PLC Programming with

STUDIO 5000

Course: Level 1

Program Allen-Bradley ControlLogix and CompactLogix PLCs with Rockwell Automation's Studio 5000 and FactoryTalk View Studio

CONTENTS

CONTENTS

Part I Introduction 4

1. Communication 9
2. Install Virtual Backplane Driver 12
3. Install 5570 Logix Emulate Controller 17
4. Install 1789-SIM 32 Point Input/Output Simulator 23

Part II Studio 5000 26

1. Create New Project, Project Path, Online/Offline 32
2. Program ladder diagram 39
3. Bit Instructions 53
4. Exercise 1 - Simple Logic 56
5. Exercise 2 - NOT Logic 68
6. Exercise 4 - OR Logic 71
7. Exercise 5 - Complex Logic 74
8. Exercise 6 - Latch and Unlatch 76
9. Q&A 79

Part III 1789-SIM 32 Point Input/Output Simulator 83

Part IV Timers 101

1. Timer On Delay (TON) 102
2. Timer Off Delay (TOF) 110
3. Retentive Timer On(RTO) 116
4. Exercise 1 - Reset Timer 123
5. Exercise 2 - Flashing Light 125

Part V Counters 126

1. Count Up (CTU) 127
2. Count Down (CTD) 132
3. Exercise - Counter Up and Down 136

Part VI Math and Compute Instructions 137

1. Exercise 1 - Math Instructions 142
2. Exercise 2 - Compute (CPT) 146

Part VII Compare Instructions 149

1. Exercise 1 - Compare Instructions 156
2. Exercise 2 - AND condition 160
3. Exercise 3 - OR condition 161

Part VIII FactoryTalk View Studio 163

1. Exercise 1 - RSLinx Enterprise Communication Setup 165
2. Exercise 2 - Graphic 172
3. Exercise 3 - Libraries 187
4. Exercise 4 - Animations 197
5. Exercise 5 - Animation with Max and Min Range 200

Part IX Introduction To Servo And Virtual Axis 205

Part X Assignments 217

1. Break Before Make (15%) 218
2. Traffic Light (15%) 220
3. Compressor (15%) 223
4. Temperature Conversion (15%) 225
5. Chemical Batch Mixer Application 225

Part I Introduction

Programmable Logic Controller

A **Programmable Logic Controller**, **PLC**, or **Programmable Controller** is a digital computer used for automation of industrial processes, such as control of machinery on factory assembly lines. Unlike general-purpose computers, the PLC is designed for multiple inputs and output arrangements, extended temperature ranges, immunity to electrical noise, and resistance to vibration and impact. Programs to control machine operation are typically stored in battery-backed or non-volatile memory. A PLC is an example of a real time system since output results must be produced in response to input conditions within a bounded time, otherwise unintended operation will result.

Features

The main difference from other computers is that PLC are armored for severe condition (dust, moisture, heat, cold, etc) and have the facility for extensive input/output (I/O) arrangements. These connect the PLC to sensors and actuators. PLCs read limit switches, analog process variables (such as temperature and pressure), and the positions of complex positioning systems. Some even use machine vision. On the actuator side, PLCs operate electric motors, pneumatic or hydraulic cylinders, magnetic relays or solenoids, or analog outputs. The input/output arrangements may be built into a simple PLC, or the PLC may have external I/O modules attached to a computer network that plugs into the PLC.

PLCs were invented as replacements for automated systems that would use hundreds or thousands of relays, cam timers, and drum sequencers. Often, a single PLC can be programmed to replace thousands of relays. Programmable controllers were initially adopted by the automotive manufacturing industry, where software revision replaced the re-wiring of hard-wired control panels when production models changed.

Many of the earliest PLCs expressed all decision making logic in simple ladder logic which appeared similar to electrical schematic diagrams. The electricians were quite able to trace out circuit problems with schematic diagrams using ladder logic. This program notation was chosen to reduce training demands for the existing technicians. Other early PLCs used a form of instruction list programming, based on a stack-based logic solver.

The functionality of the PLC has evolved over the years to include sequential relay control, motion control, process control, distributed control systems and networking. The data handling, storage, processing power and communication capabilities of some modern PLCs are approximately equivalent to desktop computers. PLC-like programming combined with remote I/O hardware, allow a general-purpose desktop computer to overlap some PLCs in certain applications.

Under the IEC 61131-3 standard, PLC's can be programmed using standards-based programming languages. A graphical programming notation called Sequential Function Charts is available on certain programmable controllers.

PLC compared with other control systems

PLCs are well-adapted to a certain range of automation tasks. These are typically industrial processes in manufacturing where the cost of developing and maintaining the automation system is high relative to the total cost of the automation, and where changes to the system would be expected during its operational life. PLCs contain input and output devices compatible with industrial pilot devices and controls; little electrical design is required, and the design problem centers on expressing the desired sequence of operations in ladder logic (or function chart) notation. PLC applications are typically highly customized systems so the cost of a packaged PLC is low compared to the cost of a specific custom-built controller design. On the other hand, in the case of mass-produced goods, customized control systems are economic due to the lower cost of the components, which can be optimally chosen instead of a "generic" solution, and where the non-recurring engineering charges are spread over thousands of sales.

For high volume or very simple fixed automation tasks, different techniques are used. For example, a consumer dishwasher would be controlled by an electromechanical cam timer costing only a few dollars in production quantities.

