Yahoo.com into your web browser, your PC does not understand where to go. It must ask the DNS server to look up the IP address of a given host names. Host names are for humans to understand. Computers understand IP addresses.

DHCP
Dynamic Host Configuration Protocol – When a device such as a computer is configured to use DHCP, A DHCP server should be available on the network. As soon as the device connects to the network, it will ask the DHCP server to automatically assign an IP address, subnet mask, DNS servers, Gateway address, etc. This address is dynamic, so the device could get a different IP address each time it's connected.

BootP
Bootstrap Protocol – Similar to DHCP, except a BootP device will get the same IP address every time it connects. The BootP server has a list of hardware addresses, and IP addresses that belong to each device. When a device (such as a PLC) connects to the network, it will give the BootP server it's hardware address. The server will then look up the hardware address in a list, and see which IP address belongs to the PLC. The BootP server will then return the IP address (and other information such as the subnet mask) to be used by the device.

Using the Host file:

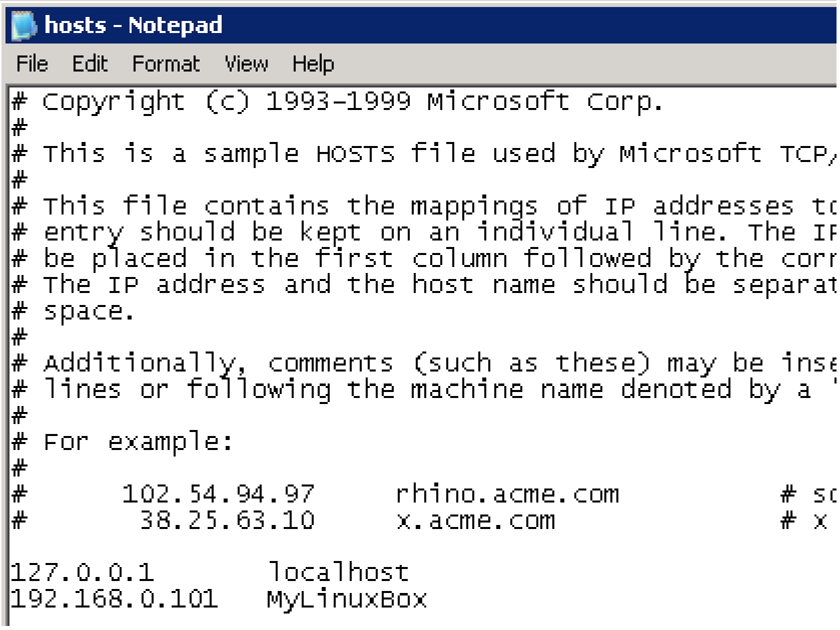
In the early days of the Internet (ARPA NET), there were no DNS servers to resolve friendly host names into IP addresses. Everyone had a text file on their computer containing every host name available, and the IP address for that host name. Eventually there were so many domain names, this was too difficult to maintain. The host file still resides on your computer today, and still can be used to resolve host names.

If DNS servers are not available on a network, you may want to use a HOST file to resolve friendly names into IP addresses.

On Windows XP, the host file is at this location: **C:\WINDOWS\system32\drivers\etc\hosts**

For Linux/Unix machines: **/etc/hosts**

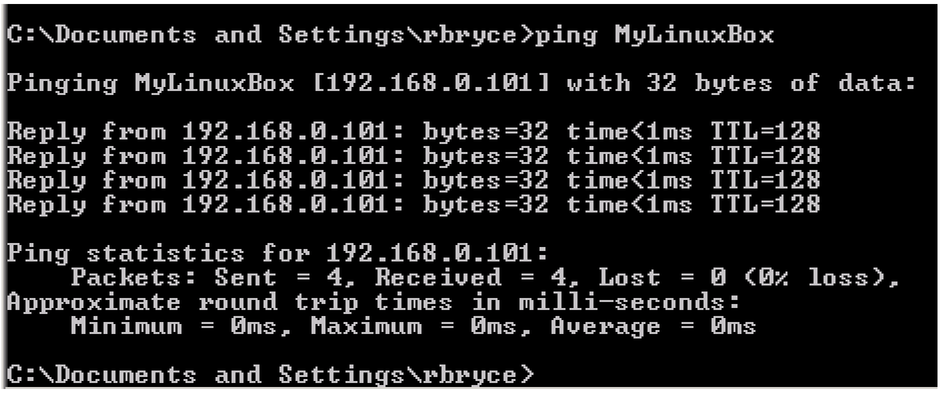
Here is an example of the host file. In this case, I added an entry for MyLinuxBox.



```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/
#
# This file contains the mappings of IP addresses tc
# entry should be kept on an individual line. The IF
# be placed in the first column followed by the corr
# The IP address and the host name should be separat
# space.
#
# Additionally, comments (such as these) may be inse
# lines or following the machine name denoted by a '
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # sc
#       38.25.63.10     x.acme.com              # x

127.0.0.1       localhost
192.168.0.101   MyLinuxBox
```

After making the change, the configuration must be saved. (Click File | Save).

From the command prompt, try to ping the friendly name:



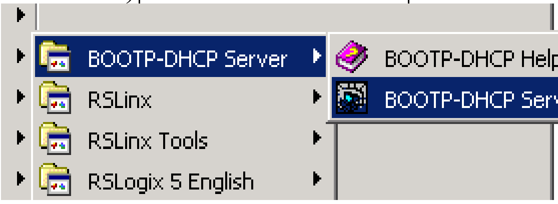
```
C:\Documents and Settings\rbryce>ping MyLinuxBox

Pinging MyLinuxBox [192.168.0.101] with 32 bytes of data:

Reply from 192.168.0.101: bytes=32 time<1ms TTL=128
Reply from 192.168.0.101: bytes=32 time<1ms TTL=128
Reply from 192.168.0.101: bytes=32 time<1ms TTL=128
Reply from 192.168.0.101: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\rbryce>
```

Notice after pinging MyLinuxBox, the replies are reported as IP addresses. This host name can also be used in the web browser's address bar if the device has a built in web server such as the 1756-ENBT module.

RSLinx -- Utilizing the BootP/DHCP Server

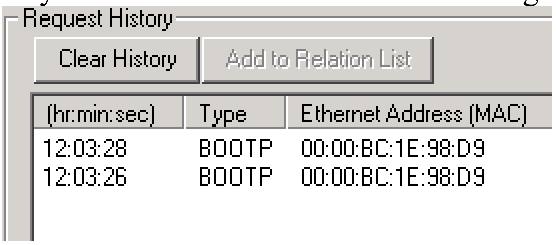
1) Write down the Ethernet Hardware Address of the device you wish to configure. This is also called the MAC address. Here is an example of a hardware address:
   00:00:BC:1E:98:D9

2) Run the BootP/DHCP utility. You can access this utility if it is installed by clicking Start | Programs (or All Programs in Windows XP)| Rockwell Software | BootP/DHCP Server | BootP DHCP Server.



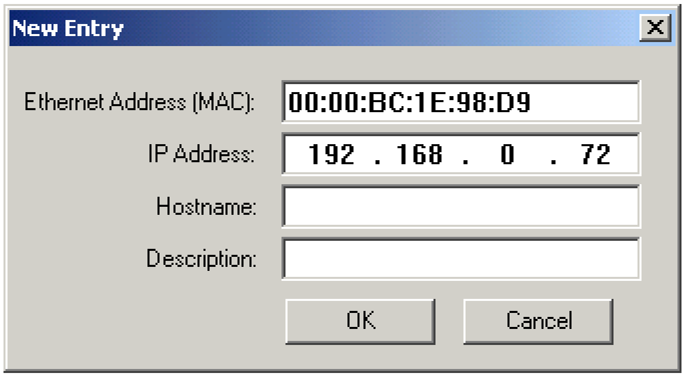
3) Once the server is open, it may as for some specific network information if this is the first time the BootP server has been run. You can obtain this information from your network administrator. For this example, we just set the subnet mask to 255.255.255.0 then click OK.

4) Power up your processor, and you should see the Ethernet device begin to request and address.



(hr:min:sec)	Type	Ethernet Address (MAC)
12:03:28	BOOTP	00:00:BC:1E:98:D9
12:03:26	BOOTP	00:00:BC:1E:98:D9

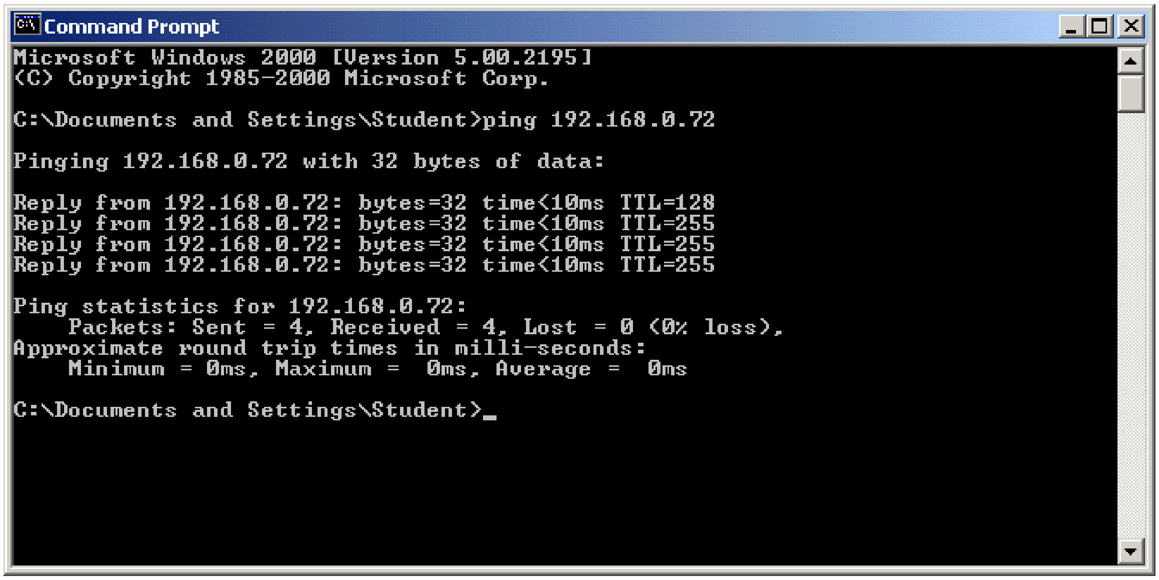
5) Double click on the device. You will then be prompted to assign an IP address to the device. Be sure you double click the right device. Entering an IP address into the wrong equipment could have disastrous consequences.



Ethernet Address (MAC): 00:00:BC:1E:98:D9
IP Address: 192 . 168 . 0 . 72
Hostname:
Description:

6) You could also enter a host name and description at this time if you wish.

7) If you wish to verify communication, you can ping the device from the command prompt. By default, the command prompt can be accessed from Start | Programs | Accessories | Command Prompt.



```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Student>ping 192.168.0.72

Pinging 192.168.0.72 with 32 bytes of data:

Reply from 192.168.0.72: bytes=32 time<10ms TTL=128
Reply from 192.168.0.72: bytes=32 time<10ms TTL=255
Reply from 192.168.0.72: bytes=32 time<10ms TTL=255
Reply from 192.168.0.72: bytes=32 time<10ms TTL=255

Ping statistics for 192.168.0.72:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms

C:\Documents and Settings\Student>_
```

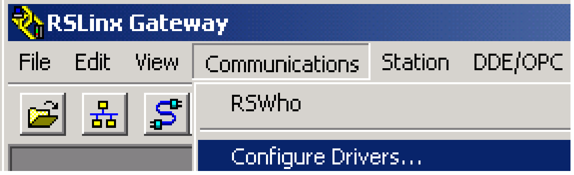
You should get replies. To ping continuously, use the -t flag after the ping command. A control-C will stop the ping command.

Setting up the Ethernet Driver in RSLinx

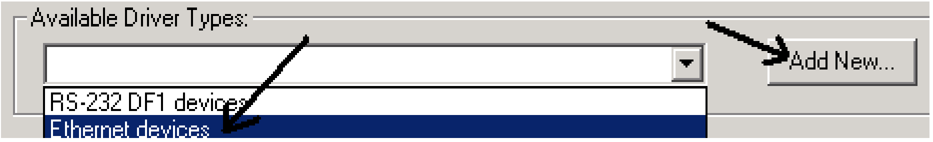
The Ethernet driver is used to make a connection to Ethernet Devices, such as an Ethernet PLC-5, or a ControlLogix system. The following steps will walk you through a sample configuration of the Ethernet driver in RSLinx.

1) Open RSLinx communication server

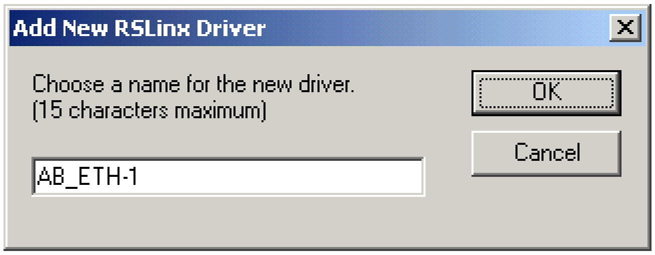
2) Click 'Communication' on the menu bar, and then choose 'Configure Drivers'.



3) From the Available driver types pull down menu, choose 'Ethernet Drives', then press the 'Add New' button.



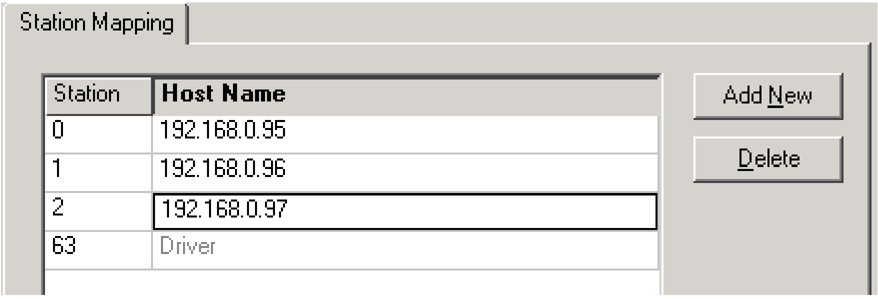
4) For this example, the name can be left at default. Press OK.



Choose a name for the new driver.
(15 characters maximum)

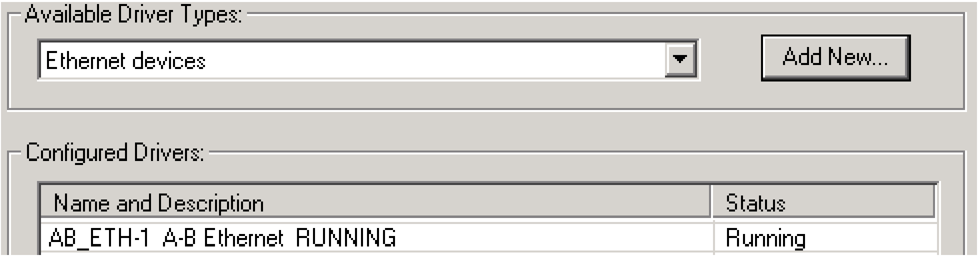
AB_ETH-1

5) Populate the list of hostnames. If you do not have a way to resolve hostnames, you can enter the IP address of the devices you wish to connect to (as shown below in the example). The IP address for each device can usually be obtained from the network administrator, drawings, the offline project, or in some cases, the IP address is displayed on the front of the module.



Station	Host Name
0	192.168.0.95
1	192.168.0.96
2	192.168.0.97
63	Driver

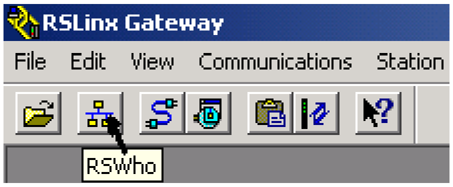
6) Press Apply, then OK. You will see the driver is now running. Close the 'Configure Drivers' screen.



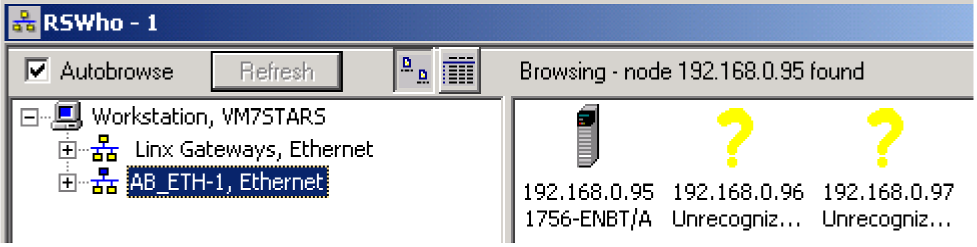
Available Driver Types: Ethernet devices

Name and Description	Status
AB_ETH-1 A-B Ethernet RUNNING	Running

7) To test your drivers, click the RSWho icon in the tool bar of RSLinx.\



8) Click the Ethernet Driver (the one you just configured) on the left side of the screen. The devices you are communicating with will appear on the right. In this example, 192.168.0.95 is a ControlLogix module, and the other devices are not present, or not a recognized PLC module.



Browsing - node 192.168.0.95 found

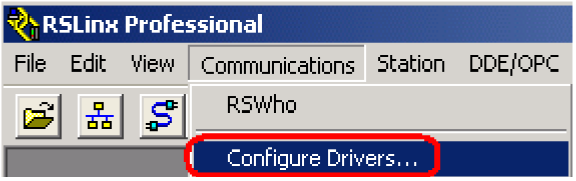
9) To go on line with a PLC, you must go to RSLogix at this point.

Configuring the DF1 Driver in RSLinx

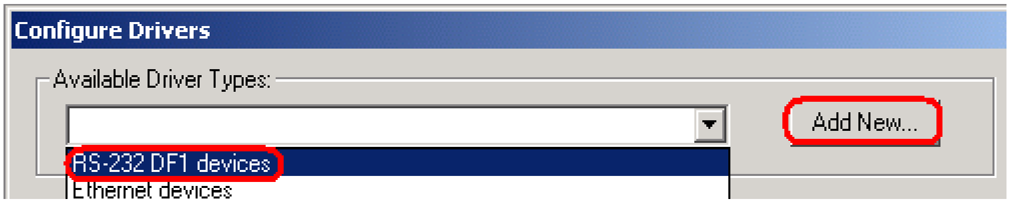
The DF1 Driver is used for point to point communication over RS232 between a COM port on a PC, and the serial port (Channel 0) of a processor. The following steps will take you through a sample configuration of the DF1 RS232 driver.

1) Open RSLinx Communication Server. If there is no short-cut on the desktop, you can access RSLinx by clicking Start | Programs | Rockwell Software | RSLinx | RSLinx

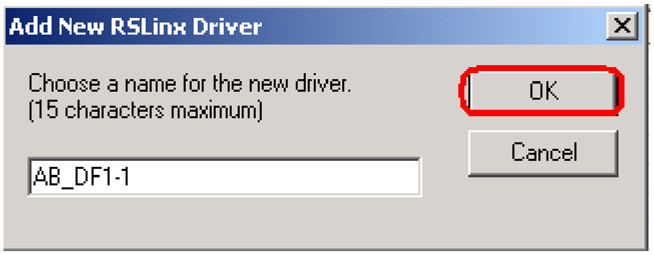
2) Click 'Communication' on the menu bar, then choose 'Configure Drivers'.



3) From the Available Driver Types pull down menu, choose 'RS232 DF1 Devices', then press the ADD NEW button.



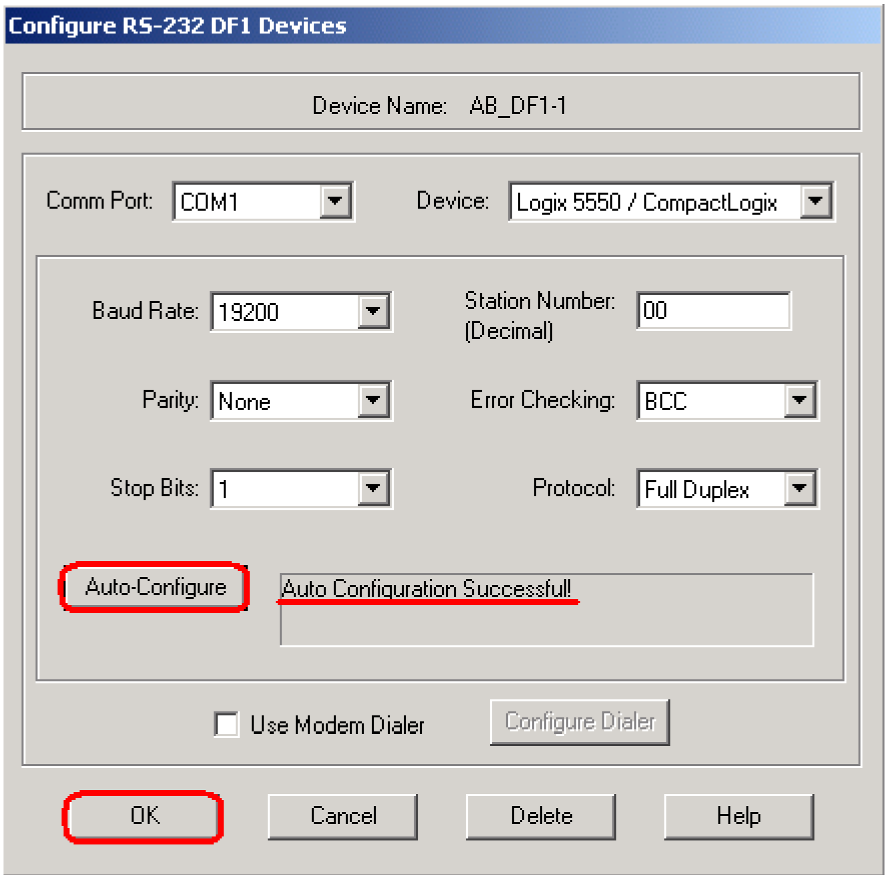
4) For this example, the name can be left at default. Press OK.



Choose a name for the new driver.
(15 characters maximum)

AB_DF1-1

5) Although the communication parameters can be entered manually, if you are currently connected to the processor, just hit the 'AutoConfigure' button. RSLinx will hit the processor with different baud rates, and different settings, until it finds a setting it gets a response on. When this happens, you will get a message that the autoconfiguration was successful. Press OK when finished.



Configure RS-232 DF1 Devices

Device Name: AB_DF1-1

Comm Port: COM1

Device: Logix 5550 / CompactLogix

Baud Rate: 19200

Station Number: (Decimal) 00

Parity: None

Error Checking: BCC

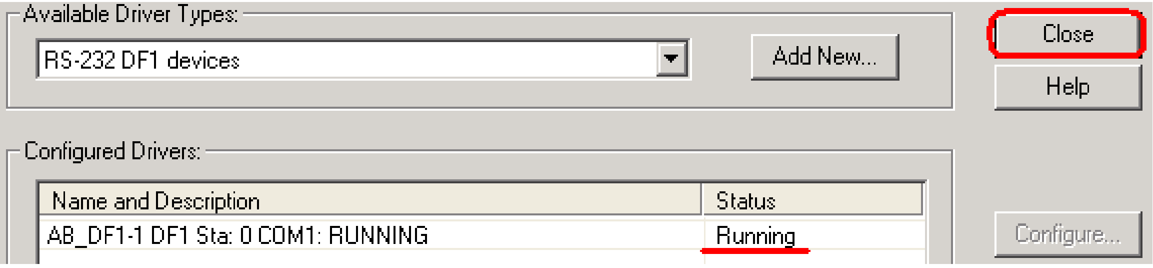
Stop Bits: 1

Protocol: Full Duplex

Auto Configuration Successful!

□ Use Modem Dialer

6) You will see the driver is now running. Close the “Configure Drivers' screen.

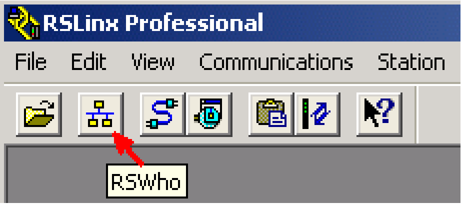


Available Driver Types: RS-232 DF1 devices

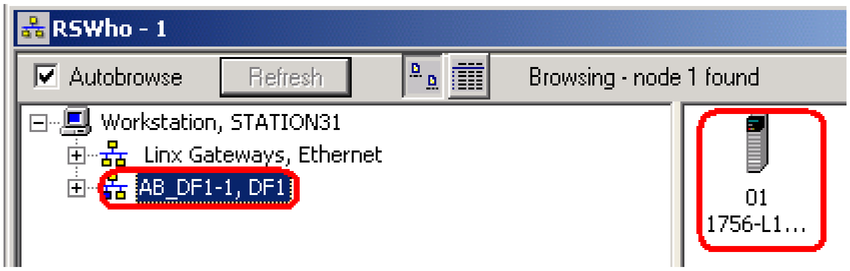
Configured Drivers:

Name and Description	Status
AB_DF1-1 DF1 Sta: 0 COM1: RUNNING	Running

7) To test your drivers, click the RSWho icon in the tool bar of RSLinx.



8) Click the DF1 driver (the one you just configured) on the left side of your screen. The devices you are communicating with will appear on the right.



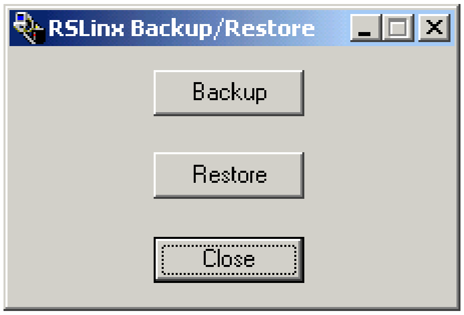
To go on line, you must go to RSLogix at this point.

RSLinx Backup Restore Utility

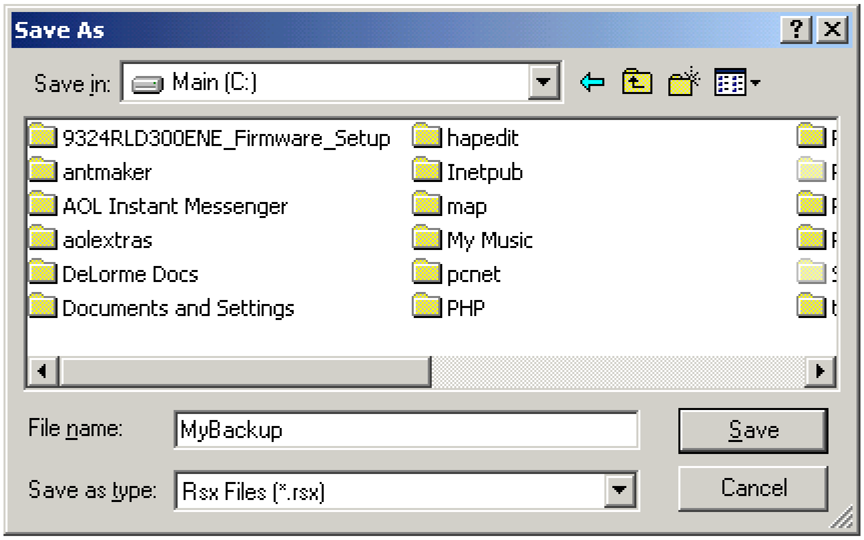
The RSLinx Backup Restore Utility can be used to backup the current driver configuration of RSLinx, or restore the configuration from a previous backup at an earlier time.

Backing up the current Configuration:

To access the Backup Restore Utility, click Start | Programs | Rockwell Software | RSLinx | Backup Restore Utility.



1) Click the 'Backup' button.
2) A dialog screen will appear asking where you want to save the backup file. Choose a location from the pull down menu. If you are saving this to a floppy disk, choose the A drive. For this example, the driver configuration will be saved to the C: Drive. You must also enter a file name, and then press SAVE.



Save in: Main (C:)

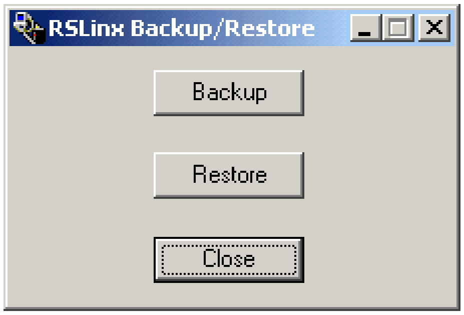
File name: MyBackup

Save as type: Rsx Files (*.rsx)

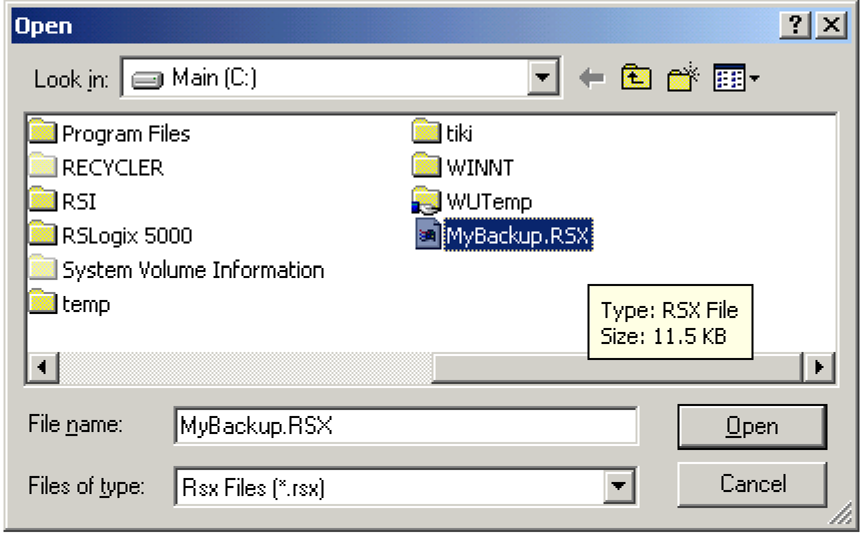
3) You will then get a message that the operation completed successfully. Press OK, then you can close the RSLinx Backup Restore Utility.

Restoring a previous Configuration:

To access the Backup Restore Utility, click Start | Programs | Rockwell Software | RSLinx | Backup Restore Utility. Be aware that RSLinx must be shut down to perform this operation. If RSLinx is not shut down, you will be prompted accordingly.



1) Click the 'Restore' button.
2) A dialog screen will appear asking where the backup file is stored at. Choose a location from the pull down menu. If the backup file was on a floppy disk, you would choose the A: drive. For this example, the backup is on the C: Drive. Click on the file you wish to restore from, the press 'Open'.



Look in: Main (C:)

Type: RSX File
Size: 11.5 KB

File name: MyBackup.RSX

Files of type: Rsx Files (*.rsx)

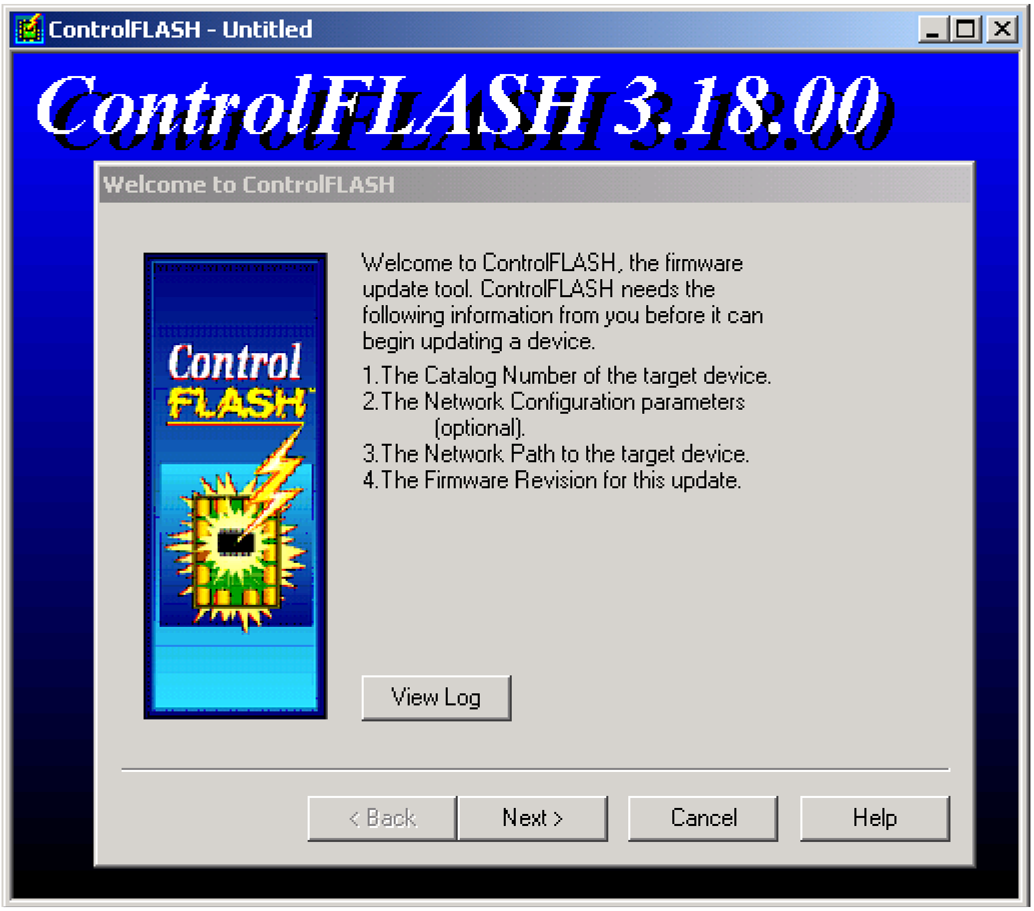
3) A dialog box should appear indicating that the operation was completed successfully. If you got an error, try the restore procedure again. In some versions of RSLinx, BRU must be ran twice if RSLinx was open.

Flashing ControlLogix Modules

In order to flash ControlLogix firmware, you need to have the Control Flash utility installed. If it is not installed on your PC, the installation program can probably be found on the same disk as RSLogix 5000.

In this example, we are going to flash a 1756- L1 processor.

1) Before starting, make sure necessary drivers are configured in RSLinx. If your drivers are not configured in RSLinx, then restore from the appropriate backup file, or consult your documentation on how to configure specific drivers to communicate with your equipment. To open the Control Flash utility click START|PROGRAMS|Flash Programming Tools|Control Flash.

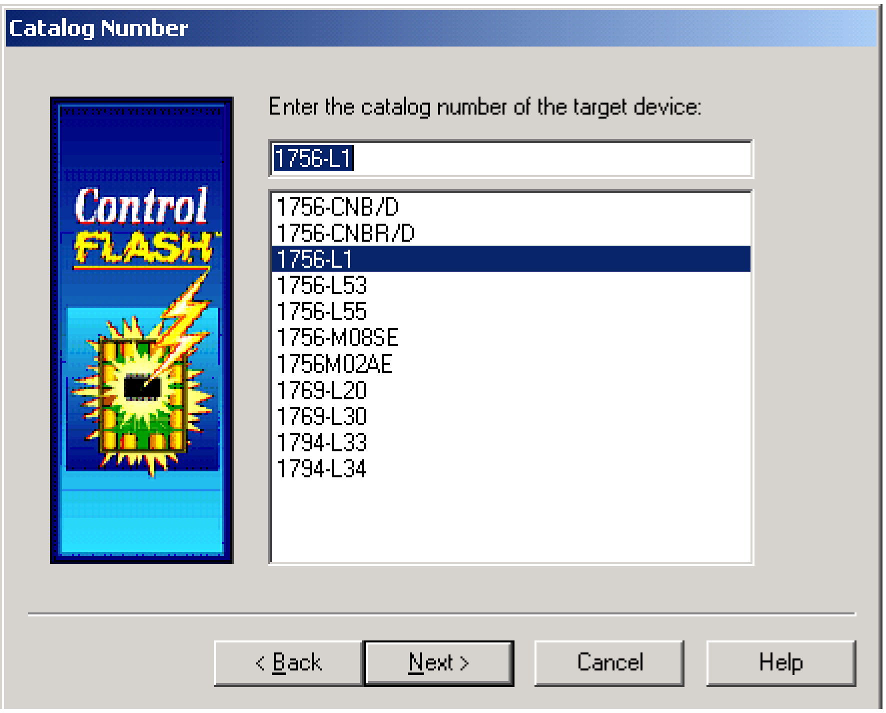


Welcome to ControlFLASH, the firmware update tool. ControlFLASH needs the following information from you before it can begin updating a device.

1. The Catalog Number of the target device.
2. The Network Configuration parameters (optional).
3. The Network Path to the target device.
4. The Firmware Revision for this update.

2) Be sure to read all instructions that are presented to you, and take all necessary precautions while flashing a module. If your flash procedure is interrupted, you could damage the module.

3) On the first screen, we are going to press the NEXT button.



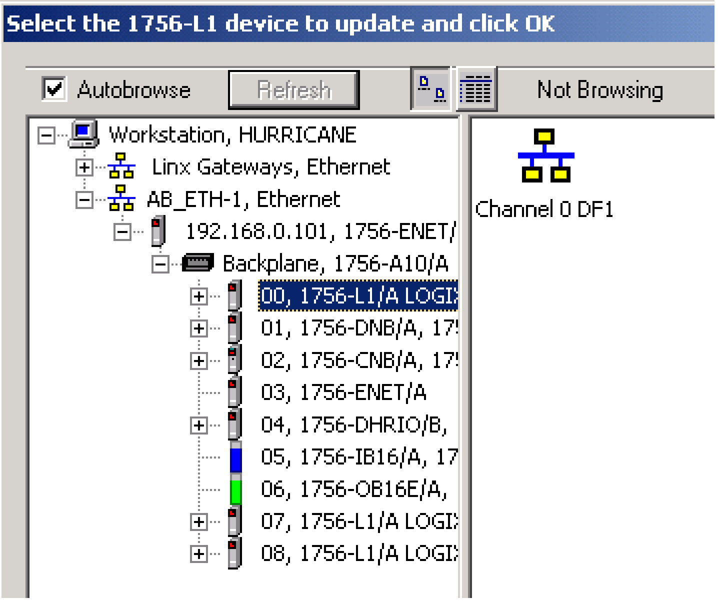
Catalog Number

Enter the catalog number of the target device:

1756-L1

1756-CNB/D
1756-CNBR/D
1756-L1
1756-L53
1756-L55
1756-M08SE
1756M02AE
1769-L20
1769-L30
1794-L33
1794-L34

4) Choose the device that you wish to flash. For this example, we will use the 1756-L1 option. Then press NEXT. Now the ControlFlash utility will ask for the path of the device you wish to flash. Drill down through RSLinx until you find the device you are wanting to flash. Highlight the device, then press OK.



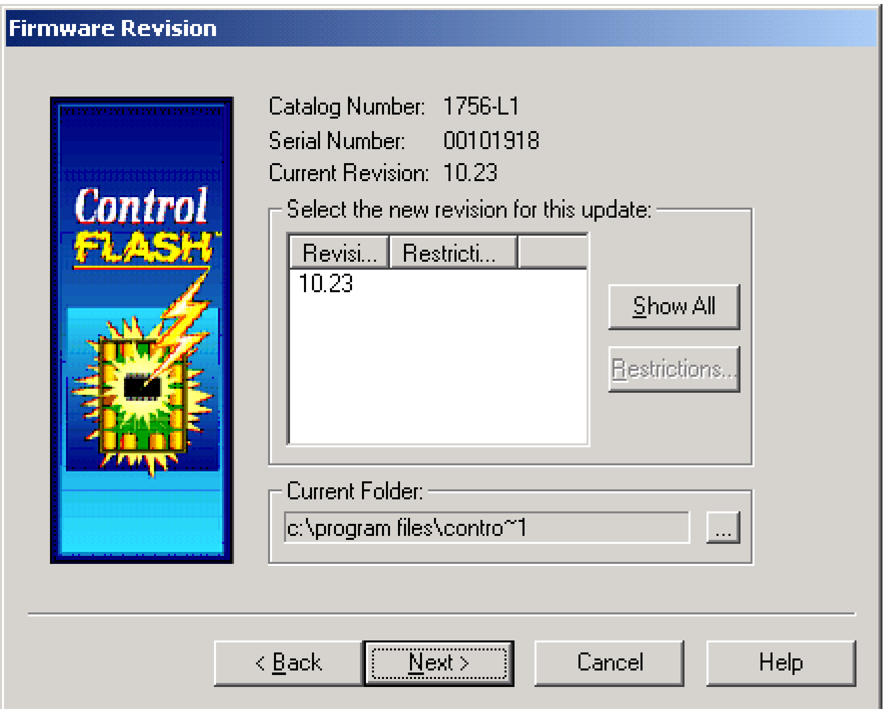
Select the 1756-L1 device to update and click OK

☑ Autobrowse

Not Browsing

Workstation, HURRICANE
Linx Gateways, Ethernet
AB_ETH-1, Ethernet
192.168.0.101, 1756-ENET/
Backplane, 1756-A10/A
00, 1756-L1/A LOGI
01, 1756-DNB/A, 17
02, 1756-CNB/A, 17
03, 1756-ENET/A
04, 1756-DHRIO/B,
05, 1756-IB16/A, 17
06, 1756-OB16E/A,
07, 1756-L1/A LOGI
08, 1756-L1/A LOGI

5) Next you are asked what firmware version you want to flash the module to. In this case, the module will be flashed to firmware version 10.23. If the processor's firmware was not the same firmware that we needed, we would highlight the revision of firmware we want for this update, and then press NEXT.



Firmware Revision

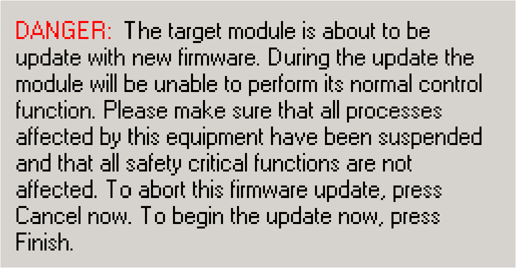
Catalog Number: 1756-L1
Serial Number: 00101918
Current Revision: 10.23

Select the new revision for this update:

Revisi...	Restricti...
10.23	

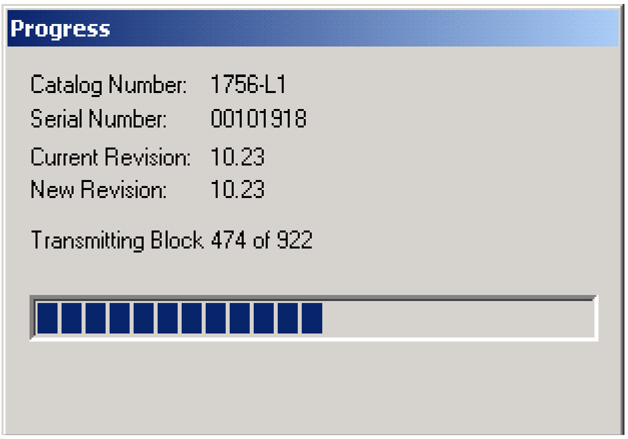
Current Folder: c:\program files\contro~1

6) Next you will receive a warning. Read the warning carefully. If you still wish to flash the module, press FINISH.



DANGER: The target module is about to be update with new firmware. During the update the module will be unable to perform its normal control function. Please make sure that all processes affected by this equipment have been suspended and that all safety critical functions are not affected. To abort this firmware update, press Cancel now. To begin the update now, press Finish.

The firmware update will now take place.

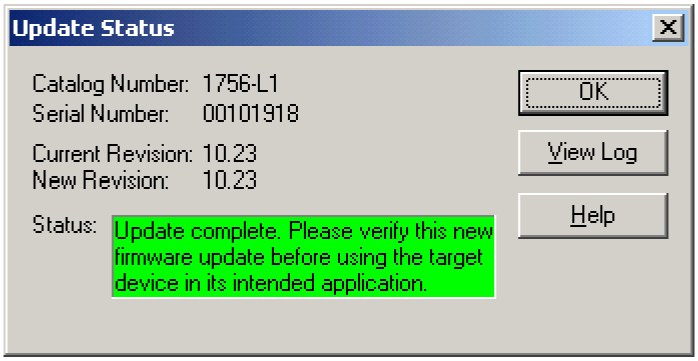


Catalog Number: 1756-L1
Serial Number: 00101918
Current Revision: 10.23
New Revision: 10.23

Transmitting Block 474 of 922

There may be more than one stage in updating a module, so be sure not to interrupt the update process until it is completely finished.

When the update is finished, you will see the following message:

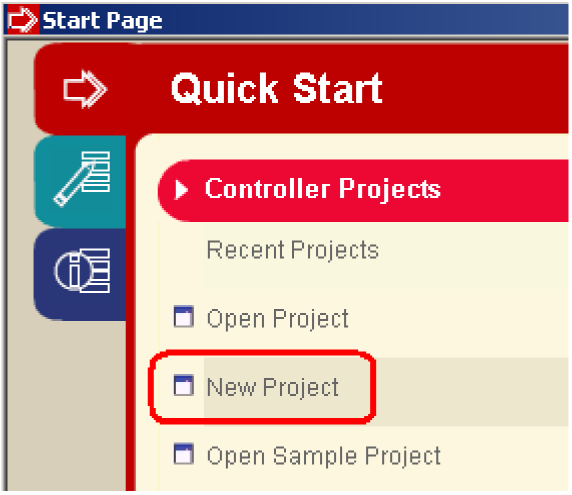


Catalog Number: 1756-L1
Serial Number: 00101918
Current Revision: 10.23
New Revision: 10.23
Status: Update complete. Please verify this new firmware update before using the target device in its intended application.

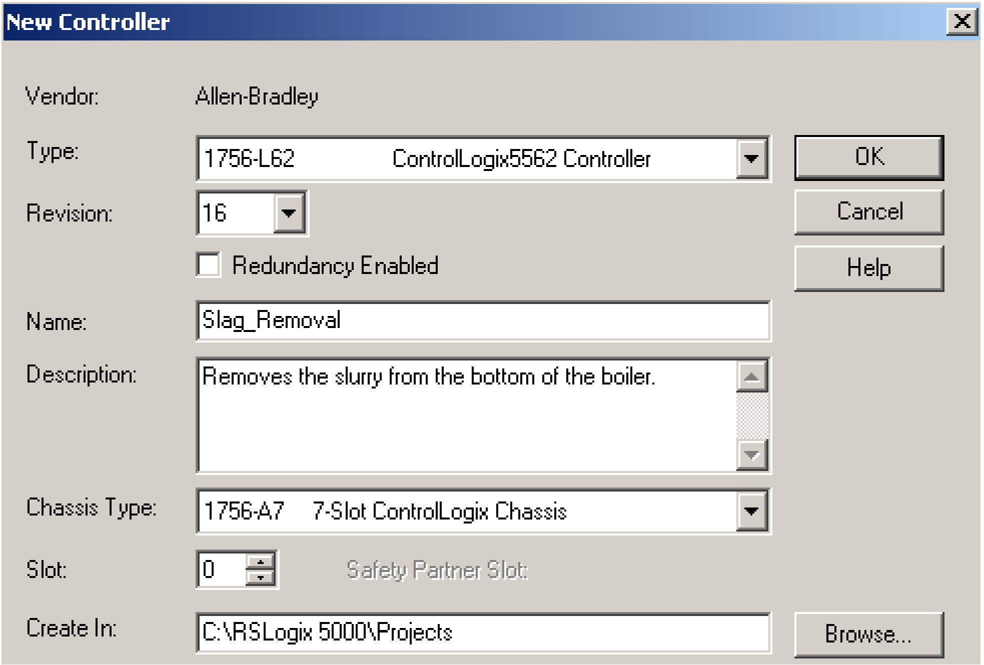
Creating a new RSLogix 5000 Project

1) Open RSLogix 5000 – You may have a short cut on the desktop, or under Start | Programs | Rockwell Software | RSLogix 5000 Enterprise Series | RSLogix 5000

2) Select “New Project”



3) Next, we are going to set up a new processor. The image below is simply for example. You will want to populate the fields according to the specifications of your own local system.


New Controller

Vendor: Allen-Bradley

Type: 1756-L62 ControlLogix5562 Controller

Revision: 16

□ Redundancy Enabled

Name: Slag_Removal

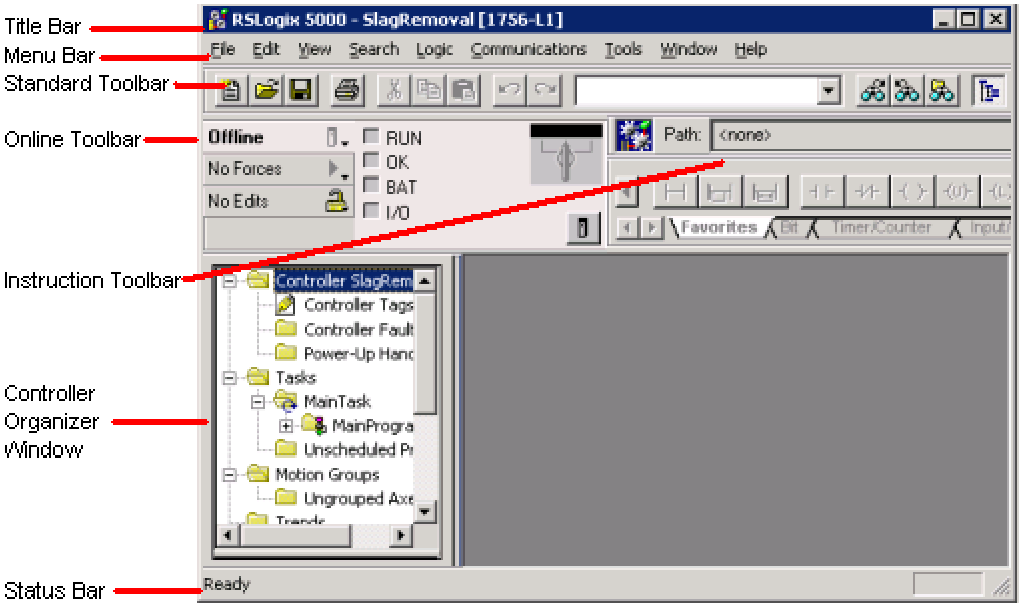
Description: Removes the slurry from the bottom of the boiler.

Chassis Type: 1756-A7 7-Slot ControlLogix Chassis

Slot: 0 Safety Partner Slot:

Create In: C:\RSLogix 5000\Projects

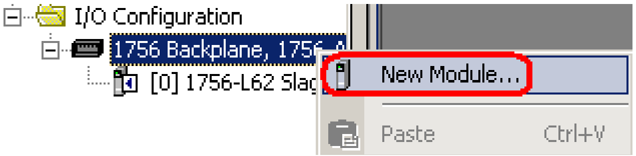
4) You have a valid project. It will not do much though because we have not configured any I/O yet. Let's take a moment to discuss some components of the RSLogix 5000 user interface.