A microcontroller-based design would be appropriate where hundreds or thousands of units will be produced and so the development cost (design of power supplies and input/output hardware) can be spread over many sales, and where the end-user would not need to alter the control. Automotive applications are an example; millions of units are built each year, and very few end-users alter the programming of these controllers. However, some specialty vehicles such as transit busses economically use PLCs instead of custom-designed controls, because the volumes are low and the development cost would be uneconomic.

Very complex process control, such as used in the chemical industry, may require algorithms and performance beyond the capability of even high-performance PLCs. Very high-speed or precision controls may also require customized solutions; for example, aircraft flight controls.

PLCs may include logic for single-variable feedback analog control loop, a "proportional, integral, derivative" or "PID controller." A PID loop could be used to control the temperature of a manufacturing process, for example. Historically PLCs were usually configured with only a few analog control loops; where processes required hundreds or thousands of loops, a distributed control system (DCS) would instead be used. However, as PLCs have become more powerful, the boundary between DCS and PLC applications has become less clear-cut.

Digital and analog signals

Digital or discrete signals behave as binary switches, yielding simply an On or Off signal (1 or 0, True or False, respectively). Pushbuttons, limit switches, and photoelectric sensors are examples of devices providing a discrete signal. Discrete signals are sent using either voltage or current, where a specific range is designated as On and another as Off. For example, a PLC might use 24 V DC I/O, with values above 22 V DC representing On, values below 2VDC representing Off, and intermediate values undefined. Initially, PLCs had only discrete I/O.

Analog signals are like volume controls, with a range of values between zero and full-scale. These are typically interpreted as integer values (counts) by the PLC, with various ranges of

accuracy depending on the device and the number of bits available to store the data. As PLCs typically use 16-bit signed binary processors, the integer values are limited between -32,768 and +32,767. Pressure, temperature, flow, and weight are often represented by analog signals. Analog signals can use voltage or current with a magnitude proportional to the value of the process signal. For example, an analog 4-20 mA or 0 - 10 V input would be converted into an integer value of 0 – 32767.

Current inputs are less sensitive to electrical noise (i.e. from welders or electric motor starts) than voltage inputs.

Example

As an example, say the facility needs to store water in a tank. The water is drawn from the tank by another system, as needed and our example system must manage the water level in the tank.

Using only digital signals, the PLC has two digital inputs from float switches (tank empty and tank full). The PLC uses a digital output to open and close the inlet valve into the tank.

If both float switches are off (down) or only the 'tank empty' switch is on, the PLC will open the valve to let more water in. Once the 'tank full' switch is on, the PLC will automatically shut the inlet to stop the water from overflowing. If only the 'tank full' switch is on, something is wrong because once the water reaches a float switch, the switch will stay on because it is floating, thus, when both float switches are on, the tank is full. Two float switches are used to prevent a 'flutter' (a ripple or a wave) condition where any water usage activates the pump for a very short time and then deactivates for a short time, and so on, causing the system to wear out faster.

An analog system might use a load cell (scale) that weighs the tank, and an adjustable (throttling) valve. The PLC could use a PID feedback loop to control the valve opening. The load cell is connected to an analog input and the valve is connected to an analog output. This system fills the tank faster when there is less water in the tank. If the water level drops rapidly, the valve can be opened wide. If water is only dripping out of the tank, the valve adjusts to slowly drip water back into the tank.

In this system, to avoid 'flutter' adjustments that can wear out the valve, many PLCs incorporate "hysteresis" which essentially creates a "deadband" of activity. A technician adjusts this deadband so the valve moves only for a significant change in rate. This will in turn minimize the motion of the valve, and reduce its wear.

A real system might combine both approaches, using float switches and simple valves to prevent spills, and a rate sensor and rate valve to optimize refill rates. Backup and maintenance methods can make a real system very complicated.

System Scale

A small PLC will have a fixed number of connections built in for inputs and outputs. Typically, expansions are available if the base model does not have enough I/O.

Modular PLCs have a chassis (also called a rack) into which is placed modules with different functions. The processor and selection of I/O modules is customised for the particular application. Several racks can be administered by a single processor, and may have thousands of inputs and outputs. A special high speed serial I/O link is used so that racks can be distributed away from the processor, reducing the wiring costs for large plants.

PLCs used in larger I/O systems may have peer-to-peer (P2P) communication between processors. This allows separate parts of a complex process to have individual control while allowing the subsystems to co-ordinate over the communication link. These communication links are also often used for HMI (Human-Machine Interface) devices such as keypads or PC-type workstations. Some of today's PLCs can communicate over a wide range of media including RS-485, Coaxial, and even Ethernet for I/O control at network speeds up to 100 Mbit/s.

Programming

Early PLCs, up to the mid-1980s, were programmed using proprietary programming panels or special-purpose programming terminals, which often had dedicated function keys representing the various logical elements of PLC programs. Programs were stored on cassette tape cartridges. Facilities for printing and documentation were very minimal due to lack of memory capacity. More recently, PLC programs are typically written in a special application on a personal computer, then downloaded by a direct-connection cable or over a network to the PLC. The very oldest PLCs used non-volatile magnetic core memory but now the program is stored in the PLC either in battery-backed-up RAM or some other non-volatile flash memory.