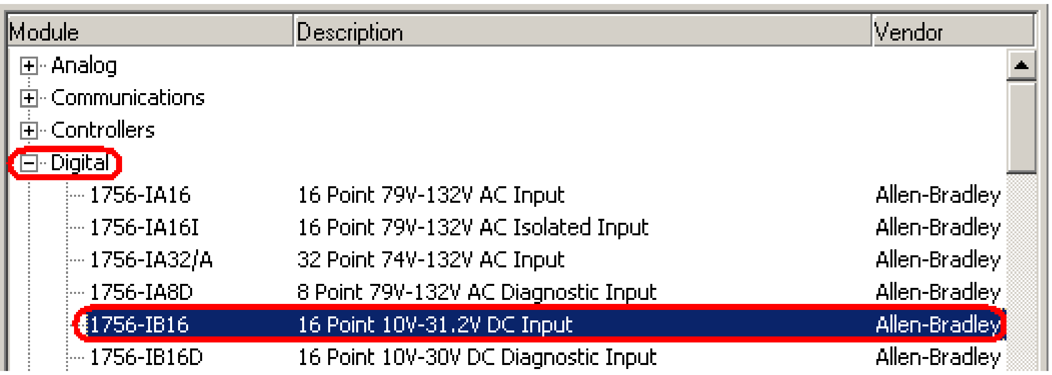
5) Next we will add the following modules to the I/O Configuration at the bottom of the controller organizer window:

1. 1756-IB16	Version 2.5	Slot 5
2. 1745-OB16E	Version 2.4	Slot 6

6) To do this, right click the backplane in the I/O Configuration folder of the project tree, and select '**New Module**'.

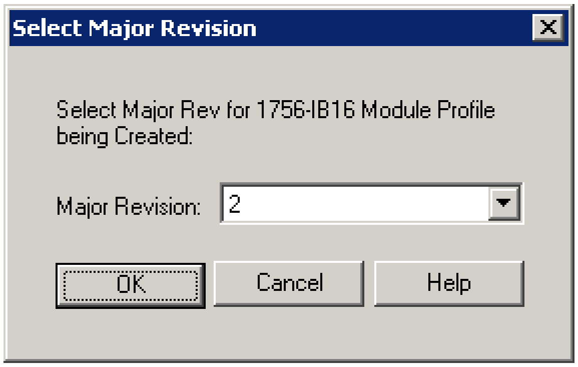


7) Expand the digital modules, and locate your 1756-IB16 module.



Module	Description	Vendor
⊞ Analog		
⊞ Communications		
⊞ Controllers		
⊟ Digital		
1756-IA16	16 Point 79V-132V AC Input	Allen-Bradley
1756-IA16I	16 Point 79V-132V AC Isolated Input	Allen-Bradley
1756-IA32/A	32 Point 74V-132V AC Input	Allen-Bradley
1756-IA8D	8 Point 79V-132V AC Diagnostic Input	Allen-Bradley
1756-IB16	16 Point 10V-31.2V DC Input	Allen-Bradley
1756-IB16D	16 Point 10V-30V DC Diagnostic Input	Allen-Bradley

8) The module is version 2.5, so the Major revision is 2, and the minor revision is 5. Click OK on the Major revision dialog box if your revision is correct.

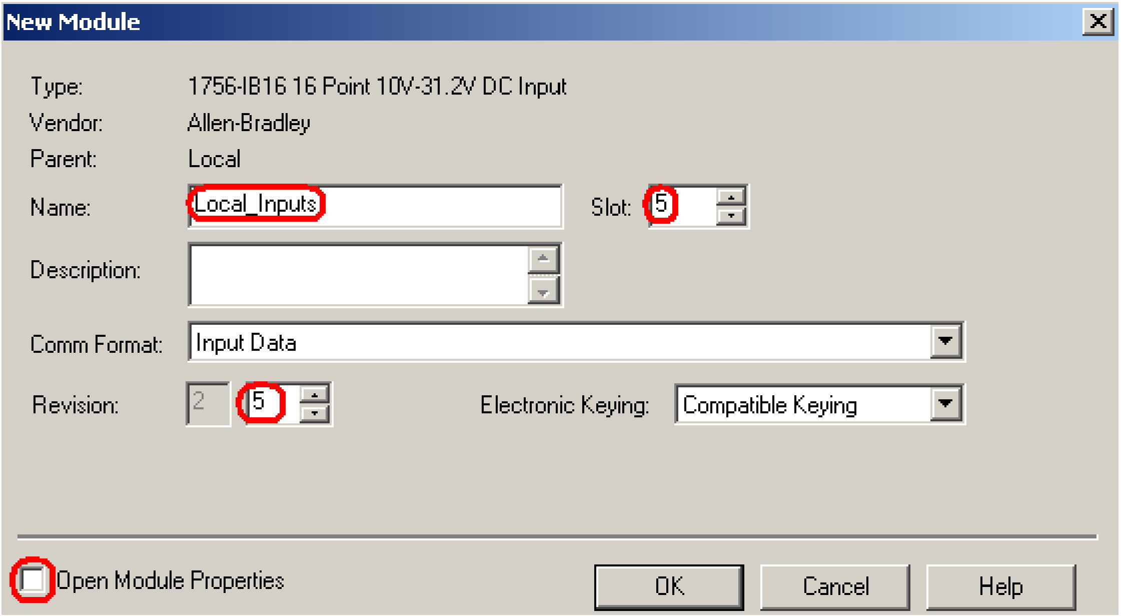


Select Major Revision

Select Major Rev for 1756-IB16 Module Profile being Created:

Major Revision: 2

9) Complete the module properties dialog box as follows, then click '**Finish**'.



New Module

Type: 1756-IB16 16 Point 10V-31.2V DC Input

Vendor: Allen-Bradley

Parent: Local

Name: Local_Inputs   Slot: 5

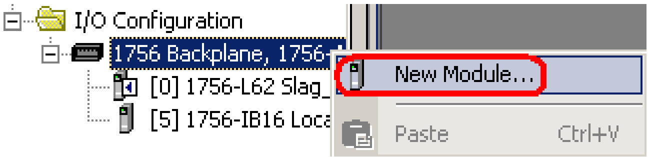
Description:

Comm Format: Input Data

Revision: 2 5   Electronic Keying: Compatible Keying

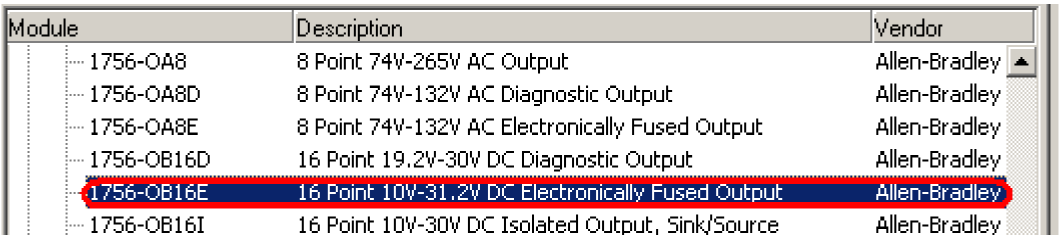
□ Open Module Properties

10)You will notice the 1756-IB16 module appears in the I/O Configuration folder. Next, we will add the 1756-OB16E. Right click the backplane in the I/O Configuration folder, and select '**New Module**'.



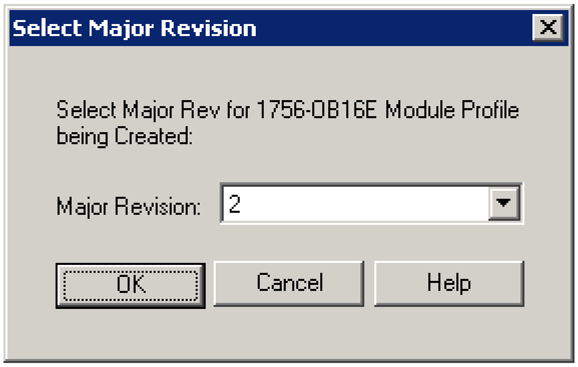
I/O Configuration
1756 Backplane, 1756-
[0] 1756-L62 Slag_
[5] 1756-IB16 Loca

11)Choose the 1756-OB16E card from the digital modules list, then press **OK**.



Module	Description	Vendor
1756-OA8	8 Point 74V-265V AC Output	Allen-Bradley
1756-OA8D	8 Point 74V-132V AC Diagnostic Output	Allen-Bradley
1756-OA8E	8 Point 74V-132V AC Electronically Fused Output	Allen-Bradley
1756-OB16D	16 Point 19.2V-30V DC Diagnostic Output	Allen-Bradley
1756-OB16E	16 Point 10V-31.2V DC Electronically Fused Output	Allen-Bradley
1756-OB16I	16 Point 10V-30V DC Isolated Output, Sink/Source	Allen-Bradley

12)Since the module is version 2.4, choose 2 as the major revision.

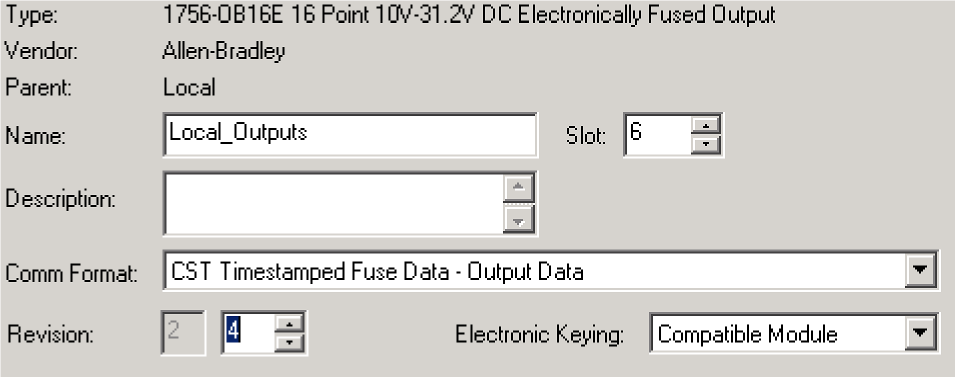


Select Major Revision

Select Major Rev for 1756-OB16E Module Profile being Created:

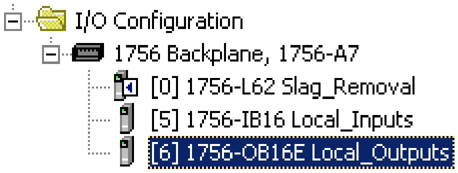
Major Revision: 2

13)Complete the module properties dialog as shown, then press '**Finish**'.

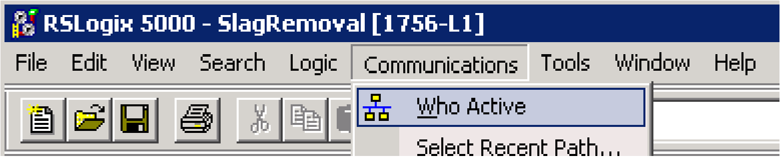


Type: 1756-OB16E 16 Point 10V-31.2V DC Electronically Fused Output
Vendor: Allen-Bradley
Parent: Local
Name: Local_Outputs
Slot: 6
Description:
Comm Format: CST Timestamped Fuse Data - Output Data
Revision: 2 4
Electronic Keying: Compatible Module

14)You will notice both modules are now in the I/O Configuration tree. Be sure to adjust this procedure to suit the modules you are actually using with your own system.


I/O Configuration
1756 Backplane, 1756-A7
[0] 1756-L62 Slag_Removal
[5] 1756-IB16 Local_Inputs
[6] 1756-OB16E Local_Outputs

15)You are ready to download. Since the communication path has not been selected yet, let's click Communications | WhoActive From the menu bar



5) Choose the path to your processor, and download.

Note: When no path has been developed, a 4 step procedure can be used to download to the processor. Make note of this page. If your download fails because a path is needed, follow these 4 steps.

A) Click Communications | Who Active
B) Browse to your ***<u>PROCESSOR</u>***
C) Highlight your ***<u>PROCESSOR</u>***
D) Click **Download**, Upload, or Go Online. In this particular case, we want to download.

You CANNOT download to any module in this case, except for the processor. The processor is the module which stores the PLC program. If your function buttons are gray (in such a way that you cannot click the button), chances are that you do not have the processor highlighted.

Working With Tags

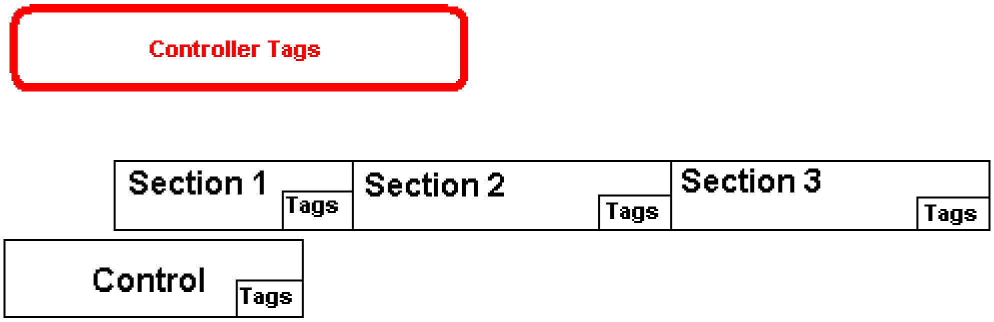
Tags are **variables** that your program reads and manipulates. There are two types of Tags: **Controller tags**, and **Program tags**.

Controller tags are **GLOBAL**, which means that any program or any controller can read or write to the tag. Program tags are **LOCAL** to the program they reside in.

Controller Tags:

If one program needs to communicate with another program, you would use a Controller Tag that each program can read from or write to. For this example, lets say a conveyor has three segments. Each segment has it's own program. When an E-Stop button is pressed, we would want all sections to shut down. Therefore the ESTOP would be a GLOBAL tag.

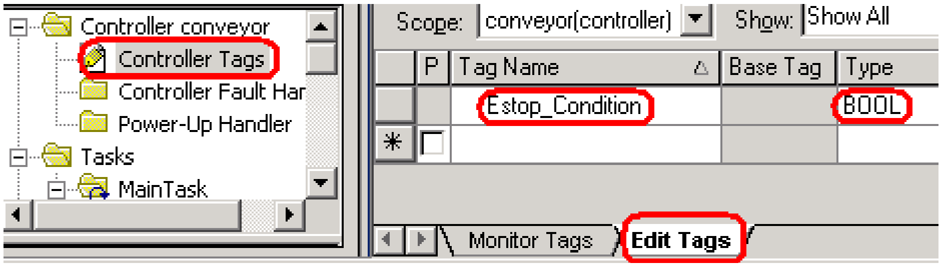
Look at the following example:



Each program has it's own tag database, that no other program can access, but if a program writes a value to a Controller Tag, any program can access the tag.

For this example, I've created a new project called conveyor with an input module in slot 5 and an output module in slot 6.

Open the Controller Tag Database, and be sure 'Edit Tags' is selected. Declare the tag called Estop_Condition, with a BOOL data type. The BOOL data type means one bit of information is to be stored. This is similar to a B3 bit in a PLC or SLC. Press enter.



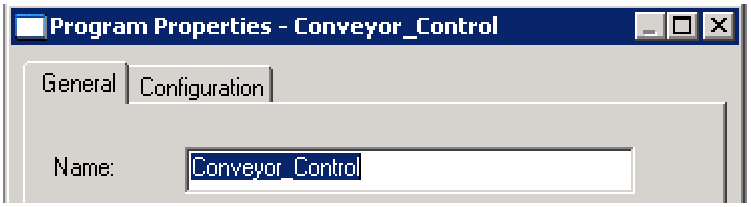
Scope: conveyor(controller) Show: Show All

P	Tag Name	Base Tag	Type
	Estop_Condition		BOOL
□			

Program Tags

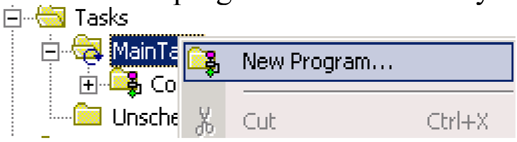
Program tags are local to the program they reside in. One main advantage of program tags is that we can create one program and copy it multiple times. The exact same program tags can be used in each instance of the program. This makes the process of building logic for similar pieces of equipment very simple.

1) Right click on the main program and choose 'Properties' Rename MainProgram to Conveyor_Control.

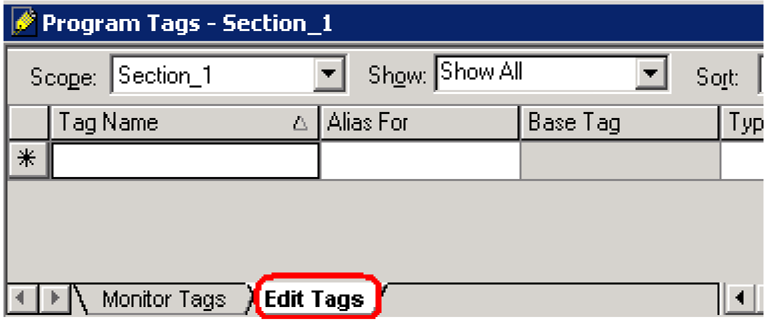

Program Properties - Conveyor_Control

Name: Conveyor_Control

2) Right click on the MainTask to add a new program. The name of your program is Section_1.



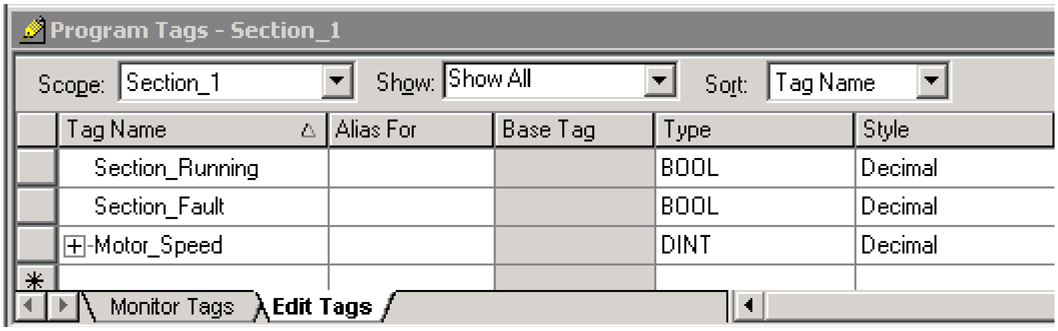
3) Expand Section_1 and open the Program Tags. Notice there are no tags in the program tag database. At the bottom of the tag editor screen, be sure 'Edit Tags' is selected.


Program Tags - Section_1

Scope: Section_1 Show: Show All Sort:

Tag Name	Alias For	Base Tag	Typ

4) Set up three program tags as shown. Pay attention to the Data Type. The data type of the tag specifies the way the data is structured within the tag. A BOOL data type will store 1 bit of information, and a DINT (Double Integer) data type will store 32 bits of information (This is usually used for numbers)

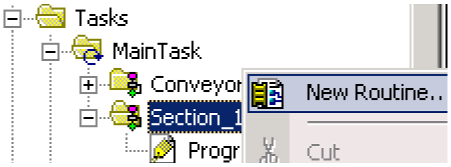


Program Tags - Section_1

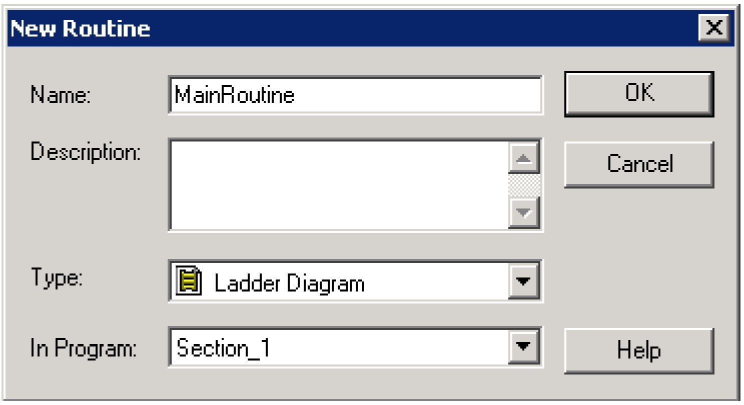
Scope: Section_1 Show: Show All Sort: Tag Name

Tag Name	Alias For	Base Tag	Type	Style
Section_Running			BOOL	Decimal
Section_Fault			BOOL	Decimal
⊞-Motor_Speed			DINT	Decimal

5) Now, let's add some logic: Right click the Section_1 program and add a new routine.



7) The name will be MainRoutine, then press OK.



New Routine

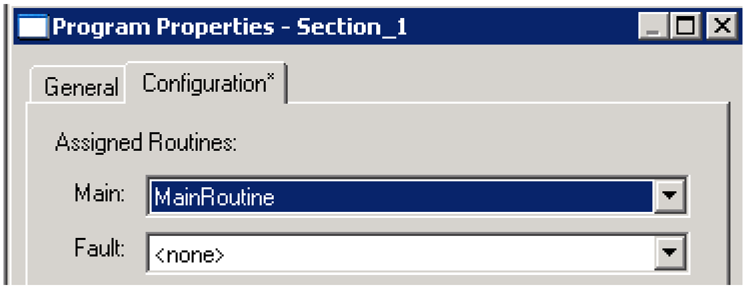
Name: MainRoutine

Description:

Type: Ladder Diagram

In Program: Section_1

8) We named this the MainRoutine, but did not configure it as the MainRoutine yet. Every program needs to have a main routine. That is the routine the processor scans within the program. You will notice in the controller organizer window, there is no 1 on the ladder icon yet. Right click the Section_1 program and go to properties. Click the Configuration tab. In the Pull down menu next to Main, choose the MainRoutine. Apply your changes, then press OK.

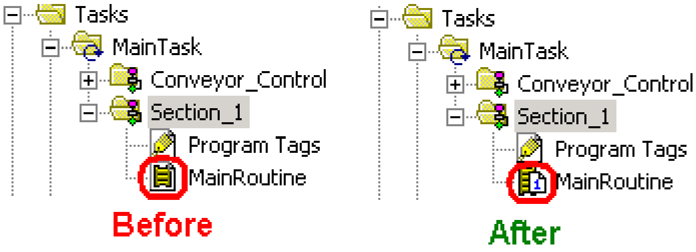


Assigned Routines:

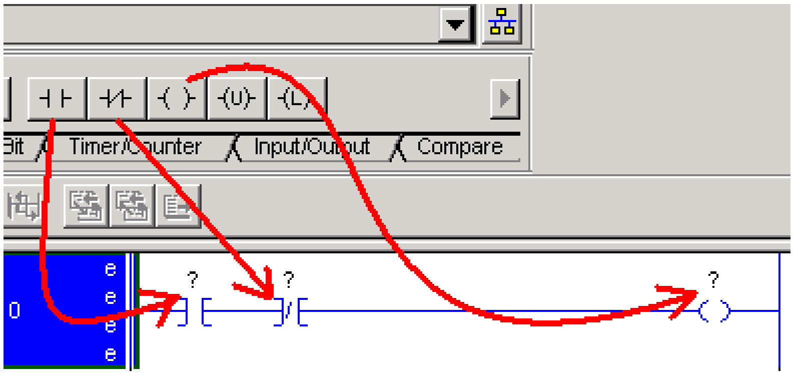
Main: MainRoutine

Fault: <none>

9) Notice the MainRoutine is now has a 1 on it...

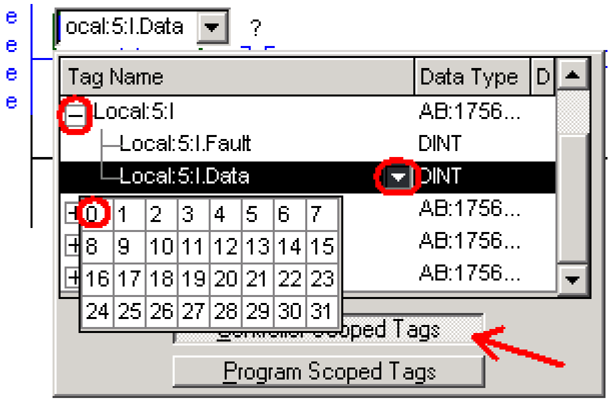


10)Open the MainRoutine, and add the following logic using the drag and drop method:



11)The question marks indicate that we need an address on each instruction. Double click the question mark on the XIC,and click the pull down tab.

12)Be sure Controller Tags is selected. Expand Local:5:I, and click the pull down tab next to data to reveal all 32 bits. Choose bit 0.

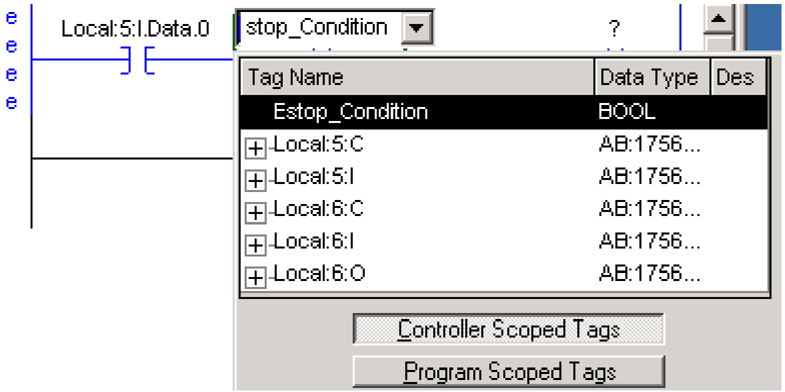


ocal:5:I.Data ?

Tag Name	Data Type	D
Local:5:I	AB:1756...	
Local:5:I.Fault	DINT	
Local:5:I.Data	DINT	
	AB:1756...	
	AB:1756...	
	AB:1756...	

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

13)For the XIO, choose the ESTOP Tag.

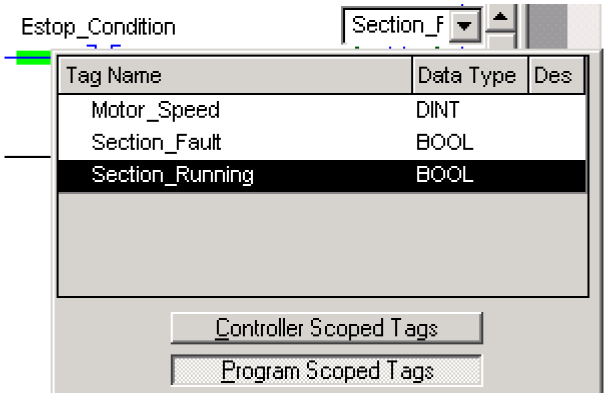


Local:5:I.Data.0

stop_Condition ?

Tag Name	Data Type	Des
Estop_Condition	BOOL	
Local:5:C	AB:1756...	
Local:5:I	AB:1756...	
Local:6:C	AB:1756...	
Local:6:I	AB:1756...	
Local:6:O	AB:1756...	

14)For the OTE, Choose 'Section Running' as a Program scoped tag.



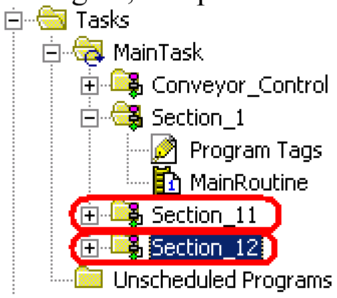
Estop_Condition

Section_F

Tag Name	Data Type	Des
Motor_Speed	DINT	
Section_Fault	BOOL	
Section_Running	BOOL	

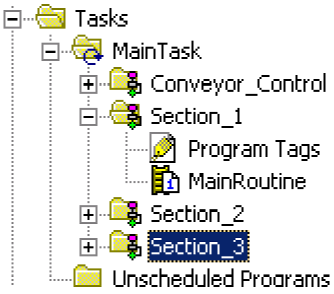
15.Now that we have one complete program (Or as complete as we need it for example... Let's copy the program and use it for other sections of the conveyor.

16)Right click the Section_1 program and choose 'Copy'. Right click on the 'MainTask' and choose paste. Then right click the 'MainTask' again, and paste a second time.



- Tasks
  - MainTask
    - Conveyor_Control
    - Section_1
      - Program Tags
      - MainRoutine
    - Section_11
    - Section_12
  - Unscheduled Programs

17)This gave us a total of 3 copies of the same program. Right click on Section_11, go to the properties, and rename it to Section_2. Right click Section_2, go to properties, and rename to Section_3. When finished, your project will look like this:



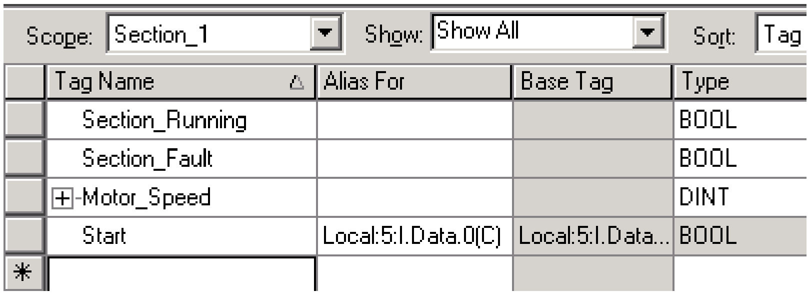
- Tasks
  - MainTask
    - Conveyor_Control
    - Section_1
      - Program Tags
      - MainRoutine
    - Section_2
    - Section_3
  - Unscheduled Programs

18)Now all we have to do is go back and change the I/O in each program to reflect the actual switch that turns on the conveyor for that section, and if we had it set up... The output the logic writes to. All the internal bits are already written for us, and the program structure is in place.

Aliasing

This process can be taken one step further. The program can be set up so nothing has to be changed in logic have to be changed after it has been copied. Aliasing allows this to be done. An Alias is a tag that is a shortcut to another tag. We can create program tags in the program tag database that point to the real world I/O. After a program has been copied, you just need to go into the program tag database, and change the address the aliases point to. Look at this example.

1) Open the program tags of Section_1. Be sure 'Edit Tags' is selected and create a tag called 'Start'. In the Alias for column, make this tag alias for Local:5:I.Data.0.

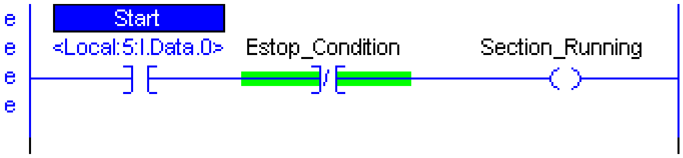


Scope: Section_1   Show: Show All   Sort: Tag

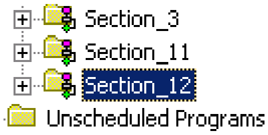
	Tag Name	Alias For	Base Tag	Type
	Section_Running			BOOL
	Section_Fault			BOOL
	⊞-Motor_Speed			DINT
	Start	Local:5:I.Data.0(C)	Local:5:I.Data...	BOOL
*				

Notice the C next to the Input module's address. This means that you are pointing to a controller tag.

2) Now go back to the MainRoutine, and change the address on the first instruction to the alias name. Notice by default the actual memory location the alias is pointing to appears below the alias name.

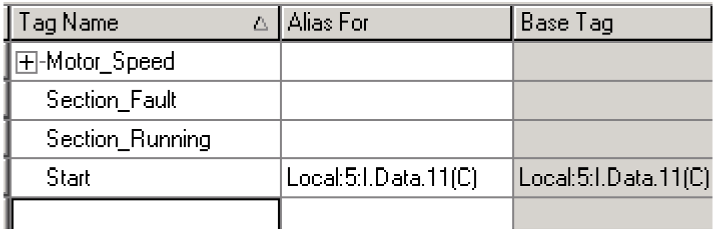


3) Now copy section on and paste it twice again into the main task. That will give us sections 11 and 12.



⊞ Section_3
⊞ Section_11
⊞ Section_12
Unscheduled Programs

4) Go into section 11, and change the Start alias to point to bit 11 of the input module.



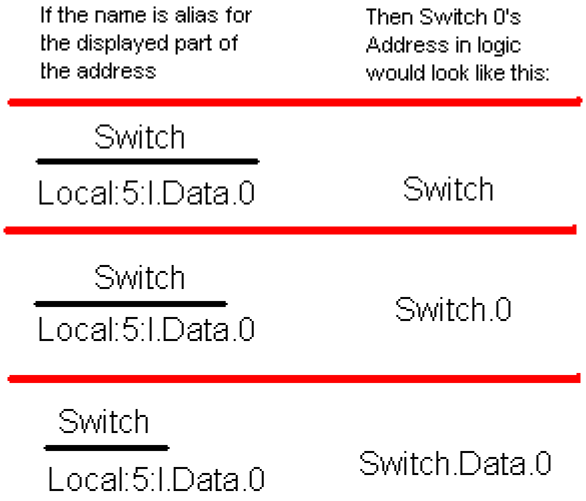
Tag Name	Alias For	Base Tag
⊞-Motor_Speed		
Section_Fault		
Section_Running		
Start	Local:5:I.Data.11(C)	Local:5:I.Data.11(C)

5) Do the same for section 12. Change the start tag to look at bit 12 of the input module.

6) When bit 11 goes high on the input module, Conveyor 11's start bit will be energized. When Bit 12 goes high on the input module, Conveyor 12's start bit will energize! You can see how aliasing would allow you to quickly develop programs that are very similar.

A Controller tag can be an alias for another controller tag. A program tag can alias another program tag. A program tag can alias a controller tag, but a controller tag CANNOT alias a program tag.

You can alias to several tag levels. The start tag you just created pointed to a bit level tag. But if it would have pointed to the data word instead, we would have to specify the bit number manually in logic. Look at the chart below:



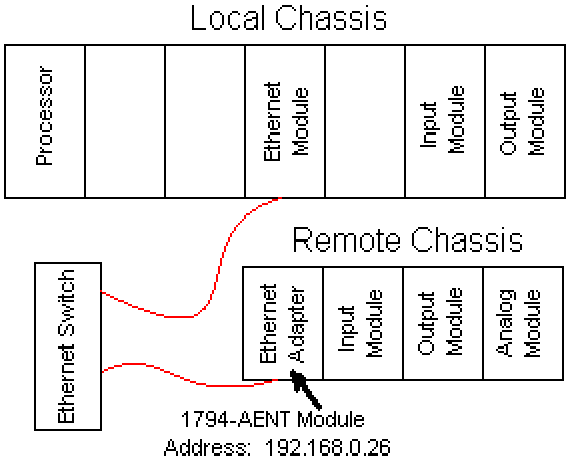
If the name is alias for the displayed part of the address	Then Switch 0's Address in logic would look like this:
Switch Local:5:I.Data.0	Switch
Switch Local:5:I.Data.0	Switch.0
Switch Local:5:I.Data.0	Switch.Data.0

Remote Chassis

In many systems, many points of I/O are located far away from the **local chassis**. The **local chassis** is the chassis where the processor in focus resides. In many cases, it is easier to mount a chassis at the remote location. A communication cable will allow the processor to control the chassis at the remote location. For example, if 256 points of I/O were 300 feet from the local chassis, it would be easier to mount a chassis at the remote location. You would then run the 256 points of I/O just a few feet to the remote chassis, then run one communication cable back to the local chassis.

The example below shows the Ethernet/IP communication protocol, however, many other protocols follow the same model such as ControlNet or Remote I/O with slightly different wiring and configuration changes.

Look at the diagram below. In this example, we have an input and output module in the local chassis. This would be for I/O in the local vicinity. For I/O in another location, we can use a remote chassis. The processor will establish a connection to this remote I/O, and will read inputs, and control outputs on this chassis. In this example, the local chassis is ControlLogix, and the remote chassis is FLEX I/O (This could also be many other types of chassis such as another ControlLogix chassis, PLC-5 Chassis, SLC Chassis, Block I/O, etc....). This procedure will assume that you have an existing program, and that IP addresses have already been assigned to the Ethernet module, and Ethernet Adapter. If you were to use ControlNet, you would assign node numbers instead of IP addresses. You can assign node numbers to these modules by physically dialing in a node number on the modules themselves. For Remote I/O, you would set up the DIP switches according to the user manuals for each module.



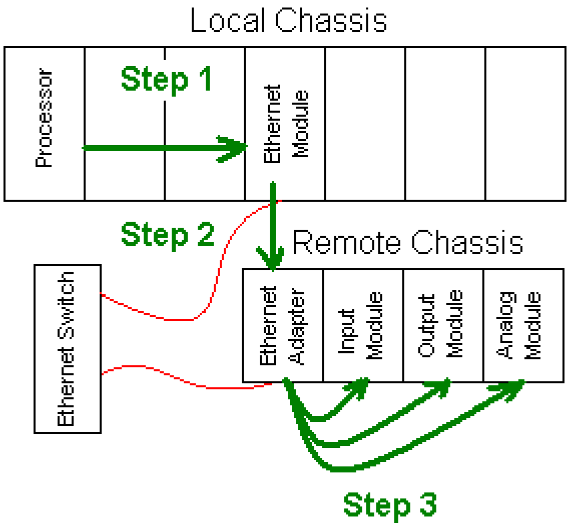
Communication path

Look at the diagram below showing the communication path. The processor is where the program resides, so the path we choose will be relative to the processor itself.

1) The processor must first connect to the 1756-ENBT module in the local chassis. If you are using ControlNet, this would be a 1756-CNB(R) module. For the older remote I/O Protocol, this would be a 1756-DHRIO module.

2) Next, we must tell the local 1756-ENBT module to connect to the adapter at the remote chassis. Remember we are using Flex I/O for this particular example, so we would connect to the 1794-AENT module (Ethernet) 1794-ACN(R)(ControlNet), or 1794-ASB (Remote I/O).

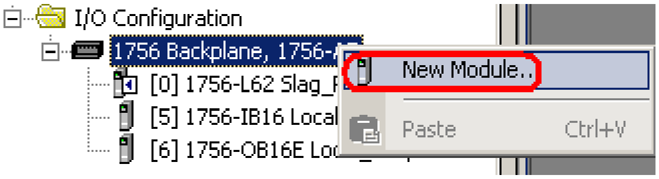
3) The next step is to have the adapter connect to the individual modules within it's chassis. We are using the following modules:
   1. Slot 0 – 1794-IB16 (DC input module)
   2. Slot 1 – 1794-OB16 (DC output module)
   3. Slot 2 – 1794-IE8 (Analog input module)



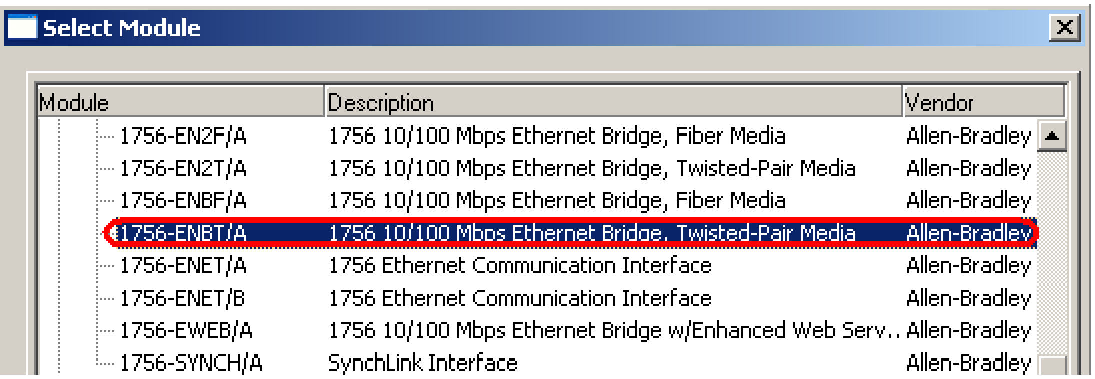
I/O Configuration

Now that we have decided the layout for our system, we need to go to the ControlLogix project, and set up the remote chassis under I/O Configuration. Recall that 3 steps will accomplish this connection: First, we connect to the Local ENBT module, then the ENBT will connect to the AENT. Next, the AENT will connect to the modules that are in it's own chassis.

1) If you do not already have the Ethernet module set up in your I/O Configuration, right click the backplane of the I/O Configuration folder, and select 'New Module'.

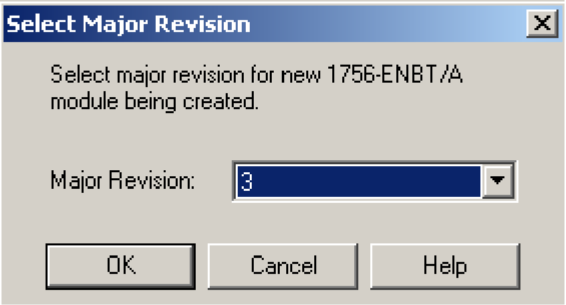


2) In the Communication category, select the 1756-ENBT module.



Module	Description	Vendor
1756-EN2F/A	1756 10/100 Mbps Ethernet Bridge, Fiber Media	Allen-Bradley
1756-EN2T/A	1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media	Allen-Bradley
1756-ENBF/A	1756 10/100 Mbps Ethernet Bridge, Fiber Media	Allen-Bradley
1756-ENBT/A	1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media	Allen-Bradley
1756-ENET/A	1756 Ethernet Communication Interface	Allen-Bradley
1756-ENET/B	1756 Ethernet Communication Interface	Allen-Bradley
1756-EWEB/A	1756 10/100 Mbps Ethernet Bridge w/Enhanced Web Serv...	Allen-Bradley
1756-SYNCH/A	SynchLink Interface	Allen-Bradley

3) Next, you will choose the major revision of the 1756-ENBT module. This revision is usually on the label on the side of the module, however, this label may not be up to date. You can use the module information from RSLinx, or type the IP address into your web browser, and click 'Browse Chassis' to determine the revision level. At the time this manual was written, the modules we use for class are 3.6. This means the Major revision is 3, and the minor revision is 6. Therefore, we must select 3 as the major revision. Press OK when finished.

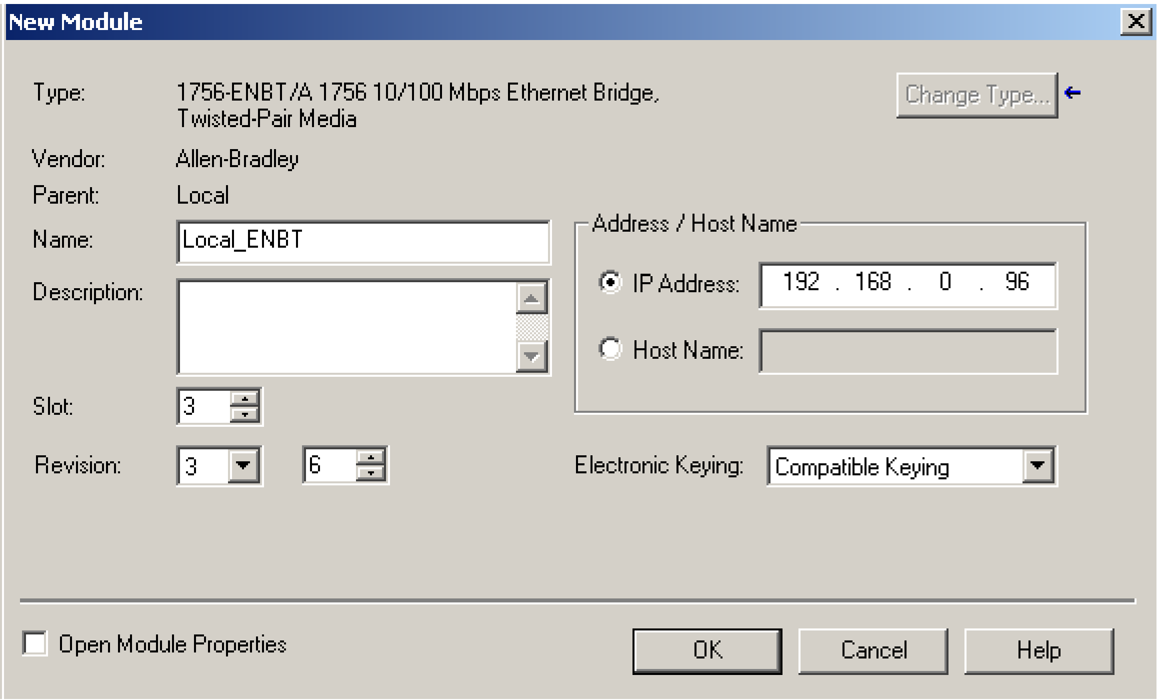


Select Major Revision

Select major revision for new 1756-ENBT/A module being created.

Major Revision: 3

4) Next, complete the dialog box for the Ethernet module. Your location may have specific naming standards, but we will name this module '**Local_Ethernet_Module**'. The **IP address** scrolls across the front of the ENBT module (assuming an IP address has been assigned). In our classroom, the ENBT module is in slot **3**, and the minor revision is 4 (Recall the module's revision was 2.4, 2 being the major, and 4 being the minor). We will leave the keying as **compatible module**. Press FINISH.



New Module

Type: 1756-ENBT/A 1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media

Vendor: Allen-Bradley

Parent: Local

Name: Local_ENBT

Description:

Slot: 3

Revision: 3 6

Address / Host Name

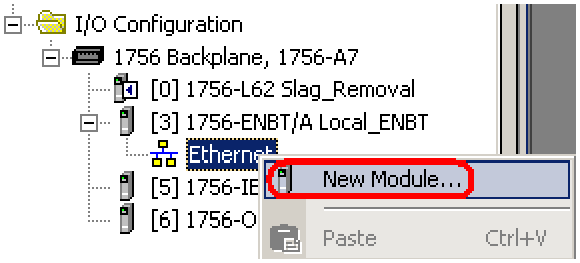
● IP Address: 192 . 168 . 0 . 96

○ Host Name:

Electronic Keying: Compatible Keying

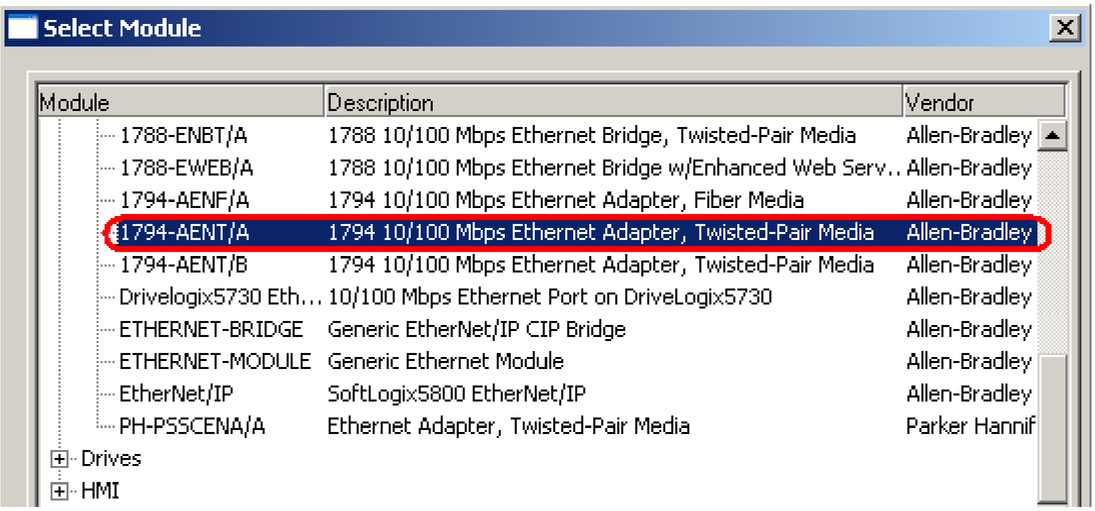
□ Open Module Properties

5) Our next step is to have the 1756-ENBT connect to the 1794-AENT module over Ethernet. In I/O Configuration, right click the Ethernet network coming from the ENBT module. Remember we are connecting to the AENT module from the Ethernet (Not directly from the backplane)



I/O Configuration
1756 Backplane, 1756-A7
[0] 1756-L62 Slag_Removal
[3] 1756-ENBT/A Local_ENBT
Ethernet
[5] 1756-IE
[6] 1756-O

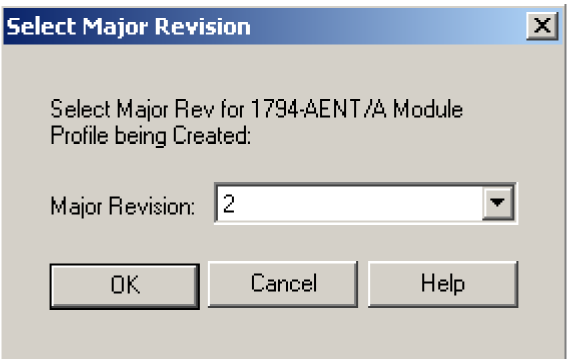
6) Choose the 1794-AENT/A (Series A)from the list of available devices in the Communications category. Press OK. (Be sure you select the **1794**-AENT/A)



Select Module

Module	Description	Vendor
1788-ENBT/A	1788 10/100 Mbps Ethernet Bridge, Twisted-Pair Media	Allen-Bradley
1788-EWEB/A	1788 10/100 Mbps Ethernet Bridge w/Enhanced Web Serv...	Allen-Bradley
1794-AENF/A	1794 10/100 Mbps Ethernet Adapter, Fiber Media	Allen-Bradley
1794-AENT/A	1794 10/100 Mbps Ethernet Adapter, Twisted-Pair Media	Allen-Bradley
1794-AENT/B	1794 10/100 Mbps Ethernet Adapter, Twisted-Pair Media	Allen-Bradley
Drivelogix5730 Eth...	10/100 Mbps Ethernet Port on DriveLogix5730	Allen-Bradley
ETHERNET-BRIDGE	Generic EtherNet/IP CIP Bridge	Allen-Bradley
ETHERNET-MODULE	Generic Ethernet Module	Allen-Bradley
EtherNet/IP	SoftLogix5800 EtherNet/IP	Allen-Bradley
PH-PSSCENA/A	Ethernet Adapter, Twisted-Pair Media	Parker Hannif
⊞ Drives		
⊞ HMI		

7) Next, select the major revision of the AENT module. We can get this information from the web browser, by typing the IP address into the browser's address bar, and click 'Module Configuration'. At the time this manual was written, the AENT module had firmware version 2.12. Therefore, we must enter 2 as the major revision.

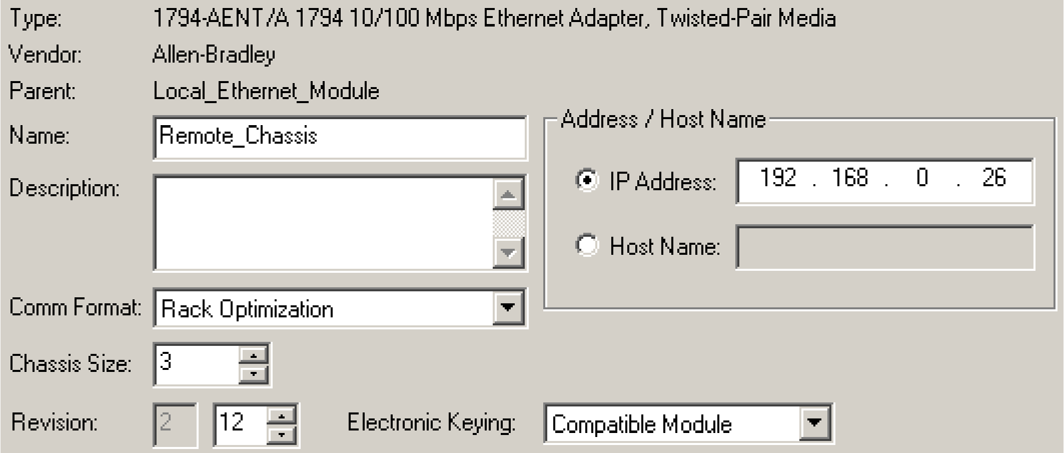


Select Major Revision

Select Major Rev for 1794-AENT/A Module Profile being Created:

Major Revision: 2

8) You may have naming conventions for remote chassis at your own location. For our classroom use, we will name the module '**Remote_Chassis**'. You can usually get the **IP address** of this module from your network administrator, schematics, or other documentation if the address was not written on the front of the module. Use the IP address the instructor assigns to you. We will leave the comm format as '**Rack Optimization**', and this will treat the three modules in the chassis as a single connection instead of having a separate connection for each module. The 1756-ENBT module only supports 64 connections in many cases. This chassis consists of **3** slots (not counting the adapter). The minor revision is **12** (because our version was 2.12), and leave the keying as '**Compatible Module**'. Press FINISH when you are finished configuring the module.



Type: 1794-AENT/A 1794 10/100 Mbps Ethernet Adapter, Twisted-Pair Media

Vendor: Allen-Bradley

Parent: Local_Ethernet_Module

Name: Remote_Chassis

Description:

Comm Format: Rack Optimization

Chassis Size: 3

Revision: 2 12

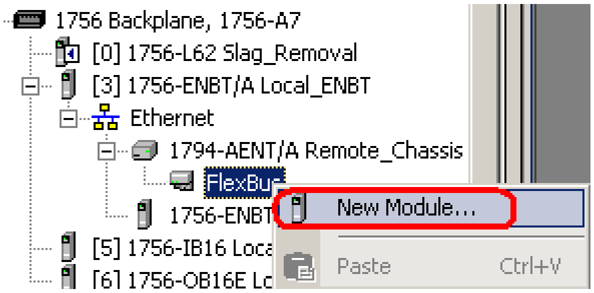
Electronic Keying: Compatible Module

Address / Host Name

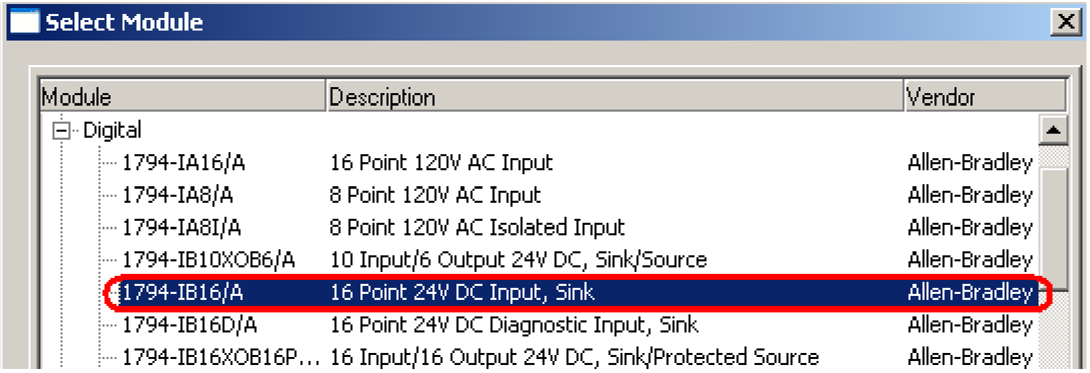
● IP Address: 192 . 168 . 0 . 26

○ Host Name:

9) Now that a connection has been made between the ENBT module and the AENT module, we need to establish a connection between the AENT module and the three modules that are in it's chassis. To add the first module (The 1794-IB16), right click the flexbus of the remote flex chassis in the I/O Configuration, and select 'New Module'.

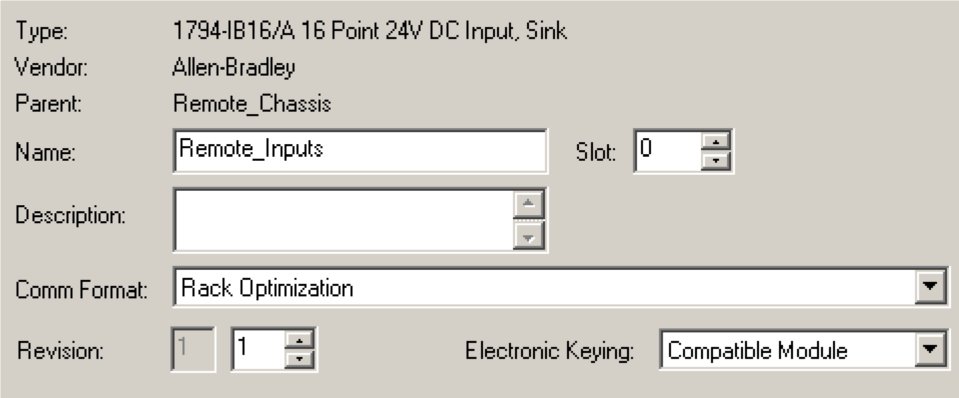


10) Choose the 1794-IB16 module from the list of digital modules, then press OK.

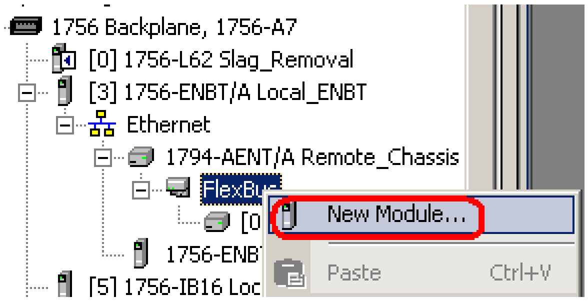

Select Module

Module	Description	Vendor
Digital		
1794-IA16/A	16 Point 120V AC Input	Allen-Bradley
1794-IA8/A	8 Point 120V AC Input	Allen-Bradley
1794-IA8I/A	8 Point 120V AC Isolated Input	Allen-Bradley
1794-IB10XOB6/A	10 Input/6 Output 24V DC, Sink/Source	Allen-Bradley
1794-IB16/A	16 Point 24V DC Input, Sink	Allen-Bradley
1794-IB16D/A	16 Point 24V DC Diagnostic Input, Sink	Allen-Bradley
1794-IB16XOB16P...	16 Input/16 Output 24V DC, Sink/Protected Source	Allen-Bradley

11) Complete the Module configuration dialog as follows, then press FINISH.

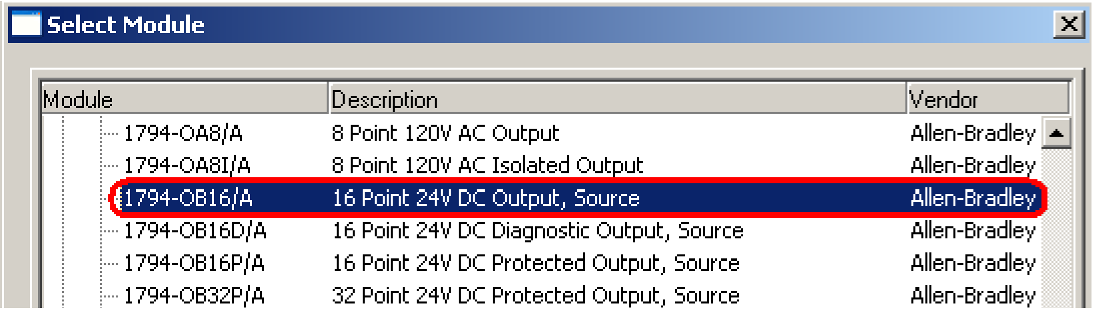

Type: 1794-IB16/A 16 Point 24V DC Input, Sink
Vendor: Allen-Bradley
Parent: Remote_Chassis
Name: Remote_Inputs
Slot: 0
Description:
Comm Format: Rack Optimization
Revision: 1 1
Electronic Keying: Compatible Module

12) Next, we will add the 1794-OB16 module to the I/O Configuration tree. Again, this module resides on the flexbus, so right click the flexbus, and select 'New Module'



```
1756 Backplane, 1756-A7
  [0] 1756-L62 Slag_Removal
  [3] 1756-ENBT/A Local_ENBT
    Ethernet
      1794-AENT/A Remote_Chassis
        FlexBus
          [0
      1756-ENB
  [5] 1756-IB16 Loc
```

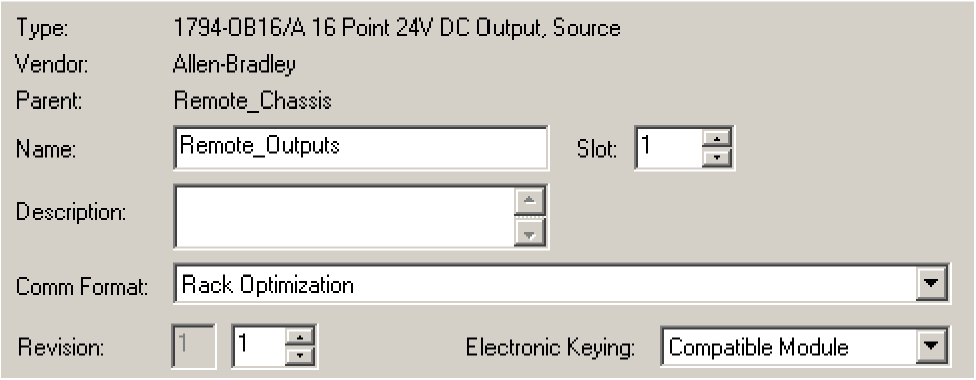
13) From the list of digital modules, select the 1794-OB16 DC output module. Press OK.



Select Module

Module	Description	Vendor
1794-OA8/A	8 Point 120V AC Output	Allen-Bradley
1794-OA8I/A	8 Point 120V AC Isolated Output	Allen-Bradley
1794-OB16/A	16 Point 24V DC Output, Source	Allen-Bradley
1794-OB16D/A	16 Point 24V DC Diagnostic Output, Source	Allen-Bradley
1794-OB16P/A	16 Point 24V DC Protected Output, Source	Allen-Bradley
1794-OB32P/A	32 Point 24V DC Protected Output, Source	Allen-Bradley

14) Complete the module configuration dialog as shown, then press FINISH.



Type: 1794-OB16/A 16 Point 24V DC Output, Source

Vendor: Allen-Bradley

Parent: Remote_Chassis

Name: Remote_Outputs

Slot: 1

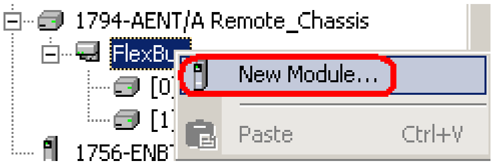
Description:

Comm Format: Rack Optimization

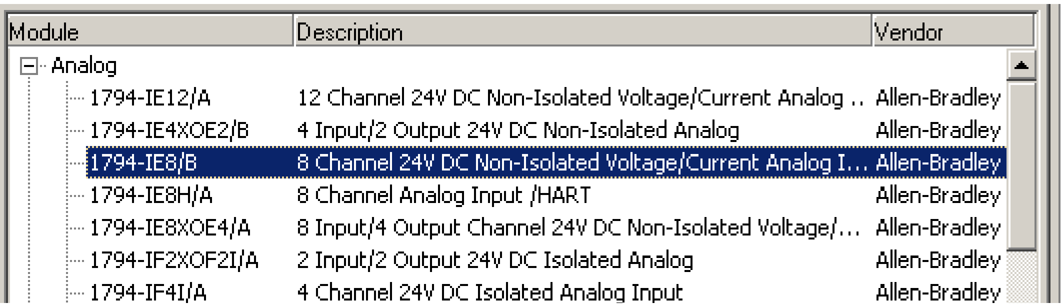
Revision: 1 1

Electronic Keying: Compatible Module

15) Next we need to add the last module to the remote chassis. This will be the 1794-IE8. This module resides on the flexbus of the flex chassis. Right click the flexbus, and select 'New Module'.

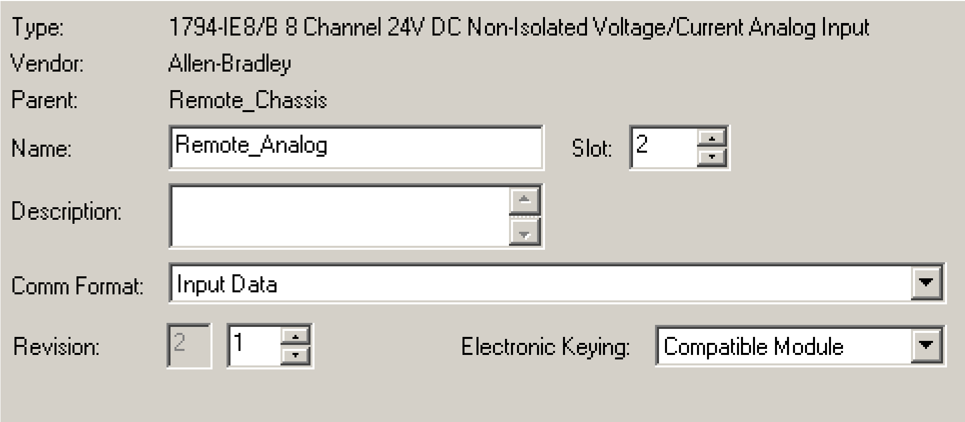


16) From the list of analog modules, choose the 1794-IE8 module, then press OK.



Module	Description	Vendor
Analog		
1794-IE12/A	12 Channel 24V DC Non-Isolated Voltage/Current Analog ..	Allen-Bradley
1794-IE4XOE2/B	4 Input/2 Output 24V DC Non-Isolated Analog	Allen-Bradley
1794-IE8/B	8 Channel 24V DC Non-Isolated Voltage/Current Analog I...	Allen-Bradley
1794-IE8H/A	8 Channel Analog Input /HART	Allen-Bradley
1794-IE8XOE4/A	8 Input/4 Output Channel 24V DC Non-Isolated Voltage/...	Allen-Bradley
1794-IF2XOF2I/A	2 Input/2 Output 24V DC Isolated Analog	Allen-Bradley
1794-IF4I/A	4 Channel 24V DC Isolated Analog Input	Allen-Bradley

17) complete the module configuration dialog as shown, then press FINISH.



Type: 1794-IE8/B 8 Channel 24V DC Non-Isolated Voltage/Current Analog Input

Vendor: Allen-Bradley

Parent: Remote_Chassis

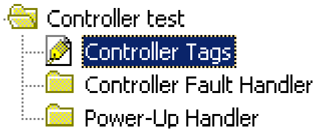
Name: Remote_Analog Slot: 2

Description:

Comm Format: Input Data

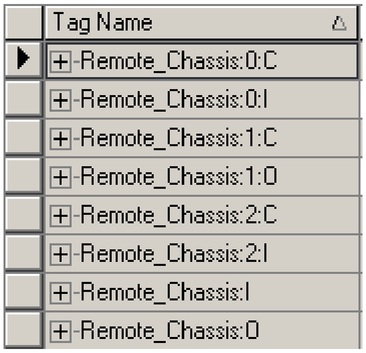
Revision: 2 1 Electronic Keying: Compatible Module

18) Now that all of our modules have been added to the I/O Configuration, RSLogix has created tags for us in the controller tag database. Let's take a look at the controller tags to see where data from these three modules will appear.



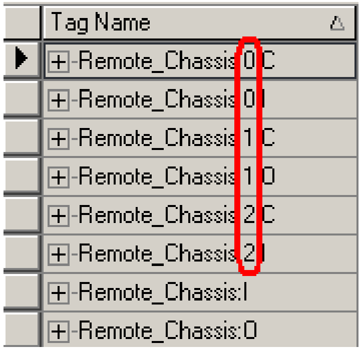
- Controller test
  - Controller Tags
  - Controller Fault Handler
  - Power-Up Handler

19) Take a look at the tag names that RSLogix generated for us.



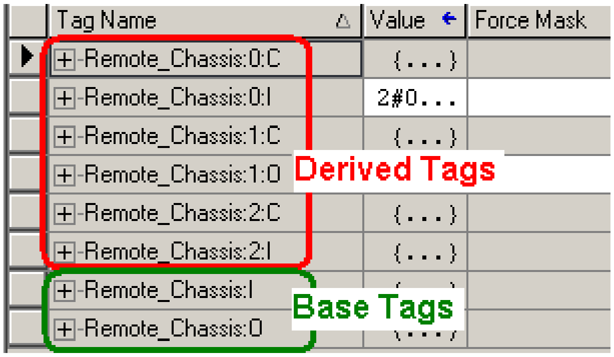
Tag Name
Remote_Chassis:0:C
Remote_Chassis:0:I
Remote_Chassis:1:C
Remote_Chassis:1:O
Remote_Chassis:2:C
Remote_Chassis:2:I
Remote_Chassis:I
Remote_Chassis:O

20) Notice the tag names assume the name of the adapter in the remote chassis. You will also notice for this example, that we have two tags for each slot. The slot number immediately follows the tag name.



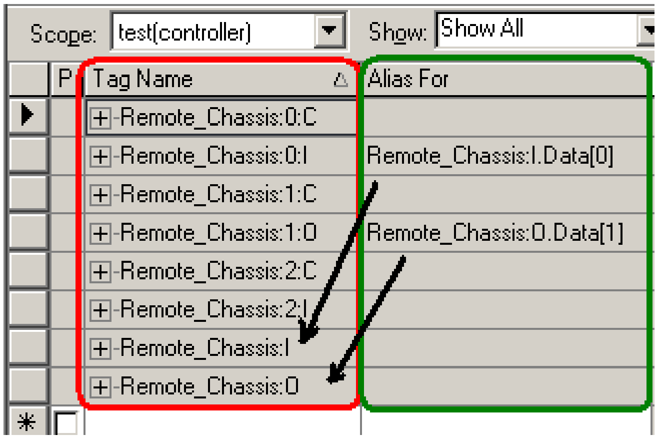
Tag Name
Remote_Chassis:0:C
Remote_Chassis:0:I
Remote_Chassis:1:C
Remote_Chassis:1:O
Remote_Chassis:2:C
Remote_Chassis:2:I
Remote_Chassis:I
Remote_Chassis:O

21) Since we established a connection to each module in the chassis individually, we have two sets of tags for this remote I/O. We have the base tags, which contain most all of the information we need to know about the chassis, and we have the derived tags which follow the same naming convention as the local I/O. The derived tags alias corresponding memory locations in the base tag that reflect the data for it's own slot when possible.



Tag Name	Value	Force Mask
⊞-Remote_Chassis:0:C	{...}	
⊞-Remote_Chassis:0:I	2#0...	
⊞-Remote_Chassis:1:C	{...}	
⊞-Remote_Chassis:1:O		
⊞-Remote_Chassis:2:C	{...}	
⊞-Remote_Chassis:2:I	{...}	
⊞-Remote_Chassis:I		
⊞-Remote_Chassis:O		

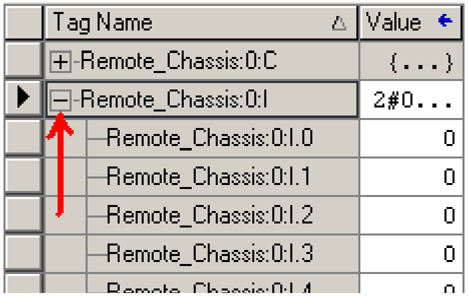
22) If you go to edit tags, you will see what based tags the derived tags are aliasing.



Scope: test(controller) Show: Show All

P	Tag Name	Alias For
	⊞-Remote_Chassis:0:C	
	⊞-Remote_Chassis:0:I	Remote_Chassis:I.Data[0]
	⊞-Remote_Chassis:1:C	
	⊞-Remote_Chassis:1:O	Remote_Chassis:O.Data[1]
	⊞-Remote_Chassis:2:C	
	⊞-Remote_Chassis:2:I	
	⊞-Remote_Chassis:I	
	⊞-Remote_Chassis:O	

23) For this course we will be using the derived tags. To see the data from the DC Input Module, you must be on line, on the monitor tags tab, then expand Remote_Chassis:0:I. You will see each input from this module. These tags can be used directly in ladder logic, or you can create another alias to use for your project.



Tag Name △	Value ←
⊞-Remote_Chassis:0:C	{...}
⊟-Remote_Chassis:0:I	2#0...
Remote_Chassis:0:I.0	0
Remote_Chassis:0:I.1	0
Remote_Chassis:0:I.2	0
Remote_Chassis:0:I.3	0
Remote_Chassis:0:I.4	0

Basic Instructions

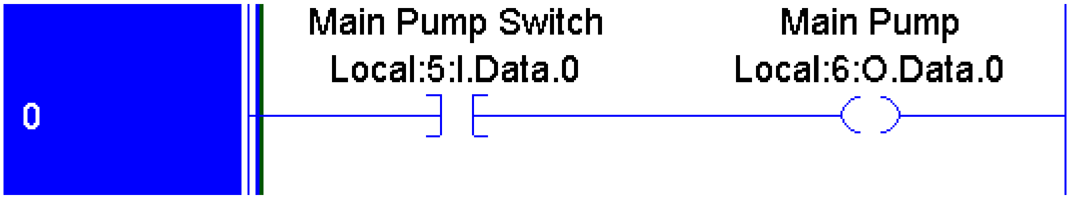
A Little History

A common programming language used in PLC's is called Ladder Logic. Ladder Logic was developed years ago to help electricians adapt to PLC's. Ladder logic is still widely in use today although this language appears to be weakening. Ladder logic is similar to Assembly Language in many ways which was widely used to program computers years ago. Since then, higher level languages such as PASCAL have come along. In the last few years we have seen a more Object Oriented approach to programming in languages such as Java. The ControlLogix processor seems to be following the Object Oriented approach with it's User Defined data types (UDTs), and event driven tasks.

Here are some of the instructions available in Ladder Logic:

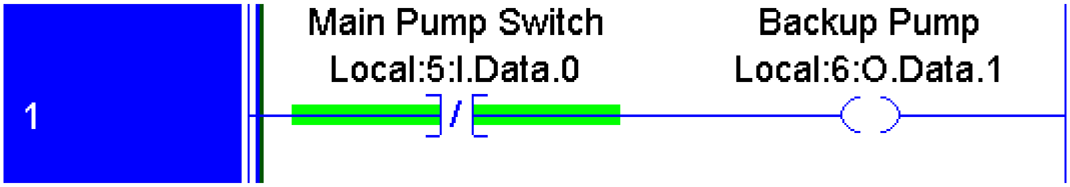
Examine If Closed (XIC)

You will find that most instructions in the SLC, PLC-5, and ControlLogix consist of three character pneumonics. The XIC looks at a given bit of memory in the processor. If this bit is on, then the XIC will intensify indicating logical continuity through the instruction. Here is what the XIC looks like in logic.



Examine If Open (XIO)

The XIO is just the opposite of the XIC instruction. The XIO looks at a bit in memory. If the bit is a 0, then the XIO is true. It will intensify indicating logical continuity through the instruction, and the next instruction in the rung will be examined. This is usually referred to as a NOT instruction because the address the instruction points to must NOT be on for the instruction to be true. Here is an example of how the XIO will appear in ladder logic:

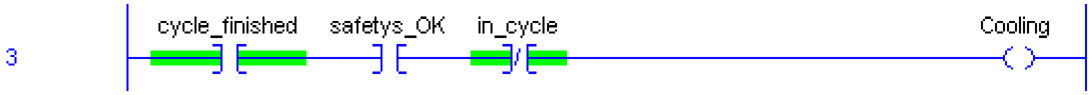


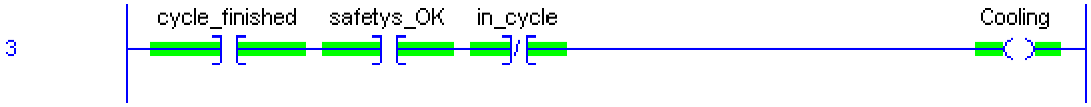
In the above example, you can see that as soon as the Main Pump Switch is shut off, a bit is set to run the backup pump.

Output To Energize (OTE)

The output to energize simply turns a bit on when it is evaluated as true, and shuts a bit off when the instruction is evaluated as false. Using the same address on an OTE in two different places in the program is considered bad programming practice. The two OTE's can interfere with each other, and makes troubleshooting difficult.

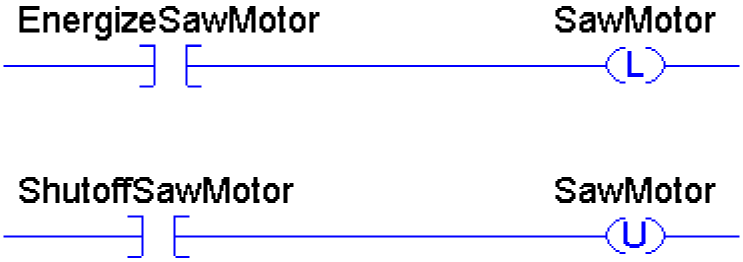
Below you will find two different states of the same rung. The first state shows the rung as false, so a zero is written to B3:1/0. The second state is true, and a 1 will be written to B3:1/0.



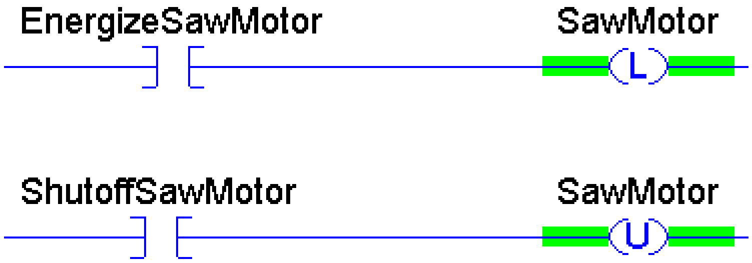


Output To Latch (OTL) and Output To Unlatch (OTU)

The Output To Latch instruction will write a 1 to it's address when true. When the OTL goes false again, the output address will remain a 1 until another instruction such as the Output to Unlatch shuts it back off. ***This is true even if the processor powers down, and is brought back up!!*** You must use caution when using the Latch/Unlatch when controlling real world devices. Here is what the Latch/Unlatch will look like in logic:



If the output address is off, both the latch and unlatch instructions are not intensified, but once the bit is turned on, you will see both the latch and unlatch intensified even though both inputs are shut off.

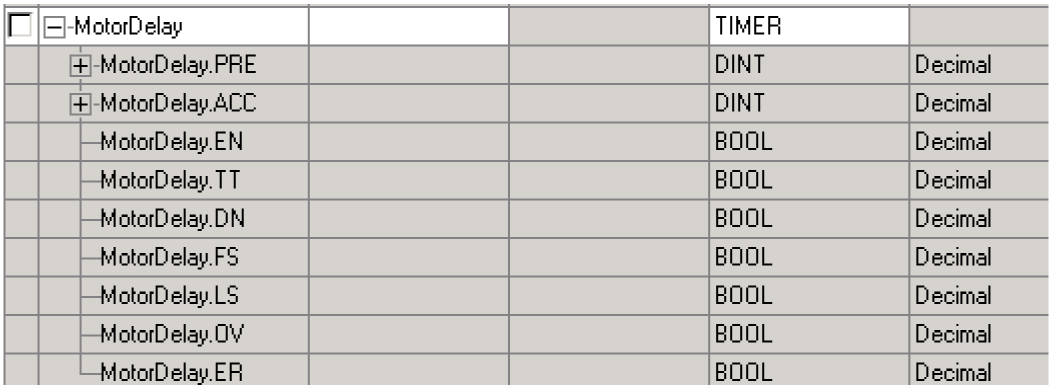


Due to the processor scan cycle, since the unlatch is placed after the latch, if both inputs were to go true, the Unlatch instruction would win, and the output address will be shut off. If the latch was after the unlatch, then the latch would be the last instruction scanned, and therefore the bit would be left in the energized state.

Timers

Timers are generally used for delaying an event from taking place, or to delay a device from shutting off either on an on transition or an off transition. There are three types of timers: The Timer ON delay (TON), Timer Off delay (TOF), and the Retentative Timer On delay (RTO).

Timers can be created as Controller Tags or Program Tags. A tag of the TIMER data type consists of the following components: Preset word (PRE), Accumulate word (ACC), Done bit (DN), Timer Timing bit (TT), and Enable bit (EN). For Timers, the Enable bit follows the rung condition.



□	⊟-MotorDelay			TIMER	
	⊞-MotorDelay.PRE			DINT	Decimal
	⊞-MotorDelay.ACC			DINT	Decimal
	MotorDelay.EN			BOOL	Decimal
	MotorDelay.TT			BOOL	Decimal
	MotorDelay.DN			BOOL	Decimal
	MotorDelay.FS			BOOL	Decimal
	MotorDelay.LS			BOOL	Decimal
	MotorDelay.OV			BOOL	Decimal
	MotorDelay.ER			BOOL	Decimal

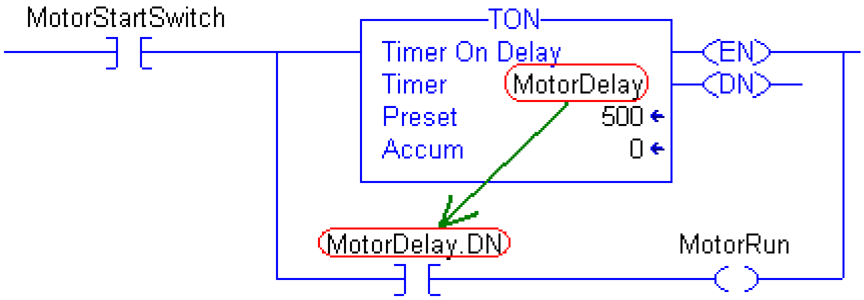
The entire timer is addressed by it's element (example: T4:0) Pieces of the timer can be used in logic however such as the DN bit on an XIC (T4:0/DN), or the Accumulated value in a MOV statement (T4:0.ACC)

Timer On Delay (TON)

The Timer On delay delays an event from taking place. Once the timer becomes true, the enable bit becomes true instantly. The timer will also start timing instantly, so the TT bit becomes high. Since the timer is timing, the accumulated value will increment.

Once the Accumulated value reaches the preset, the done bit (DN) will go high, and the timer will stop timing. The accumulated value remains at (or near) the preset until the rung goes false again. Here is what a typical timer might look like in logic:

When the switch is energized, the timer will begin timing. When the ACC value reaches the PRE value, the DN bit goes high, and the main motor will start. Since the Time Base is .001, therefore 5000 (preset) times .001 (timebase) = 5 second delay.



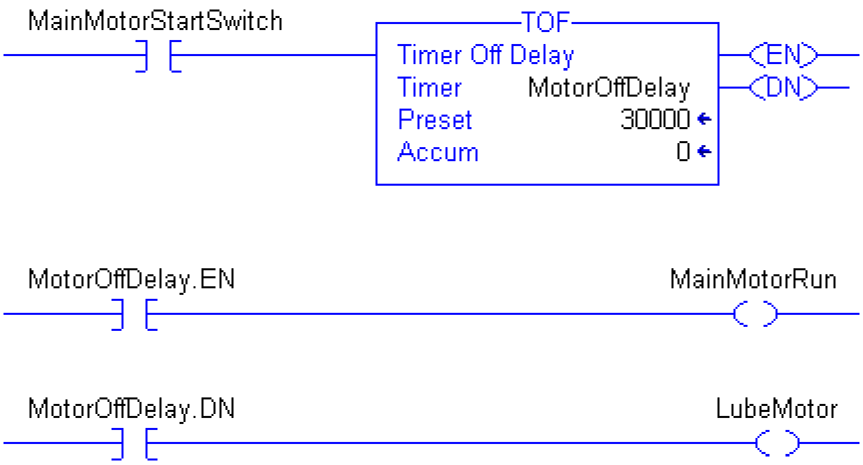
Timer Off Delay (TOF)

The Off Delay Timer is generally used to delay an event from shutting off. Image a lube system on a large motor. As long as the main motor is turning, the lube pump should be running. When the main motor shuts off, you wouldn't want to shut off the lube pump immediately because the main motor needs time to coast down to zero RPM's. The Main motor could run off the EN bit, and the Lube motor could run off the DN bit.

On the Off delay timer, as soon as the rung goes true, The EN bit goes true as it does for all timers. Since the Off delay timer does not delay the DN bit from shutting off, the DN bit goes high immediately. Remember, the TOF instruction delays the DN bit from shutting off, not turning on. (Plus if we are delaying the DN bit from shutting off, it needs to be high to begin with). While the rung is true, the timer is not timing, and the ACC value is at zero.

When the rung is shut off, the EN bit shuts off immediately. The ACC value will start timing until it reaches PRE then the DN bit will shut off.

Here is what the TOF instruction might look like in logic:

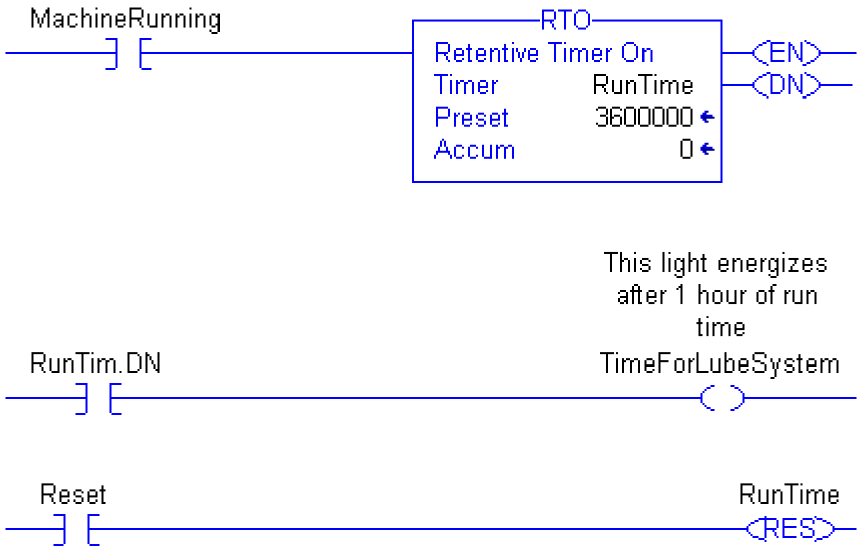


When the motor switch is energized, both the main motor and the lube motor will energize immediately. When the main motor switch is shut off, the main motor shuts off immediately, but since the TOF delays the DN bit from shutting off, the Lube motor will shut off 30 seconds later. Warning: Using the RES instruction on a TOF instruction could cause unpredictable operation.

Retentative On Delay Timer (RTO)

The RTO instruction works a lot like the TON instruction with one main exception: When the rung goes false on the RTO instruction, it will retain the ACC value. When the rung becomes true again, the ACC value will pick up from where it left off. One good application for the RTO would be an hour meter to indicate total runtime for machinery.

Since the RTO does not reset itself when the rung goes false, the RES instruction must be used to reset a timer. Here is a practical application:



In this example, once the machine accumulates 1 hour of run time, a light might come on indicating that a lubrication needs to be engaged. Once the operator lubricates the machine, he can reset the hour meter.

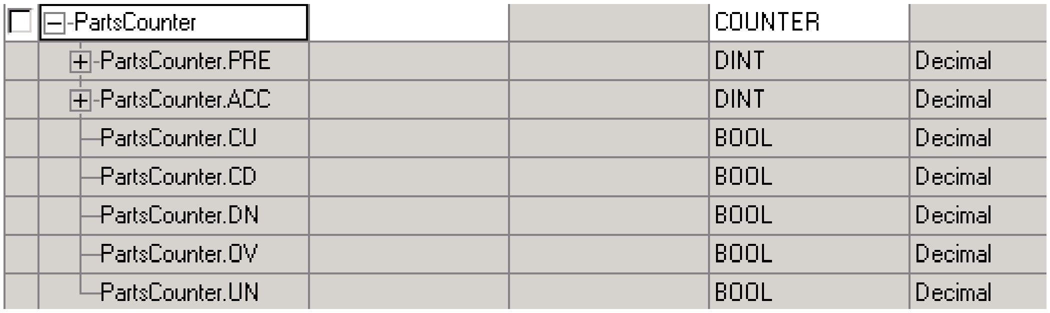
Counters

Counters count rung transitions. The CTU runs the accumulated value of the counter up on the false to true rung transition, and the CTD instruction runs the accumulated value down. The CTU and CTD can be used in conjunction with each other.

Counters consist of the following components:

ACC	Accumulated Value	PRE	Preset Value
CD	Count Down Bit	CU	Count Up bit
OV	Overflow Bit	UN	Underflow bit

Tags used for counters are declared with the COUNTER data type. Here is an example:

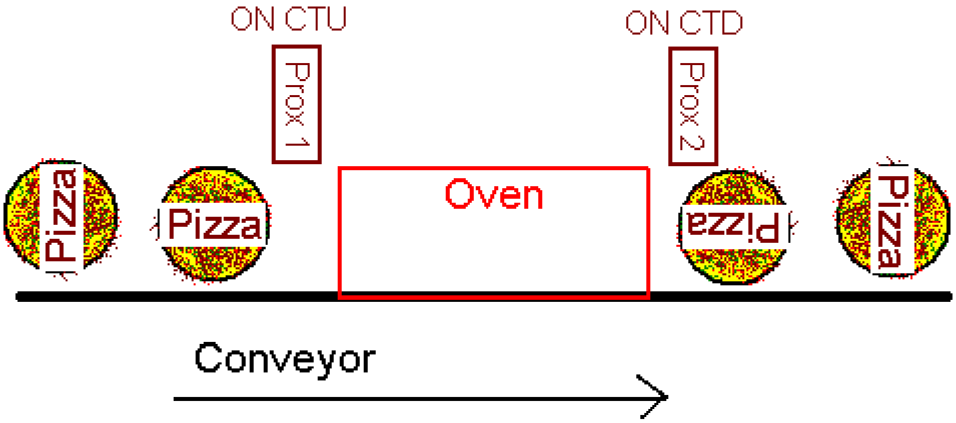


□	⊟-PartsCounter			COUNTER	
	⊞-PartsCounter.PRE			DINT	Decimal
	⊞-PartsCounter.ACC			DINT	Decimal
	PartsCounter.CU			BOOL	Decimal
	PartsCounter.CD			BOOL	Decimal
	PartsCounter.DN			BOOL	Decimal
	PartsCounter.OV			BOOL	Decimal
	PartsCounter.UN			BOOL	Decimal

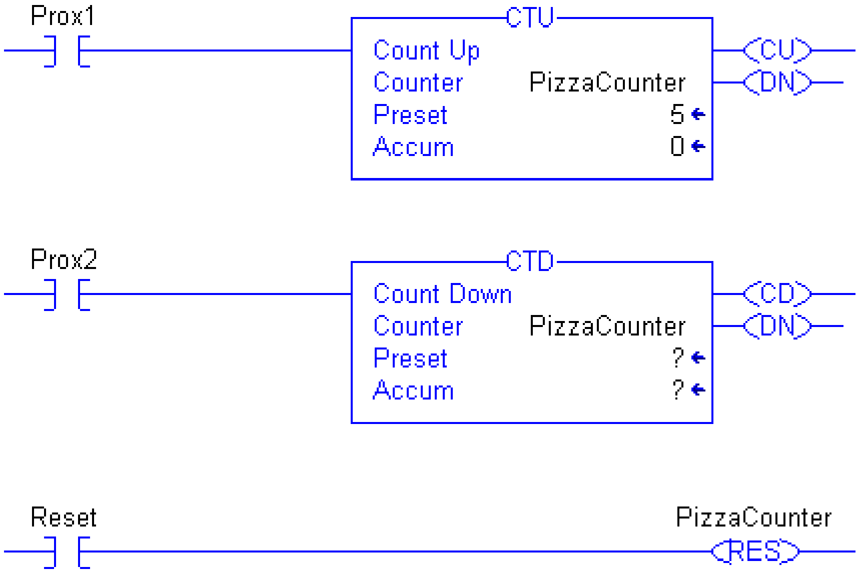
For the CTU instruction: The CU bit is high when the CTU instruction is true. The ACC value increments by the value of 1 each time the CU bit goes high. When the ACC reaches the PRE, the DN bit will be set. The CTU will continue to increment the accumulated value until it reaches the maximum possible value for a 32 bit signed integer (2147483647). If the CU bit goes high one more time, the OV bit will be set, and the ACC value will go to -2147483648. Each time the CU bit goes high, the ACC value will still continue to increment (become less negative).

For the CTD instruction: The CD bit is high when the CTD instruction is true. The ACC value decrements by the value of 1 each time the CD bit goes high. Any time the ACC is above or equal to the PRE, the DN bit will remain set. The DN bit is reset if the ACC falls below the PRE at any time. The CTD will continue to decrement the accumulated value until it reaches the minimum possible value for a 32 bit signed integer (-2147483648). If the CD bit goes high one more time, the UN bit will be set, and the ACC value will go to 2147483647. Each time the CD bit goes high, the ACC value will still continue to decrement (become less positive).

Here is a practical example of a CTU/CTD implementation:



Each time a pizza goes into the oven, the ACC value is incremented by one. Each time a pizza comes out of the oven, the ACC value is decremented by one. Therefore, the ACC value represents how many pizzas are in the oven at any given time. The DN bit could be used to shut the conveyor down if pizzas are going into the oven and not coming out!



```
Prox1                          CTU
                               Count Up
                               Counter      PizzaCounter    (CU)
                               Preset                  5    (DN)
                               Accum                   0

Prox2                          CTD
                               Count Down
                               Counter      PizzaCounter    (CD)
                               Preset                  ?    (DN)
                               Accum                   ?

Reset                                        PizzaCounter
                                                (RES)
```

Using the GSV Command (Accessing the system time)

The PLC-5 and SLC-500 had a STATUS FILE which could used by logic at any time simply by referring to the memory location where the time and date were stored.

The system time in ControlLogix works much differently. When a new project is created, no variables exist in the tag databases. If you are going to use a variable in logic, it must first be created first.

The system time in ControlLogix is called the WALLCLOCKTIME object. First an array must be be allocated in the tag database, then we must use a GSV (Get System Value) command to continuously load the time from the system into the new array.

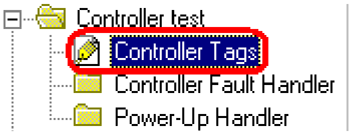
You should access the help file in RSLogix 5000 for a complete description of the WALLCLOCKTIME object. To access the help file, click Help on the menu bar, then click Contents. Click the FIND tab. If this is the first time you are using the Help|Find feature, you may be prompted to select next, then finish to build the help database. Type wallclocktime in step 1, then in step 3, double click 'accessing the wallclocktime object'.

For this example, we are simply going to create an array of seven elements in the controller tag database, then use the GSV command to populate the array with the system time. The purpose of each of the seven elements are as follows:

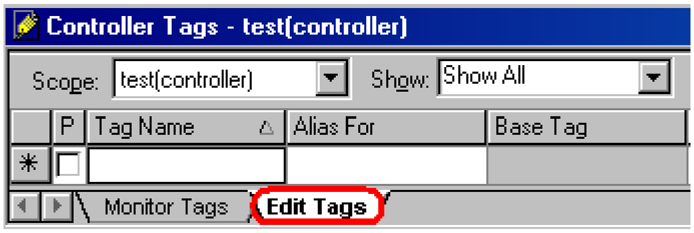
Element 0 – Year
Element 1 – Month
Element 2 – Day
Element 3 – Hour
Element 4 – Minute
Element 5 – Second
Element 6 – Microsecond

Let's get started....

1) First we need to open the controller tag database at the top of your controller organizer window.



2) Next, let's be sure the 'edit tags' tab is selected so we can add a new tag.



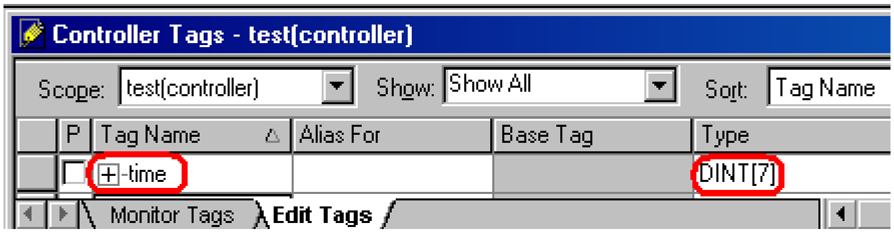
Controller Tags - test(controller)

Scope: test(controller)

Show: Show All

	P	Tag Name	Alias For	Base Tag
✳	□			

3) The new tag we create will be called 'time', and the type will be 'DINT[7]'. Recall that the [7] will create an array of 7 elements. Press enter to accept the tag.



Controller Tags - test(controller)

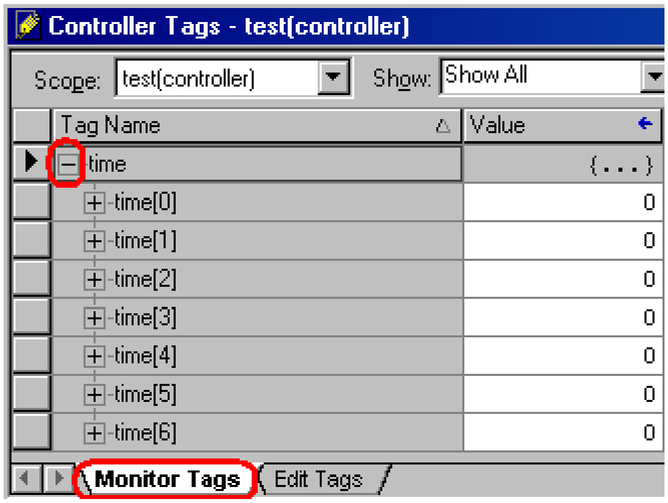
Scope: test(controller)

Show: Show All

Sort: Tag Name

	P	Tag Name	Alias For	Base Tag	Type
	□	⊞-time			DINT[7]

4) Let's look at the Array. Click “Monitor Tags”. Click the “+” next to the tag name, and you will see that seven elements have been created.



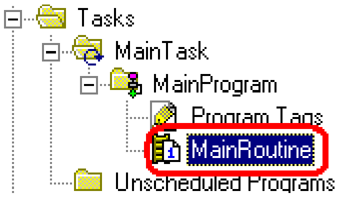
Controller Tags - test(controller)

Scope: test(controller)

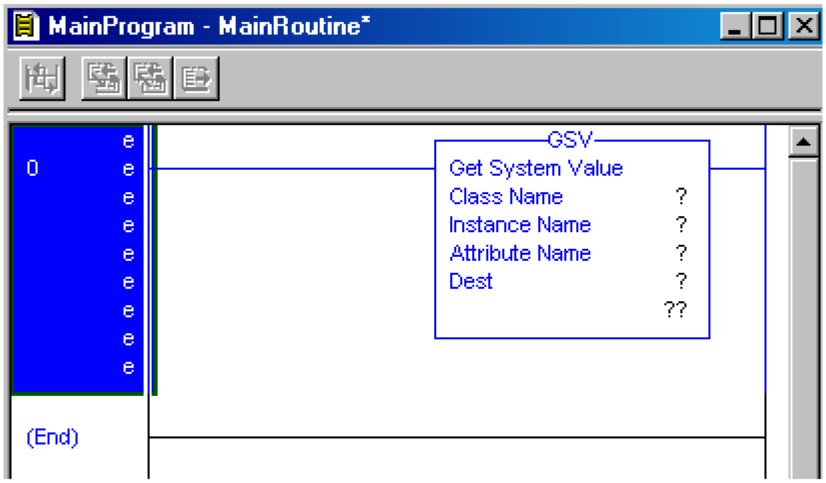
Show: Show All

	Tag Name	Value
▶	⊟ time	{...}
	⊞-time[0]	0
	⊞-time[1]	0
	⊞-time[2]	0
	⊞-time[3]	0
	⊞-time[4]	0
	⊞-time[5]	0
	⊞-time[6]	0

5) Next we need to choose which routine we will be adding the GSV to. Recall that the purpose of this GSV is to extract the wall clock time from the system, and load it into the tag we just created. For this example, we will go to the MainRoutine of the MainProgram. You can access the routines from the controller organizer window.

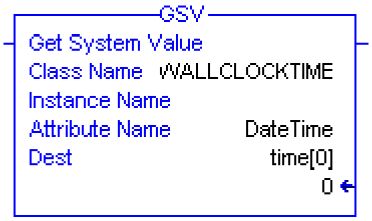


6) Be sure the end rung is highlighted, then type GSV (Then press "Enter") You will notice the GSV command has been added to you logic.



GSV
Get System Value
Class Name ?
Instance Name ?
Attribute Name ?
Dest ?
??

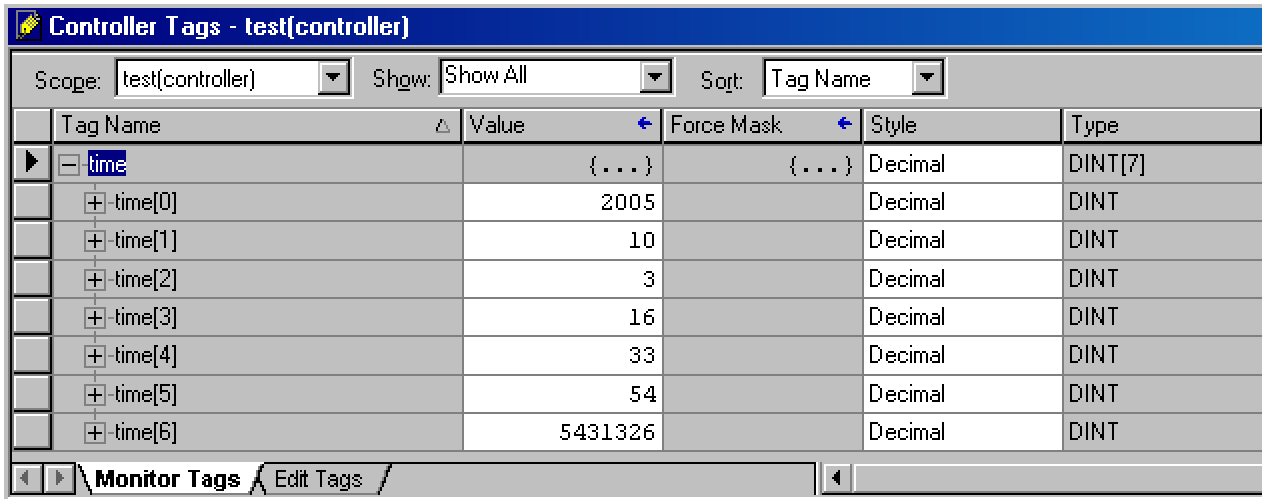
7) Double click the "?" next to Class Name. The class will be WALLCLOCKTIME. There is only one instance available of this class, so we don't need to select one. The Attribute will be DateTime. The destination should be time[0].