Early PLCs were designed to be used by electricians who would learn PLC programming on the job. These PLCs were programmed in "ladder logic", which strongly resembles a schematic diagram of relay logic. Modern PLCs can be programmed in a variety of ways, from ladder logic to more traditional programming languages such as BASIC and C. Another method is State Logic, a Very High Level Programming Language designed to program PLCs based on State Transition Diagrams.

Recently, the International standard IEC 61131-3 has become popular. IEC 61131-3 currently defines five programming languages for programmable control systems: FBD (Function block diagram), LD (Ladder diagram), ST (Structured text, similar to the Pascal programming language), IL (Instruction list, similar to assembly language) and SFC (Sequential function chart). These techniques emphasize logical organization of operations.

While the fundamental concepts of PLC programming are common to all manufacturers, differences in I/O addressing, memory organization and instruction sets mean that PLC programs are never perfectly interchangeable between different makers. Even within the same product line of a single manufacturer, different models may not be directly compatible.

User Interface

PLCs may need to interact with people for the purpose of configuration, alarm reporting or everyday control. A Human-Machine Interface (HMI) is employed for this purpose.

A simple system may use buttons and lights to interact with the user. Text displays are available as well as graphical touch screens. Most modern PLCs can communicate over a network to some other system, such as a computer running a SCADA (Supervisory Control And Data Acquisition) system or web browser.

Communication

PLCs usually have built in communications ports usually 9-Pin RS232, and optionally for RS485 and Ethernet. Modbus or DF1 is usually included as one of the communications protocols. Others' options include various fieldbuses such as DeviceNet or Profibus. Other communications protocols that may be used are listed in the List of automation protocols.

History

The PLC was invented in response to the needs of the American automotive industry. Before the PLC, control, sequencing, and safety interlock logic for manufacturing automobiles was accomplished using relays, timers and dedicated closed-loop controllers. The process for updating such facilities for the yearly model change-over was very time consuming and expensive, as the relay systems needed to be rewired by skilled electricians. In 1968 GM Hydramatic (the automatic transmission division of General Motors) issued a request for proposal for an electronic replacement for hard-wired relay systems.

The winning proposal came from Bedford Associates of Bedford, Massachusetts. The first PLC, designated the 084 because it was Bedford Associates eighty-fourth project, was the result. Bedford Associates started a new company dedicated to developing, manufacturing, selling, and servicing this new product: Modicon, which stood for MOdular DIgital CONtroller. One of the people who worked on that project was Dick Morley, who is considered to be the "father" of the PLC. The Modicon brand was sold in 1977 to Gould Electronics, and later acquired by German Company AEG and then by Schneider Electric, the current owner.

One of the very first 084 models built is now on display at Modicon's headquarters in North Andover, Massachusetts. It was presented to Modicon by GM, when the unit was retired after nearly twenty years of uninterrupted service.

The automotive industry is still one of the largest users of PLCs, and Modicon still numbers some of its controller models such that they end with eighty-four. PLCs are used in many different industries and machines such as packaging and semiconductor machines. Well known PLC brands are Allen-Bradley, Mitsubishi Electric, ABB Ltd., Koyo, Honeywell, Siemens, Modicon, Omron, General Electric and Panasonic (a brand name of Matsushita).

1. COMMUNICATION

Connecting PLCs to a computer

All hardware devices you connect to a computer need a "Driver" in order for the software running on that computer to communicate with the device. If you for example connect a modem or a printer to your computer you need to install the driver for that particular device. With this driver you tell the computer (configure) what protocol you are going to use and what media the device is connected. A printer for example could be connected to a computer via a parallel cable, a serial cable, USB, Ethernet or even wireless using the infrared port.

In order to communicate with a PLC there are too many different options to just install a single driver. For Allen Bradley PLC's you have to use a program called "RSLinx". This software package has all the drivers for the different PLC's, networks and their associated communication hardware available.

So, what is RSLinx actually?

RSLinx is a communication server used to connect programming software like the RSLogix product family to Allen Bradley PLCs.