GSV
Get System Value
Class Name WALLCLOCKTIME
Instance Name
Attribute Name DateTime
Dest time[0]
0

8) Now you can download your work, and go on line in Run mode. If your GSV command is working properly, the GSV command will be extracting the time from the system, and loading the wall clock time (of 7 elements) into our time tag starting with element 0. To test this, go back to your controller tags, Be sure the time tag is still expanded, and you will see data in the following format:

time[0] = Year
time[1] = Month
time[2] = Day
time[3] = Hour
time[4] = Minute
time[5] = Second
time[6] = Microsecond



Controller Tags - test(controller)

Scope: test(controller) Show: Show All Sort: Tag Name

Tag Name	Value	Force Mask	Style	Type
time	{...}	{...}	Decimal	DINT[7]
time[0]	2005		Decimal	DINT
time[1]	10		Decimal	DINT
time[2]	3		Decimal	DINT
time[3]	16		Decimal	DINT
time[4]	33		Decimal	DINT
time[5]	54		Decimal	DINT
time[6]	5431326		Decimal	DINT

On line Editing for ControlLogix

There are five basic steps in performing an edit on line.

1) Start Edits,
2) Make Changes,
3) Accept edits,
4) Test Edits, and
5) Assemble edits.

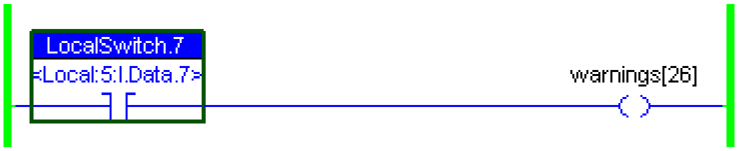
Note: Beginning with Version 13, '**Finalize Edits**' will accept, test, and assemble all in one step. Unless you are very experienced, it is recommended that you follow the full 5 step procedure.

Although these steps seem very simple there are a few rules to watch out for.

- You cannot change the data type of existing tags. If you create a new tag with the wrong data type, you must delete the tag, and declare it again.
- You cannot make an on line edit if the key switch is in Run Mode.
- You do not need to perform an on line edit to directly change a value in the data table such as the preset of a timer or counter.
- If the processor is in program mode, you do not need to test and assemble after accepting.
- If the processor is in program mode, and a rung is deleted, there is no warning.

Let's walk through the 5 step procedure:

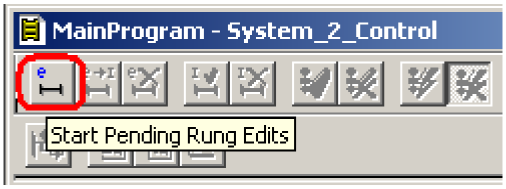
Look at the rung below. Our objective is to transfer control of the output to LocalSwitch.6. If you click on bit LocalSwitch.7 and attempt to make a change, nothing happens.



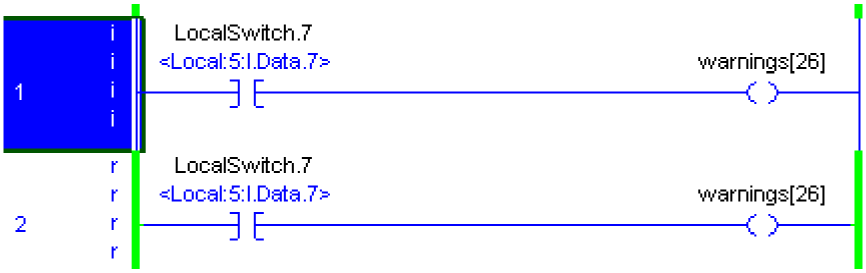
Step 1) Start Rung Edits

The first step is to put the rung into edit mode. There are several ways this can be done:

- Double click the rung number
- Right click the rung number and start rung edits
- From 'Logic' on the menu bar, click On line Edits, then start pending rung edits
- Click the start rung edit icon in the on line editing tool bar just above the ladder view



Notice that RSLogix made a copy of the rung for us to work with. By looking at the power rails, you can see the bottom rung is being executed by the processor, and the top rung is the one you need to make edits to. You will also notice the e (edit) or i (insert) and r (replace) in the margin are lower case. This means the edits are not in the processor yet. If you are adding new logic instead of modifying existing logic, this is the step where you add a new rung.



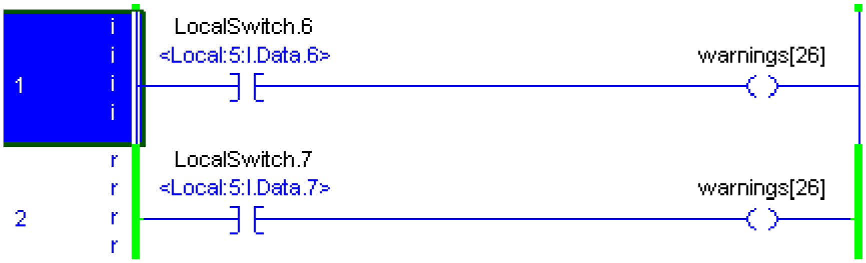
Step 2) Make Changes

Now that the rung is in edit mode, changes can be made.

If you added a new rung in step #1, this is where you need to add your logic to the new rung.

Be careful not to add any logic that will fault the processor or cause damage to personnel or equipment. Notice the i (insert) and r (replace) zones are in lower case. This means the changes are in RAM only, and have not been sent to the processor.

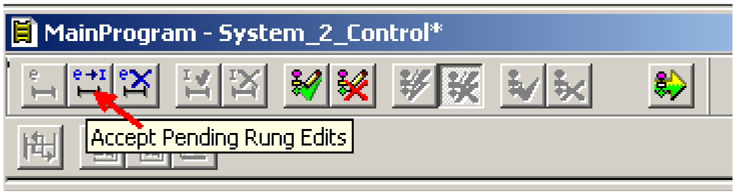
In this example, bit 7 is being changed to bit 6 on the input.

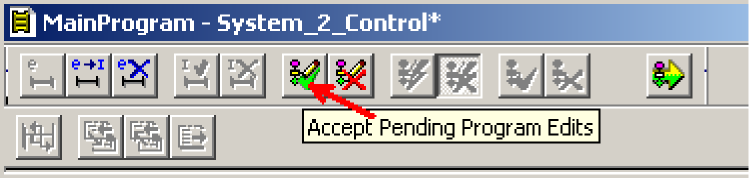


Step 3) Accept Edits

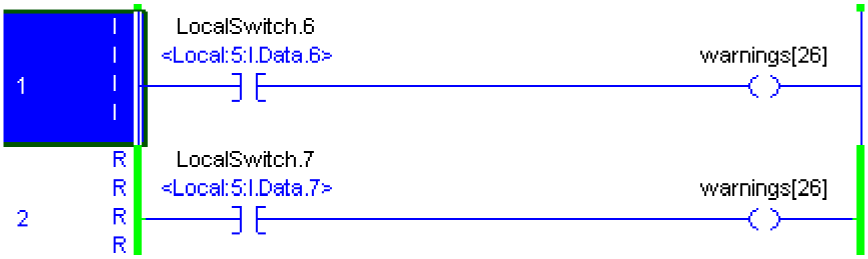
Now that your rung is set up as you need it, it's time to send the edits to the processor. You can accept pending **rung** edits (This would just accept the **rung** you have selected), or you can accept pending **program** edits (This would accept all the edits in the current **program**) There are several ways to perform the next three steps.

- Right click the rung number, and accept edits
- Click Logic | On line Edits | Accept (rung or program edits) from the menu bar
- Click one of the Accept Edits icons in the on line editing tool bar as shown below

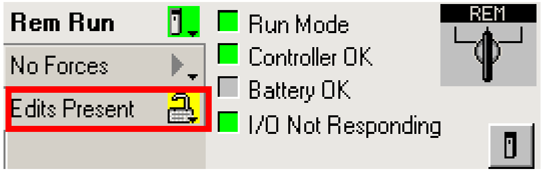




Notice in the margin rung 1 is marked for insertion, and rung 2 is marked for removal. The I's and R's are capitol because the edits are now in the processor. Look at the power rails. You can see the old rung is still being executed by the processor.



You will also see that pending edits exist by looking at the on line tool bar.

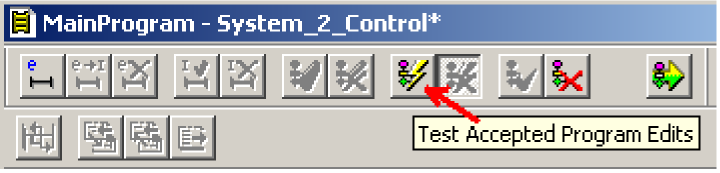


Step 4) Test Edits

When you test edits, the new or modified rungs will become active. The old rungs will be left in the processor until we are sure our new rungs are working properly. Be aware that if you change an output address, there might no longer be logic writing to that address. This means that you could abandon a bit in the ON state.

You can test your edits by doing one of the following actions:

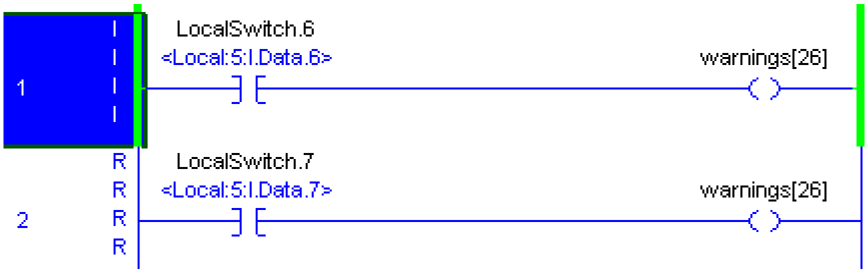
1) Right click the rung number
2) Choose Logic | On line Edits | Test accepted program edits from the menu bar
3) Click the Test icon in the on line edit tool bar above your logic window.



If you are modifying an input type address you should also be careful. If the rung was previously true, you may want to make sure your new logic is also going to be true at the moment you accept, or the the output may shut off.

Let's test the edits, and you will notice the new rung(s) are active. If the edits do not work the way you anticipated, you can un-test to revert to the old rung while you make other changes to the new rung.

Notice the power rails:

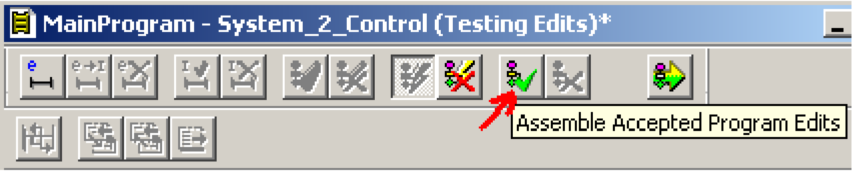


Step 5) Assemble Edits

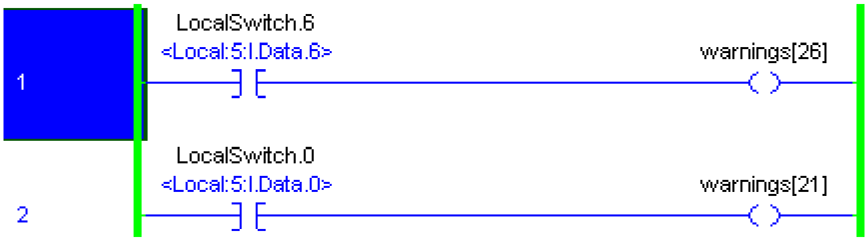
If you logic is working properly, go ahead and assemble the edits. Assembling removes the old rung, and the edit zone markers. After Assembling, you may want to save your work to the hard drive.

You can assemble by using one of the following methods:

1) Right click the rung number, and choose accept edits (if available in your version)
2) Click Logic | On line Edits | Assemble accepted program edits from the menu bar.
3) Click the Assemble Edits icon in the on line edits tool bar.


MainProgram - System_2_Control (Testing Edits)*
Assemble Accepted Program Edits

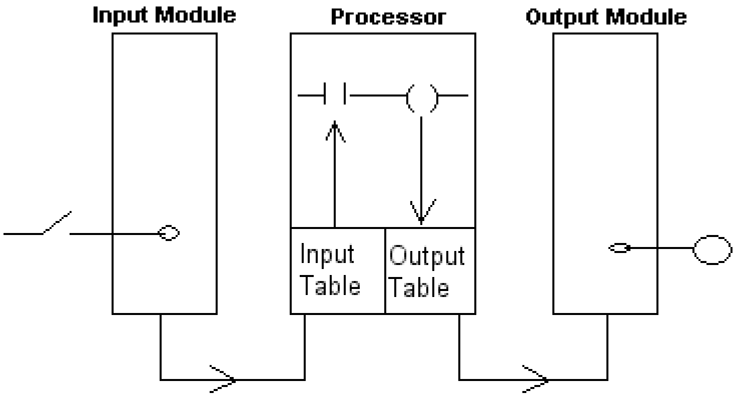
Notice the Logic now appears to be normal:



Forcing I/O

Forcing can be used for troubleshooting, and to some extent simulates real world jumpers. Leaving forces in the processor, or depending on forced I/O to make your equipment run is considered bad practice.

Look at the diagram below:



Under normal circumstances, the following events take place:

1. The switch is shut
2. A 1 appears in the input tag
3. The XIC instruction goes true
4. The OTE is enabled
5. A 1 is written to the output tag
6. The light will energize on the output module

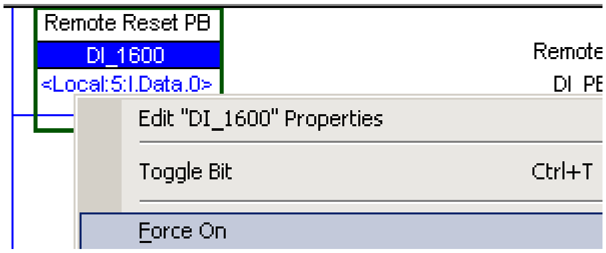
Forcing the input:
If you place a jumper across the switch, you would have the same effect as the switch always being shut. A 1 would always be in the data table, the logic would be true, and the light would energize. The same effect applys to forcing. Forcing the input on would result in a 1 in the input data table for the switch, and all logic would be executed as if the switch was shut. The opposite applys to an OFF force. An Off force would be similar to cutting a lead on the switch. A zero would result in the input data table.

Forcing the output:
If you place a jumper to the output, the output table would still be a zero if the logic is false. Information does not flow from the output device to the output data table. Therefore, any XIC instruction that is looking at the output bit would also be false. The same applies to forcing. If you force an output device, the output data table will still be controlled by the ladder logic.

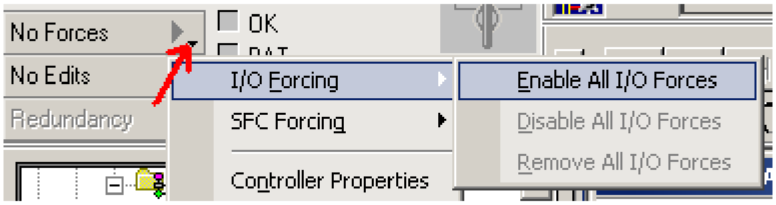
Note: Even though forcing an output does not directly effect the data table, The field device itself could feed an input back into the processor causing other things to happen in logic. Know your system before using the force feature.

There are several ways to force I/O. Forcing can be applied from ladder logic, or from the Controller Tag database. Internal memory locations cannot be forced. You can only force real I/O, Aliases to real I/O, or producer/consumer tags.

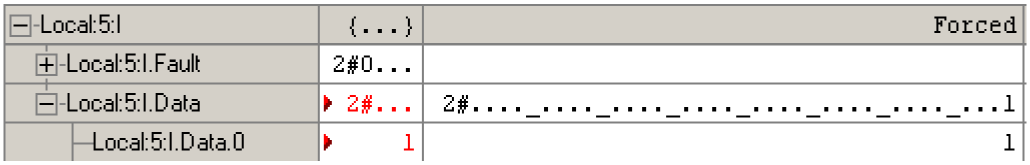
In this example, we will force an input directly from ladder logic. Right click on the input address, and choose 'Force On'.



Notice the force light on the processor begins to flash (if your process has a force light) indicating that forces are installed, but not enabled. The force can be enabled from the on line tool bar as shown below:



The force light on your processor will now be solid amber indicating that installed forces have been enabled. If we go to the data table, you will see that the input bit is on, and it is red indicating that a force has been enabled on the input. You will also see that the force mask reflects which bits have been forced in the data word. Forcing can be preformed directly from the force mask as well.



⊟-Local:5:I	{...}	Forced
⊞-Local:5:I.Fault	2#0...	
⊟-Local:5:I.Data	2#...	2#...._...._...._...._...._...._...._...1
Local:5:I.Data.0	1	1

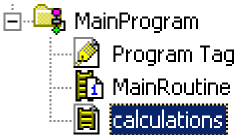
In the force mask, the value of 1 indicates a bit has been forced on. The value of 0 indicates an off force, and a period indicates no force is installed on a particular bit.

JSR Instruction

We discussed the use of the JSR instruction in the main routine that instructs the processor to execute a subroutine. A subroutine can also behave as a function, and with the JSR instruction, we can pass values to that subroutine, and the subroutine can perform math on this value, and return an answer back to the JSR statement. This can be very useful when the subroutine must be called repeatedly throughout a program scan to perform operations such as converting Celsius to Fahrenheit.

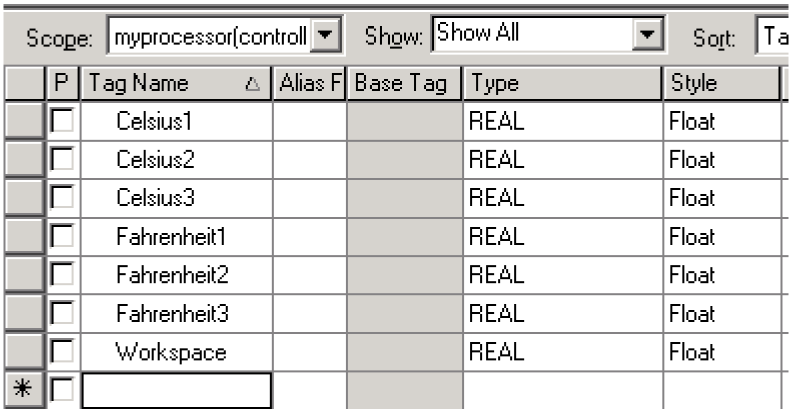
This example will walk you through setting up the JSR instruction to pass several values (Celsius) to a subroutine. The values will then be converted, and returned to the JSR statement in Fahrenheit units.

1) Add the a routine called "Calculations" to the MainProgram.



MainProgram
- Program Tags
- MainRoutine
- calculations

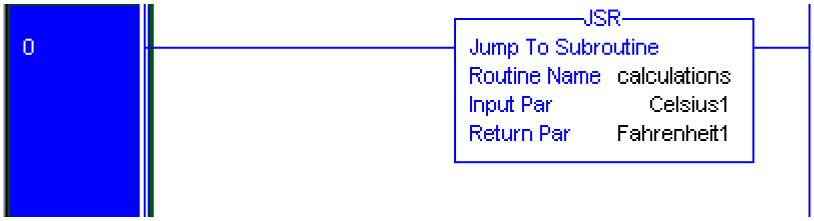
2) Add the following tags to the CONTROLLER tag database as shown:
   1. Celsius1 as REAL
   2. Celsius2 as REAL
   3. Celsius3 as REAL
   4. Fahrenheit1 as REAL
   5. Fahrenheit2 as REAL
   6. Fahrenheit3 as REAL
   7. Workspace as REAL



Scope: myprocessor(controll ▼ Show: Show All ▼ Sort: Ta

	P	Tag Name △	Alias F	Base Tag	Type	Style
	□	Celsius1			REAL	Float
	□	Celsius2			REAL	Float
	□	Celsius3			REAL	Float
	□	Fahrenheit1			REAL	Float
	□	Fahrenheit2			REAL	Float
	□	Fahrenheit3			REAL	Float
	□	Workspace			REAL	Float
*	□					

3) Next, on the MainRoutine, we'll add a JSR instruction that will pass one parameter (Celsius1) to the calculations routine. When we get a value returned from the calculations routine, we will place this value into Fahrenheit1. To add the JSR instruction, type the following text onto a rung of the MainRoutine: **jsr calculations1 1 Celsius Fahrenheit1**

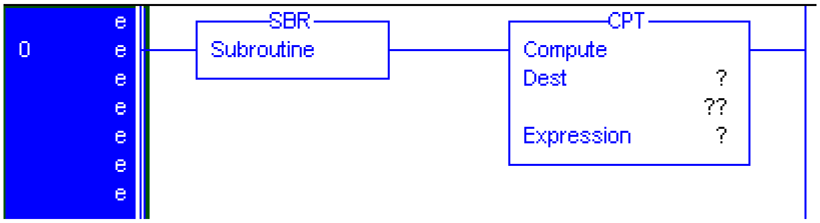


0

JSR
Jump To Subroutine
Routine Name calculations
Input Par Celsius1
Return Par Fahrenheit1

4) So far, we have the project set up to pass the value of Celsius1 to the subroutine called "Calculations". This subroutine will perform the math function necessary to perform the conversion, and will return a value back to the JSR statement when the subroutine is finished. When the JSR statement receives this returned value, it will store the value into the tag called Fahrenheit1.

5) We are ready to set up our subroutine. Let's Open the "Calculations" routine, and type the following text into rung 0: **SBR CPT**

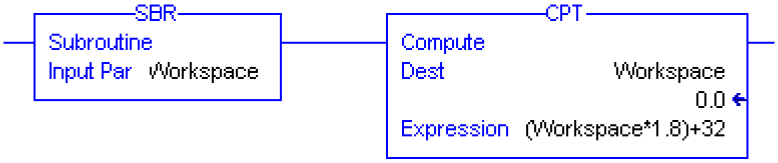


0

SBR
Subroutine

CPT
Compute
Dest ?
??
Expression ?

6) When this subroutine (calculations) receives a value from the JSR statement, it will store this value in a location declared by the SBR tag (our workspace). The CPT statement will then manipulate this workspace to convert the workspace to Fahrenheit. Set up these two instructions as shown below: (You can right click the SBR statement to add an input parameter.)

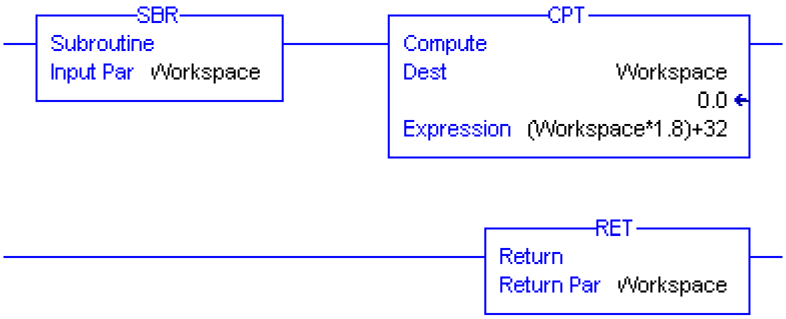


SBR
Subroutine
Input Par Workspace

CPT
Compute
Dest Workspace
0.0
Expression (Workspace*1.8)+32

7) Now, all that's left to do is to return the value of the workspace back to the JSR instruction, so the JSR can store the return value into the Celsius tag. Click the end rung in calculations, and type: **RET Workspace**

8) When finished, your calculations routine should appear as shown below:

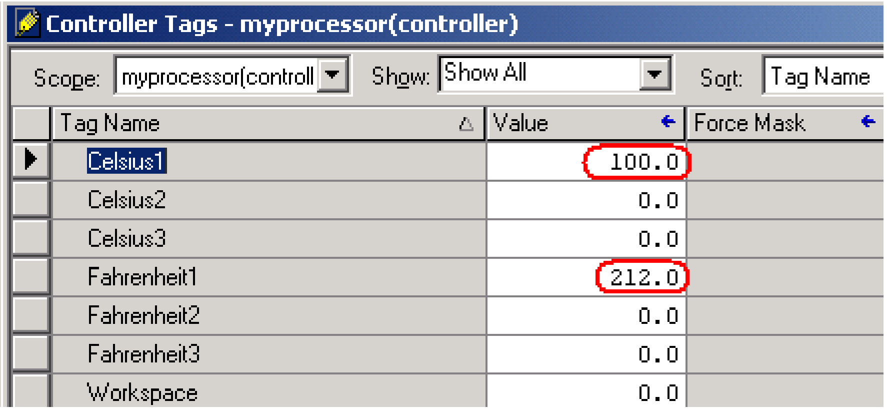


SBR
Subroutine
Input Par Workspace

CPT
Compute
Dest Workspace
0.0
Expression (Workspace*1.8)+32

RET
Return
Return Par Workspace

9) Now, let's go to the Controller tag database. When you enter a Celsius value into the Celsius1 tag, this value should then be converted to Fahrenheit, and returned into the Fahrenheit1 tag.

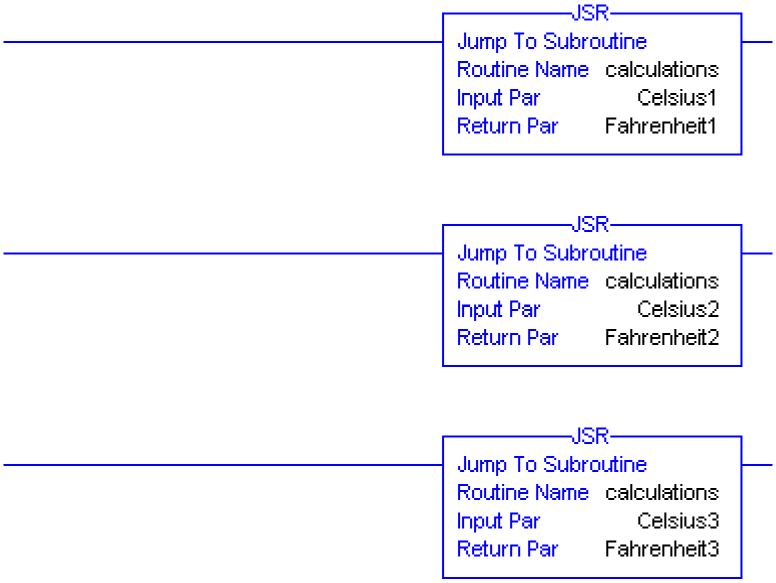


Controller Tags - myprocessor(controller)

Scope: myprocessor(controll Show: Show All Sort: Tag Name

Tag Name	Value	Force Mask
Celsius1	100.0	
Celsius2	0.0	
Celsius3	0.0	
Fahrenheit1	212.0	
Fahrenheit2	0.0	
Fahrenheit3	0.0	
Workspace	0.0	

10) Now, add two more JSR statements to the MainRoutine that will convert Celsius2 and Celsius3 to Fahrenheit values as well.



```
JSR
Jump To Subroutine
Routine Name   calculations
Input Par          Celsius1
Return Par      Fahrenheit1

JSR
Jump To Subroutine
Routine Name   calculations
Input Par          Celsius2
Return Par      Fahrenheit2

JSR
Jump To Subroutine
Routine Name   calculations
Input Par          Celsius3
Return Par      Fahrenheit3
```

Introduction to Function Blocks

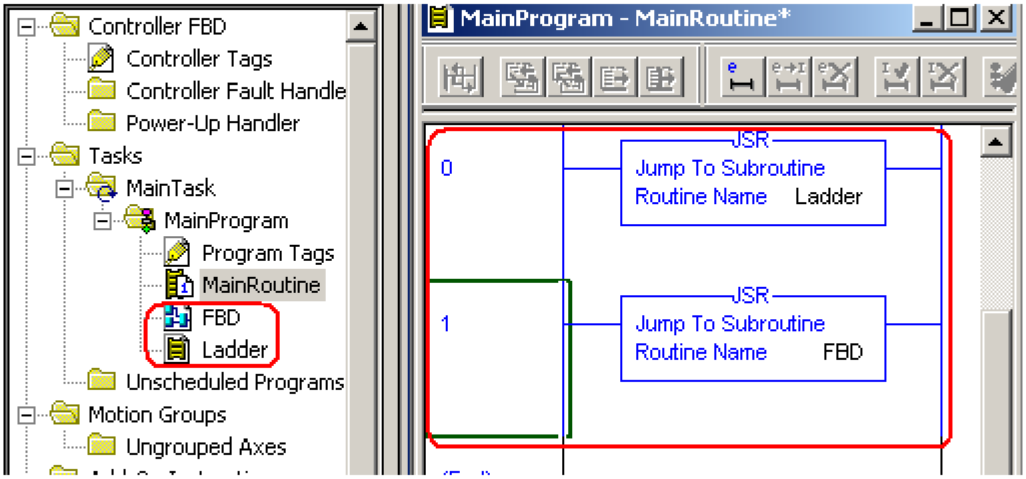
Function blocks are a method of programming that uses graphic objects on a sheet to represent logical functions. To program and view function blocks, you must have the appropriate license key installed on your computer. Function block routines can consist of multiple sheets of these block diagrams.

The advantage of using function blocks is that they are more graphical, and represent information flow very easily.

The disadvantage of function blocks is that if not used properly, the project becomes very difficult to follow.

In this example, we will be converting a simple line of logic into a function block diagram for the purpose of understanding the basic operation of this language.

In this example, I've created two routines, Ladder, and FBD. As you can see, the FBD is just another subroutine of the MainProgram, so we must add a JSR in this case, so it will execute.



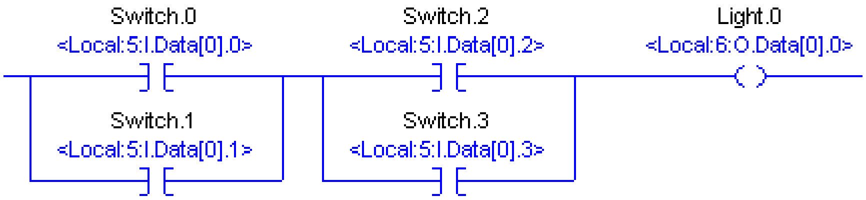
- Controller FBD
  - Controller Tags
  - Controller Fault Handle
  - Power-Up Handler
- Tasks
  - MainTask
    - MainProgram
      - Program Tags
      - MainRoutine
      - FBD
      - Ladder
  - Unscheduled Programs
- Motion Groups
  - Ungrouped Axes

MainProgram - MainRoutine*

0 JSR
Jump To Subroutine
Routine Name Ladder

1 JSR
Jump To Subroutine
Routine Name FBD

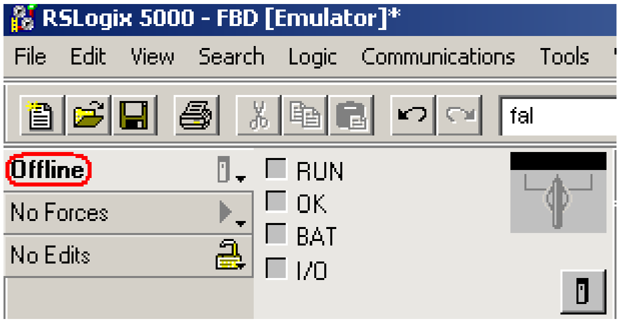
In the LADDER routine, we have a simple line of logic as follows:



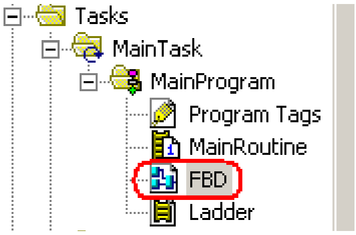
This logic can be expressed by the following statement in english:
If (**switch.0** OR **switch.1**) AND (**switch.2** OR **switch.3**) then **light.0**

Next, we'll convert this to a function block, and we will have the exact same result.

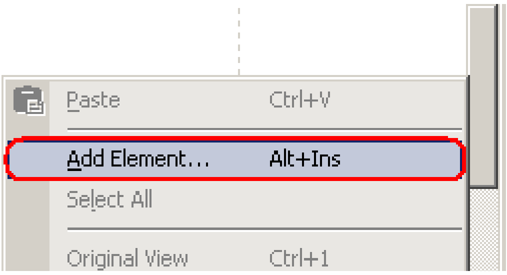
1) First, let's go to the FBD routine. Online editing is possible for function blocks, however, for this example, we'll do this offline.


RSLogix 5000 - FBD [Emulator]*
File Edit View Search Logic Communications Tools
fal
Offline
No Forces
No Edits
RUN
OK
BAT
I/O

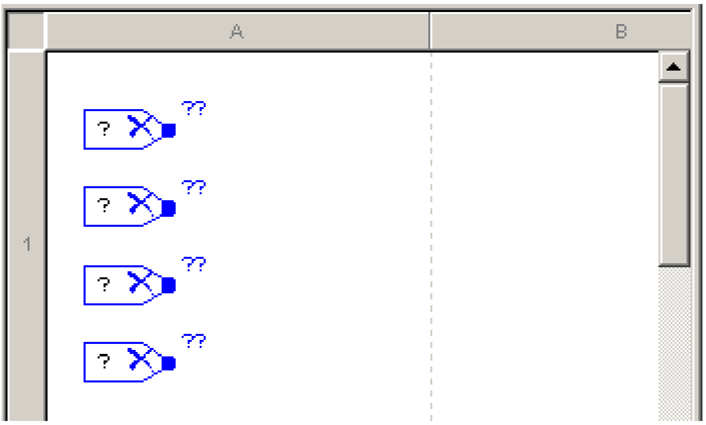
2) Now open the FBD routine in your Controller Organizer window:



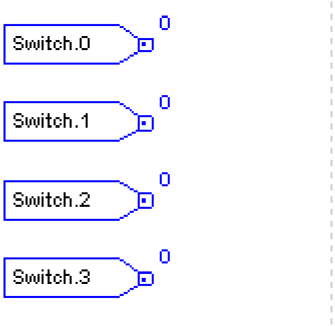
3) Next, well add our input references. To do this, right click the FBD sheet, and add for IREF elements (InputREFerences)



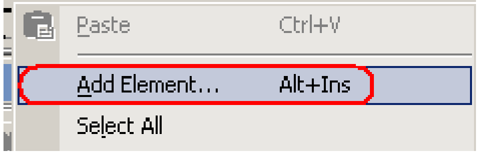
4) When you are finished, your sheet should appear similar to what is shown below...



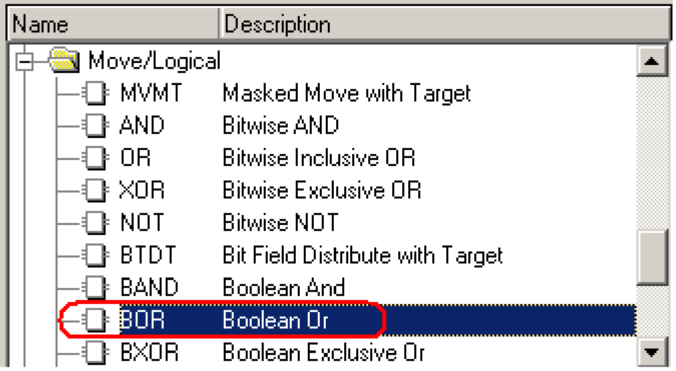
5) Next, we'll address the Input Reference elements. Double click the '?' on each of the input reference elements, and browse to the switches we are using for this conversion as shown:



6) Next, we will add two Boolean OR instructions. Right click your sheet, and choose "Add Element"

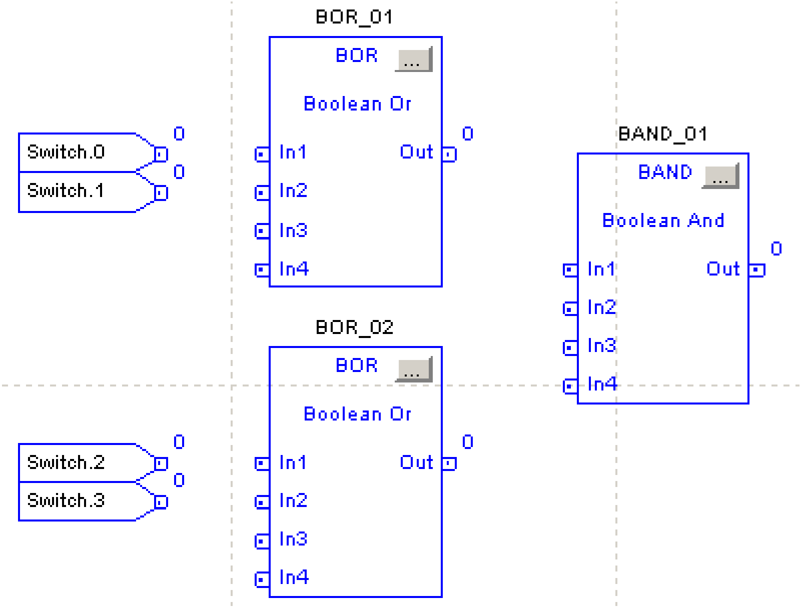


7) Under the Move/Logical category, choose the “Boolean OR” instruction.

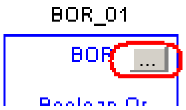


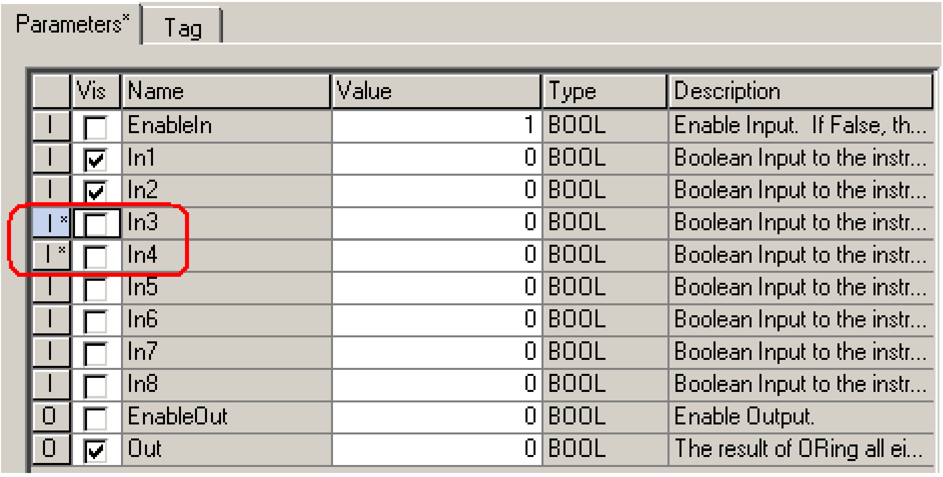
Name	Description
Move/Logical	
MVMT	Masked Move with Target
AND	Bitwise AND
OR	Bitwise Inclusive OR
XOR	Bitwise Exclusive OR
NOT	Bitwise NOT
BTDT	Bit Field Distribute with Target
BAND	Boolean And
BOR	Boolean Or
BXOR	Boolean Exclusive Or

8) Repeat this to add a second “BOR” statement, and a Boolean AND statement (BAND). Arrange your sheet as follows:



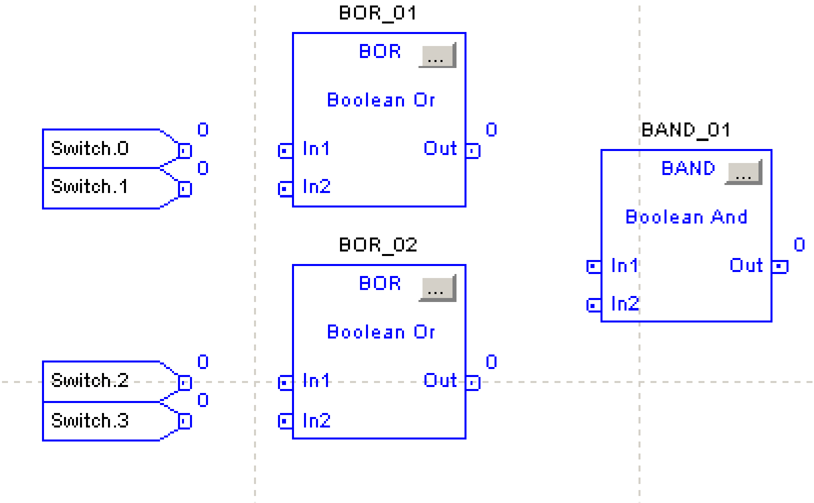
9) Now, we only need 2 inputs for each of these Boolean statements, so click the Elipsis, and uncheck inputs 2 and 3 on each of the 3 statements you just added:



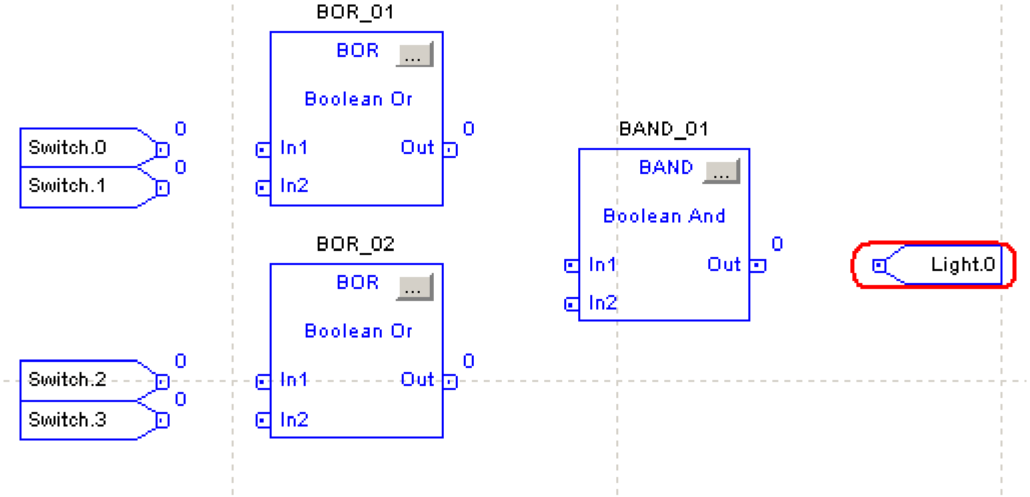


	Vis	Name	Value	Type	Description
I	☐	EnableIn	1	BOOL	Enable Input. If False, th...
I	☑	In1	0	BOOL	Boolean Input to the instr...
I	☑	In2	0	BOOL	Boolean Input to the instr...
I *	☐	In3	0	BOOL	Boolean Input to the instr...
I *	☐	In4	0	BOOL	Boolean Input to the instr...
I	☐	In5	0	BOOL	Boolean Input to the instr...
I	☐	In6	0	BOOL	Boolean Input to the instr...
I	☐	In7	0	BOOL	Boolean Input to the instr...
I	☐	In8	0	BOOL	Boolean Input to the instr...
O	☐	EnableOut	0	BOOL	Enable Output.
O	☑	Out	0	BOOL	The result of ORing all ei...

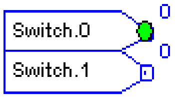
10) When finished, your sheet will look similar to the image below:

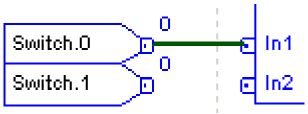


11) Now we need to add an output reference. Right click your sheet and add the output reference. Label this reference “light.0” as shown:

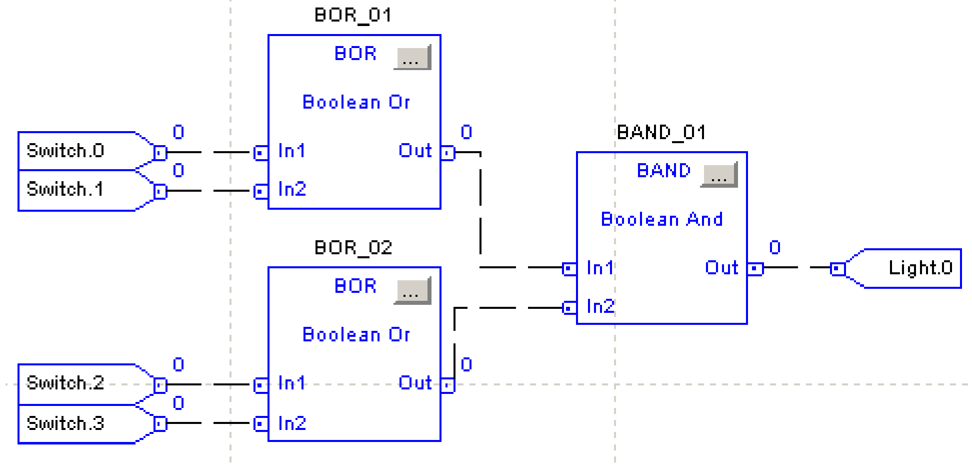


12) Now we need to connect the nodes. Move your mouse over the top of the IREF for switch.0. This tip will turn green to indicate you are at the correct location to make a connection. Click on the green tip, and draw a line to the In1 node of BOR_01

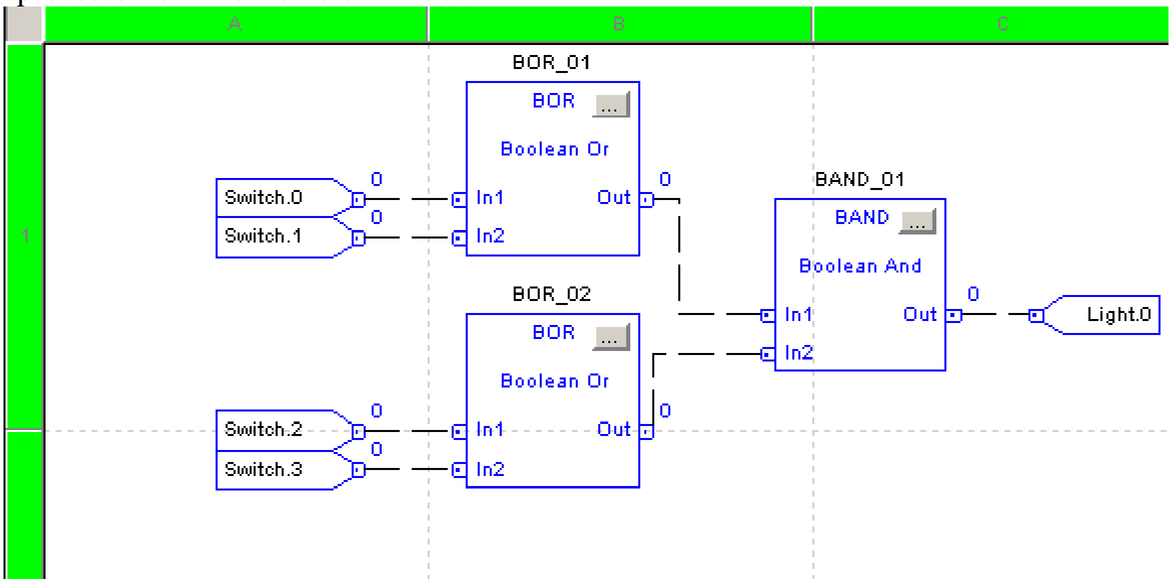




13) Repeat this for the other connections as shown:



14) Now, remove your logic from the LADDER routine, and download your work. Take the processor to run or remote run mode.

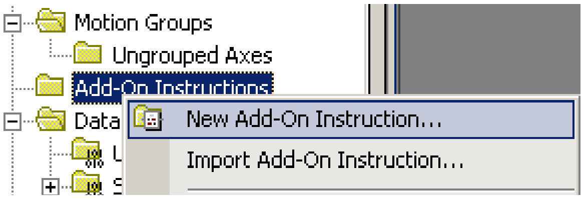


15) Now just as before, if you energize switch.0 OR switch.1, this is not enough to energize your output. You must also energize switch.2 OR switch.3 as before.

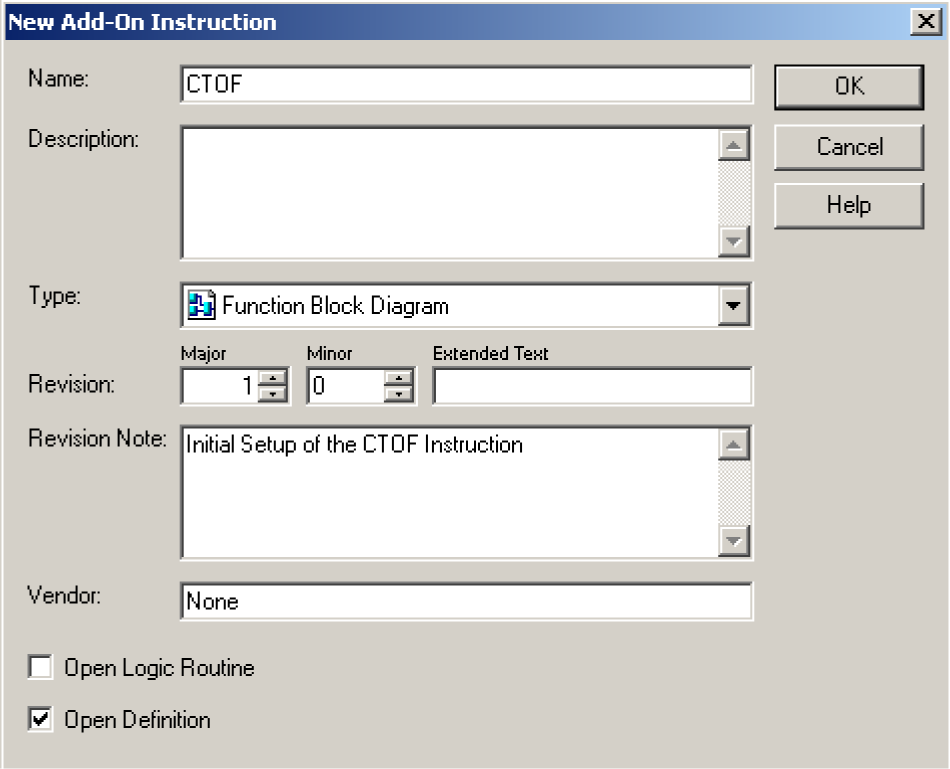
Creating an Add-On Instruction

Add-on instructions allow you to create your own instructions which can be used in your project. Add-on instructions can be in the form of Ladder Logic, Function Block Diagrams, or Structured Text. This can be useful if you have complex algorighms, and need to make the process of troubleshooting easier, or if you have common algorithms that will be used many times throughout a project. To graphically view the flow of information, we are going to use a Function Block Diagram for this particular example. The add-on instruction we write will convert Celsius to Fahrenheit.

1) Right click the Add-On Instruction folder, and select "New Add-On Instruction"



2) Complete the "New Add-On Instruction" dialog box as follows, then press OK.



Name: CTOF

Description:

Type: Function Block Diagram

Revision: Major 1 Minor 0 Extended Text

Revision Note: Initial Setup of the CTOF Instruction

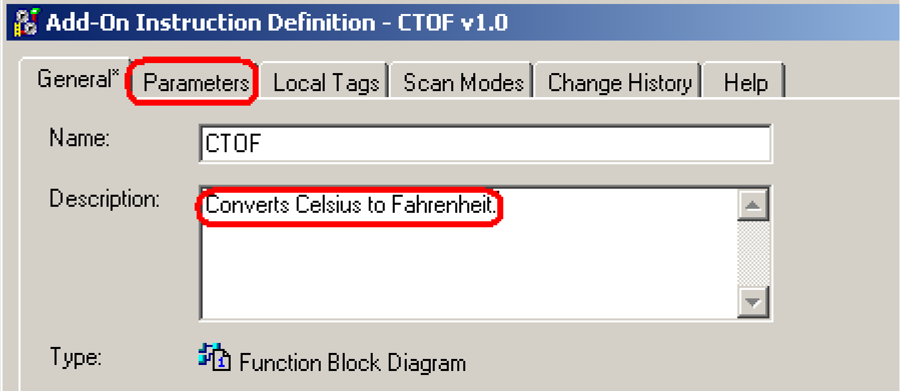
Vendor: None

☐ Open Logic Routine

☑ Open Definition

3) The Definition box will now appear.

4) Enter your description, then go to the “Parameters” tab.

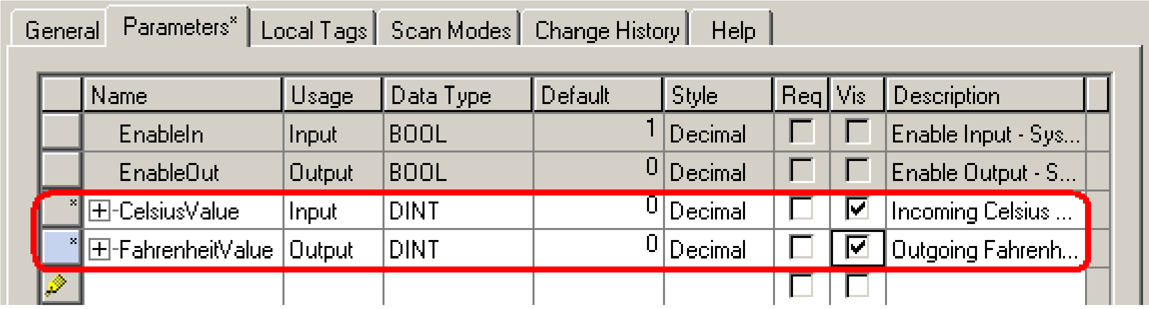


Name: CTOF

Description: Converts Celsius to Fahrenheit

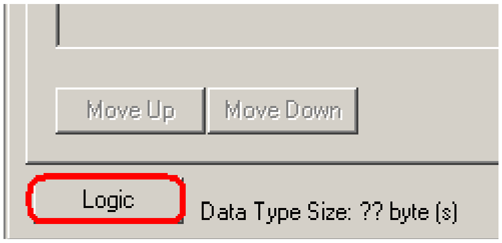
Type: Function Block Diagram

5) Configure your parameters as follows. The CelsiusValue will be an INPUT parameter, and the FahrenheitValue will be an OUTPUT Parameter. Be sure to turn on the VIS(visibility) property for each of the parameters specified so we will have nodes to tie references to when the instruction is added to logic.



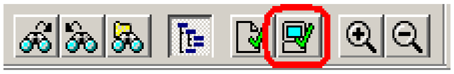
Name	Usage	Data Type	Default	Style	Req	Vis	Description
EnableIn	Input	BOOL	1	Decimal	□	□	Enable Input - Sys...
EnableOut	Output	BOOL	0	Decimal	□	□	Enable Output - S...
⊞-CelsiusValue	Input	DINT	0	Decimal	□	☑	Incoming Celsius ...
⊞-FahrenheitValue	Output	DINT	0	Decimal	□	☑	Outgoing Fahrenh...
					□	□	

6) Now, click the “Logic” button on the bottom of the Parameters screen.

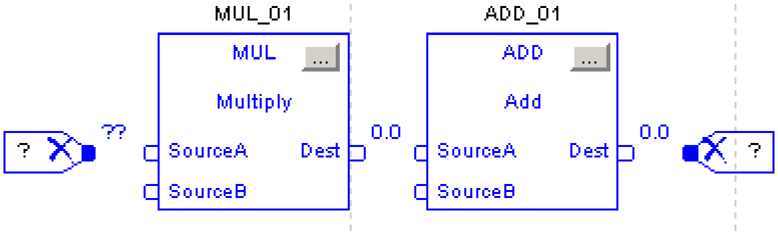


Data Type Size: ?? byte (s)

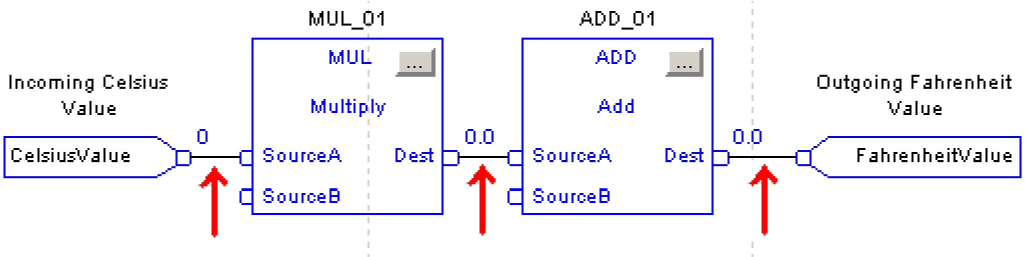
7) Verify the project to update your tables. If you have errors, we'll work them out later.



8) Right click your function block sheet to add an IREF (Input Reference), MUL (Multiply), ADD (Addition), and OREF (Output Reference) as shown. This will be the logic which runs when the Add-On Instruction is executed.

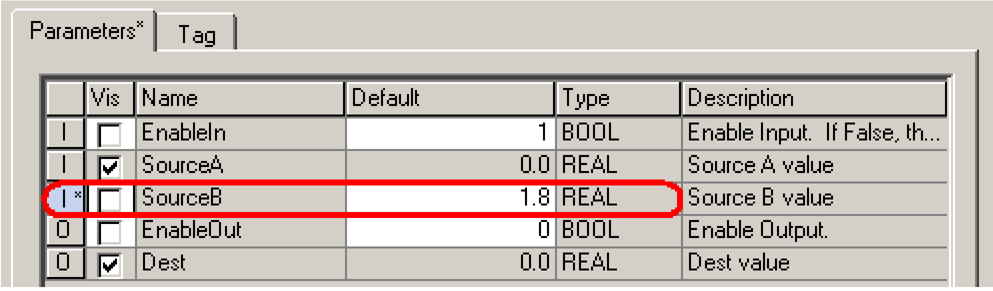


9) Now, we'll set up our parameters... Double click the "?" on the IREF instruction, and browse to the tag we created called "Celsius Value".... Do the Same for the OREF instruction, but it will point to the FahrenheitValue Tag.



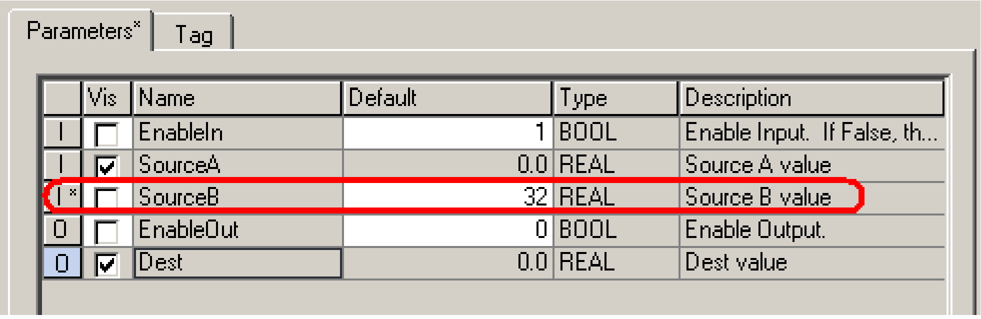
10) Now, we need to configure "Source B" for each instruction. Source B for the MUL instruction should be 1.8, and Source B for the ADD instruction should be 32. Click the Elipsis on each instruction to change these values (The three dots "...") You can uncheck the visibility property of each instruction for Source B because we will be using a static value that will not be changing.

11) The MUL Instruction will be set up as follows:



	Vis	Name	Default	Type	Description
I	□	EnableIn	1	BOOL	Enable Input. If False, th...
I	☑	SourceA	0.0	REAL	Source A value
I *	□	SourceB	1.8	REAL	Source B value
O	□	EnableOut	0	BOOL	Enable Output.
O	☑	Dest	0.0	REAL	Dest value

12) And the ADD instruction will be configured like this:

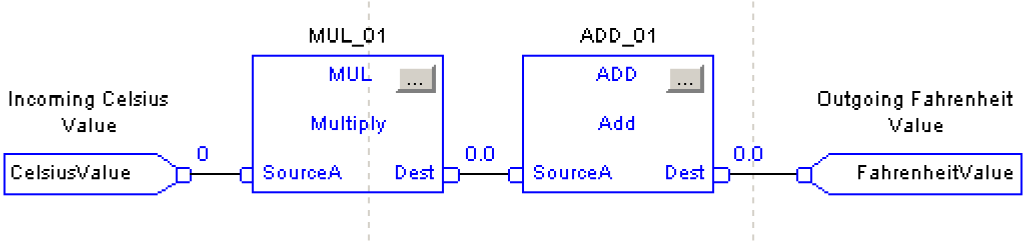


Parameters* | Tag

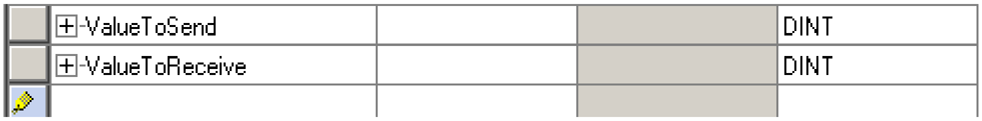
	Vis	Name	Default	Type	Description
I	□	EnableIn	1	BOOL	Enable Input. If False, th...
I	☑	SourceA	0.0	REAL	Source A value
I *	□	SourceB	32	REAL	Source B value
O	□	EnableOut	0	BOOL	Enable Output.
O	☑	Dest	0.0	REAL	Dest value

13) Be sure to save your work periodically to avoid loosing any work we've done in the event of a power outage, or inadvertent shutdown of RSLogix.

14) Now, our instructions routine will appear as follows:

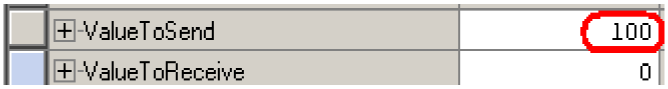


15) We are ready to utilize the instruction. First, we must create a few tags that will send a value to the instruction, and receive a value from the instruction. Go to the Controller Tag Database, and create the following Tags (You must be in "Edit Tags" mode to create these tags)



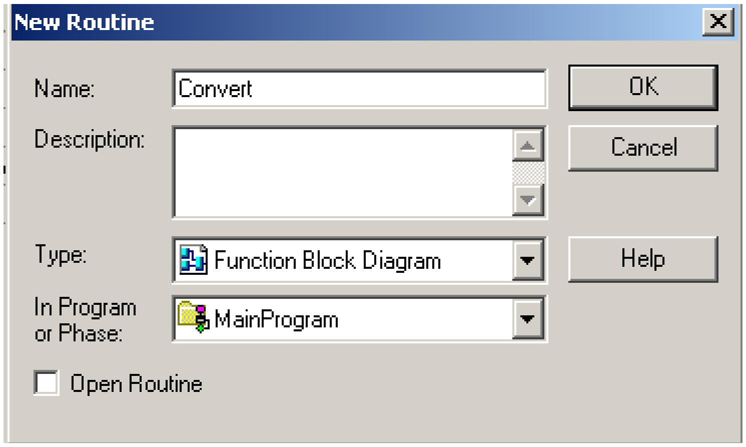
	⊞-ValueToSend			DINT
	⊞-ValueToReceive			DINT
✎				

16) Under "Monitor Tags", enter a value into the "ValueToSend" tag. Later this value will be converted, and the result will be stored into the "ValueToReceive" Tag.



	⊞-ValueToSend	100
	⊞-ValueToReceive	0

17) Now, we can create a new program routine to utilize our new instruction. Right click on the MainProgram, and add a new routine called "Convert" as follows: (Be sure to create it as a function block for this example).



Name: Convert

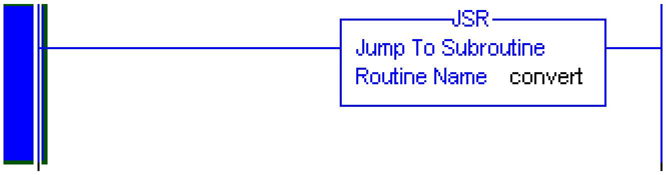
Description:

Type: Function Block Diagram

In Program or Phase: MainProgram

☐ Open Routine

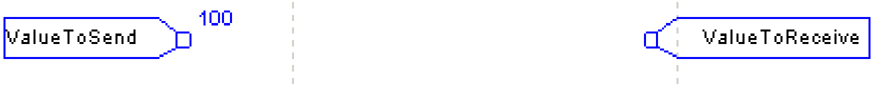
18) Now, open the MainRoutine of the MainProgram, and add a "JSR Convert" instruction as shown:


JSR

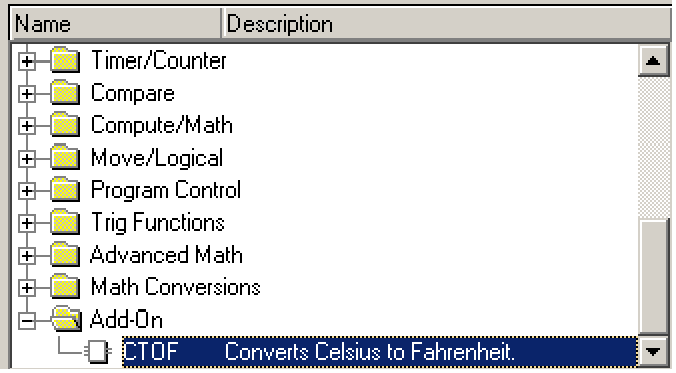
Jump To Subroutine

Routine Name   convert

19) Now, open the routine you just created, and add an IREF and and OREF to your sheet. The IREF will contain our ValueToSend tag, and the OREF will contain our ValueToReceiveTag.

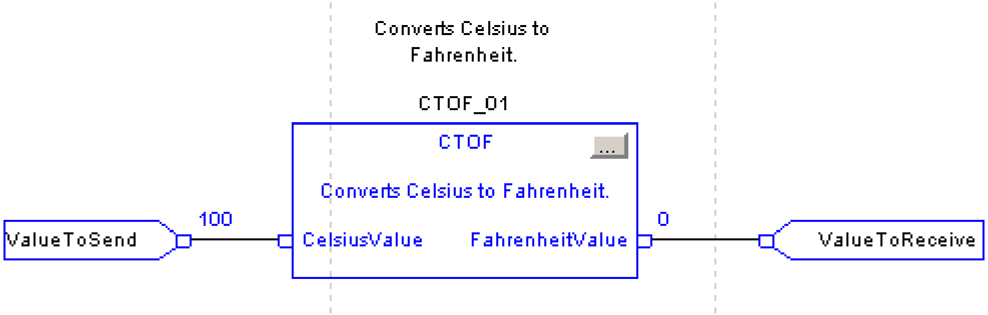


20) The next step is to add our CTOF instruction we created. Right click your sheet to add the element. The CTOF instruction will be in the Add-On folder at the bottom of your list.

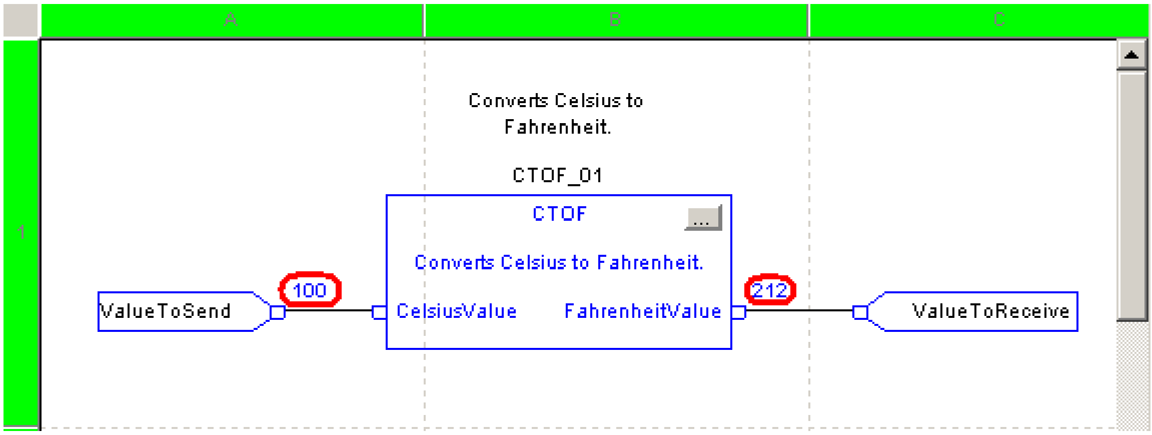


Name	Description
Timer/Counter	
Compare	
Compute/Math	
Move/Logical	
Program Control	
Trig Functions	
Advanced Math	
Math Conversions	
Add-On	
CTOF	Converts Celsius to Fahrenheit.

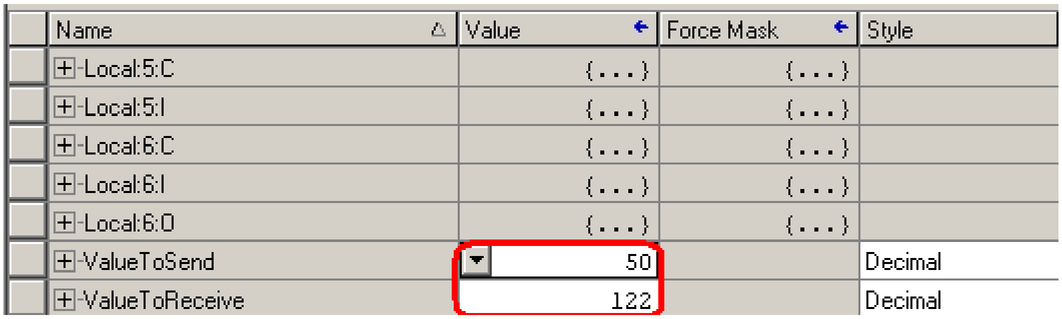
21) Draw your links from the IREF and to the OREF as shown. No other configuration for this instruction is necessary.



22) Download your work. Verify that when you change a value in your ValueToSend, the ValueToReceive changes to the converted value.



23) Since we don't have any logic at this point to populate the “ValueToSend” tag, you can manually change the value from Controller Tags.



Name	Value	Force Mask	Style
⊞-Local:5:C	{...}	{...}	
⊞-Local:5:I	{...}	{...}	
⊞-Local:6:C	{...}	{...}	
⊞-Local:6:I	{...}	{...}	
⊞-Local:6:O	{...}	{...}	
⊞-ValueToSend	50		Decimal
⊞-ValueToReceive	122		Decimal

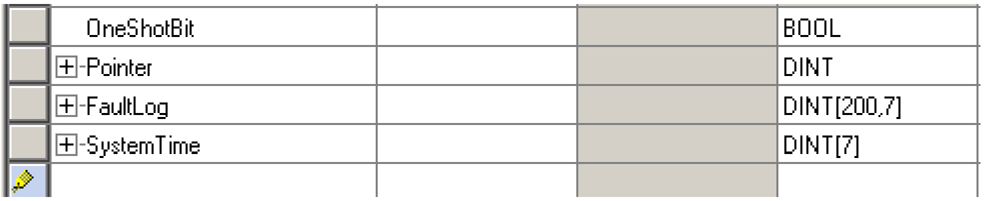
Indirect Addressing

Indirect addressing allows you to have a variable address. Although this can save much time and memory while programming, indirect addressing can be difficult to troubleshoot if you are unfamiliar with the system.

In this example, we will create a fault log. Each time a fault occurs, we will log the time and date to the NEXT element of the fault log array. An Array is simply a group of elements which all have the same name, but I unique index number. In the PLC-5, for example, the Timer file (T4) was an array of timers. Each timer had a different "Index" number, such as T4:0, T4:1, etc.

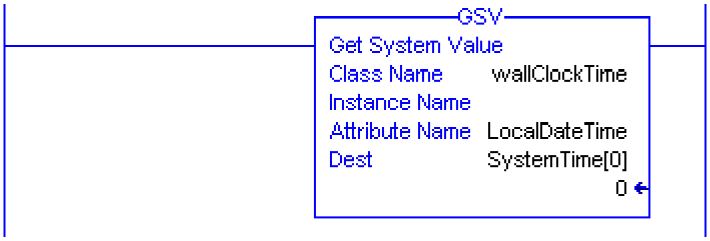
We will represent a fault with a simple switch. We will create the fault log as a two dimensional array. The first element of the array will represent a fault number (We'll allow up to 200 faults).... The second element of the array will log the time at which the fault log occurred, which will be seven elements: Year, Month, Day, Hour, Minute, Second, and Microsecond. We will also need a pointer that we can change to index us through the fault log, and a OneShot that will only log one time per fault (instead of logging to the fault log every scan). The GSV command will be used to extract the time from the system, and store this time to a temporary array that we can copy from later on.

1) First, Let's create our Variables. We'll just add these variables to the Controller Tag Database as follows:



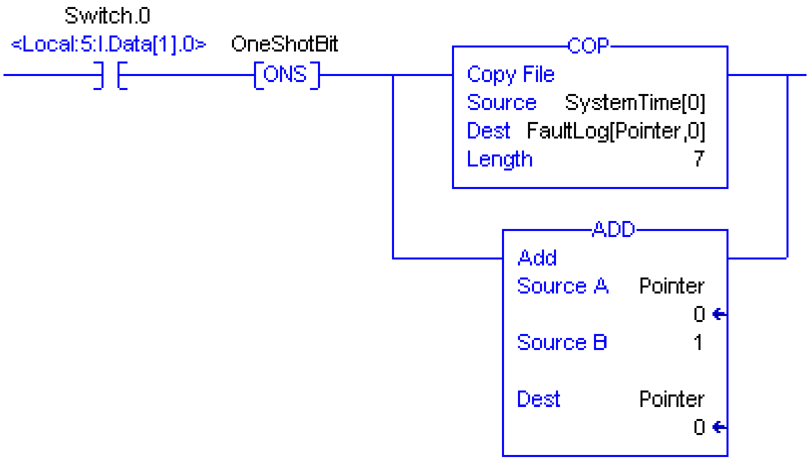
OneShotBit			BOOL
⊞-Pointer			DINT
⊞-FaultLog			DINT[200,7]
⊞-SystemTime			DINT[7]

2) Next, we need to set up the GSV command to get the time from the system, and store this time to the tag we just created. If you need to adjust the clock, you can do that easily from Controller Properties.

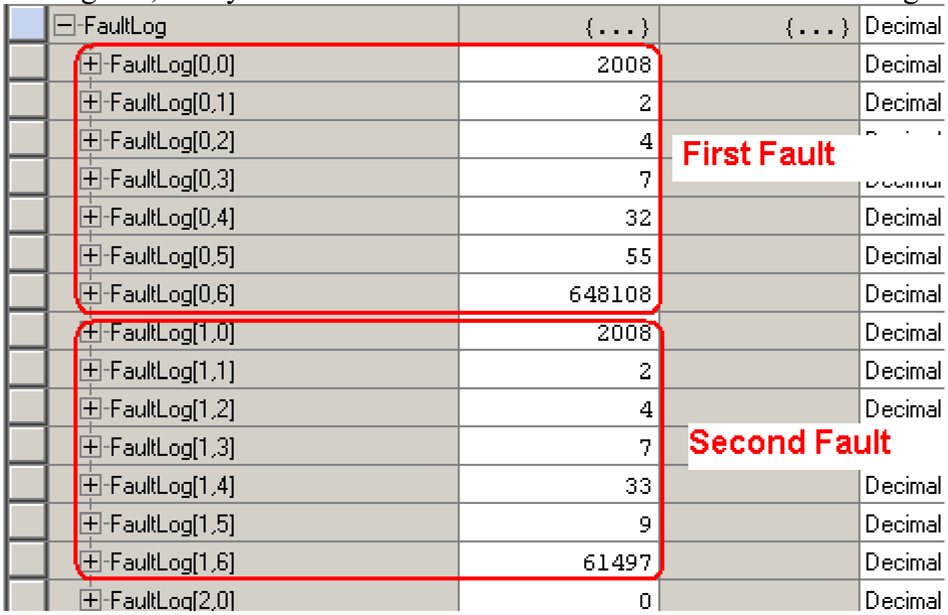


GSV
Get System Value
Class Name wallClockTime
Instance Name
Attribute Name LocalDateTime
Dest SystemTime[0]
0

3) Now, we'll set up the logic as follows. You will need to manually type the destination of the COP instruction because it is an indirect address.
   1. When the switch is energized, the COP instruction will execute.
   2. Since the value of the 'pointer' tag is currently 0, seven elements (the entire system time) will be copied to FaultLog[0,0].
   3. The add statement will then increment the pointer.
   4. The next time a fault occurs, the system time will be written to FaultLog[1,0]...
   5. The process repeats until the log file is full.



4) Now you can download your work, and you will see the logic in operation. Each time the switch is energized, the system time is written to a different area of the faultlog.

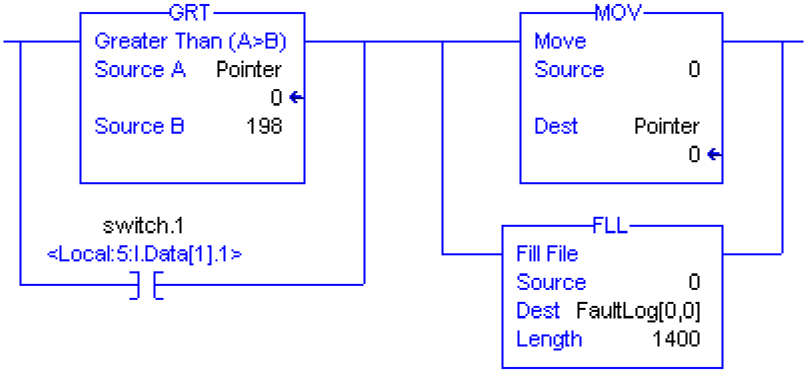


FaultLog	{...}	{...}	Decimal
FaultLog[0,0]	2008		Decimal
FaultLog[0,1]	2		Decimal
FaultLog[0,2]	4		
FaultLog[0,3]	7		
FaultLog[0,4]	32		Decimal
FaultLog[0,5]	55		Decimal
FaultLog[0,6]	648108		Decimal
FaultLog[1,0]	2008		Decimal
FaultLog[1,1]	2		Decimal
FaultLog[1,2]	4		Decimal
FaultLog[1,3]	7		
FaultLog[1,4]	33		Decimal
FaultLog[1,5]	9		Decimal
FaultLog[1,6]	61497		Decimal
FaultLog[2,0]	0		Decimal

First Fault

Second Fault

5) Now the only problem we have is that once 200 faults are reached, we would index past the FaultLog file, and the processor would fault. We need to add some logic to prevent this from happening. In this logic, we'll say that if the pointer goes above 198, reset the pointer to 0, and reset the fault log. The FLL instruction will fill the fault log file with 0's. We'll also give you the ability to reset the fault log with a switch. Add the logic as shown:

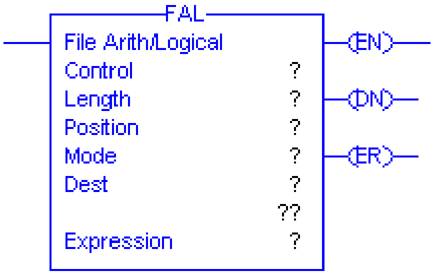


6) The length of the FLL is 1400 elements because our FaultLog file is an array of 200 x 7. Download your work, and verify the fault log can be cleared.

FAL Instruction

The FAL instruction is very versatile. It can copy data from one file to another, copy data from a file to an element, or an element to a file. The main purpose of the FAL instruction, however is to perform math (or manipulate) the data as it is moved across to another file.

Here is what the FAL Instruction looks like in logic:



The **CONTROL** element is just a workspace for the FAL to perform it's job. This workspace keeps track of the status of the FAL instruction.

The **LENGTH** is the number of elements in the file you are operating on.

The **POSITION** indicates the progress of the FAL instruction as it operates on the file. The instruction is considered done (DN bit set) when the position equals the length.

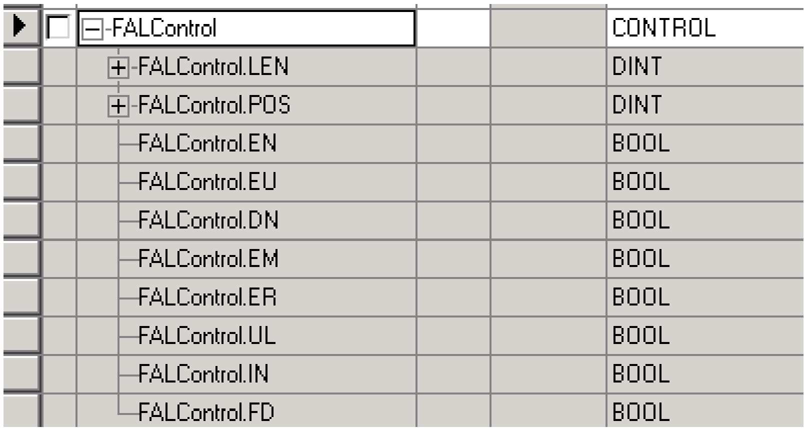
There are three **MODES** of operation:

ALL – All calculations are performed in the current scan.
INCREMENTAL – One element is operated on for each false to true rung transition.
NUMERIC – A certain number of elements are operated on per scan once the rung is true.

The **DESTINATION** is where the data is stored after the expression has been executed. This can be in the form of a single element such as Unit4Temperature, or an array such as UnitTemperature[0] – UnitTemperature[199]

The **EXPRESSION** is the 'formula' used in the calculation.

First, we need to declare a CONTROL element. We are going to declare this in the Controller Tag database, although it could be used as a program tag as well. Recall that the control element stores information about the status and progress of the FAL instruction.



-FALControl	CONTROL
-FALControl.LEN	DINT
-FALControl.POS	DINT
FALControl.EN	BOOL
FALControl.EU	BOOL
FALControl.DN	BOOL
FALControl.EM	BOOL
FALControl.ER	BOOL
FALControl.UL	BOOL
FALControl.IN	BOOL
FALControl.FD	BOOL

You will notice when you expand the FALControl element, there are several sub elements:

EN – This is the ENABLE bit. It is set when the rung goes from false to true.
In INCREMENTAL mode, this bit will follow the rung condition.
In NUMERIC mode, or ALL mode, the EN bit remains set until the instruction is finished.

DN – This is the DONE bit. It is set when the instruction completes all operations. In NUMERIC mode, the DN bit is reset when the instruction is complete if the rung is false.... Otherwise, the EN bit will reset the DN bit when it goes true again.

ER – This is the ERROR bit. It is set if the instruction generates an overflow, and the instruction stops it's operation. If this happens, the ER bit is reset by the ladder logic. The POS will indicate the position of the value (in the file) that caused the overflow condition to occur.

LEN – This is the length of the number of elements you are operating on.

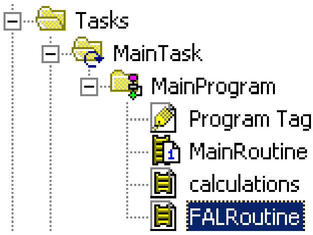
POS – This is the position of the FAL instruction as it operates through the file.

Other bits such as (EM) Empty (EU) Queue (IN) Inhibit (FD) Found, and (UL) Unload are not used for this particular instruction.

Let's make it work

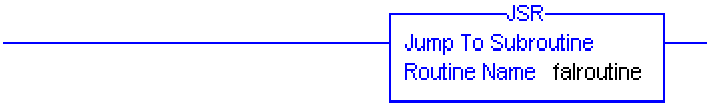
In this example, we are going to take 10 elements from an Array called Celsius, and convert them all from Celsius to Fahrenheit, The result will be stored into an array of 10 elements called Fahrenheit.

1) First, let's create a routine that we are going to use just for FAL calculations (Right click the MainProgram and add a routine called "FALRoutine" as shown:



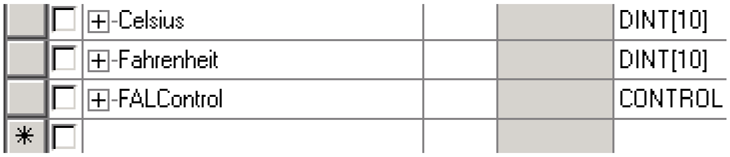
```
Tasks
  MainTask
    MainProgram
      Program Tags
      MainRoutine
      calculations
      FALRoutine
```

2) Next, we need a way for this FALRoutine to execute, so we'll go back to the MainRoutine, and create a JSR statement that will instruction the processor to execute FALRoutine.



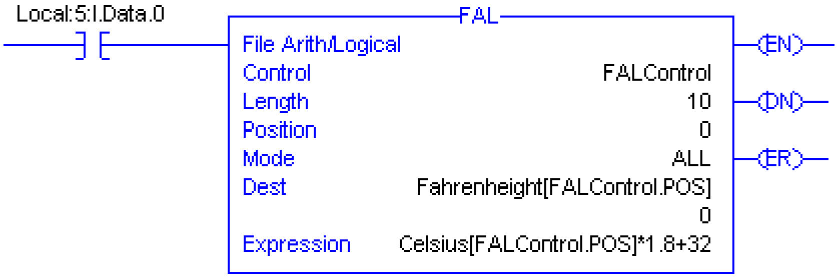
```
JSR
Jump To Subroutine
Routine Name   falroutine
```

3) Now, let's create the tags we need in the controller tag database to make this work.
   1. Celsius as DNT[10]
   2. Fahrenheit as DINT[10]
   3. FALControl as Control



	□	⊞-Celsius			DINT[10]
	□	⊞-Fahrenheit			DINT[10]
	□	⊞-FALControl			CONTROL
✳	□				

4) Now let's add the FAL instruction into the FALRoutine as shown. You will notice we are using Indirect addressing because the FAL instruction does not automatically index the position. We have to use indirect addressing to change our position within the file as the FAL instruction executes.



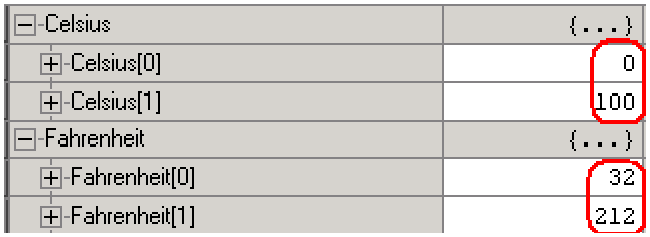
Local:5:I.Data.0

FAL

File Arith/Logical

Control	FALControl
Length	10
Position	0
Mode	ALL
Dest	Fahrenheight[FALControl.POS] 0
Expression	Celsius[FALControl.POS]*1.8+32

5) Now, go back to the controller tag database,and open your Celsius and Fahrenheit Arrays. When you place a value in the Celsius array, it should be converted to Fahrenheit, and stored into the Fahrenheit Array when you throw switch 0 to activate the FAL instruction.



⊟-Celsius	{...}
⊞-Celsius[0]	0
⊞-Celsius[1]	100
⊟-Fahrenheit	{...}
⊞-Fahrenheit[0]	32
⊞-Fahrenheit[1]	212

Working with User Defined Data Types

A User Defined Data type allows the user to create his own data structure offline. To understand user defined Data Types, you must first understand pre-defined data types. In the PLC-5, T4:0 was a variable with the timer data type. T4:0 contained several members: T4:0.ACC, T4:0.PRE, T4:0/DN, T4:0/EN, and T4:0/TT.

In ControlLogix, a variable such as lubedelay can be created, and given the data type of "Timer". Once this assignment is made, the variable "lubedelay" can be expanded to reveal all of it's components: lubedelay.ACC, lubedelay.PRE, lubedelay.DN, lubedelay.TT, and lubedleay.EN. All of these variables are updated by a timer instruction in logic.

The timer object is always on the 'menu' as a pre defined data type, but you don't have any timers by default. You have to set up a variable in the tag database, and assign that variable the timer data type. You would now have a timer.

Instead of using the data structure of a timer, you may want to create your own data structures. For example, you may want to create a data type called 'TankAlarm', and have the alarm data type consist of some members such as HighLevel, and LowLevel.

Now any tag you assign a TankAlarm data type will inherit both the HighLevel and LowLevel members.

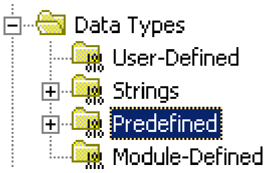
Data structures and data types are not directly usable by the logic. Setting up a User Defined Data type merely puts an item on a menu. You don't actually have an instance of the item until you declare a tag with your data type.

For example: Steak and Shake sells hamburgers (or steak burgers). Seeing the hamburger on a menu does not allow you to utilize the product in any way until you order at least one instance of the menu item. The same is true for the timer example we used earlier. The timer data structure is always available, but you don't have any timers until you create a tag with the timer data type.

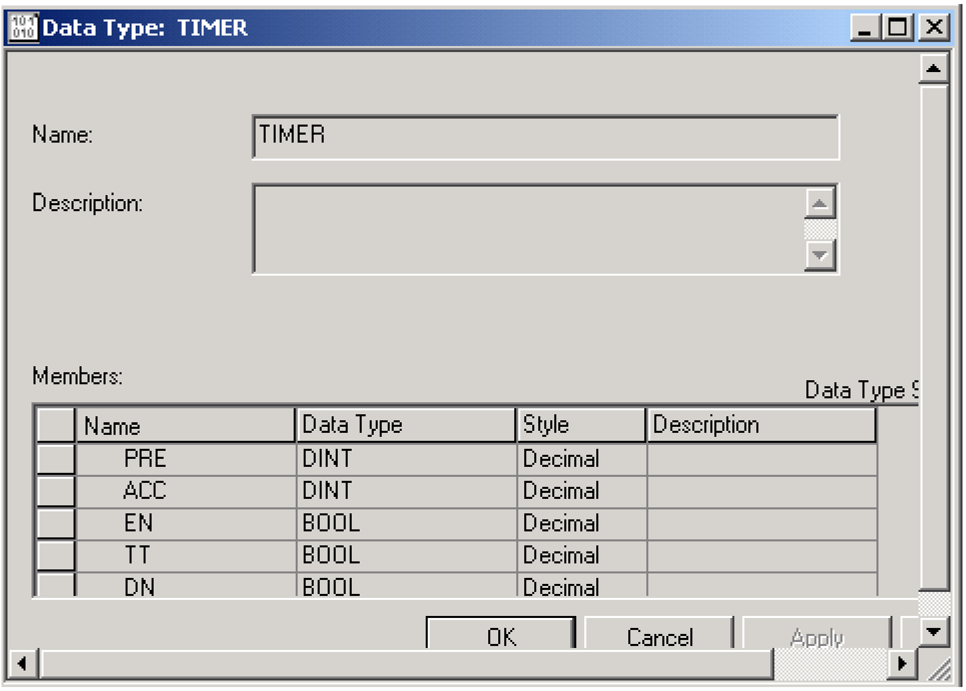
Let's take a look at the timer data structure to see what members it consists of, and then we will create our own data structure for TankAlarms.

Simple UDT's

In an open project, locate the Data Types folder in the controller organizer window. Under the Data Types folder, you will find the Predefined data types. Expand the Predefined data types folder, and double click TIMER.



Notice the members of the Timer Data Structure. Any tag we assign in the tag database as a timer will inherit all of these members.



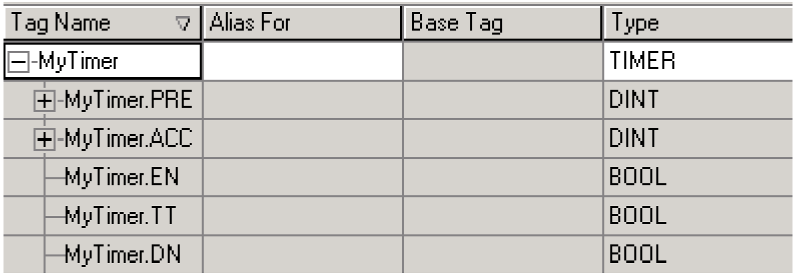
Name: TIMER

Description:

Members:

	Name	Data Type	Style	Description
	PRE	DINT	Decimal	
	ACC	DINT	Decimal	
	EN	BOOL	Decimal	
	TT	BOOL	Decimal	
	DN	BOOL	Decimal	

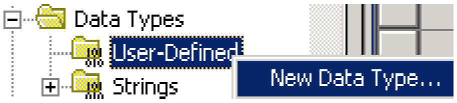
When you create MyTimer with the timer type, the tag can be expanded to reveal all the components.:



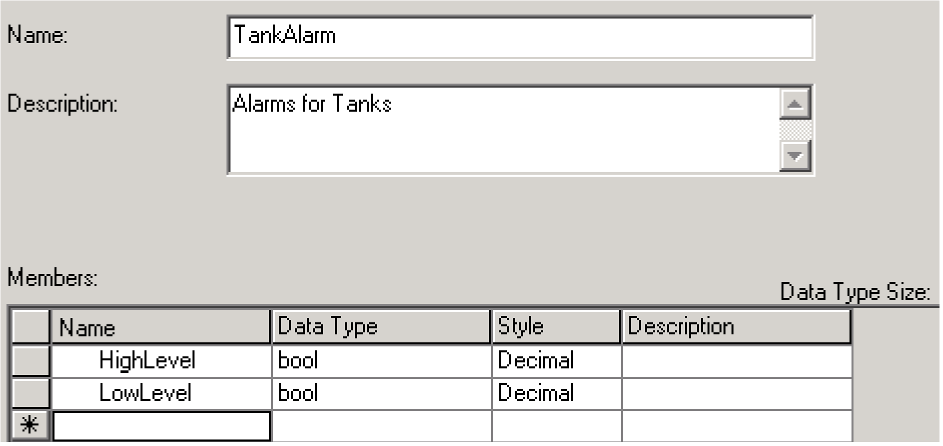
Tag Name	Alias For	Base Tag	Type
⊟-MyTimer			TIMER
⊞-MyTimer.PRE			DINT
⊞-MyTimer.ACC			DINT
MyTimer.EN			BOOL
MyTimer.TT			BOOL
MyTimer.DN			BOOL

Next, we are going to create a User Defined Data structure called "TankAlarm". This data structure will have two members: HighLevel, and LowLevel.

Right Click on the User-Defined folder under Data Types, and select "New Data Type".



The name of this data type will be TankAlarm. The two members, HighLevel, and LowLevel, will each have a bool data type.



Name: TankAlarm

Description: Alarms for Tanks

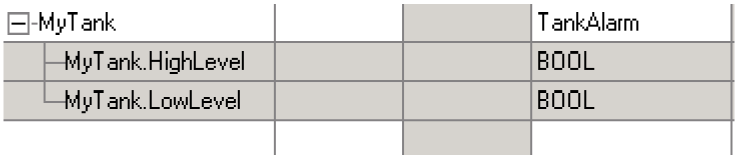
Members:

Data Type Size:

	Name	Data Type	Style	Description
	HighLevel	bool	Decimal	
	LowLevel	bool	Decimal	
✱				

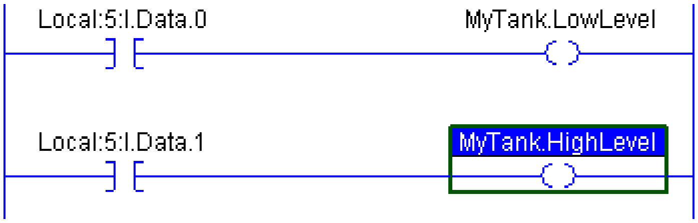
Now that “TankAlarm” is an available data type, we can now create an instance of this data type in the tag database.

In this example, I created a tag called MyTank, and gave it the TankAlarm data type. MyTank inherited the members of the TankAlarm data type. We can now have XIC's and   OTE's in logic that use MyTank.LowLevel, and MyTank.Highlevel variables.



⊟-MyTank			TankAlarm
MyTank.HighLevel			BOOL
MyTank.LowLevel			BOOL

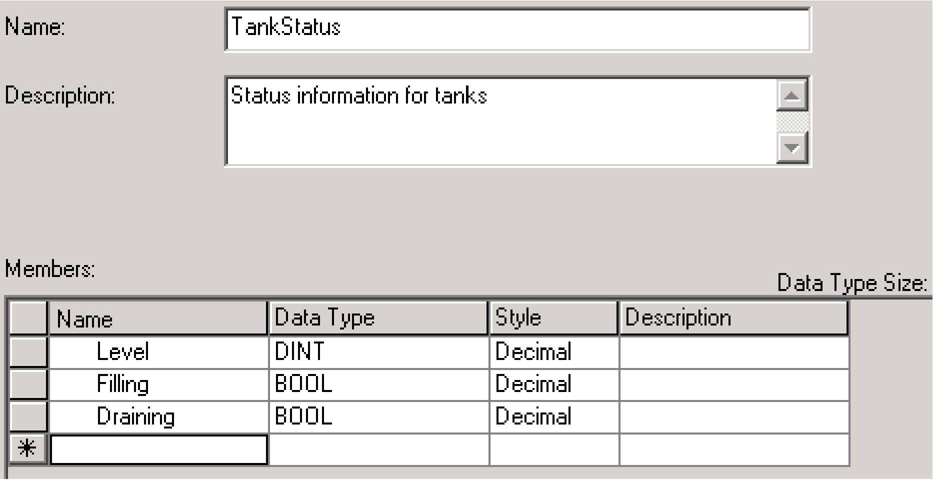
Here is an example of what logic might look like to populate these bits:



Nesting UDT's

Additional information may be needed for MyTank. Status information may also be available. The TankStatus data type will consist of three members: Level, Draining, and Filling. Let's go ahead and set up this data type, then we will nest TankAlarm, and TankStatus members into a tank data type. This will provide us with all the information we need about MyTank all in one area. Lets do this one step at a time.

First, Create a UDT for TankStatus as shown (Right click the User-Defined folder and select 'new data type'):



Name: TankStatus

Description: Status information for tanks

Members:

Data Type Size:

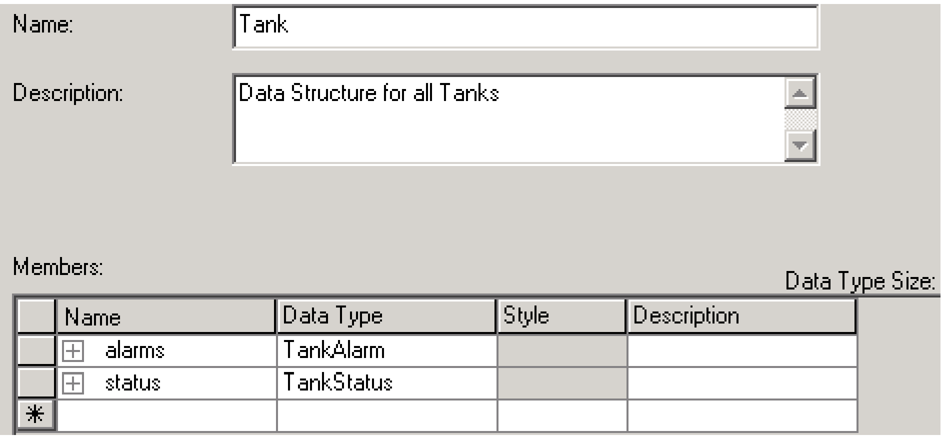
	Name	Data Type	Style	Description
	Level	DINT	Decimal	
	Filling	BOOL	Decimal	
	Draining	BOOL	Decimal	
*				

Notice that Level is an actual value, so it will have to be DINT. Filling and Draining are either true or false, so their type is BOOL. Apply your changes.

We can use the TankStatus Data type in the tag database as it is, but we would have to create a separate tagname, which is not good. Remember our goal is to organize data.

To have all the data under one tagname, we are going to have to create another data structure that has the members TankStatus and TankAlarm. Each of those data structures have their own members.

Create the data structure as shown:



Name: Tank

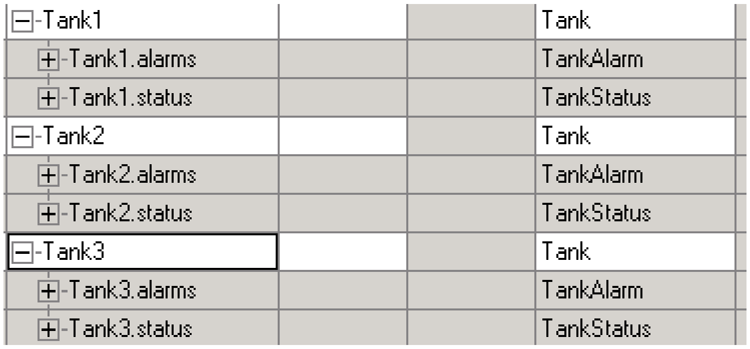
Description: Data Structure for all Tanks

Members:

Data Type Size:

	Name	Data Type	Style	Description
	⊞ alarms	TankAlarm		
	⊞ status	TankStatus		
✱				

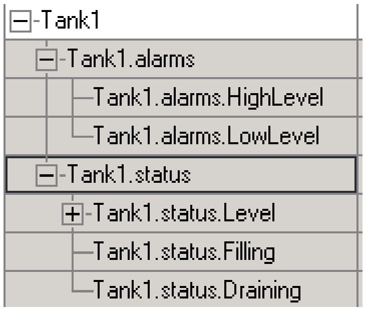
Now go to the Controller Tag Database, and create 3 tags, Tank1, Tank2, and Tank3 each having the data type of 'Tank'. Look what happens!



⊟-Tank1			Tank
⊞-Tank1.alarms			TankAlarm
⊞-Tank1.status			TankStatus
⊟-Tank2			Tank
⊞-Tank2.alarms			TankAlarm
⊞-Tank2.status			TankStatus
⊟-Tank3			Tank
⊞-Tank3.alarms			TankAlarm
⊞-Tank3.status			TankStatus

All three Tanks had the 'Tank' data type. (Recall that the tank data type consisted of two members, alarm and status.

Remember also that alarm and status had their own structures as well. Expand the alarm and status tags.



```
⊟-Tank1
  ⊟-Tank1.alarms
     ─Tank1.alarms.HighLevel
     └Tank1.alarms.LowLevel
  ⊟-Tank1.status
     ⊞-Tank1.status.Level
     ─Tank1.status.Filling
     └Tank1.status.Draining
```

All the data for each Tank is now well organized. These tags should then be incorporated into the input and output data mapping routines.

Producer/Consumer Model

In older PLC systems, such as the PLC-5, a message instruction had to be executed in order to get data from one processor to another. Although this worked very well, there was seldom a guarantee when data would arrive at it's destination, nor would data be transferred if the processor was in program mode.