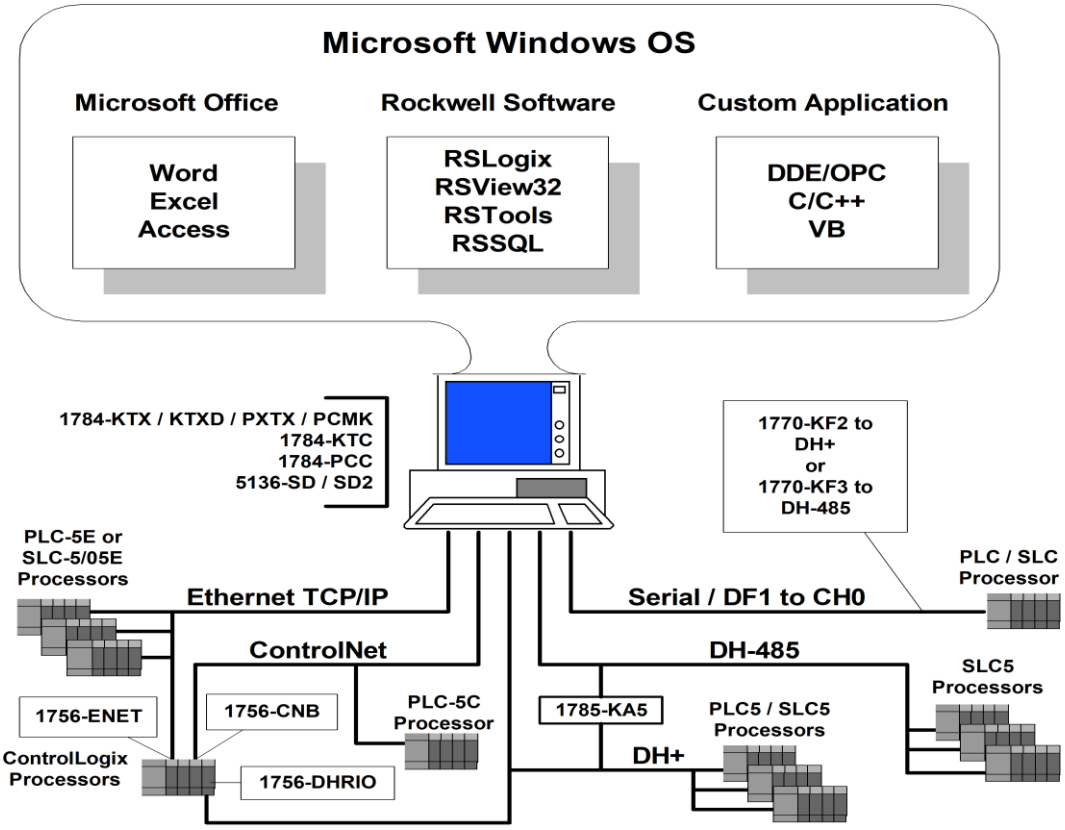
Communication package

First RSLinx acts as a communications package between Allen Bradley PLC's and RSLogix programming software. In order to do this you need at least RSLinx Lite although the other packages of RSLinx will do this also. I will discus the different packages of RSLinx later. RSLinx Lite comes bundled with several software packages from Rockwell Software like RSLogix , Panelbuilder, Drive Explorer and more.

A DDE/OPC Server

RSLinx also functions as a DDE or OPC Server to DDE or OPC compliant applications. RSLinx Lite cannot work as a DDE or OPC server. RSLinx can pull data from any PLC data

table and serve it up to applications that request data from it such as Microsoft Excel, Access or Microsoft Word. (see picture on the following page).



Protocol

What is a protocol:

You can compare a protocol with a language. It is a set of rules that has to be used by 2 devices to be able to "understand" each other. Like a phone conversation, you need to speak the same language on both sides to get a conversation going. When you talk to a person on a phone, you don't know what "Media" is being used. Of course you know yours; you are on a regular phone or a cell phone. But you don't know how it is connected to the other side. The same goes for PLC's, you can connect 2 Ethernet PLC's together using fiber ,wireless or traditional CAT5 cable. But if one PLC is for example Allen Bradley and the other a Siemens PLC, you will have the 2 devices connected together but no conversation is possible because they use a different protocol (Language). As with a phone, you can dial the right amount of digits to get a person on the phone in another country but if they do not speak the same language nothing will work.

So, after reading the above section you realize that 2 things have to be selected in RSLinx in order to set up the communication between the computer and the PLC.

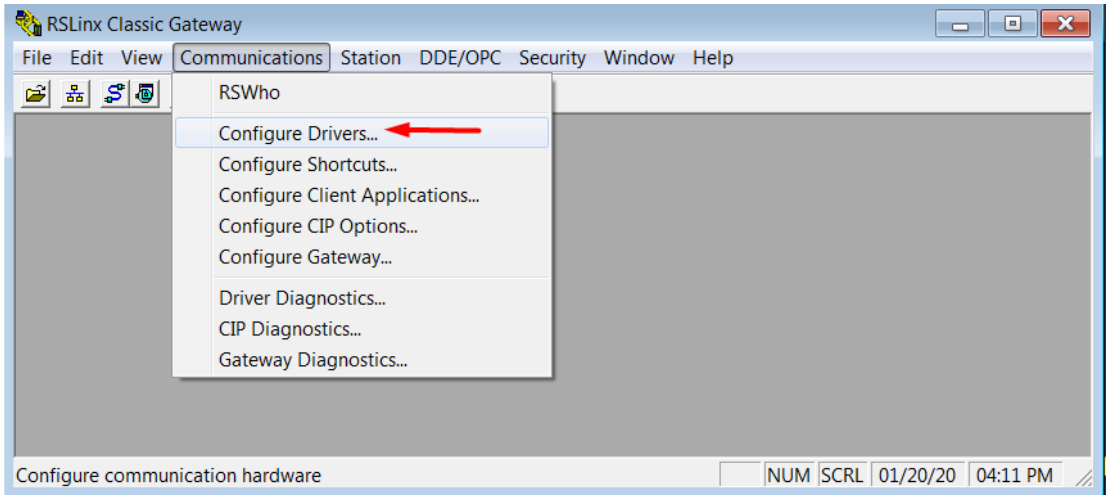
- The Protocol you plan on using.
- The hardware you use to connect to the network.