With the Producer/Consumer model, the transfer of data between processors takes place without any logic, and the user sets the RPI (Requested Packet Interval) for the rate at which data should be updated.

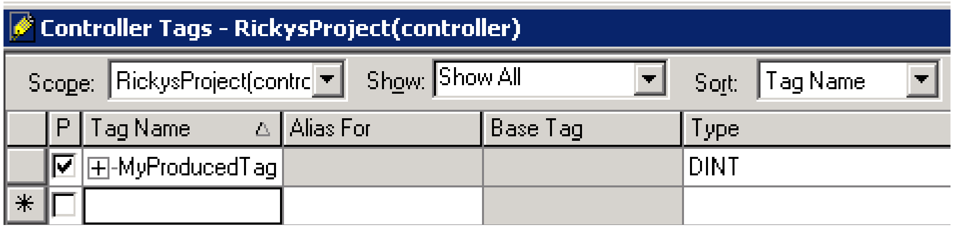
The producer consumer model is very easy to understand, and to trace where data is coming from.

The Producer/Consumer model can transfer data between tags which have a DINT data type, an array, or a User-Defined Data Type (UDT's)

The Produced Tag

The producer has the easy job. In the Controller Tag Database, a user simply creates a tag, and marks it as produced. In this case, MyProducedTag is a DINT. The 'P' in the left-most column makes the tag produced.

When we mark a tag as produced, we are simply allowing the tag to be served to consumers. We can populate this produced tag through logic, and the consumed tags in other processors will receive the data we place into the produced tag.

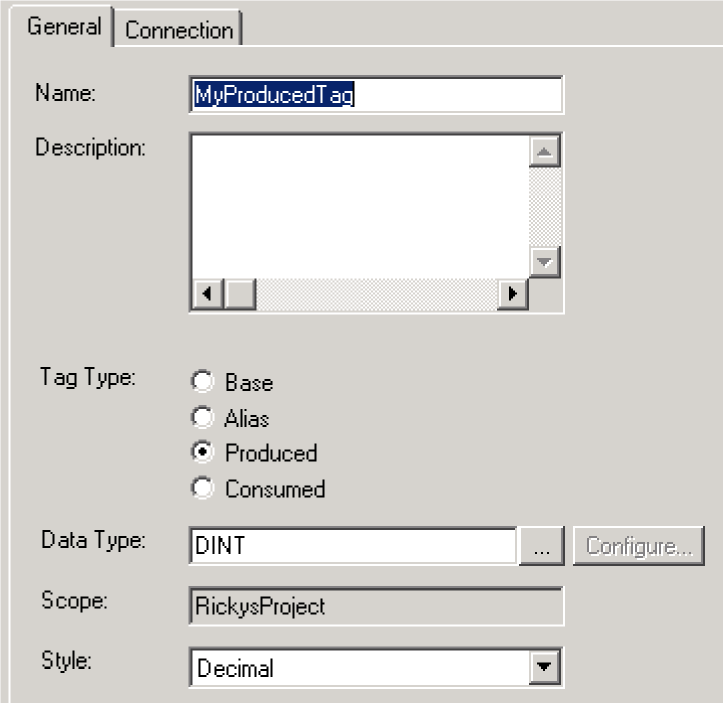


Controller Tags - RickysProject(controller)

Scope: RickysProject(contrc Show: Show All Sort: Tag Name

	P	Tag Name △	Alias For	Base Tag	Type
	☑	⊞-MyProducedTag			DINT
*	□				

If you were to right click on the tag, and go to the tag's properties, we would have a few more options:



Name: MyProducedTag

Description:

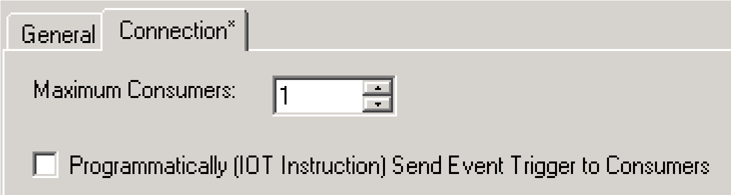
Tag Type: ○ Base ○ Alias ● Produced ○ Consumed

Data Type: DINT

Scope: RickysProject

Style: Decimal

If we click on the 'Connection' tab, we can set the maximum number of consumers which are allowed to connect to the produced tag. The allowed values are 1 to 256. You can also send event triggers to consumers using the IOT instruction in logic.



Maximum Consumers: 1

□ Programmatically (IOT Instruction) Send Event Trigger to Consumers

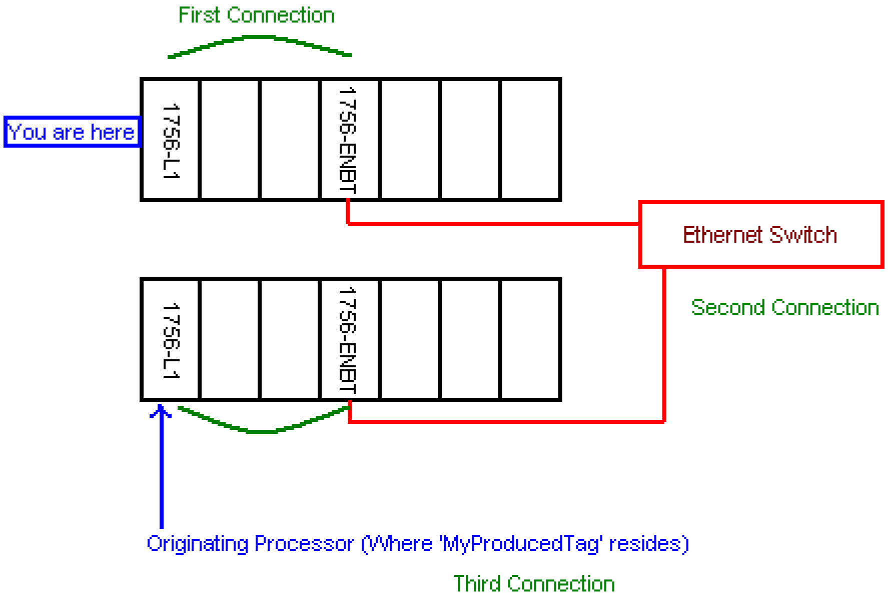
The Consumed Tag

In the previous few pages, we discussed how to set up a tag in the **producing** processor. Now we will go to the program which resides in the processor which will be **consuming** data from this produced tag.

Setting up a consumed tag involves an extra step. First, we build a path to the processor where the tag is being produced. This is done in I/O Configuration. Then the consumed tag can be created.

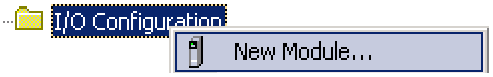
Step 1: Building the path to the producer.

In the I/O configuration tree, we will have to build a path to the processor where the produced tag resides. First, we will have to establish a connection to our local Ethernet or ControlNet module. Next, we will have our local communication module connect to a communication module in the chassis where the produced tag resides. Last, we must tell the communication module in the remote chassis to connect to the processor which is producing data. Look at the diagram below:

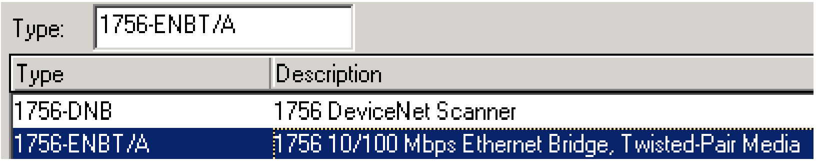


This connection is made under I/O Configuration.

First, right click on the I/O Configuration folder to connect to the local communication module if it not already set up.



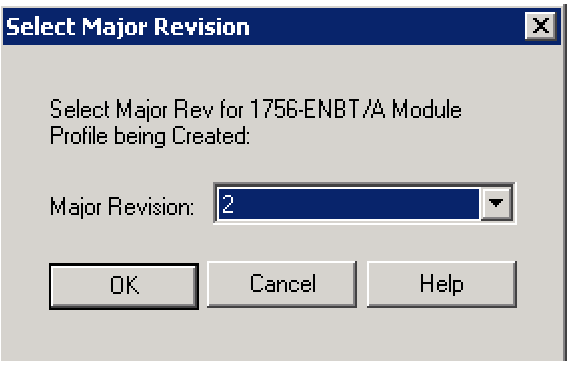
Be sure to select the communication module you will be using For this example, we are using a 1756-ENBT module.



Type: 1756-ENBT/A

Type	Description
1756-DNB	1756 DeviceNet Scanner
1756-ENBT/A	1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media

You can get the major revision from a web browser, RSLinx, or from the label on the side of the module if the label is correct. Our module is version 2.4 (2 is major, 4 is minor).

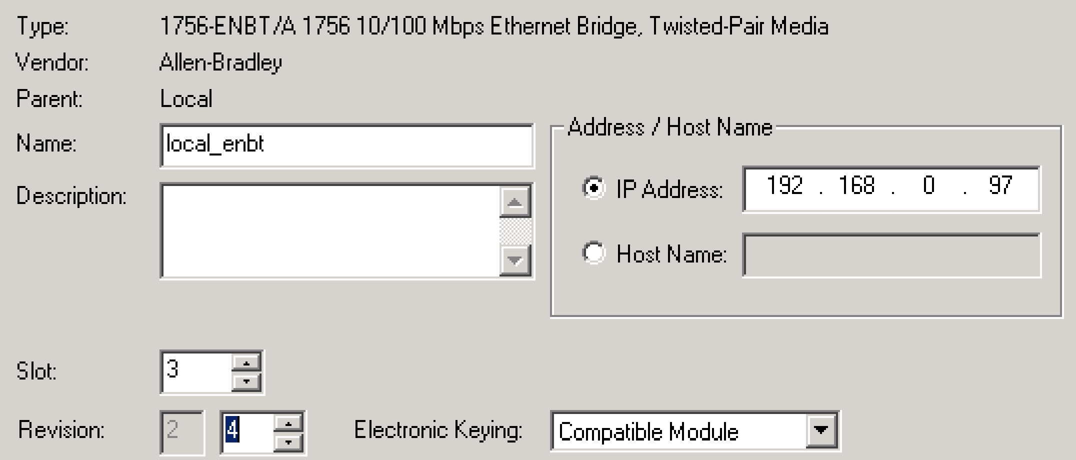


Select Major Revision

Select Major Rev for 1756-ENBT/A Module Profile being Created:

Major Revision: 2

We named this module local_enbt. This module resides in slot 3, and the IP address is 192.168.0.97. The minor revision was 4. (Adjust these settings for what you are using at your own station)

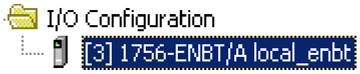


Type: 1756-ENBT/A 1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media
Vendor: Allen-Bradley
Parent: Local
Name: local_enbt
Description:

Address / Host Name
● IP Address: 192 . 168 . 0 . 97
○ Host Name:

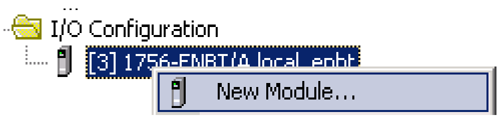
Slot: 3
Revision: 2 4
Electronic Keying: Compatible Module

When finished with this step, your local communication module will appear in the I/O configuration similar to the image shown below:



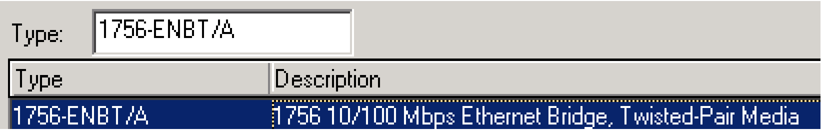
I/O Configuration
[3] 1756-ENBT/A local_enbt

Second, we must add the communication module of the remote chassis to the I/O Configuration. We must right click on our local communication module, and tell our local communication module to connect to the remote communication module as shown:



I/O Configuration
[3] 1756-ENBT/A local_enbt
New Module...

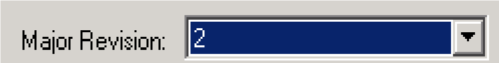
We are connecting to another 1756-ENBT module.



Type: 1756-ENBT/A

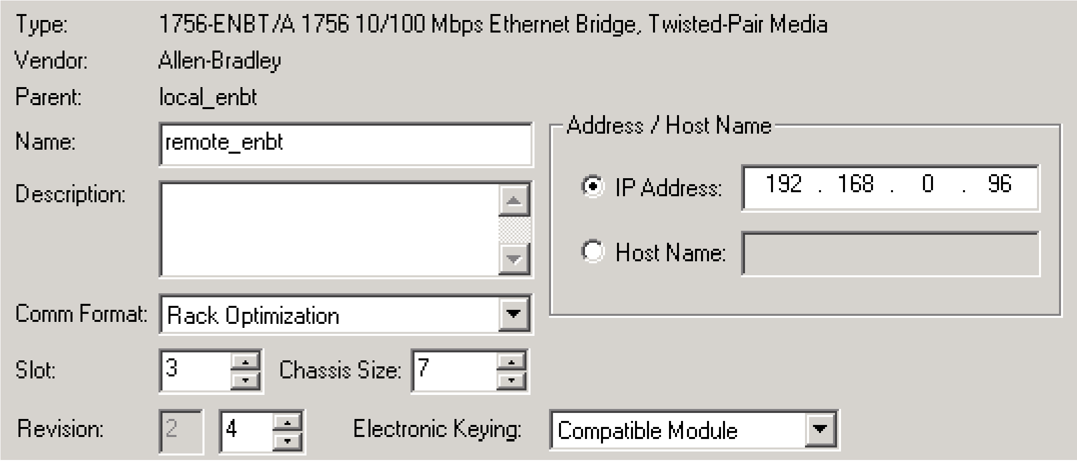
Type	Description
1756-ENBT/A	1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media

Again, the remote ENBT module is version 2.4, so the Major Revision is 2.



Major Revision: 2

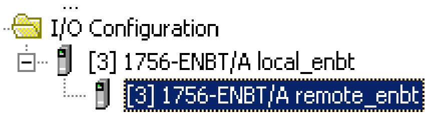
We will name the module remote_enbt. It has the IP address of 192.168.0.96. It is in slot 3 of a 7 slot chassis. We choose rack optimization for the comm format, so data from all modules can be received through a single connection. The minor revision is 4. Adjust your settings as necessary for your station.



Type: 1756-ENBT/A 1756 10/100 Mbps Ethernet Bridge, Twisted-Pair Media
Vendor: Allen-Bradley
Parent: local_enbt
Name: remote_enbt
Description:
Comm Format: Rack Optimization
Slot: 3 Chassis Size: 7
Revision: 2 4 Electronic Keying: Compatible Module

Address / Host Name
● IP Address: 192 . 168 . 0 . 96
○ Host Name:

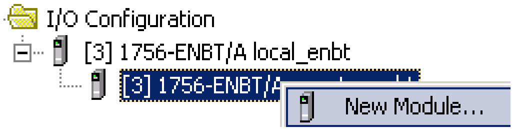
When finished adding the remote enbt module, your I/O configuration will look similar to the image below:



I/O Configuration
[3] 1756-ENBT/A local_enbt
[3] 1756-ENBT/A remote_enbt

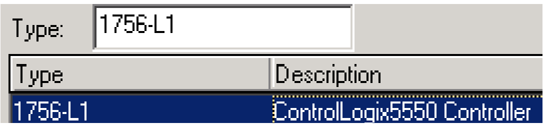
Third, we must make a connection to the processor from the remote_enbt module.

Right click the remote enbt module, and we will add a new module as we did before.



I/O Configuration
[3] 1756-ENBT/A local_enbt
[3] 1756-ENBT/A

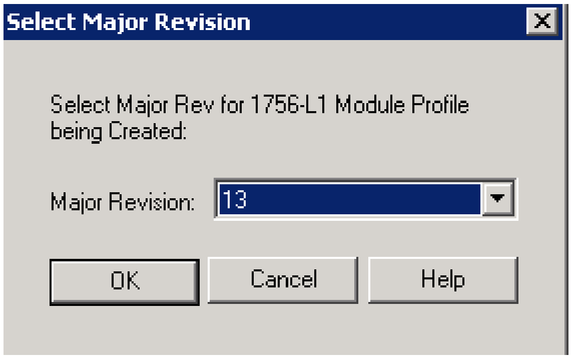
Most of the stations have the 1756-L1 processor. Be sure to select the type of processor that resides in the station which is producing the data.



Type: 1756-L1

Type	Description
1756-L1	ControlLogix5550 Controller

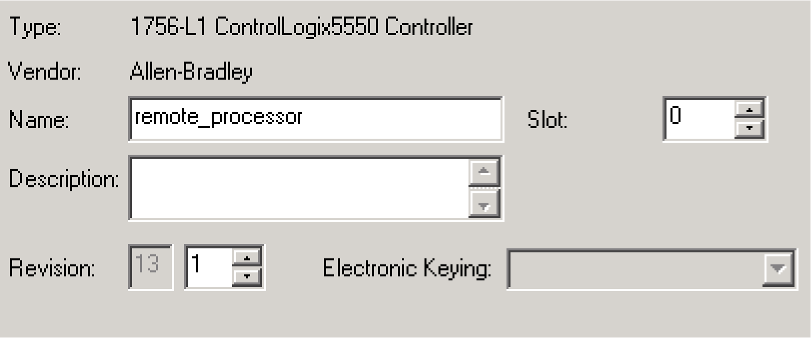
Select the revision level of the processor you are connecting to. The version was 13 when this document was created.



Select Major Rev for 1756-L1 Module Profile being Created:

Major Revision: 13

The processor is named remote_processor and resides in slot 0.


Type: 1756-L1 ControlLogix5550 Controller

Vendor: Allen-Bradley

Name: remote_processor

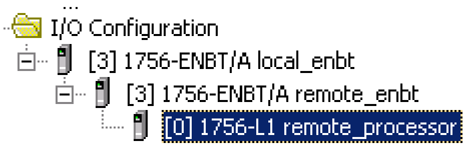
Slot: 0

Description:

Revision: 13 1

Electronic Keying:

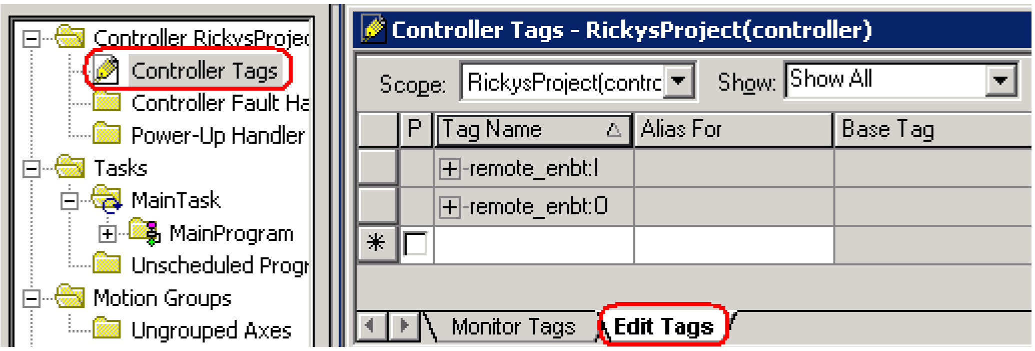
When finished, your I/O Configuration will appear similar to what is shown below:


```
I/O Configuration
  [3] 1756-ENBT/A local_enbt
    [3] 1756-ENBT/A remote_enbt
      [0] 1756-L1 remote_processor
```

Step 2: Creating the consumed tag.

Now that we have a path to the processor which is producing the tag, we can create our consumed tag.

First, go to the controller tag database, and click 'Edit Tags'.

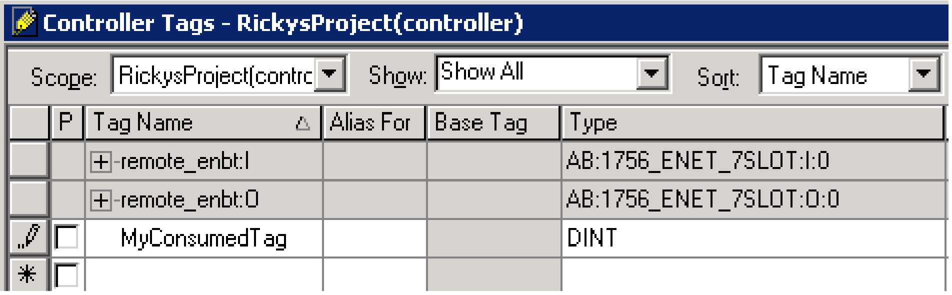


Controller Tags - RickysProject(controller)

Scope: RickysProject(contrc Show: Show All

	P	Tag Name	Alias For	Base Tag
		⊞-remote_enbt:I		
		⊞-remote_enbt:O		
✳	□			

Second, we will create a tag called 'MyConsumedTag'. Leave it as a DINT.

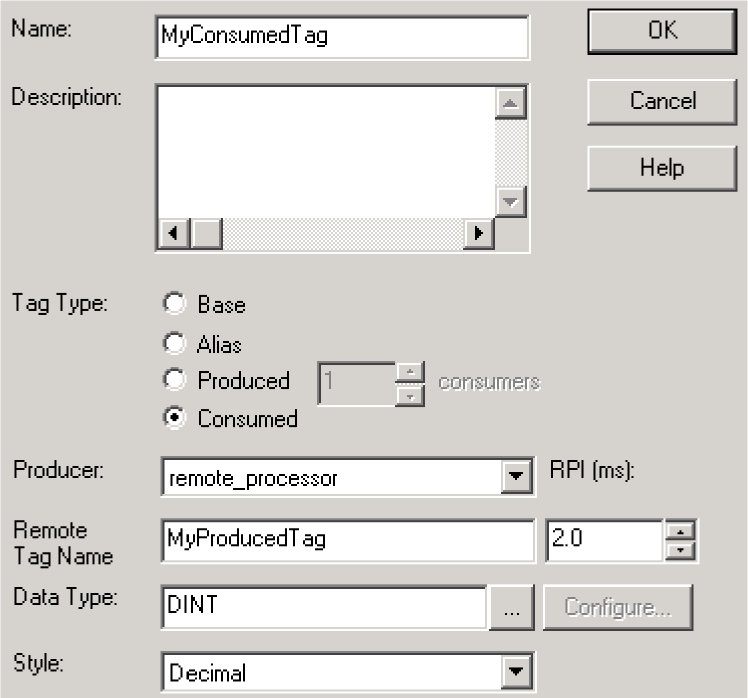


Controller Tags - RickysProject(controller)

Scope: RickysProject(contrc Show: Show All Sort: Tag Name

	P	Tag Name	Alias For	Base Tag	Type
		⊞-remote_enbt:I			AB:1756_ENET_7SLOT:I:0
		⊞-remote_enbt:O			AB:1756_ENET_7SLOT:O:0
✎	□	MyConsumedTag			DINT
✳	□				

Third, right click on the tag, and choose 'edit tag properties'. The tag type is consumed. From the Pull-Down tab you can select remote_processor as the producer. (This is the processor we connected to in I/O Configuration. The name of the tag in the remote_processor is called MyProducedTag, and is a DINT. The RPI is fine for this exercise, but if you wish to conserve bandwidth on the network, the RPI can be increased so the tag will update less often.



Name: MyConsumedTag

Description:

Tag Type: ○ Base ○ Alias ○ Produced 1 consumers ● Consumed

Producer: remote_processor

RPI (ms): 2.0

Remote Tag Name: MyProducedTag

Data Type: DINT

Style: Decimal

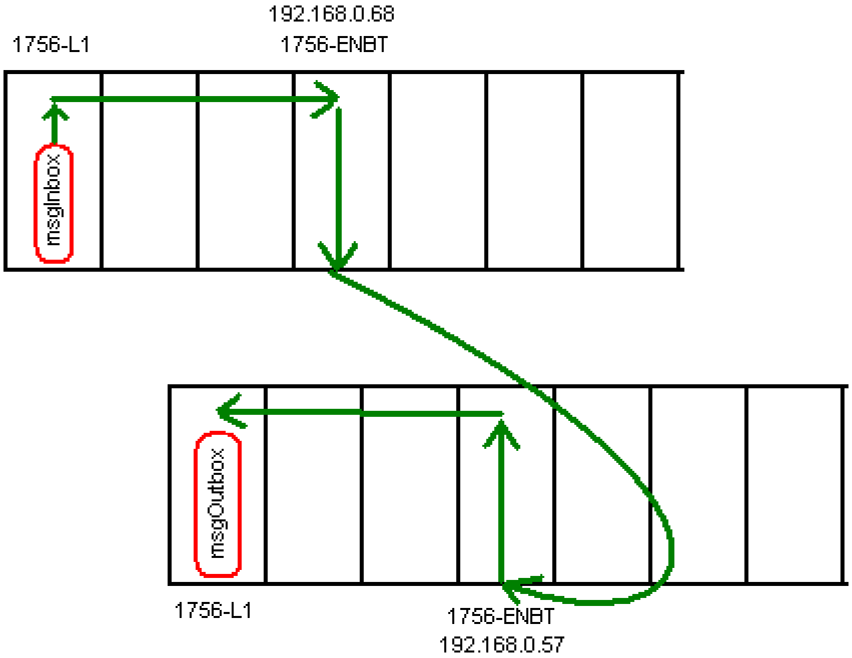
You are ready to download and test your work.

ControlLogix Messaging

The producer consumer model is very efficient for transferring data between processors, but if the data transfer does not need to occur at periodic intervals, you may be able to conserve network bandwidth using the message instruction. Using the message instruction, data can even be received (or sent) from another processor, even if that processor is not present in the I/O Configuration tree.

For this example, we will set up a message instruction that will read data from another ControlLogix processor, and store that data in a memory location in our own controller.

Below, you can see the path that we will take to connect to the target processor. Once, the connection is made, **msgInbox** will then receive data from **msgOutbox** each time the message instruction is executed. Data actually flows in the opposite direction of the communication path for a data read because the connection has to be made first, then we can begin receiving data from the target.



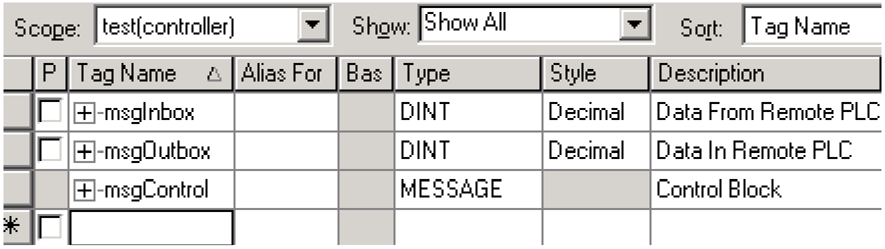
1) First let's set up the memory locations we will be using for this transfer. We will create three new items in the controller tag database:
   1. msgOutbox as DINT
   2. msgInbox as DINT
   3. msgControl as message

msgOutbox will be the memory location that another controller can read from. This tag has nothing to do with our own message instruction, but we will populate this tag with data, so the station reading from your controller can test their connection. You will be reading the msgOutbox tag of another station later in this exercise.

msgInbox is where the data will be stored within our own controller. After the message instruction executes, whatever data was stored in the msgOutbox of the target processor will appear in the msgInbox tag of your own processor.

msgControl is simply the workspace for the message instruction to be able to operate. It stores information about the message instruction such as when it is enabled, waiting, or done.

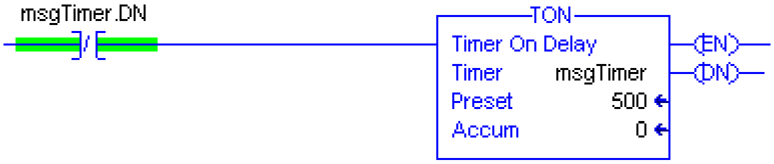
2) There are several ways we can add these tags. You can right click 'Controller Tags' at the top of the controller organizer window, and add a new tag, or you can open the controller tag database, and add the tags to the bottom of the spreadsheet. Note: You must be in 'Edit Tags' mode to add tags to the controller tag database. This can be done either on line or offline. Add these three tags to the controller tag database as shown:



Scope: test(controller) Show: Show All Sort: Tag Name

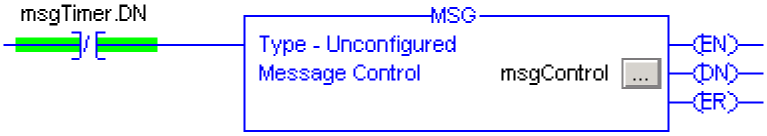
	P	Tag Name △	Alias For	Bas	Type	Style	Description
	□	⊞-msgInbox			DINT	Decimal	Data From Remote PLC
	□	⊞-msgOutbox			DINT	Decimal	Data In Remote PLC
		⊞-msgControl			MESSAGE		Control Block
*	□						

3) Next, add a self-running timer as shown. This timer will be used to trigger the message instruction. You will have to declare msgTimer as Timer. You can do this by creating another entry in the tag database, or right-click msgTimer once you've typed it into the timer instruction, and then select 'new msgTimer' to declare the new variable for use in logic. This procedure will make the entry in the tag database for you.



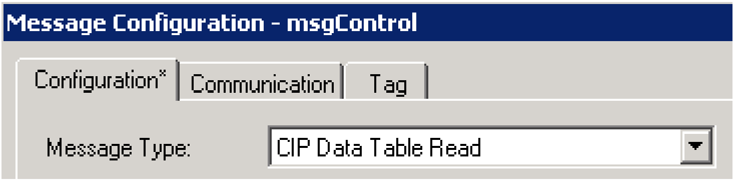
Notice the preset on the timer is 500. Because the time base for ControlLogix is milliseconds only, the preset for this timer is half of one second. We'll be using the DN bit from this timer to trigger the message instruction.

4) Next, enter the following rung of logic for the message instruction. All variables in this rung have already been declared.



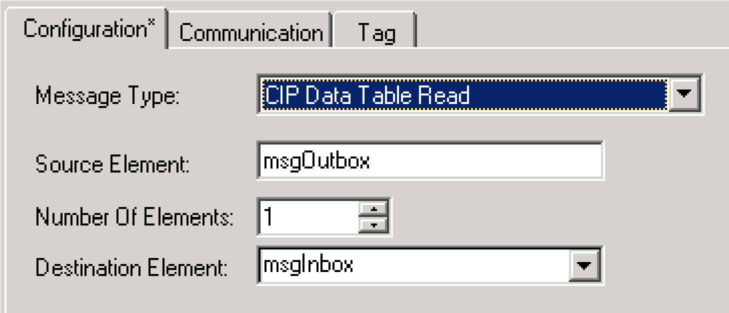
5) Click the ellipsis to configure the message instruction. The ellipsis is the 3 dots next to the control element

6) The message instruction is very versatile, and there are a lot of options. The message instruction can be used to initiate block transfers, read or write module status, and even reset an electronic fuse on an output module. For this exercise, we are just interested in a CIP data table read. CIP (Control and Information Protocol) is the means by which ControlLogix processors natively exchange data through the message instruction.



Message Type: CIP Data Table Read

7) Next, you are asked for the source and the destination element. The source is the name of the tag from which we are getting data, and the destination is the memory location in which this data is to be placed. Recall that we are getting data from the msgOutbox tag of the remote processor, and we will be placing that data into our own msgInbox tag. The length will be just one element.



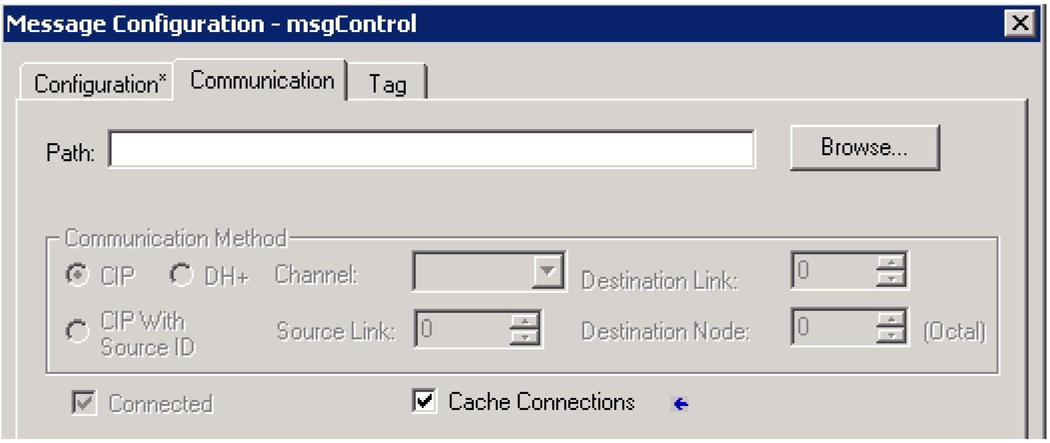
Message Type: CIP Data Table Read

Source Element: msgOutbox

Number Of Elements: 1

Destination Element: msgInbox

8) We have told the processor what the source and destination tags are, however, this PLC could reside on a network of many ControlLogix systems. We have not yet specified what path the controller needs to take to connect to the msgOutbox tag. Go to the 'communication' tab, and we'll discuss how to configure the communication path.



Path:

Communication Method

● CIP ○ DH+ Channel:

Destination Link: 0

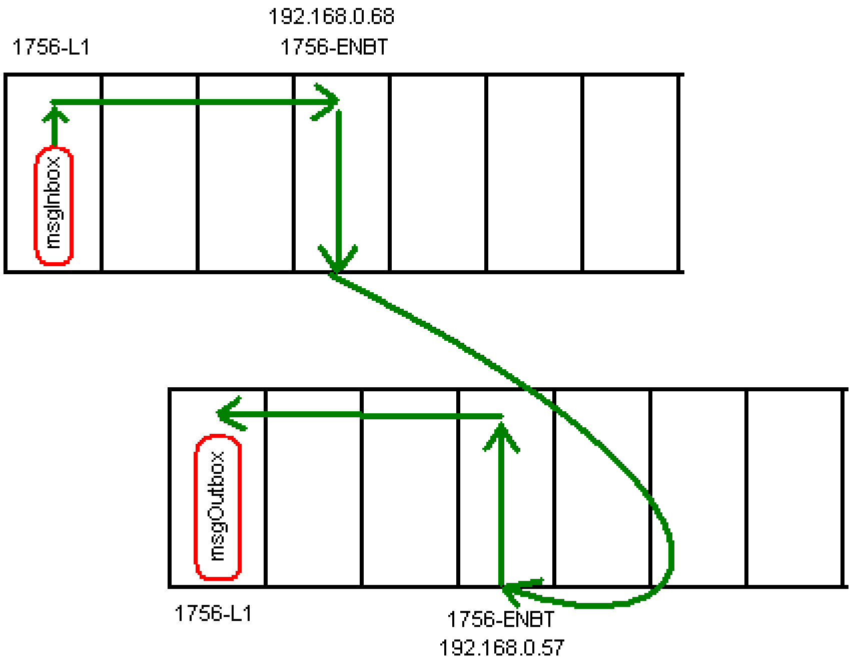
○ CIP With Source ID

Source Link: 0

Destination Node: 0 (Octal)

☑ Connected

☑ Cache Connections



9) Look again at our communication path:

From our PLC (Top left), we must first connect to the backplane. The next step would be to connect to our own Ethernet module (or ControlNet if you have a ControlNet connection). Next, we will specify what our exit port is (we will come out the front port of the Ethernet module. Next, we specify which IP address to connect to on the Ethernet network (or ControlNet node), then we go to the backplane, and then connect to the processor in slot 0. The chart below will help you to determine what the connection path will be **(alternate between step A and B)**:

Step A: **To get to:**	 **Specify:**	***Step B:*** **For module on:**	 **Specify:**
The Backplane	1	The Backplane	Slot #
DF1 Port (Of controller)	2	DF1 Network	DF1 Address
ControlNet Port of CNB	2	ControlNet Network	Node #
Ethernet port (of ENBT)	2	DH+ Network	Node #
DH+ port CH A (DHRIO)	2	Ethernet Network	IP Address
DH+ port CH B (DHRIO)	3		

Note for step B you can also use the following formats: address:port, DNS name, or DNS name:port)

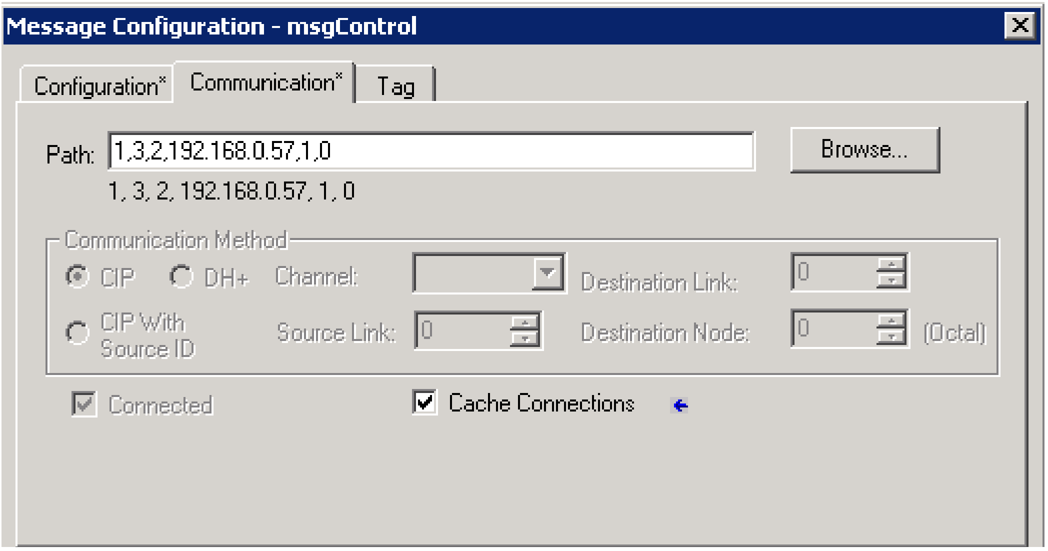
Using this chart you will come up with the path **1,3,2,192.168.0.57,1,0**

10)How did we get this path?

1. From the processor you are programming, you must first choose '**1**' to get to the backplane in step A.
2. Step B tells us to specify the slot number we need to go to next. This would be the slot # for the 1756-ENBT module. (Slot **3**)
3. Go back to Step A. Since we are now on-board the ENBT module, you must specify a **2** to get out the front port of the module.
4. Back to Step B... Since we came out of port 2, we are now on the Ethernet network, so we must specify which address to go to. For this example, the address was **192.168.0.57**.
5. Go back to Step A. Now that we are on-board the Ethernet module, specify **1** to get to the backplane.
6. Back on Step B we specify which slot to connect to. That would be the processor in Slot **0**.

Note: If the target processor was already in your I/O Configuration you could just browse to the processor.

11)You path will appear as shown



Message Configuration - msgControl

Path: 1,3,2,192.168.0.57,1,0

1, 3, 2, 192.168.0.57, 1, 0

Communication Method

● CIP ○ DH+ Channel: Destination Link: 0

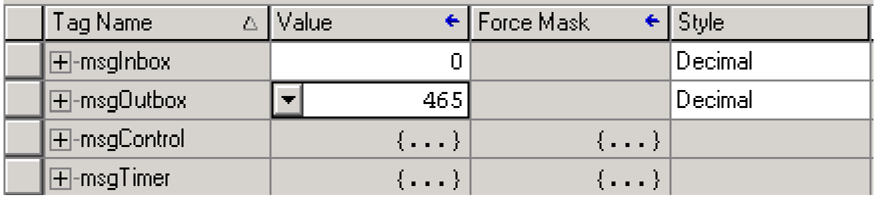
○ CIP With Source ID Source Link: 0 Destination Node: 0 (Octal)

☑ Connected ☑ Cache Connections

12)Apply your changes then press OK.

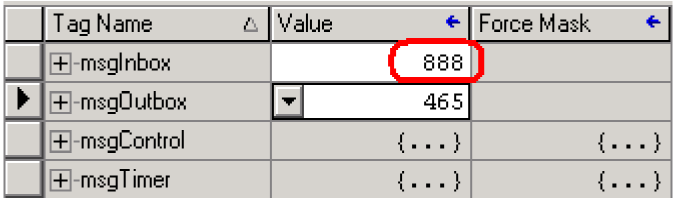
13)Download to your processor, and put the processor into 'run' or 'remote run' mode.

14)In the Controller Tag database, populate the msgOutbox tag with a value. You must be in 'Monitor Tags' to inject a value, then press enter.



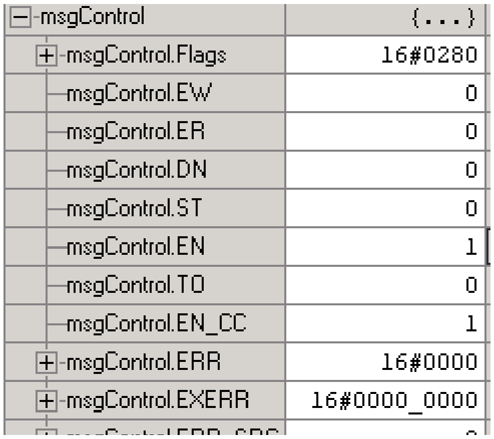
Tag Name	Value	Force Mask	Style
+-msgInbox	0		Decimal
+-msgOutbox	465		Decimal
+-msgControl	{...}	{...}	
+-msgTimer	{...}	{...}	

15)If your MSG instruction is working properly, and if the msgOutbox tag is set up properly in the remote processor, you should receive this value in your msgInbox.



Tag Name	Value	Force Mask
+-msgInbox	888	
+-msgOutbox	465	
+-msgControl	{...}	{...}
+-msgTimer	{...}	{...}

16)Open the control element for the message instruction, and you will see the the status bits change as the MSG instruction executes. For a more detailed explanation of these bits, and error codes you might find in this control element, refer to the help file in RSLogix 5000. *(Not all status words are shown below)*



−-msgControl	{...}
+-msgControl.Flags	16#0280
msgControl.EW	0
msgControl.ER	0
msgControl.DN	0
msgControl.ST	0
msgControl.EN	1
msgControl.TO	0
msgControl.EN_CC	1
+-msgControl.ERR	16#0000
+-msgControl.EXERR	16#0000_0000

17) If you was using this for an application, you would write logic next to populate the msgOubox tag with information you need another controller to receive *(Using your own tag names)*

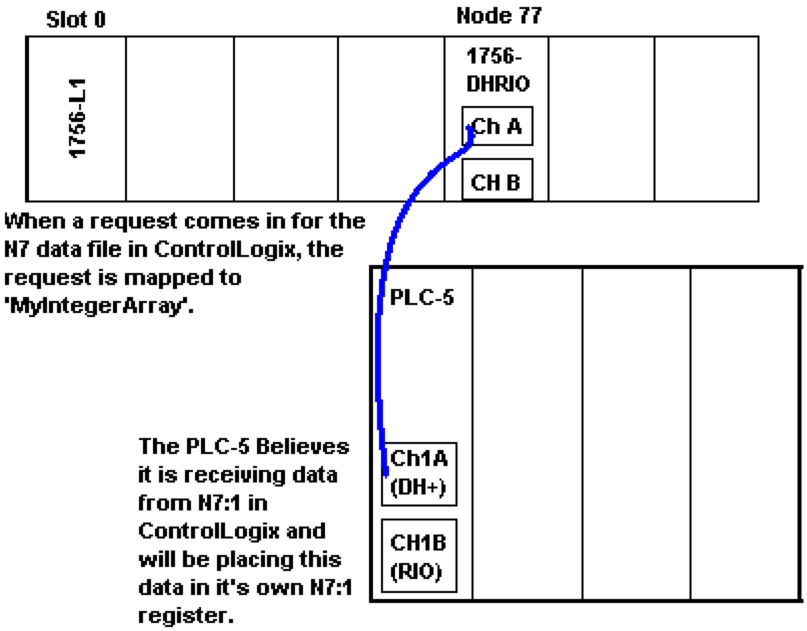
Initiating a message instruction from the PLC to read a value from a ControlLogix processor.

In this example, the PLC-5 will initiate a message read instruction to read a value from the ControlLogix processor. There are several steps involved in this process:

A: The PLC initiates a message read instruction to read a value from what it believes to be another PLC-5 processor. The PLC-5 does not have the ability to directly read a controller tag in the ControlLogix processor.

B: The 1756-DHRIO module must be configured to map any messages received to slot 0 (for this example) because that is where the processor resides.

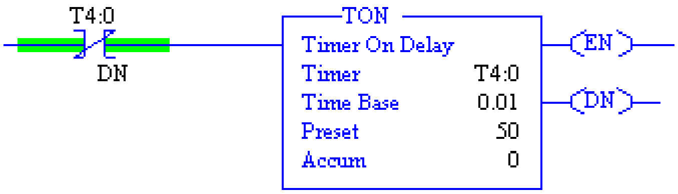
C: Since the ControlLogix processor does not support the PLC-5 data table structure, it must map all messages from a legacy PLC/SLC file number to an array in it's controller tag database.



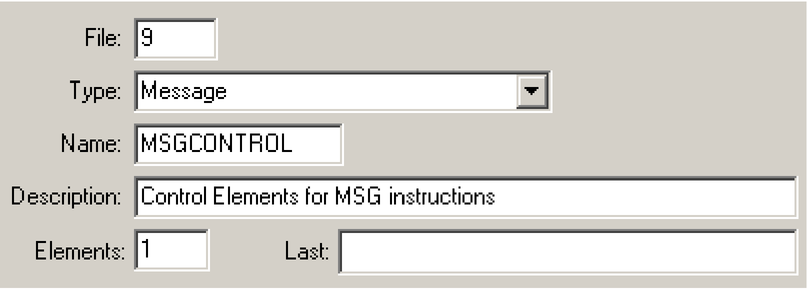
When a request comes in for the N7 data file in ControlLogix, the request is mapped to 'MyIntegerArray'.

The PLC-5 Believes it is receiving data from N7:1 in ControlLogix and will be placing this data in it's own N7:1 register.

1) In the PLC-5, create a self-running timer.



2) Create a Control file for the message instruction to operate. Do do this, Right click on the data file folder and create a new data file as shown.



File: 9

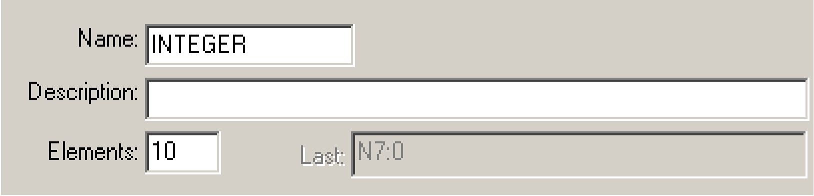
Type: Message

Name: MSGCONTROL

Description: Control Elements for MSG instructions

Elements: 1 Last:

3) Next, Right click on the N7 Data file, and go to the file properties. We are going to expand the data table so we have a place for the message instruction to store data. Give yourself 10 elements. This should be plenty of space for future use in our classroom.

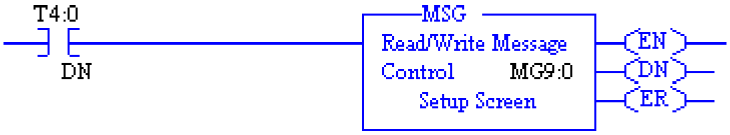


Name: INTEGER

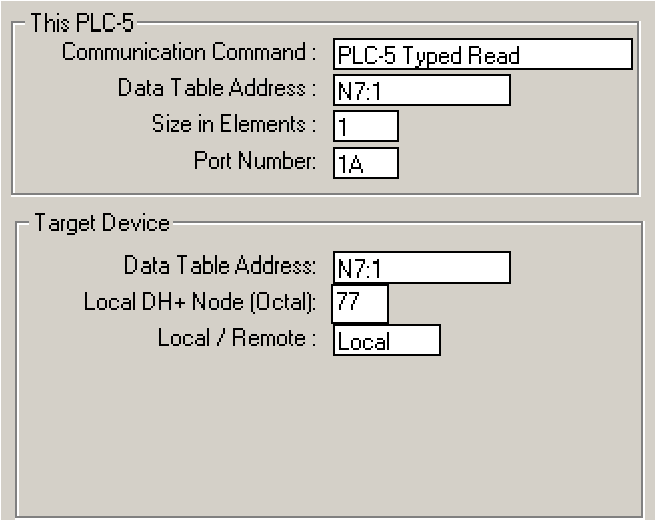
Description:

Elements: 10 Last: N7:0

4) Now, write the message instruction as shown. Then the Setup Screen will appear



5) Next configure the message instruction similar to what is shown below:



This PLC-5

Communication Command : PLC-5 Typed Read

Data Table Address : N7:1

Size in Elements : 1

Port Number: 1A

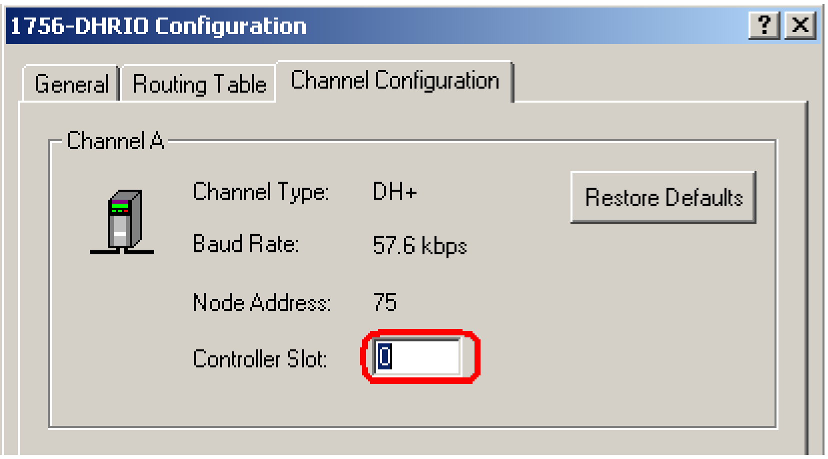
Target Device

Data Table Address: N7:1

Local DH+ Node (Octal): 77

Local / Remote : Local

6) In the RSWho Screen, locate the 1756-DHRIO module. Right click on the module, and go to 'Module Configuration'. Be sure the processor is set to Slot 0.



Channel A

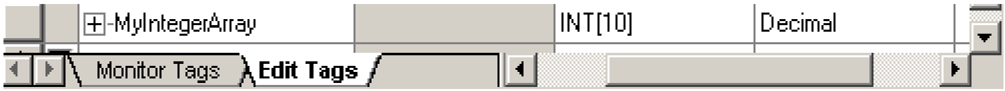
Channel Type: DH+

Baud Rate: 57.6 kbps

Node Address: 75

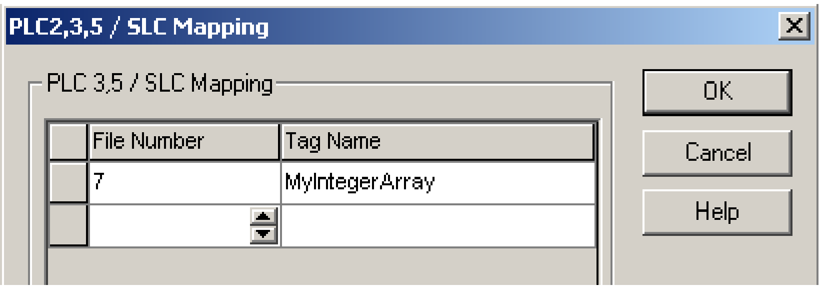
Controller Slot: 0

7) In the ControlLogix processor, we must create an array that simulates the N7 integer file in a PLC5. Go to the Controller tag database, and add a tag as shown.