2. INSTALL VIRTUAL BACKPLANE DRIVER

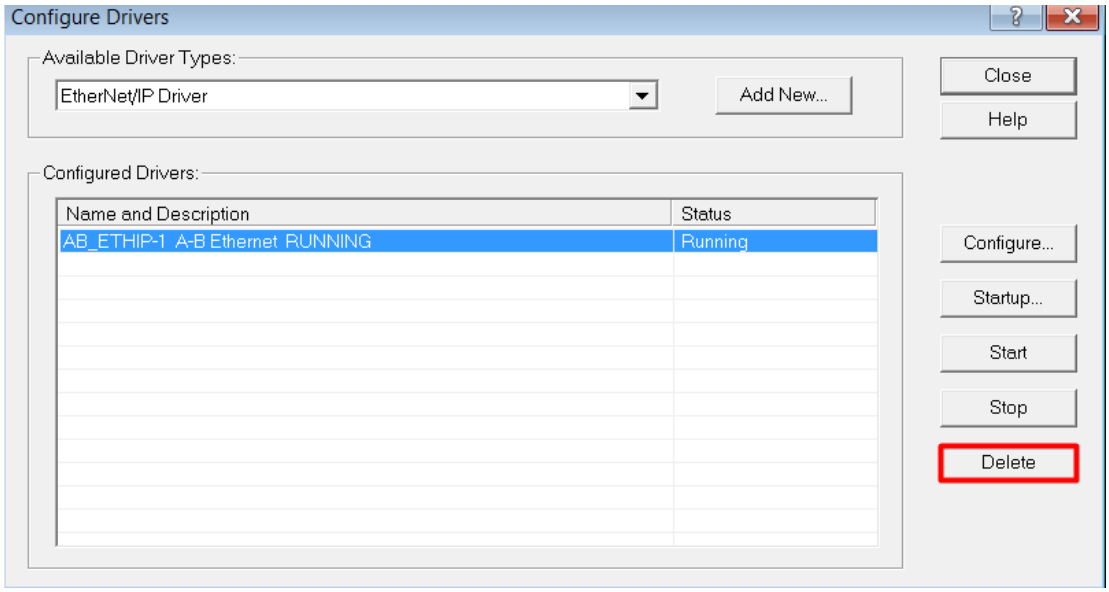
RSLinx® Classic is the most widely-installed Communications software in automation today. All editions of **RSLinx® Classic** deliver the ability to browse your automation networks, configure and diagnose network devices.

Let's setup a drive in RSLinx® Classic by double click on the RSLinx icon located on the desktop.

Go to menu and select **Communications** -> **Configure Drivers**.

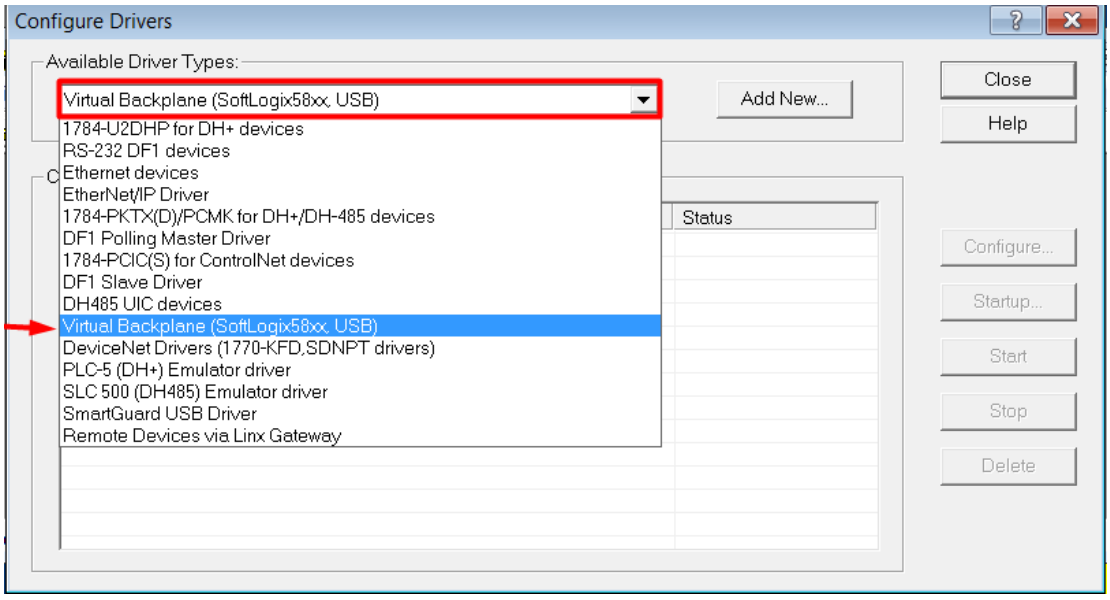


Installed drivers will display in the Configured Drivers window. You can delete driver by highlight the driver then click on the **Delete** button to apply the change. Do not worry if you deleted a driver by accident. You can always add them back.

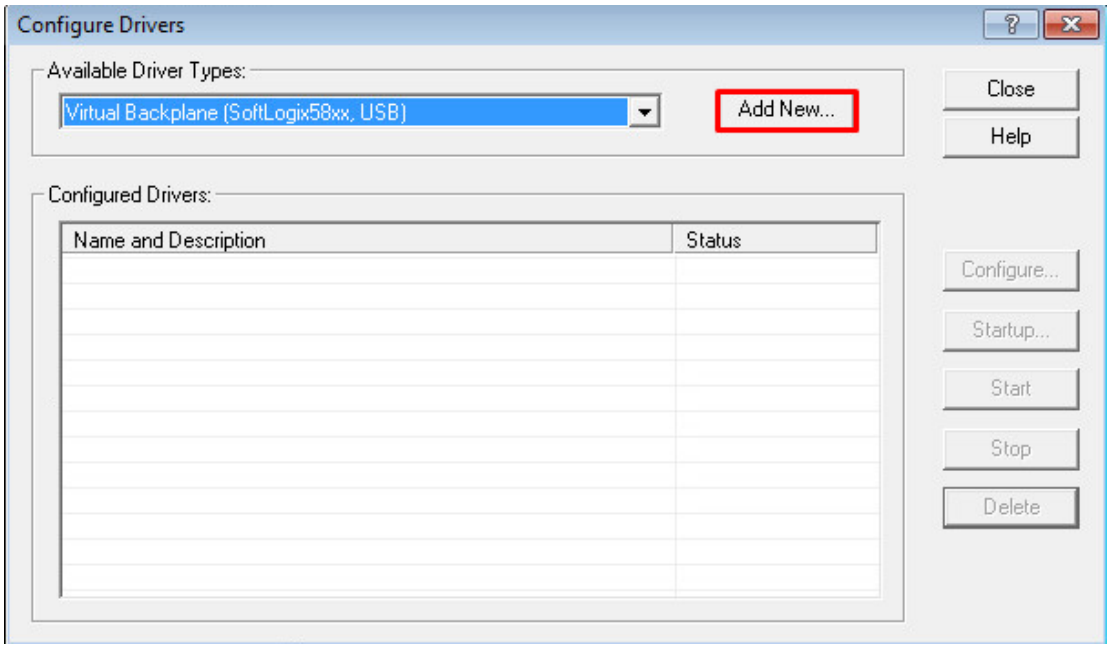


Let's install a **Virtual Backplane (SoftLogix58xx, USB)** driver. Virtual Backplane driver can be used for Emulate 5000 or SoftLogix5800 controller. Only one driver can be installed for this type.

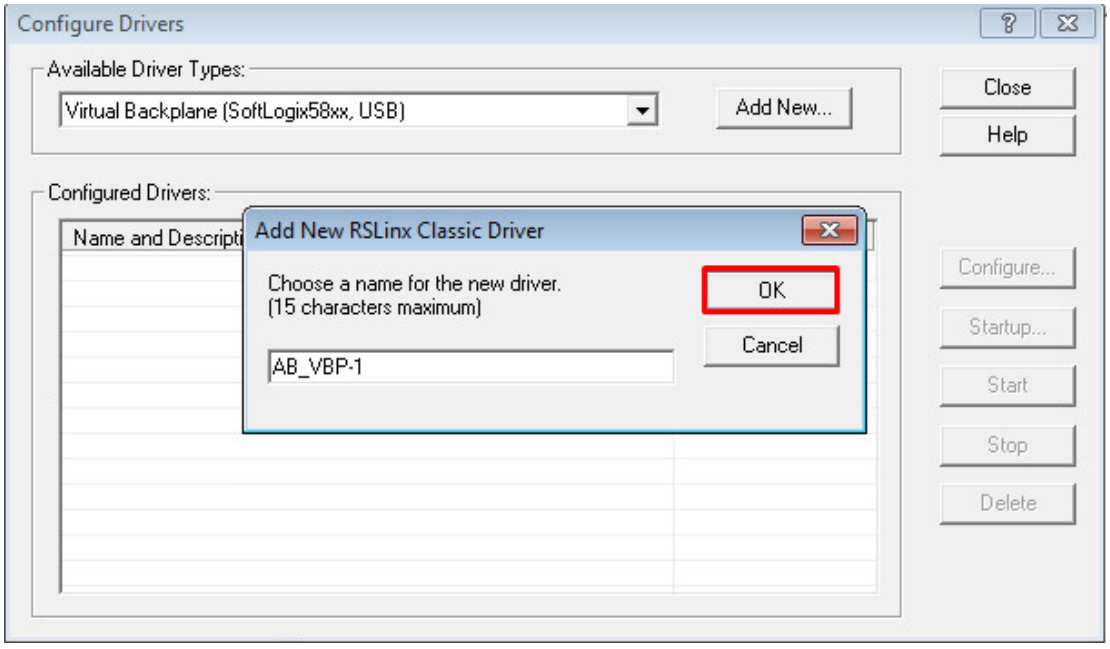
First click on the Available Driver Types drop down menu then select **Virtual Backplane (SoftLogix58xx, USB)**.