MyIntegerArray	INT[10]	Decimal

8) .Next, this array must be mapped to file 7. When the ControlLogix receives a request for file 7, it will then know to go to MyInteger array to get the data. To set up this mapping, click Logic on the menu bar in RSLogix 5000, then choose 'Map PLC/SLC Messages'. Set up the mapping as shown, then download your work and ensure the logic is working properly.



PLC2,3,5 / SLC Mapping

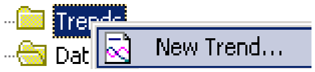
PLC 3,5 / SLC Mapping

File Number	Tag Name
7	MyIntegerArray

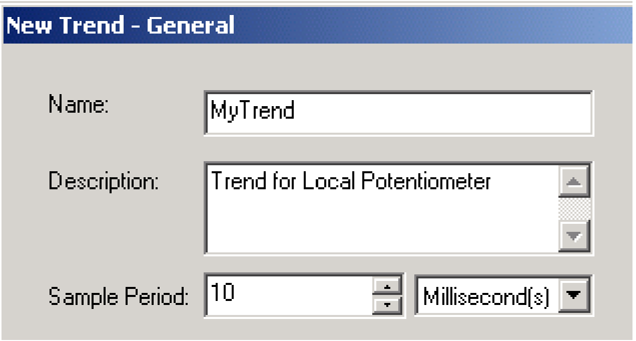
Trending

Trending is used to graph data over time. This can be either an analog signal or a discrete signal. You are allowed 8 pens per trend chart with a maximum of 255 trending charts per project. This is a simple procedure that will guide you through the creation of a trend chart in your project.

1) Right click the **trends** folder in the Controller Organizer Window, and select '***new trend***'.



2) In the new trend dialog window, name your trend, and add a description. For this example, we will leave the sample period at default. Click '***Next***'.



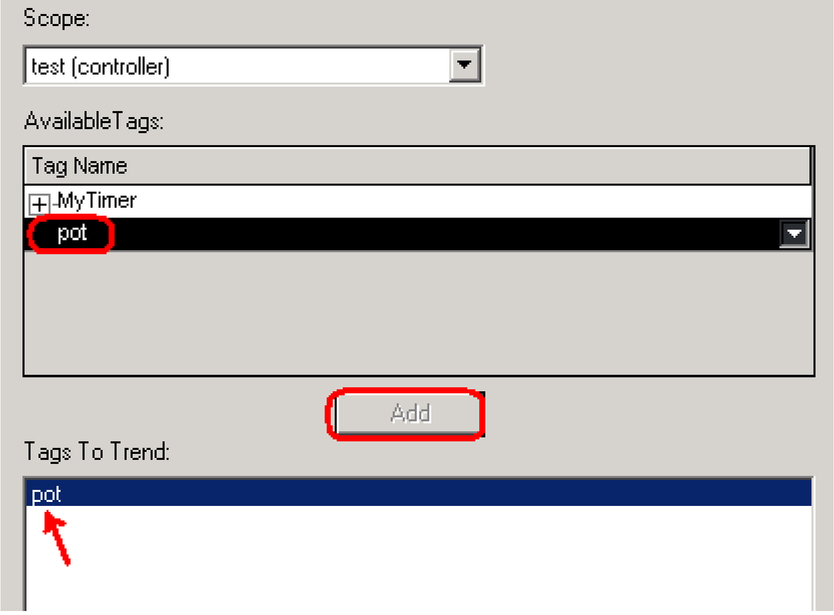
New Trend - General

Name: MyTrend

Description: Trend for Local Potentiometer

Sample Period: 10 Millisecond(s)

3) Choose the tag you wish to trend, then press the '***Add***' button. You will notice your tag is now in the list of tags to trend. Click ***Finish***.



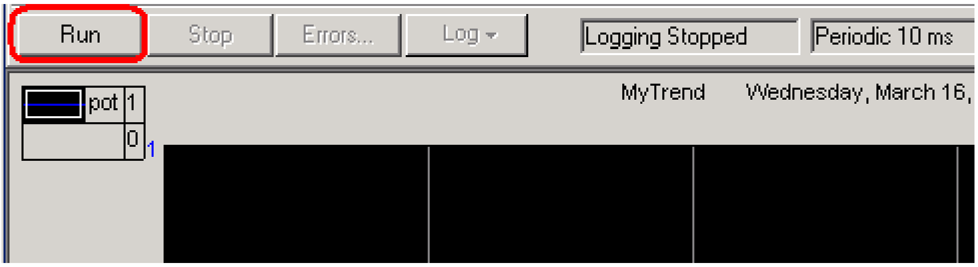
Scope: test (controller)

AvailableTags:

Tag Name
MyTimer
pot

Tags To Trend: pot

4) Next, Click the 'Run' button in the upper left corner of the trending chart.

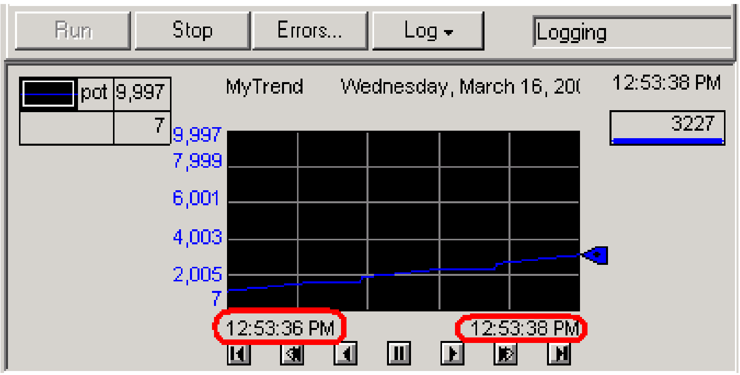


Logging Stopped

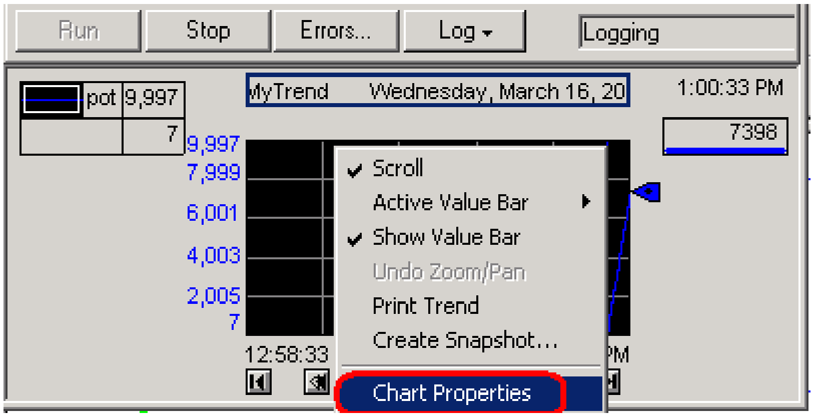
Periodic 10 ms

MyTrend Wednesday, March 16,

5) You will see your chart start tracking. You will notice the chart runs very fast. You only have a time period of 2 seconds currently being displayed from the left to the right side of the screen.



6) We are going to change this to 2 minutes so the chart will run much slower. This will give us a better indication of what the signal is doing over time. The X axis runs left to right (This is Time), and the Scale runs from bottom to top. To reconfigure the X axis, right click on the chart, and choose '***Chart Properties***'.



7) On the X tab, change the time span to **2 minutes**. Then press ***Apply***, and ***OK***.

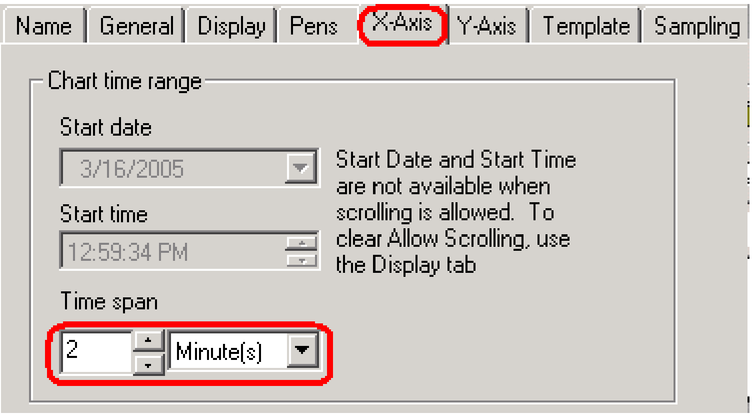


Chart time range

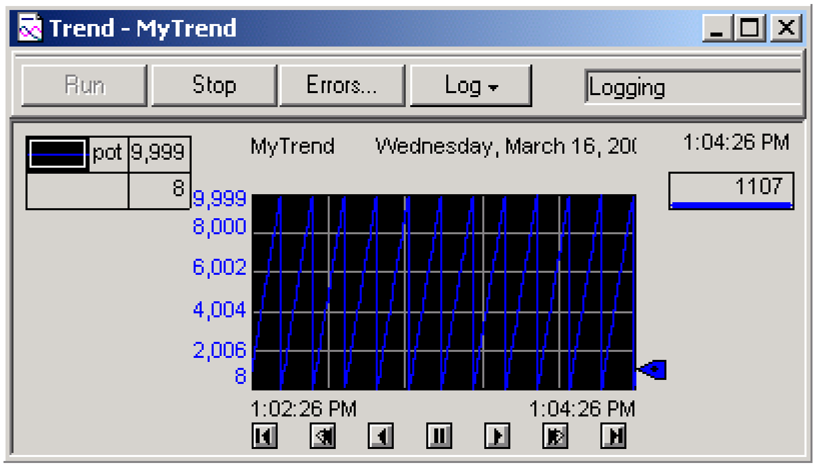
Start date: 3/16/2005

Start time: 12:59:34 PM

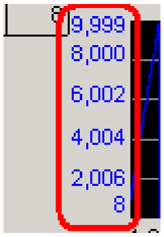
Start Date and Start Time are not available when scrolling is allowed. To clear Allow Scrolling, use the Display tab

Time span: 2 Minute(s)

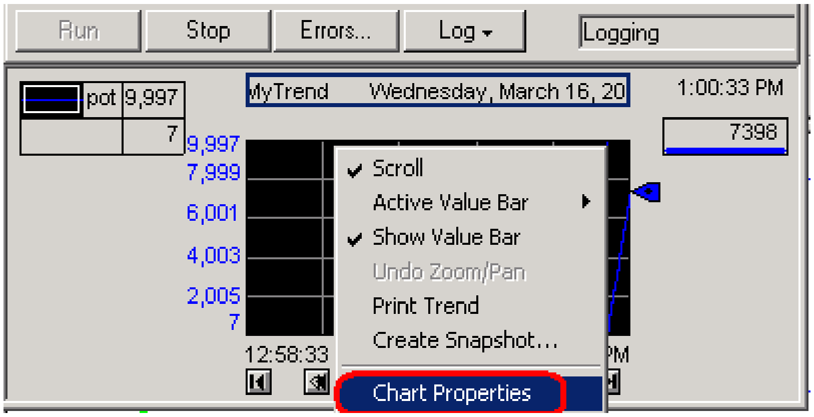
8) You will see your chart is now tracking over a 2 minute time period.



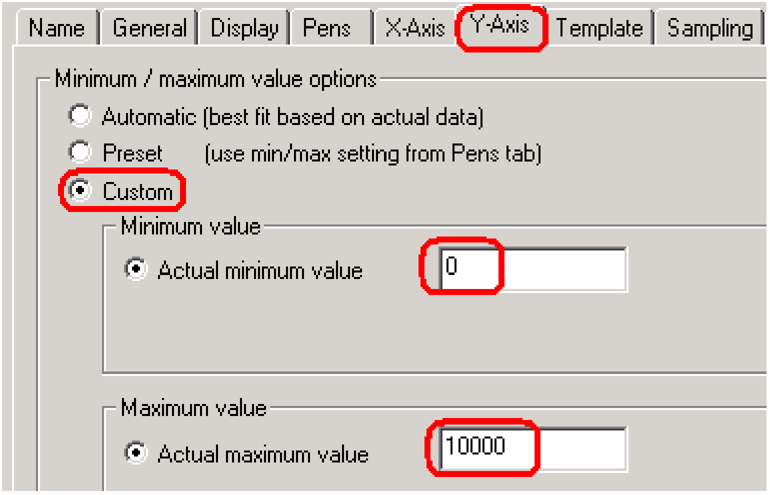
9) You will see the scale is not reflecting the full range possible for the analog signal. This is because the chart defaults to automatic mode. It will take the minimum and maximum value and adjust the scale accordingly. We will account for this in the next step.



10)Right click on the chart, and go back to '***Chart Properties***'



11)On the 'Y Axis' tab, you will notice 3 options. Automatic Mode is what we were running by default. This adjust the scale based on actual data. You can also choose preset if you like, and enter a different min/max value for each pen on the pens tab. The last option is to choose 'Custom'. This will let us lock in our own values for this particular trend chart. We will use custom for this example, and enter 0 as the minimum, and 10000 as the maximum. These values can be adjusted based on the data you are getting from your analog source. Press ***Apply***, then ***OK*** when finished.



Minimum / maximum value options

○ Automatic (best fit based on actual data)

○ Preset (use min/max setting from Pens tab)

● Custom

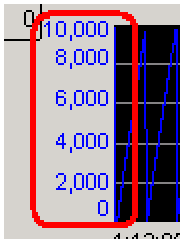
Minimum value

● Actual minimum value 0

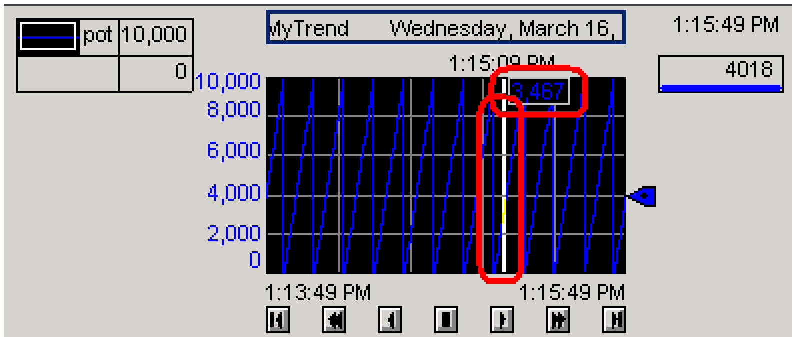
Maximum value

● Actual maximum value 10000

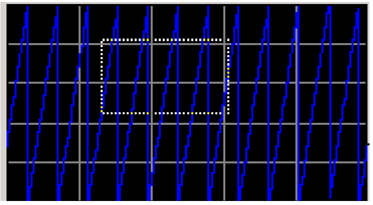
12)You will see the scale on the chart is now locked in with the custom values you entered from the '***Y Axis***' tab.

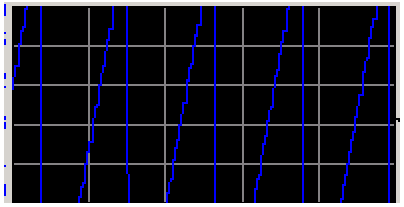


13)By clicking anywhere on the chart, a value bar will appear. This value bar will indicate the exact value of the tag at that moment in time.

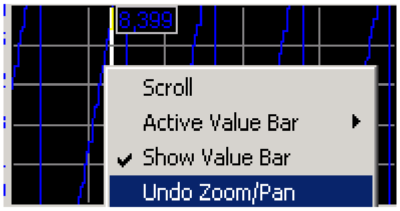


14)If you click on an area of the chart, and drag your mouse, you will draw a box around a certain area of the graph. When you release your mouse, you will be zoomed in on that particular area of the chart. You will know you are in zoom mode by a magnifying glass on your mouse cursor.

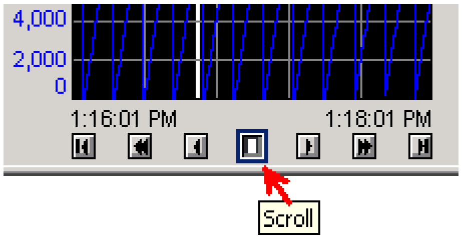




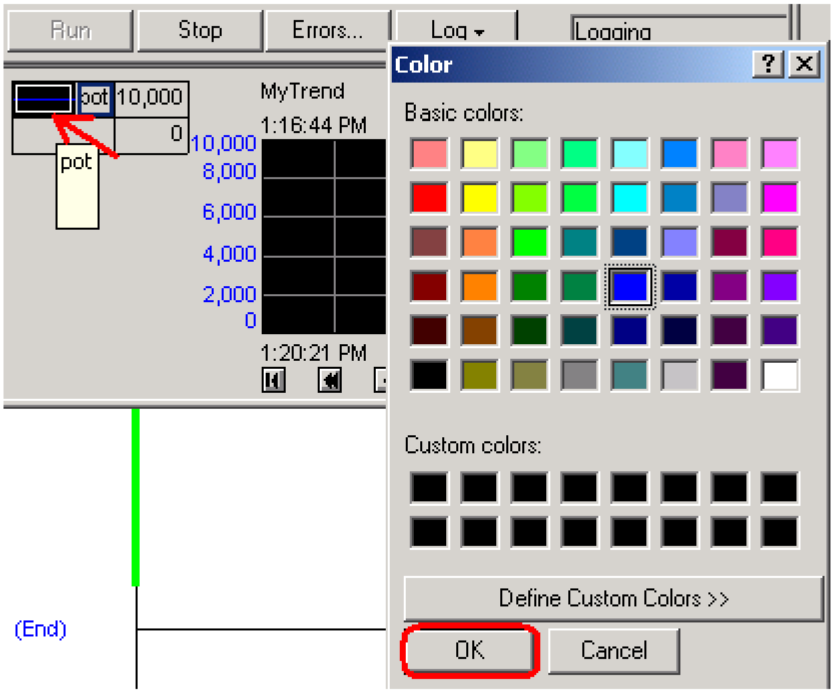
15)To get out of Zoom mode, right click on the chart, and Undo Zoom/Pan.



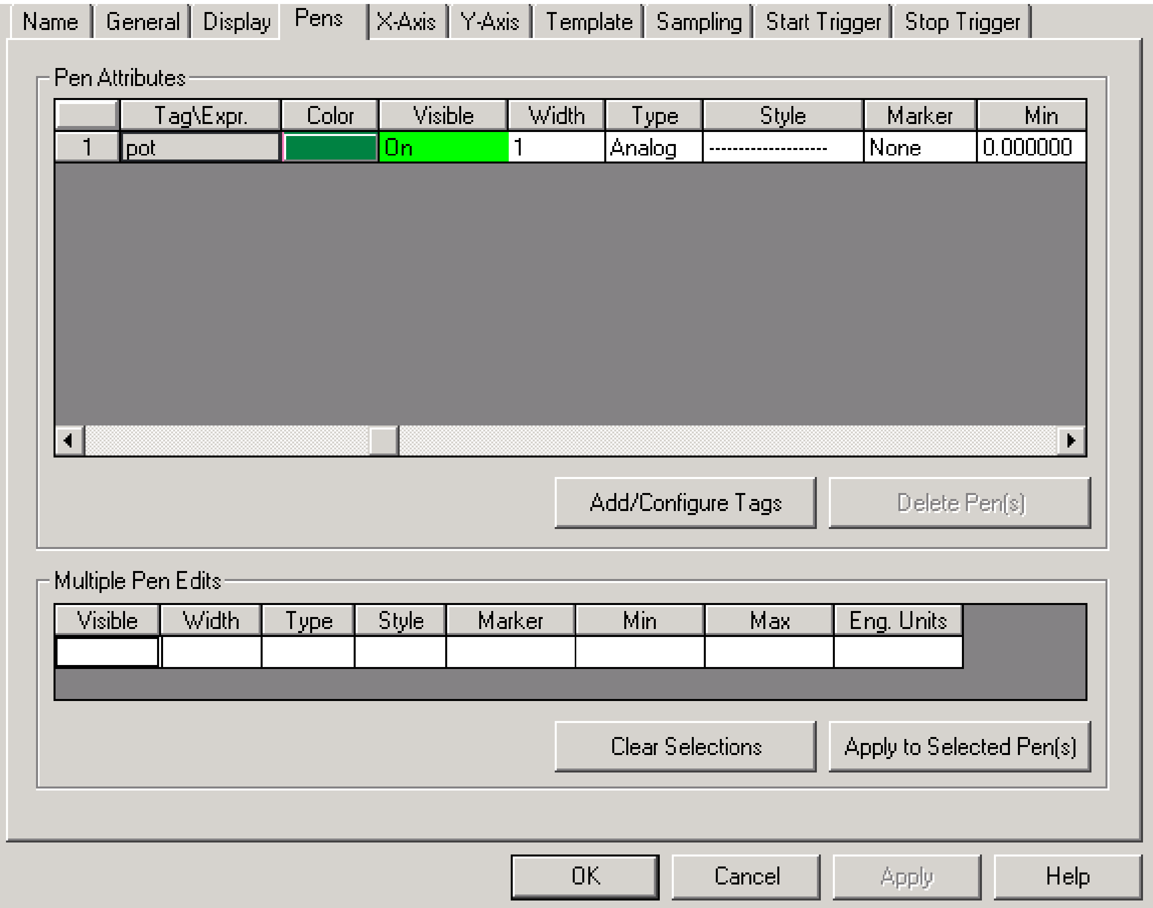
16)Next, you must restart your chart by un-pausing the graph.



17)Your chart should now be tracking. If you wish to change the color of your pen, Double-Click the pen in the upper left hand corner of your chart. A pallet will appear. Choose your new color from the pallet, then press ***OK.***



18)You know know the basic method for setting up a trending chart. At this time, take a few minutes to explore the other options you have for the trending chart. You can add more pens if you like under the 'pens' tab. This will allow you to track more addresses. You can also change the width of each pen, make them visible, or invisible, etc.... Try to change the color of the background.... If you have questions, ask the instructor.



Pen Attributes

	Tag\Expr.	Color	Visible	Width	Type	Style	Marker	Min
1	pot	🟩	On	1	Analog	-------------------	None	0.000000

Multiple Pen Edits

Visible	Width	Type	Style	Marker	Min	Max	Eng. Units

PID Demonstration -- ControlLogix

Terminology

Notice: This document is for the purposes of understand Proportional, Integral, and Derivative only, and how various settings react based on a sample program. In no way does this document reflect how to tune PID loops at your location due to the wide variety of plant applications. In no way are you qualified to tune a PID loop based solely on the information contained in this document. Although I believe all documentation to be accurate, it is your responsibility to verify the accuracy at the time you refer to the document based on overall changes in Industrial automation, the particular plant example you are concerned with, and possible errors in this text.

CV – Control Variable – This is the output (control) from the PID loop. The control variable can be used to control a valve or heating bands. The PID loop will continuously read the input (process variable), and decide where to set the control variable to achieve the set point.

PV – Process Variable – This is the input (status) of the process. The process variable can read a tank level or a temperature. The PID loop continuously reads this value to determine how much output to provide to the process to achieve the set point.

SP – Set Point – The value of the process variable we desire to achieve. The set point can be described in terms of a temperature we want to maintain, a tank level we desire, a particular flow rate, etc.

ERROR – The difference between the current value of the set point and the process variable. (In other words, the difference between where we are and where we should be)

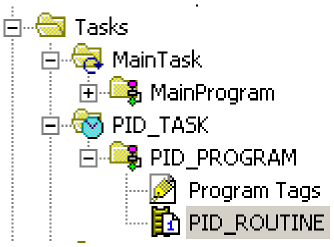
Proportional – Output that is proportional to the amount of error. For example: as the process variable approaches the set point the output (CV) will decrease (or in some cases increase) proportionally.

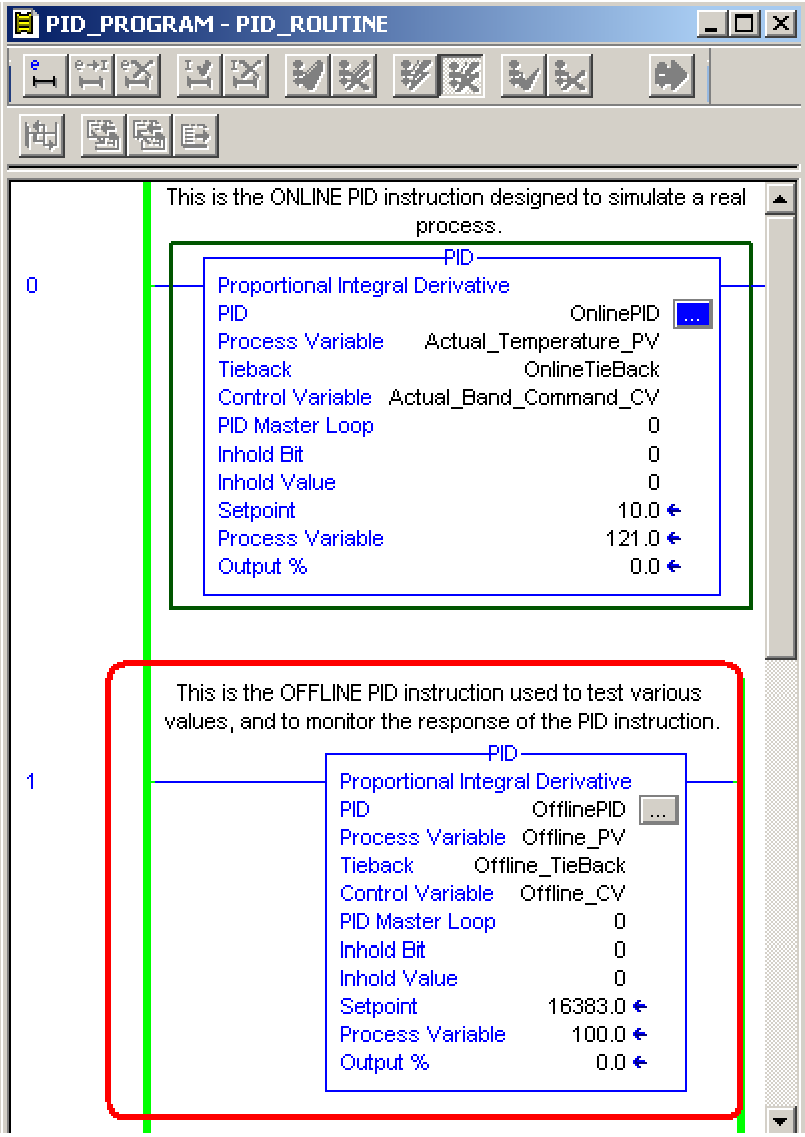
Integral – Repeats the proportional influence over time. For example: The longer we stay below the set point, the more output is provided (or less in some cases). Integral is based on time.

Derivative – Output influence that is based on the rate of change of error. For example, if we have a huge steam demand, a heating unit may start to cool down rapidly. At first, the error is not enough to provide a large output based on proportional influence, and not enough time has passed for integral influence to provide enough output to quickly pull the process back to the set point. Derivative is not used in every process. For many processes, just the proportional and integral alone is enough to control a loop.

Tieback – A value used to implement "bumpless transfer" when the loop is in manual mode. This helps to ensure a more stable output when the loop is placed into manual mode.

1) In the PID Routine, you will find an OFFLINE PID instruction. This instruction is not currently controlling any process. We are going to use this instruction simply for the purpose of seeing how the proportional controller alone reacts to error.





0

This is the ONLINE PID instruction designed to simulate a real process.

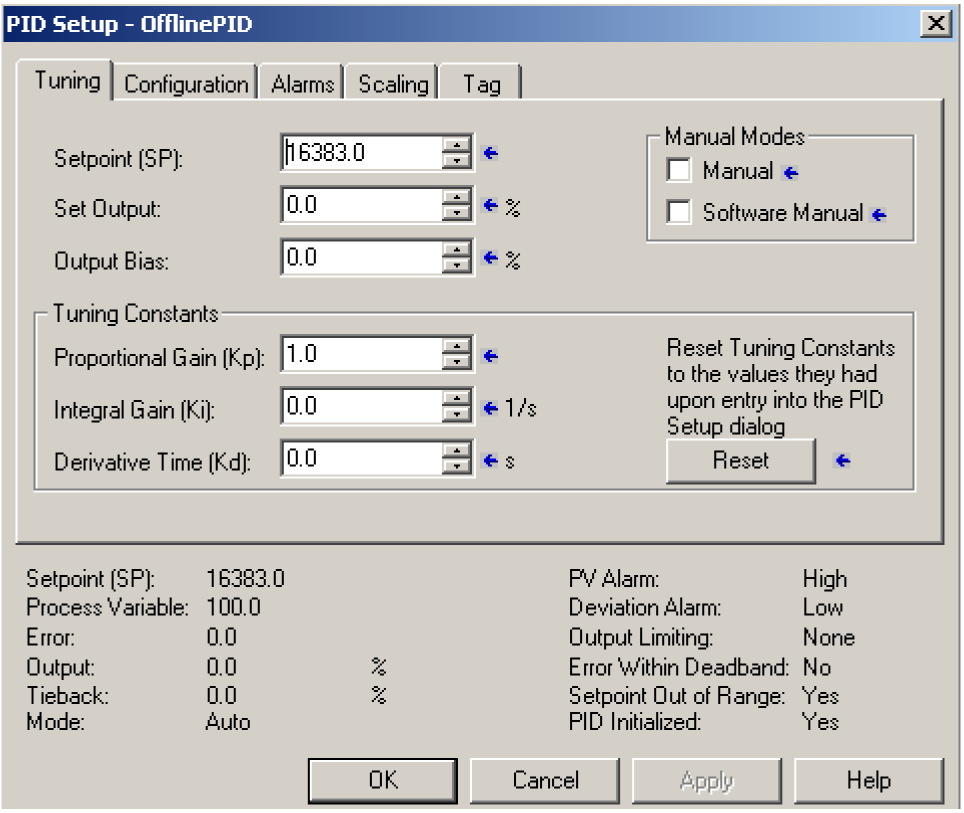
PID
Proportional Integral Derivative
PID OnlinePID
Process Variable Actual_Temperature_PV
Tieback OnlineTieBack
Control Variable Actual_Band_Command_CV
PID Master Loop 0
Inhold Bit 0
Inhold Value 0
Setpoint 10.0
Process Variable 121.0
Output % 0.0

1

This is the OFFLINE PID instruction used to test various values, and to monitor the response of the PID instruction.

PID
Proportional Integral Derivative
PID OfflinePID
Process Variable Offline_PV
Tieback Offline_TieBack
Control Variable Offline_CV
PID Master Loop 0
Inhold Bit 0
Inhold Value 0
Setpoint 16383.0
Process Variable 100.0
Output % 0.0

2) Click the ellipsis within the PID instruction to get to the setup screen, and you will see various parameters to control the PID instruction. PV is the process variable. This is the value returned from a sensor. We will also be discussing the CV (control variable) later in this lesson. This is the output for the PID loop. Set up the status parameters as follows:



PID Setup - OfflinePID

Tuning | Configuration | Alarms | Scaling | Tag

Setpoint (SP): 16383.0

Set Output: 0.0 %

Output Bias: 0.0 %

Manual Modes

□ Manual

□ Software Manual

Tuning Constants

Proportional Gain (Kp): 1.0

Integral Gain (Ki): 0.0 1/s

Derivative Time (Kd): 0.0 s

Reset Tuning Constants to the values they had upon entry into the PID Setup dialog

Reset

Setpoint (SP):	16383.0		PV Alarm:	High
Process Variable:	100.0		Deviation Alarm:	Low
Error:	0.0		Output Limiting:	None
Output:	0.0	%	Error Within Deadband:	No
Tieback:	0.0	%	Setpoint Out of Range:	Yes
Mode:	Auto		PID Initialized:	Yes

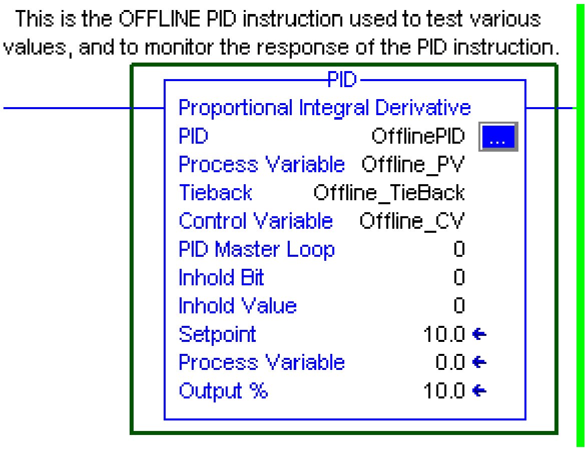
3) You will notice that the Controller Gain (Kp) is going to be our proportional gain. Be sure this gain is set for the value of 1 for our first experiment. Ki and Kd are your integral and derivative timing values. We will work with these later.

4) Next we are going to change the SP to the value of 10. When you changed the Set point to the value of 10, how much output did the controller provide? What is the error?

5) If we change the SP to the value of 30, how much output would you expect from the controller? If the PV is still at 0, what is the error going to be?

6) Try changing the SP to 30 and see if you get the results you would expect.

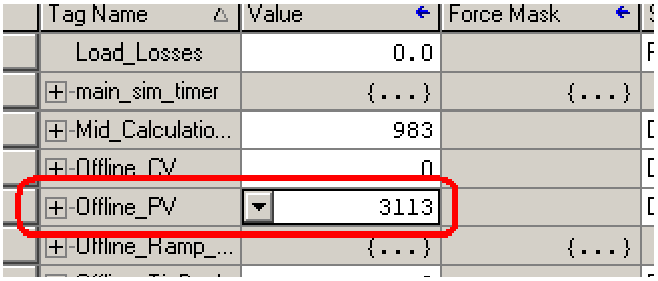
7) Now lets imagine for a moment that the output we are providing is starting to cause things to happen in the real world.... Such as a flow control valve that is starting to open. We are continuously reading the flow rate from this system. The sensor that is measuring the flow rate is calibrated in such a way that if we get a value of 16383 (20mA), then our flow rate is 100%. The value of 0 returned from the sensor will reflect a flow rate of 0. We can find the conversion from amperage to raw data in the manual for the analog module. Look at your PID equation. What memory location represents the value being returned from the sensor?



This is the OFFLINE PID instruction used to test various values, and to monitor the response of the PID instruction.

PID
Proportional Integral Derivative
PID OfflinePID
Process Variable Offline_PV
Tieback Offline_TieBack
Control Variable Offline_CV
PID Master Loop 0
Inhold Bit 0
Inhold Value 0
Setpoint 10.0
Process Variable 0.0
Output % 10.0

8) Remember that the value we are sending to the flow valve is called the Control Variable. The value being returned from a sensor which is measuring actual flow is called the Process Variable. The process variable is going to show up in memory location Offline_PV. Remember how our sensor is calibrated. If we see the value of 16383 at Offline_PV, then we know we have 100% flow. We are going to simulate feedback from a flow meter by entering the value 3113 into Offline_PV. After entering the value of 3113 in Offline_PV, go back to the Setup Screen for your PID instruction.



Tag Name	Value	Force Mask	S
Load_Losses	0.0		F
+-main_sim_timer	{...}	{...}	
+-Mid_Calculatio...	983		D
+-Offline_CV	0		D
+-Offline_PV	3113		D
+-Offline_Ramp_...	{...}	{...}	

9) With your Set point still at 30:

1. What is the PV reading?
2. What is the CV reading?
3. What is the ERROR?
4. Note that 3113 is 19% of 16383.

10) Increase Offline_PV to the value of 3768 (23%). What happens to the output as the Process Variable begins to reach the set point?

11) Now lets see how a change in Kp gain causes the controller to react. Reset Offline_PV back to 0, go to the setup screen for the PID loop in Ladder 9 and change the Kc Gain to the value of 2. If you were to change the set point to the value of 10, how much output would you expect from the controller?

12) Change the SP to 10. Did you get the value you expected to see?

13) What output would you expect at the CV if you change the SP to 30?

14) Change the SP to 30. Did you get the output you expected?

15) Change Offline_PV to 3768 (23%). What output would you expect to see with a SP of 30?

16) Go back to the Setup Screen for the OFFLINE PID instruction in Ladder 10. Was your answer correct?

17) Now you understand how the proportional controller alone works. The controller provides an output that is proportional to the error. Think about how the proportional controller alone works. Under normal circumstances, can the proportional controller alone bring the process variable up to the set point?

18) Why or Why Not?

19) Think about a heating process:

When you start the heating process, the controller will provide a very large output. As you approach the set point, the output will decrease, until the output is zero at the set point. With an output of zero, you are not putting any heat into the system. Due to ambient and load losses, the system will begin to cool. Eventually, at some point below the set point, the amount of heat the controller is adding to the system will be equal to the amount of heat being removed from the system. At this point your system has reached steady state. The error between the steady state temperature and the set point is called the OFFSET.

If the load is increased, then more heat is being drawn out of the system. The controller is no longer providing sufficient output to maintain the previous offset. A greater error must be present in order to increase the amount of heat the controller is adding to the system.

Now verify your answer to question 10. We will demonstrate this effect in the next example.

1--SETPOINT---------------PV=SP (CV=0)

2____PV little below SP (CV =10)

3____PV much below SP (CV=20)

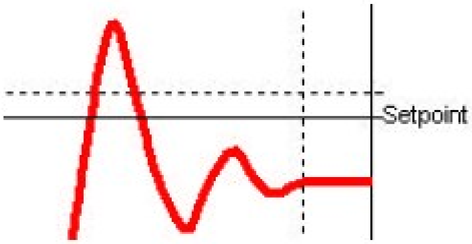
Position 1) If the PV was at the SP, then the controller is not adding any power to compensate for ambient and load losses. Temperature cannot be sustained.

Position 2) If our PV was at this point, we have an error of 10, and therefore the processor is calling for 10% power. Here we can reach steady state if the losses and loads combined are not taking out more heat energy than we are putting into the system. If the load increased, we could not maintain this temperature since our output can only be 10% at this position.

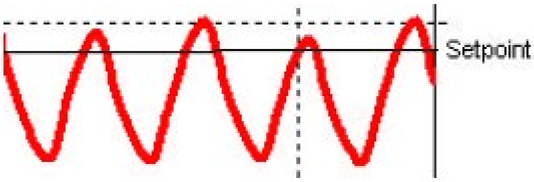
Position 3) If we was at position 2 and our load began to increase, we are taking more heat out of the system than what we are putting in. The temperature begins to decrease. This causes an increase in the error, and the controller will increase power. The system will reach steady state at a lower temperature. This will be the temperature at which the controller is providing enough output to make up for the increased load.

20) If the Kp is increased, then the offset is going to be decreased because we can have the same output with a smaller error. However, if Kp is increased too high, the controller will become unstable and start to oscillate. Allen Bradley states in the Instruction Set Reference Manual that a good rule of thumb used to set Kp is half the value which causes oscillations. This will vary depending on the process.

21) In this example Kp is set okay. The PV overshot the SP, but did eventually reach steady state. Since this is is an example of proportional control only. Steady state is actually being reached at a lower temperature than the set point. The reason is because the temperature at which we reached steady state provided the exact error we need for the proportional controller to generate enough output to make up for losses and load. If Kc is increased, then the error will be decreased, and the steady state temperature will be closer to SP.



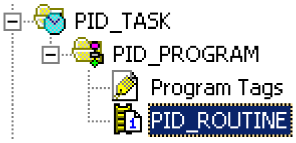
22) In this example, we increased Kp too much. This caused the controller to become unstable and start to oscillate. The minimum value for Kp to cause this type of oscillation is important for calibration. In many cases, once you find the value of Kp that causes the controller to oscillate, set Kp to half this value. The amount of time required for one complete oscillation is called the “natural period” of a sine wave. This natural period will be useful when setting the integral and derivative later on in this lesson.



23) In the next lesson, we are going to simulate a real world process. During this lesson you are going to determine what is the maximum value we can use for Kp without the controller becoming unstable.

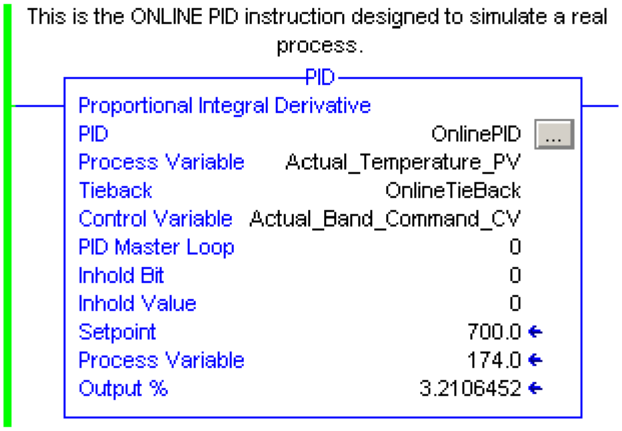
Proportional Gain Worksheet

24) To start this exercise, Open the PID Routine of the PID task.



PID_TASK
PID_PROGRAM
Program Tags
PID_ROUTINE

25) Notice the ONLINE PID instruction in your ladder diagram:



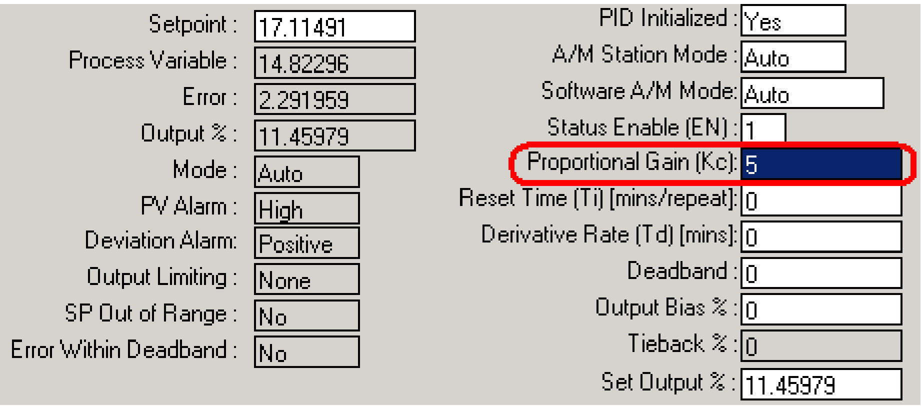
This is the ONLINE PID instruction designed to simulate a real process.

PID

Proportional Integral Derivative

PID	OnlinePID
Process Variable	Actual_Temperature_PV
Tieback	OnlineTieBack
Control Variable	Actual_Band_Command_CV
PID Master Loop	0
Inhold Bit	0
Inhold Value	0
Setpoint	700.0
Process Variable	174.0
Output %	3.2106452

26) Click the Elipsis to get to the setup screen within your PID instruction.

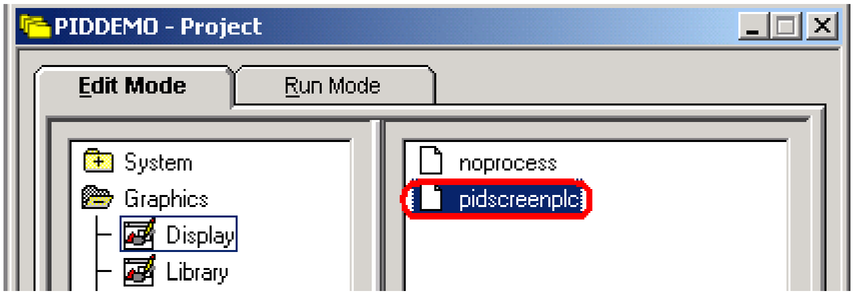


Setpoint :	17.11491	PID Initialized :	Yes
Process Variable :	14.82296	A/M Station Mode :	Auto
Error :	2.291959	Software A/M Mode:	Auto
Output % :	11.45979	Status Enable (EN) :	1
Mode :	Auto	Proportional Gain (Kc):	5
PV Alarm :	High	Reset Time (Ti) [mins/repeat]:	0
Deviation Alarm:	Positive	Derivative Rate (Td) [mins]:	0
Output Limiting :	None	Deadband :	0
SP Out of Range :	No	Output Bias % :	0
Error Within Deadband :	No	Tieback % :	0
		Set Output % :	11.45979

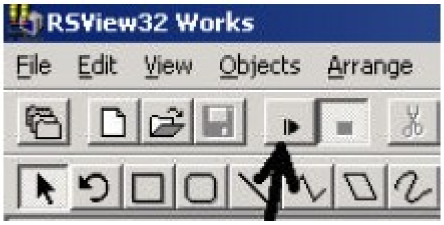
27) Set the Controller Gain (Kc) to 35

28) Next, follow your instructor to open a project in RSView. RSView will be the Graphical User Interface (GUI) for a heating process. The program in your PLC-5 will simulate the process. You can find RSView by clicking Start|Programs|Rockwell Software|RSView32|RSView 32 Works. Once RSView is open, Click File, and then open the project called piddemo.rsv

29) Next, Double click on the + next to Graphics. The single click display. A list of the screens available for this project show up on the right hand side. Double click the display called pidscreensplc.



30) Click the Start button on your tool bar.



31) Now we are set up for the next exercise. In this next exercise you are going to adjust the proportional gain (Kp) to find out what value causes the loop to become unstable. Use the following chart: For each row, go to RSLogix and enter the Kc value into the PID controller in Ladder 8. Come back to RSView, With your load at 0, change the load on the system to 100%. Watch the response. Circle whether the controller is still stable after the transition, or whether it has gone unstable. Then change your load from 100% to 0% and watch the graph in RSView. Record the stability, then move to the next row.

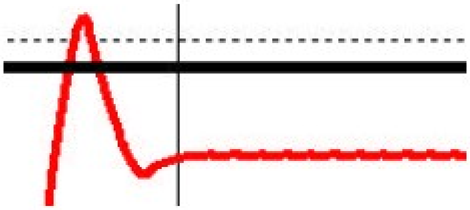
Proportional Gain Kc	Load transition 0 to 100%	Load transition 100 to 0%
35	Stable | Unstable	Stable | Unstable
40	Stable | Unstable	Stable | Unstable
45	Stable | Unstable	Stable | Unstable
50	Stable | Unstable	Stable | Unstable

32) What value of Kp started causing instability during certain load transitions?

____

33) What is the natural period of the oscillations (in minutes)?__________________________

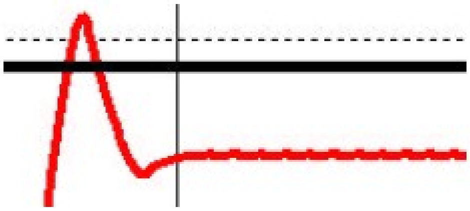
34) Your Set point is 700 degrees. Record the steady state temperature under 0 load with your new Kp. _______You will see that the steady state temperature is significantly below the set point. That is because of the error required to generate enough output to make up for ambient losses. Put your load at 100% and record the steady state temperature again. _________You will see that it is even lower. Because we are taking more heat out of the system, we must have greater error to generate more output.



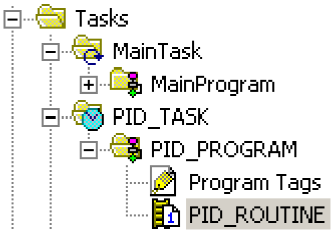
35) We will compensate for this offset in the next lesson

Integral Worksheet

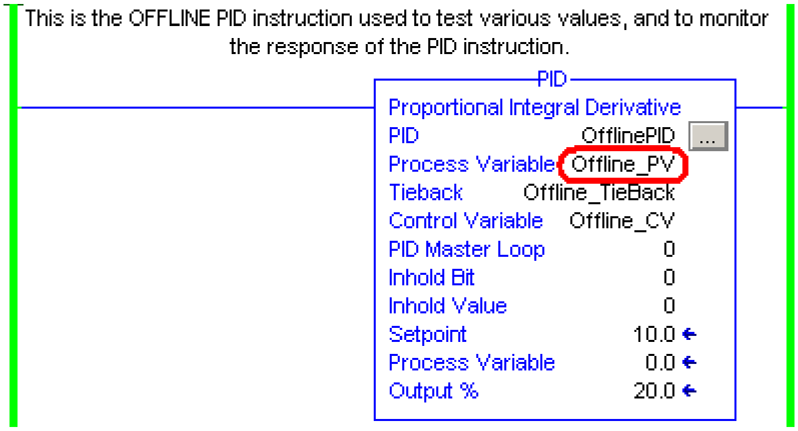
36) You noticed in the last lesson that by using the proportional control alone, we cannot achieve the desired set point, and sometimes cannot even get the PV close to the SP without the loop becoming unstable. We need another component of the controller that can increase the output to push the PV up to the SP. The integral controller controls proportional repeats per minute. Therefore, the longer the PV stays below the SP, the more output is generated. The closer the PV gets to the SP, the slower the output increases. Remember proportional control generates an output simply based on the amount of error. The integral output takes this one step further. If we still have an error, then we are going to continuously keep adding output ever so slightly until we reach the set point.



37) The PID_ROUTINE has a PID instruction set up that will allow us to see the effect of the controller alone when it is not attached to a real world process. Lets bring up RSLogix and double click the PID_ROUTINE in the project tree.

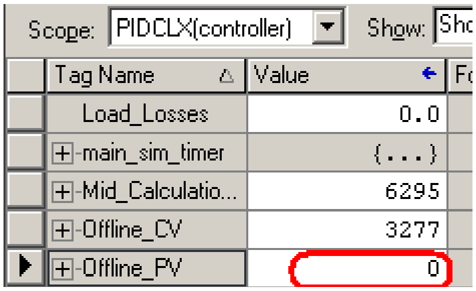


38) Notice the Process Variable is Offline_PV. Before we start, lets set the Process Variable Offline_PV to 0 in the Controller Tag Database. If this were a real process, the feedback from the process would show up in Offline_PV. We will do this manually for this example.



This is the OFFLINE PID instruction used to test various values, and to monitor the response of the PID instruction.

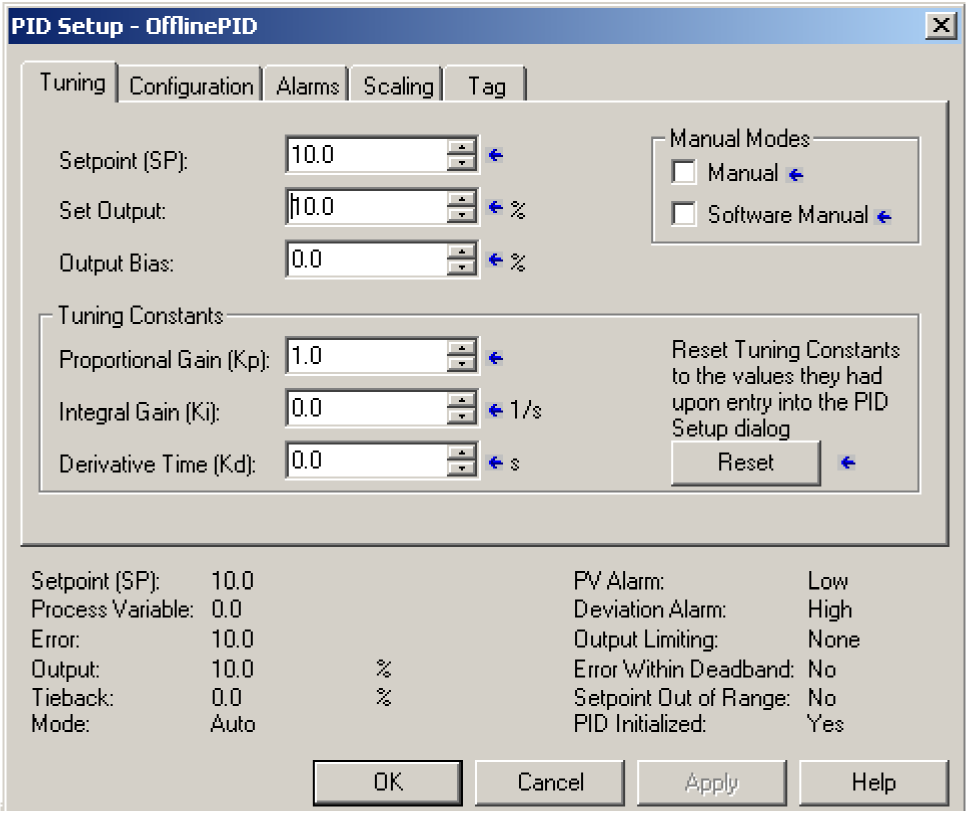
PID
Proportional Integral Derivative
PID OfflinePID ...
Process Variable Offline_PV
Tieback Offline_TieBack
Control Variable Offline_CV
PID Master Loop 0
Inhold Bit 0
Inhold Value 0
Setpoint 10.0 ←
Process Variable 0.0 ←
Output % 20.0 ←



Scope: PIDCLX(controller) ▼ Show: Sho

Tag Name △	Value ←	F
Load_Losses	0.0	
⊞-main_sim_timer	{...}	
⊞-Mid_Calculatio...	6295	
⊞-Offline_CV	3277	
⊞-Offline_PV	0	

39) Click ‘Setup Screen’ on the Offline PID instruction. Configure your PID instruction as follows:



Setpoint (SP): 10.0

Set Output: 10.0 %

Output Bias: 0.0 %

Manual Modes

☐ Manual

☐ Software Manual

Tuning Constants

Proportional Gain (Kp): 1.0

Integral Gain (Ki): 0.0 1/s

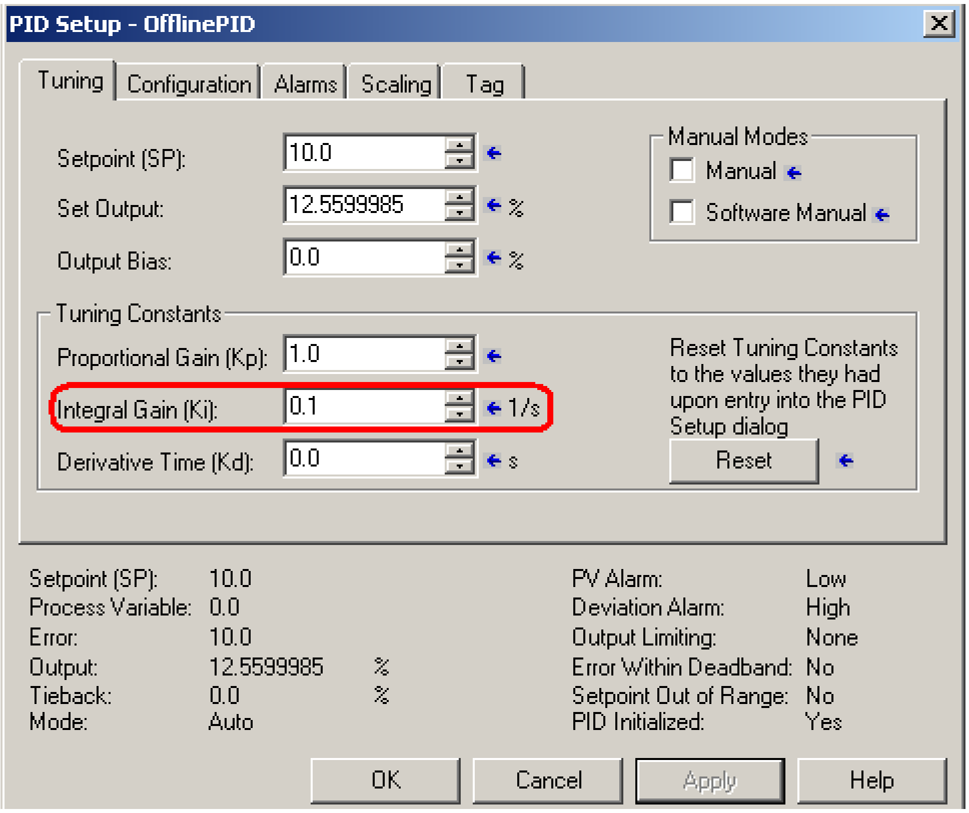
Derivative Time (Kd): 0.0 s

Reset Tuning Constants to the values they had upon entry into the PID Setup dialog

Setpoint (SP):	10.0		PV Alarm:	Low
Process Variable:	0.0		Deviation Alarm:	High
Error:	10.0		Output Limiting:	None
Output:	10.0	%	Error Within Deadband:	No
Tieback:	0.0	%	Setpoint Out of Range:	No
Mode:	Auto		PID Initialized:	Yes

40) Recall that from a previous lesson, using the proportional control only, if our error is 10%, and the Controller gain is 1.0, then the output is going to be 10%. Here we have a condition where our PV is below the set point and is not increasing (just like in our online heating process once it reached steady state with proportional gain only).

41) Change your Reset Ki to the value of **.1**. If you make a mistake in entering this value (such as entering 1), enter 0 to clear the integral influence on the output, then enter **.1** again.



Setpoint (SP): 10.0
Set Output: 12.5599985 %
Output Bias: 0.0 %

Manual Modes
□ Manual
□ Software Manual

Tuning Constants
Proportional Gain (Kp): 1.0
Integral Gain (Ki): 0.1 1/s
Derivative Time (Kd): 0.0 s

Reset Tuning Constants to the values they had upon entry into the PID Setup dialog
Reset

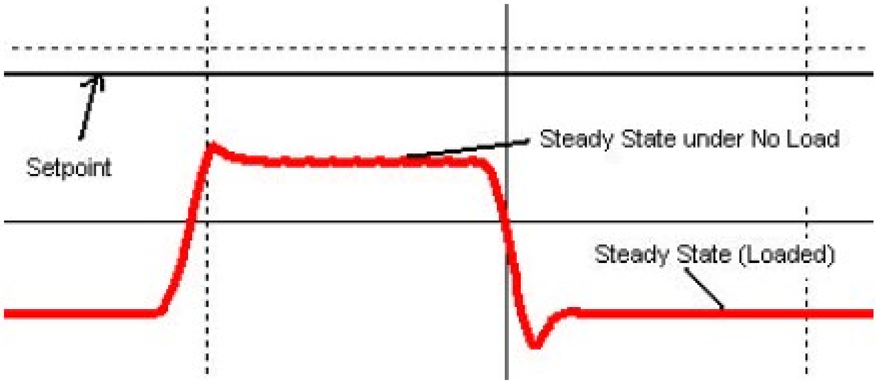
Setpoint (SP):	10.0		PV Alarm:	Low
Process Variable:	0.0		Deviation Alarm:	High
Error:	10.0		Output Limiting:	None
Output:	12.5599985	%	Error Within Deadband:	No
Tieback:	0.0	%	Setpoint Out of Range:	No
Mode:	Auto		PID Initialized:	Yes

42) Observe what is happening to the CV%. Since we entered Ki as **repeats per second**, after 10 seconds, the proportional gain will be repeated, and you will have an output value of 20. After 20 seconds you will have an output value of 30. If the PV starts to increase, the error will decrease, and the CV% will increase more slowly. Even if the PV is only 1% below the SP, the CV% will still be increasing (although very slowly by this time) until the PV is equal to the SP.

43) Change your Reset Ki to **0** to remove the integral influence, then set Ki to **.05**. You will see the Integral control is repeating the proportional gain every 20 seconds now which is half as fast. Change your Ki to **0**, then observe the value of the output when you set Ki to **.2**. The output will increase much faster. If the value of Ki is set too high it could cause the system to become unstable.

44) Now that you understand how integral control works, lets try this on your heating process.

45) Go back to RSView, and for review, change your load to 0% and observe how the loop reacts. Then change your load to 100% and observe how the loop reacts. Also notice that the steady state process variable is well below the set point.



46) Now we are going to change the Ki variable. What was the natural period (in minutes) that you recorded when the loop was unstable?___________ A rule of thumb is that the Ti variable is equal to the natural period. This will change (sometimes drastically) based on your application. Ti is the rate in minutes per repeat. To convert Ti to Ki, use the following formula according to Allen Bradley's Instruction Set Reference Manual.:

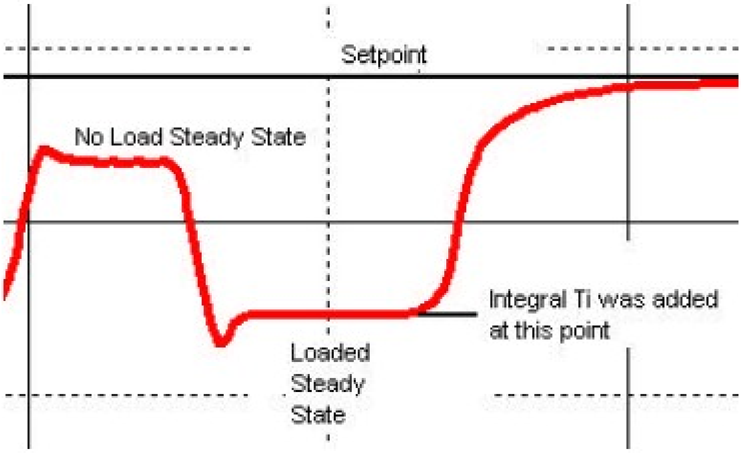
Kp = Kc (Kp is Proportional gain in independent mode, and Kc is Controller gain dependent mode.)

Ki = Kc/60*Ti

What is your answer for Ki? __________

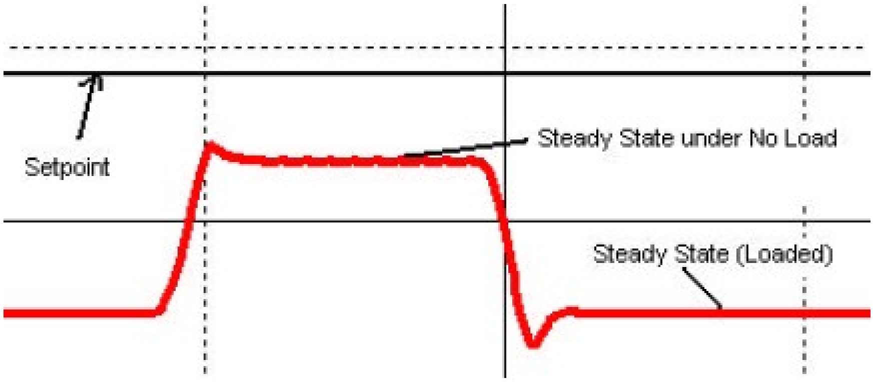
47) Go to RSLogix, and look at the PID_ROUTINE. Set the Ki value of the ONLINE PID instruction. Immediately go back to RSView and observe what effect Integral control is having on your system.

48) You can see that right away when Ki is added to the equation, the output increases sharply due to the large error. The integral control continues to add output to the control variable at increasing slower rates until the PV is driven up to achieve the Set point. Now change your load between 0 and 100 percent to see how stable the loop is. You will see fluctuations immediately, however, no matter what the load, the Set point is achieved and the loop becomes stable within a reasonable period of time.



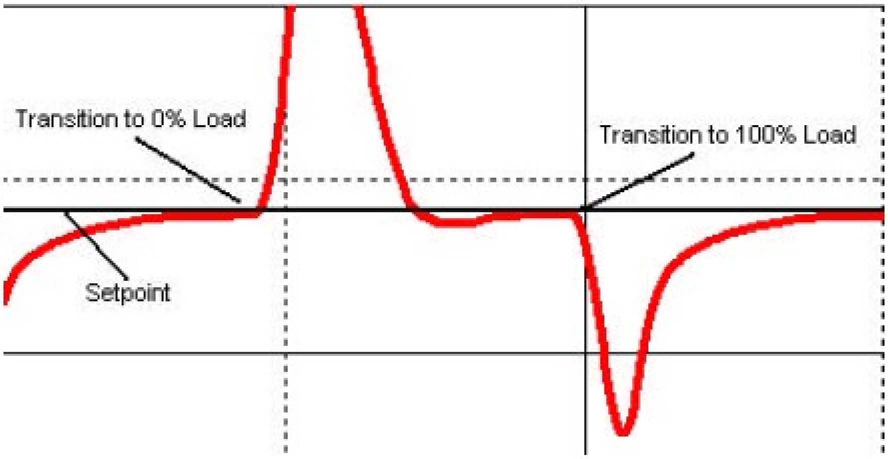
49) When removing the load, the PV may go out of range of the chart. The reason for that is that for 100% load, the control variable required is very high. When the load is removed, and is no longer removing heat from the system, the heat bands are still hot, and take some time to begin to cool.

50) Similarly, when heating up the system, the heat bands are cold, and require some amount of time to become hot enough to start adding heat to the system. For this example, the power level the PID controller is wanting to achieve is called the requested power. The amount of energy the heating bands are adding to the system is called the actual power. When there is a change in requested power, it takes some time for the actual power to acheive the requested power level.



51) Now Compare these 2 graphs. To the left is a graph of how the system reacted with proportional control only. Notice the steady state temperatures. The larger the load, the lower the temperature of the system. Even at no load, due to ambient losses, the steady state temperature was well below the set point.

52) Now that we have Ki added to our controller, you will see that we now achieve the SP.



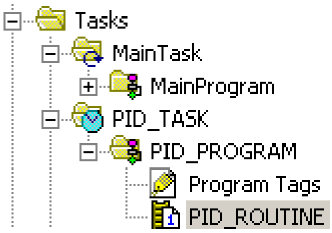
Derivative Worksheet

53) The proportional and integral control work well in maintaining a desired temperature. The last part of PID that we are going to discuss is the DERIVATIVE component. With derivative action, the controller output is proportional to the rate of change of the measurement or error. You do not always need derivative action, and on some processes, it should never be used. Since derivative is based on the rate of change of error, if you have a noisy PV, large amounts of gain can be generated causing the loop to become unstable. Derivative action on noisy loops can cause the valve to become 'jittery' and decrease the life of the valve. An example might be a flow process at a water treatment facility. The material flowing through the system might have various degrees of viscosity causing intermittent large changes in the PV. If the controller attempts to act on these changes, the loop could become unstable. Because of this effect, derivative action usually causes more harm than good for flow control processes and therefore should not be used.

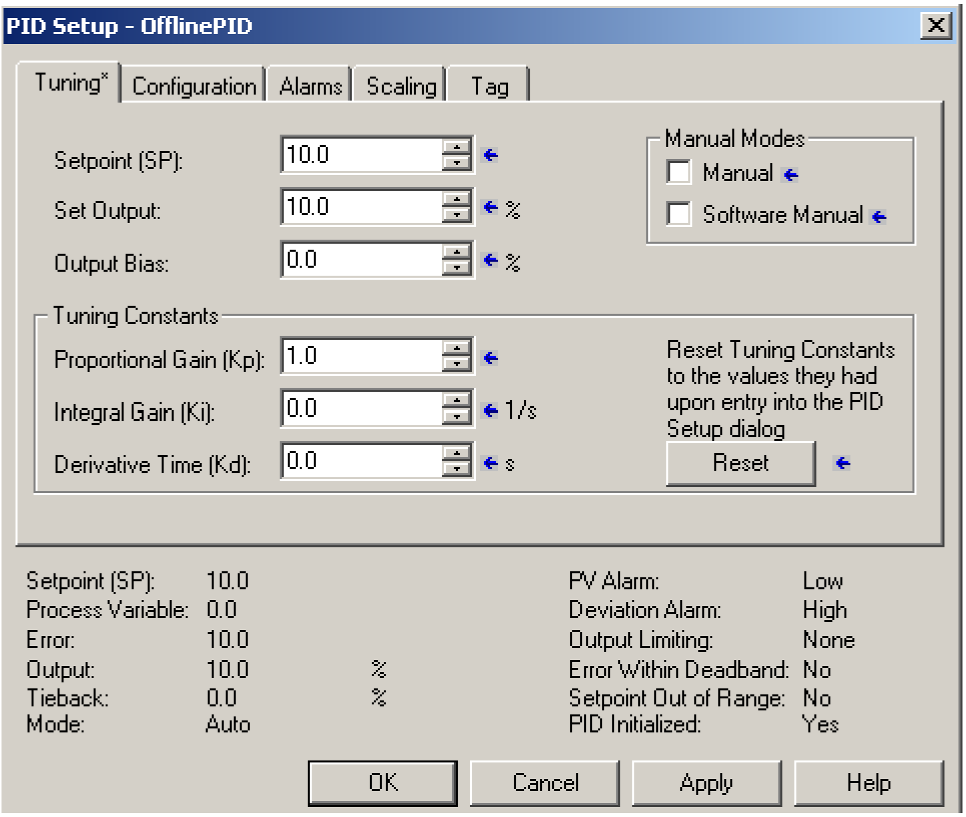
54) Imagine a heating process. Derivative action can be useful on a heating process by anticipating changes. If load is dramatically increased on a system, the PV is going to begin to decrease rapidly. The proportional and integral gain are not going to start adding large amounts of output to the CV until the error increases significantly. The derivative action will immediately detect that the PV is rapidly decreasing, and start to generate output immediately.

55) The Instruction Set Reference Manual states that a good rule of thumb to use when setting the derivative Kd, is one-eighth the value of Ki.

56) Because of the nature of the heating process, we have been using, Derivative action would not be significantly noticed. We are going to use our OFFLINE PID instruction in the PID_ROUTINE to observe the effect of the derivative on a process.



57) Let's go into the Setup Screen for our OFFLINE PID instruction, and configure the parameters as follows:



PID Setup - OfflinePID

Setpoint (SP): 10.0

Set Output: 10.0 %

Output Bias: 0.0 %

Manual Modes

□ Manual

□ Software Manual

Tuning Constants

Proportional Gain (Kp): 1.0

Integral Gain (Ki): 0.0 1/s

Derivative Time (Kd): 0.0 s

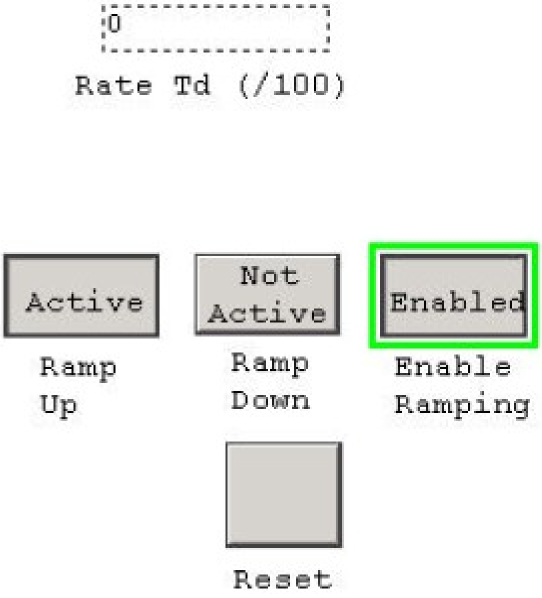
Reset Tuning Constants to the values they had upon entry into the PID Setup dialog

Setpoint (SP):	10.0		PV Alarm:	Low
Process Variable:	0.0		Deviation Alarm:	High
Error:	10.0		Output Limiting:	None
Output:	10.0	%	Error Within Deadband:	No
Tieback:	0.0	%	Setpoint Out of Range:	No
Mode:	Auto		PID Initialized:	Yes

58) In an actual process, a rule of thumb is that Td is 1/8th of the natural period of the unstable sine wave you recorded earlier. Utilize the following formula to calculate the Kd from the Td variable:

Kd = Kc * Td * 60

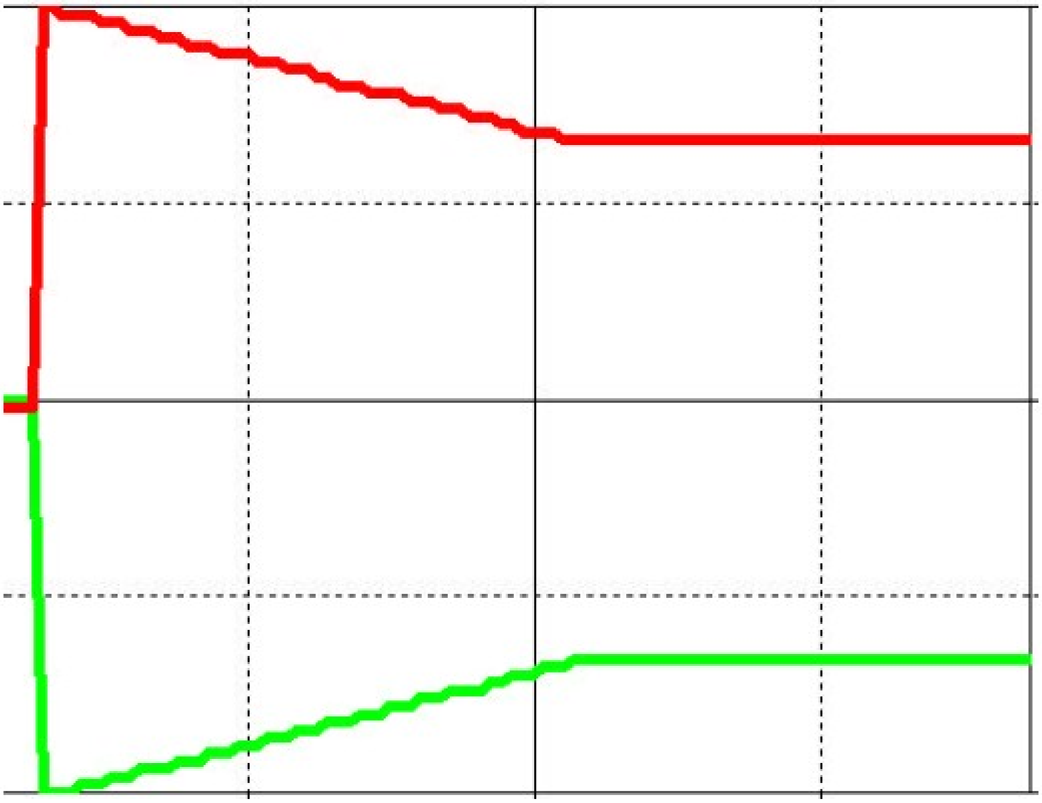
59) Since derivative gain is based on a rate of change, we are going to use a signal generator to feed a ramping PV into our system. Go back to your RSView project, and open the display called “noprocess”. This screen will allow you to control the ramping signal generator, monitor the CV%, and change the Kd of the controller.



60) Set your Rate Kd at 0. Enable your ramping action by clicking the enable button, and then start your signal generator ramping up.

61) Watch your trending chart in RSView. You will see the PV (green line) at the bottom of your chart. This PV might be representing a temperature. The red line represents your power output. Since our Kc gain is set to 1, the CV is an exact mirror of the PV.

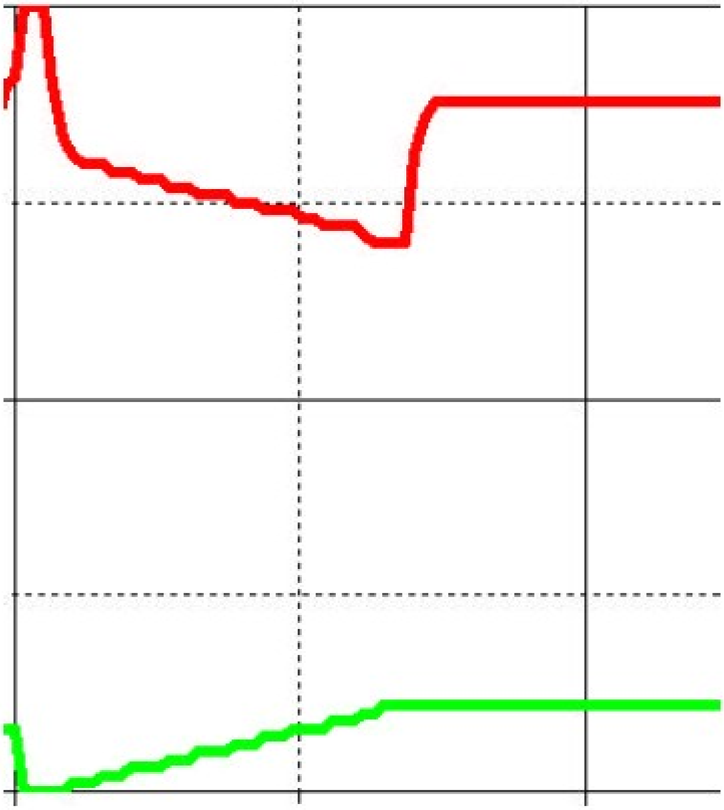
62) The chart below is what you observed with no derivative action. Notice that the output power decreases proportionally as the PV increases. If the PV stops increasing and remains the same, the Output power stops decreasing and remains the same as well.



63) Next we are going to put some derivative into the process, so you can see its effect on the output power. Before we do this, lets try to anticipate how the derivative action is going to change the output power. Remember the derivative action attempts to resist change. As long as our PV is changing, we are going to have some derivative action. If our PV stops changing, then we are no longer going to have the derivative action. The derivative sees that our temperature is increasing rapidly. Therefore, it will attempt to decrease the CV as long as this fast rate of change exists. Similarly, if the derivative sees the temperature falling rapidly, it will attempt to add output to the controller to compensate.

64) To keep you from having to switch back to RSLogix, you have a numeric entry in RSView that will allow you to change the Kd value. We are going to enter the value of 30 into RSView, and this value will show up on the setup screen of the PID instruction as .3.

65) Press the RESET button in RSView, and then enable the ramping action. Start your PV ramping up. Notice the way the Control Variable reacts. Immediately it is decreased because of the rate of change of the PV. Now stop the ramping action. You will see the CV goes back to the proportional value, and derivative action no longer exists.

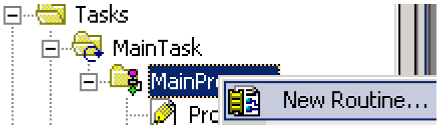


Getting Started with Structured Text in RSLogix 5000

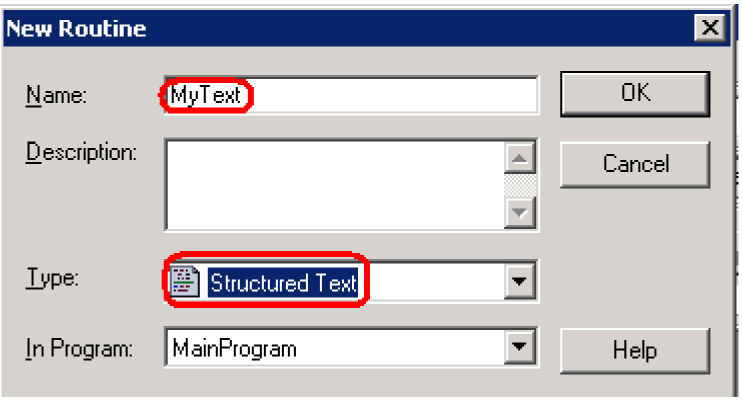
For complex algorithms, ladder logic can be too cumbersome and time consuming to program. A higher level programming language can greatly simplify your calculations. Variables can be assigned, and operated on, and loops can be used for multiple operations in the same scan.

With object oriented programming quickly becoming a standard, more programmers are beginning to use structured text as the primary means of programming PLC's. Ladder logic is based on many of the same concepts as Assembly Language. Higher level programming languages are available today eliminating the need to program with ladder logic. When this happens, those who troubleshoot and program PLC systems must be able to decipher a structured text routine.

To create a new structured text routine, right click on a program, and add a new routine as shown.



Name the routine, and specify the type as a Structured Text Routine.


New Routine

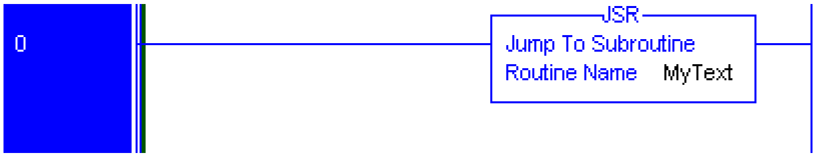
Name: MyText

Description:

Type: Structured Text

In Program: MainProgram

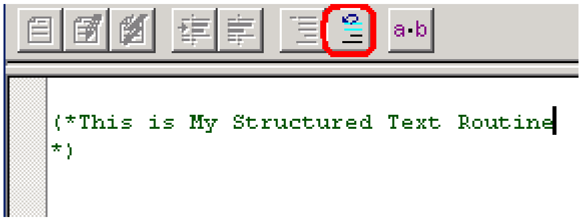
Be sure to add a JSR statement to the MainRoutine so this subroutine will execute.



Jump To Subroutine
Routine Name MyText

If you aren't use to structured text, at first these examples will seem a bit cumbersome and hard to get used to at first, but working with multiple addresses, and performing complex operations at once is much easier and simpler.

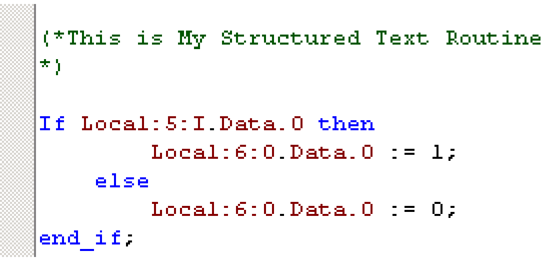
The **comment** flags allow you to document the structured text program. (Note: // can be used to comment a single line as well)



```
(*This is My Structured Text Routine
*)
```

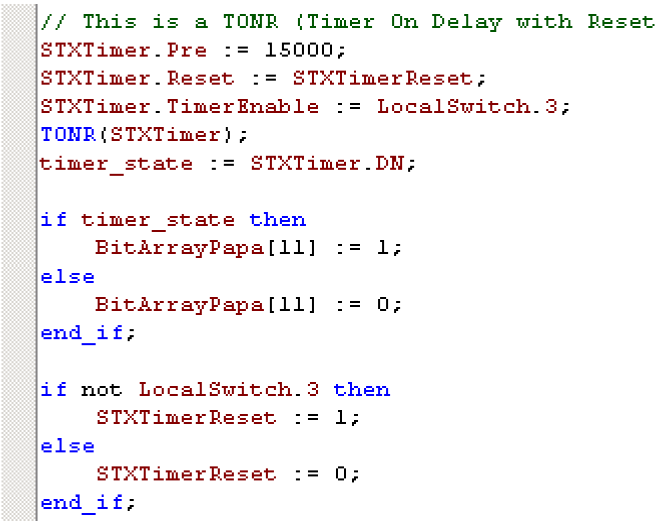
The **IF** conditional checks an expression. If the expression immediately following the IF statement returns a 1, the operation after the THEN statement will be executed. If the expression returns a 0, the operation after the ELSE statement will be executed.

In this example, if Switch 0 is ON, Light 0 will be turned on... If Switch 0 is off, light 0 will be shut off.



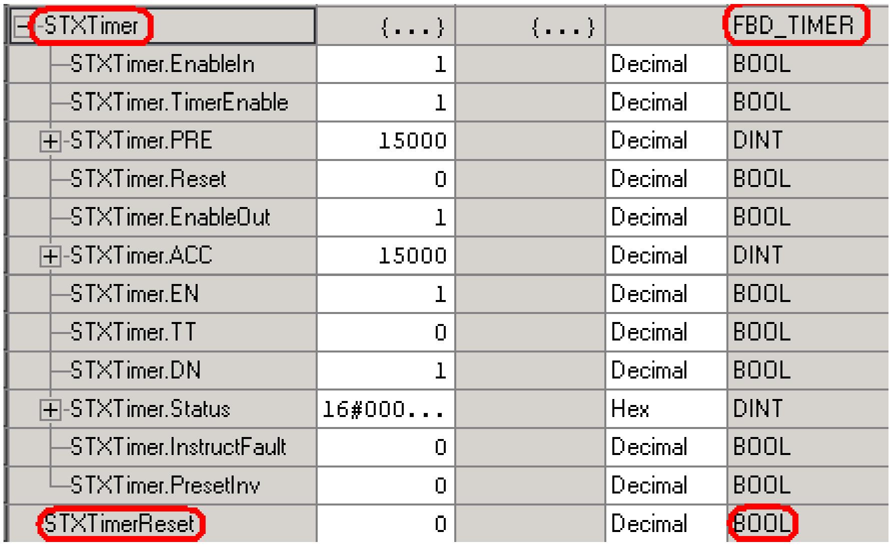
```
(*This is My Structured Text Routine
*)

If Local:5:I.Data.0 then
        Local:6:O.Data.0 := 1;
    else
        Local:6:O.Data.0 := 0;
end_if;
```

Here is an example of a Timer On Delay with Reset:



```
// This is a TONR (Timer On Delay with Reset
STXTimer.Pre := 15000;
STXTimer.Reset := STXTimerReset;
STXTimer.TimerEnable := LocalSwitch.3;
TONR(STXTimer);
timer_state := STXTimer.DN;

if timer_state then
    BitArrayPapa[11] := 1;
else
    BitArrayPapa[11] := 0;
end_if;

if not LocalSwitch.3 then
    STXTimerReset := 1;
else
    STXTimerReset := 0;
end_if;
```

STXTimer and STXTimerReset are declared in the tag database:



⊟-STXTimer	{...}	{...}		FBD_TIMER
STXTimer.EnableIn	1		Decimal	BOOL
STXTimer.TimerEnable	1		Decimal	BOOL
⊞-STXTimer.PRE	15000		Decimal	DINT
STXTimer.Reset	0		Decimal	BOOL
STXTimer.EnableOut	1		Decimal	BOOL
⊞-STXTimer.ACC	15000		Decimal	DINT
STXTimer.EN	1		Decimal	BOOL
STXTimer.TT	0		Decimal	BOOL
STXTimer.DN	1		Decimal	BOOL
⊞-STXTimer.Status	16#000...		Hex	DINT
STXTimer.InstructFault	0		Decimal	BOOL
STXTimer.PresetInv	0		Decimal	BOOL
STXTimerReset	0		Decimal	BOOL

You can now download and test your work.

For more information on Structured text, please read AB document #1756-PM001.

Hands On Troubleshooting

Your instructor will guide you through the following troubleshooting utilities:

Search/Find

Cross Reference

Trending

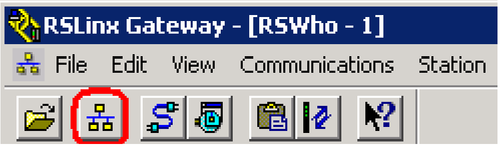
Extensive time will be provided for you to practice tracing down outputs, cross referencing through a plant program, and learning which inputs are required to energize various outputs.

Connecting to a Spreadsheet using DDE

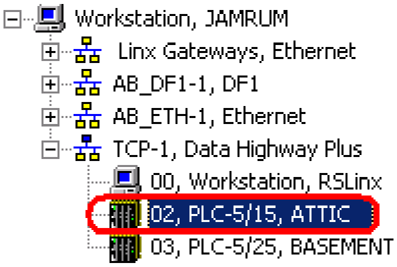
Using a spreadsheet such as OpenOffice.org Calc, or Excel, you can create a DDE (Dynamic Data Exchange) link through RSLinx. This can be useful to replace the custom data monitor feature that existed in RSLogix 5 and 500. Using a spreadsheet has a few advantages over the custom data monitor, such as the ability to perform calculations, create charts and graphs, multiple columns of data, etc... This tutorial assumes you already have a running driver and are connected to the PLC using professional version of RSLinx.

1) The first step in creating the link is to open RSLinx communication server.

2) Open the RSWho screen, and locate your processor. This can be a PLC-5, SLC-500, ControlLogix, etc... The same procedure applies to each family of processors. For this example, we will use the PLC-5

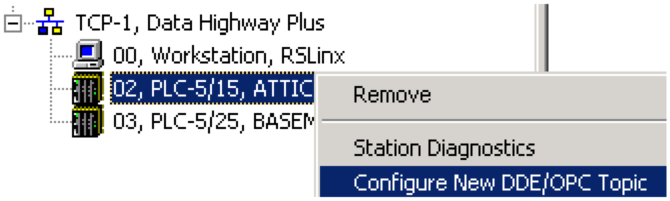


3) Next, you will want to determine which driver your processor is under. For this example, I will use the TCP Driver since I'm going through a gateway. You may use the Ethernet driver, or DF1 driver as well. Expand the driver, and locate your processor. *(Note: If you are connecting to a ControlLogix system over Ethernet, you will need to expand the Ethernet driver, drill through the Ethernet module, and across the backplane to find the processor.)*

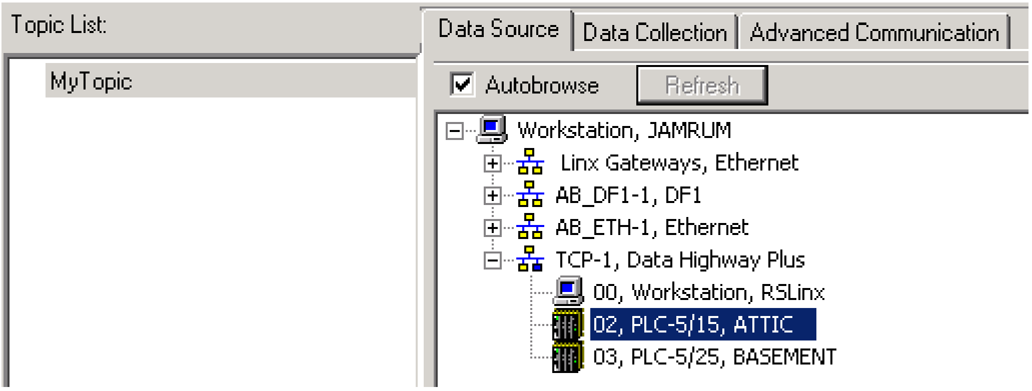


Workstation, JAMRUM
Linx Gateways, Ethernet
AB_DF1-1, DF1
AB_ETH-1, Ethernet
TCP-1, Data Highway Plus
00, Workstation, RSLinx
02, PLC-5/15, ATTIC
03, PLC-5/25, BASEMENT

4) Once, you have found the processor, right click the processor and choose '**Configure new DDE/OPC Topic**'.



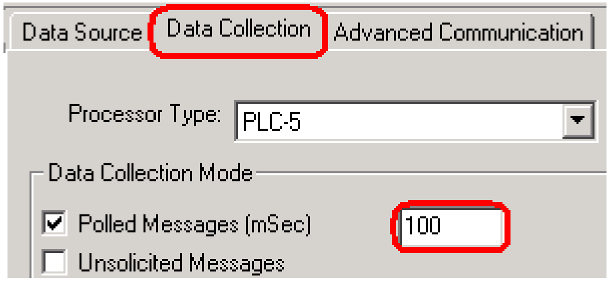
5) Next, type the name of your topic. In this example, the topic will be named '**MyTopic**', and the processor I am pointing this topic to is already highlighted.


Topic List:

MyTopic

☑ Autobrowse

6) Click the 'Data Collection' Tab. Change the Poll rate to 100ms for this example to improve the update time of our link. *(Note: In most situations, you want to keep the poll time as slow as possible to reduce load on the network.)*

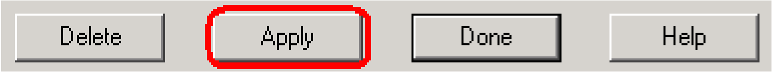


Processor Type: PLC-5

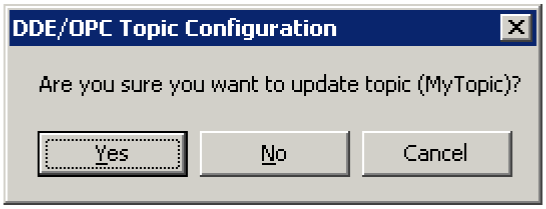
☑ Polled Messages (mSec) 100

□ Unsolicited Messages

7) Press '**Apply**'.

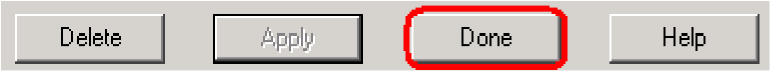


8) You will be asked to update the topic. Click '**Yes**'.


DDE/OPC Topic Configuration

Are you sure you want to update topic (MyTopic)?

9) Now click 'Done'. (Do not click Clone!)

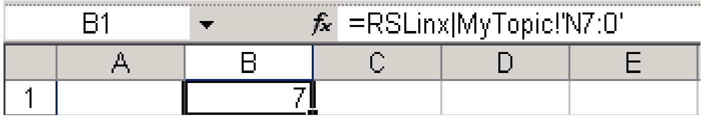


10) Next, open your spreadsheet program.

11) In a cell of your spreadsheet, enter the formula in the following format:

=RSLinx|MyTopic!'DataTableAddress'

12) The cell in your spreadsheet should now be displaying the value of the data table (or tag) that you chose to view.


B1 =RSLinx|MyTopic!'N7:0'

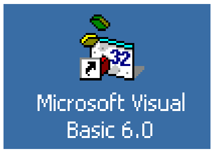
	A	B	C	D	E
1		7			

Using DDE with Visual Basic 6

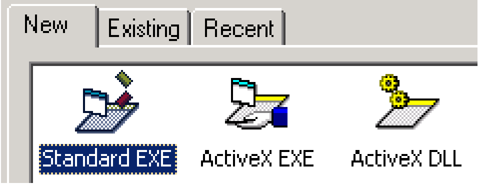
This tutorial assumes you already have a Topic set up in a professional version of RSLinx. If you have not yet configured the topic, see the section titled '**Connecting to a Spreadsheet using DDE**'. (I'm also assuming VB6 is installed on your computer) For this example, we will create a simple program that can view a memory location, and write to a memory location in the controller.

Acquiring Data from the processor

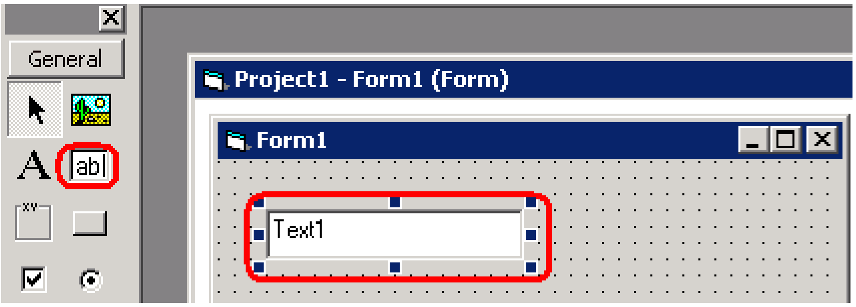
1) Open Microsoft Visual Basic 6.



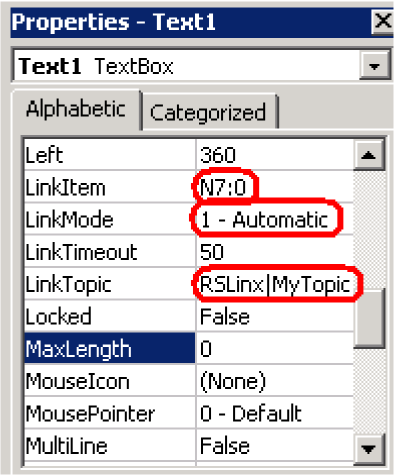
2) Choose to create a standard executable file (standard exe)



3) Click the texbox object from the general toolbox, and draw a textbox onto your form.

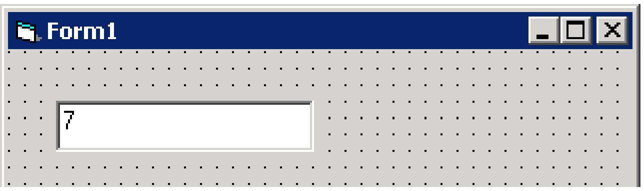


4) On the Lower left side of your IDE, you will see a properties window. (If it is not there, turn it on from the view menu. We will be setting the LinkTopic, LinkItem, and LinkMode as shown for this example. RSLinx is the application we are going through to acquire the data, MyTopic is the topic we created within RSLinx to point to the processor, and the LinkItem is the memory location you wish to acquire data from for display in the textbox object. By putting the LinkMode in automatic, the textbox will update at the intervals we specified under the data collection tab in RSLinx.

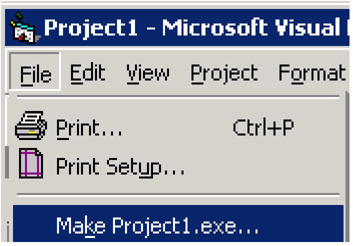


Left	360
LinkItem	N7:0
LinkMode	1 - Automatic
LinkTimeout	50
LinkTopic	RSLinx|MyTopic
Locked	False
MaxLength	0
MouseIcon	(None)
MousePointer	0 - Default
MultiLine	False

5) You will see your textbox is now updating in real time.



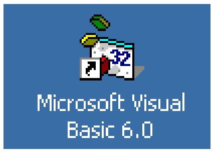
6) To build this project as an executable, click File | Make Project1.exe. You can then store the executable on your desktop or any convenient location.



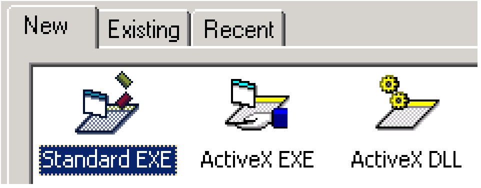
Sending data to the processor

In the last exercise, we acquired data from the processor. Occasionally, you may want to send data to the processor. We will be using the same application, RSLinx, and the same topic within that application for this exercise (MyTopic). We will change the preset of a timer such as T4:0.PRE. Since the operator will be entering a value, we don't want to get that value automatically. (We don't know if the operator has finished entering a value or not). We will want the operator to hit a button that sends the value he entered to the processor.

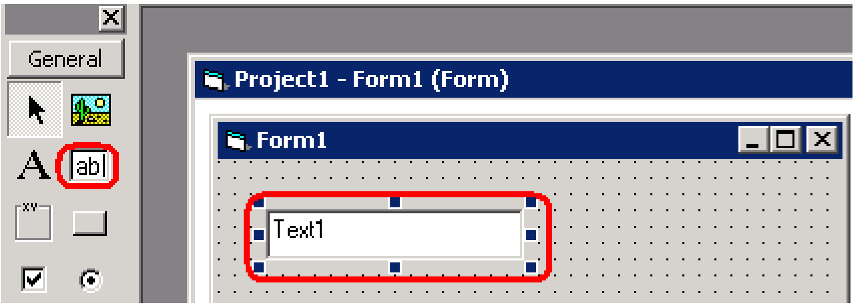
1) Open VB6 as before.



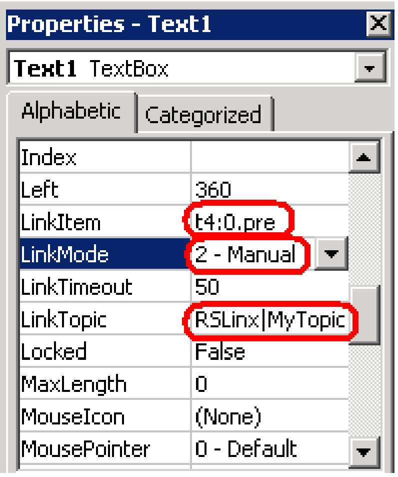
2) This will be a standard executable



3) Using the general toolbox, grab the textbox object, and draw the object on your form.



7) On the Lower left side of your IDE, you will see a properties window. (If it is not there, turn it on from the view menu. We will be setting the LinkTopic, LinkItem, and LinkMode as shown for this example. RSLinx is the application we are going through to acquire the data, MyTopic is the topic we created within RSLinx to point to the processor, and the LinkItem is the memory location you wish to acquire data from for display in the textbox object. By putting the LinkMode in manual this time, an event will be required to trigger the linkpoke command such as a command_click event. (Be sure you are looking at the properties of the textbox)



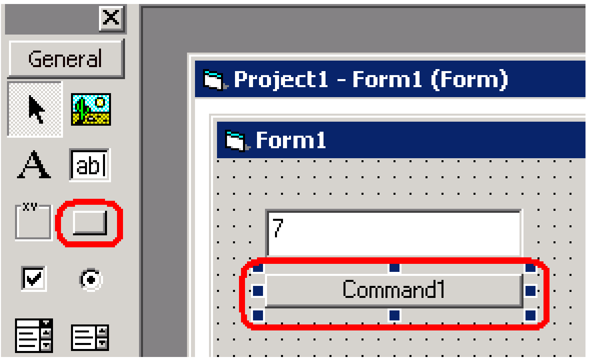
Properties - Text1

Text1 TextBox

Alphabetic | Categorized

Index	
Left	360
LinkItem	t4:0.pre
LinkMode	2 - Manual
LinkTimeout	50
LinkTopic	RSLinx|MyTopic
Locked	False
MaxLength	0
MouseIcon	(None)
MousePointer	0 - Default

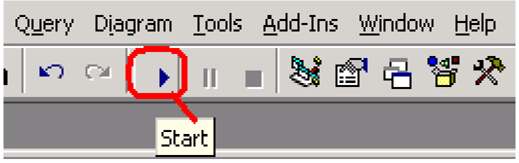
4) Next, grab a command button from the general toolbox. Draw the command button onto the form.



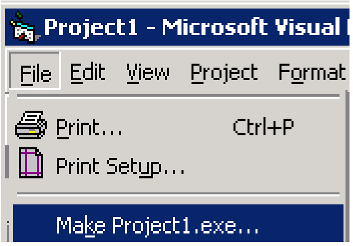
5) Double click on the command button to get to the code window. In the command1_click event (subroutine), type the following text:

Text1.LinkPoke

6) Text1 is the name of the textbox. When you press the period, the intellisense will appear giving you a list of options available for Text1. We chose the LinkPoke method because we want to send whatever value is in text 1. When the command button is pressed, the code for the command1_click event will execute, and the Text1.LinkPoke method will be executed. You can test run your project to verify it's operation.



7) When finished, you can close the form you started, and build your project as we did before.


Project1 - Microsoft Visual
File
Edit
View
Project
Format
Print...
Ctrl+P
Print Setup...
Make Project1.exe...