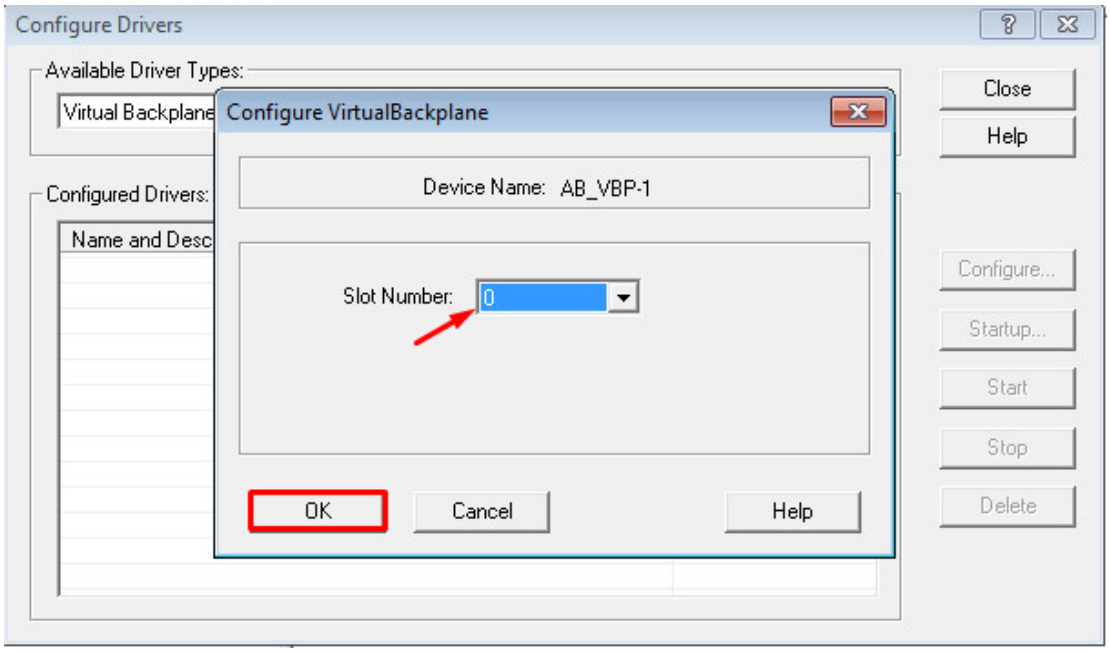
Click on the **Add New...** button to continue.



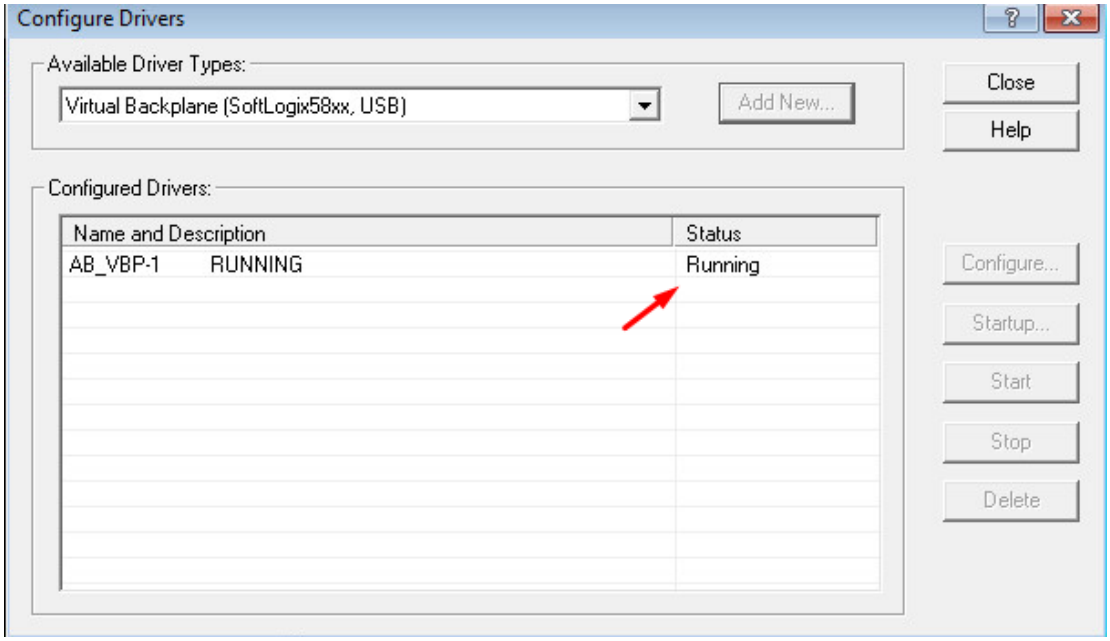
Keep the driver name as **AB_VBP-1**. Click **OK** button to continue.



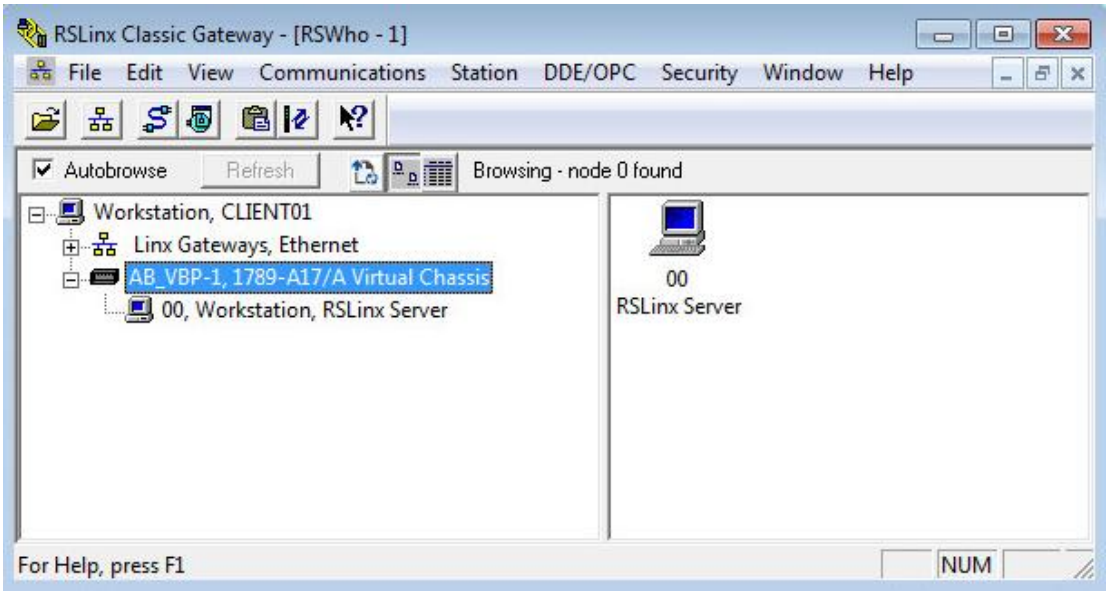
Use the default Slot Number (**0**), click the **OK** button to finish the installation. Do not change the default slot number otherwise will cause communication issue with RSLinx.



Finally check to make sure the driver is in Running under status. Click the **Close** button to exit the Configure Drivers window.



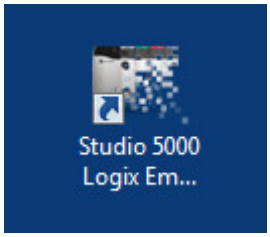
Close or minimize the RSLinx Classic window.


RSLinx Classic Gateway - [RSWho - 1]
File Edit View Communications Station DDE/OPC Security Window Help
Autobrowse
Refresh
Browsing - node 0 found
Workstation, CLIENT01
Linx Gateways, Ethernet
AB_VBP-1, 1789-A17/A Virtual Chassis
00, Workstation, RSLinx Server
00
RSLinx Server
For Help, press F1
NUM

3. INSTALL 5570 LOGIX EMULATE CONTROLLER

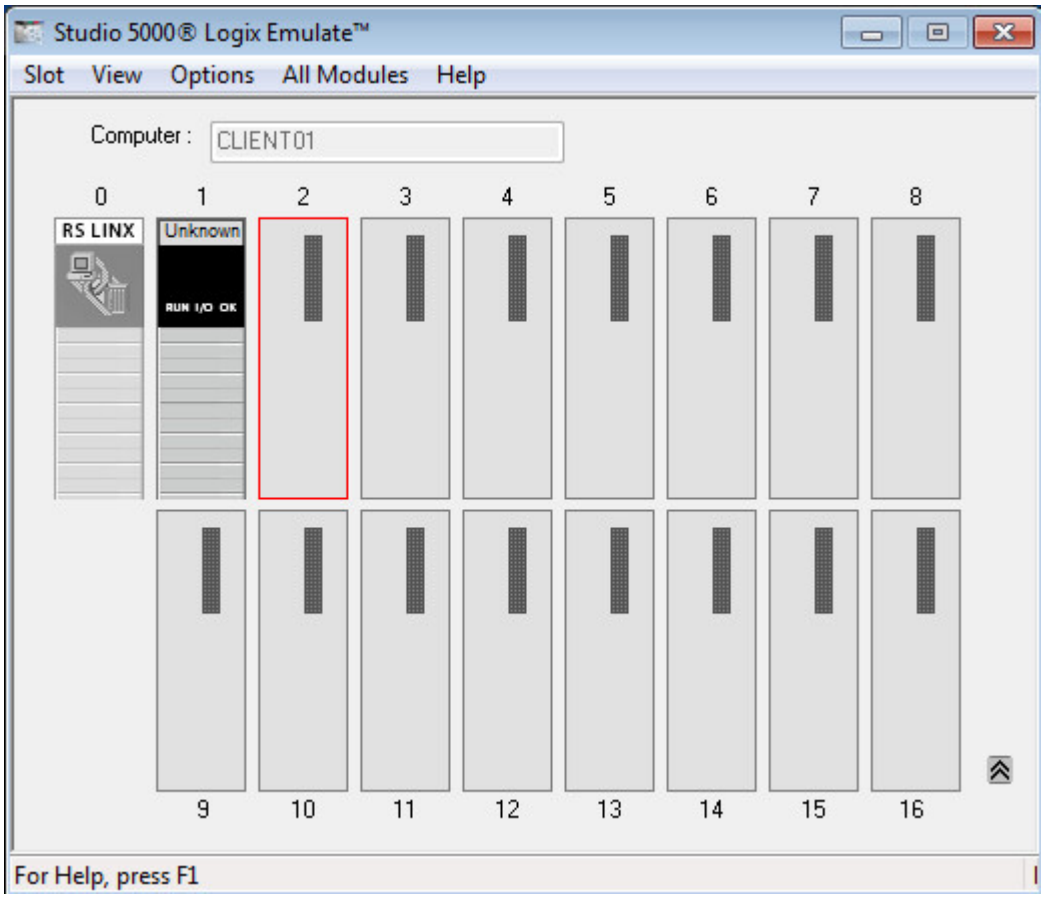
RSLogix Emulator 5000 is a software simulator for the Allen Bradley line of Logix **5000** controllers (ControlLogix®, CompactLogix®, FlexLogix®, SoftLogix5800® and DriveLogix®). The goal is to mimic the function of a PLC without the actual hardware and thus do advanced debugging.

Let's install a controller in the Studio 5000® Logix Emulate chassis by double click on the Studio 5000 Logix Emulate icon located on the desktop.



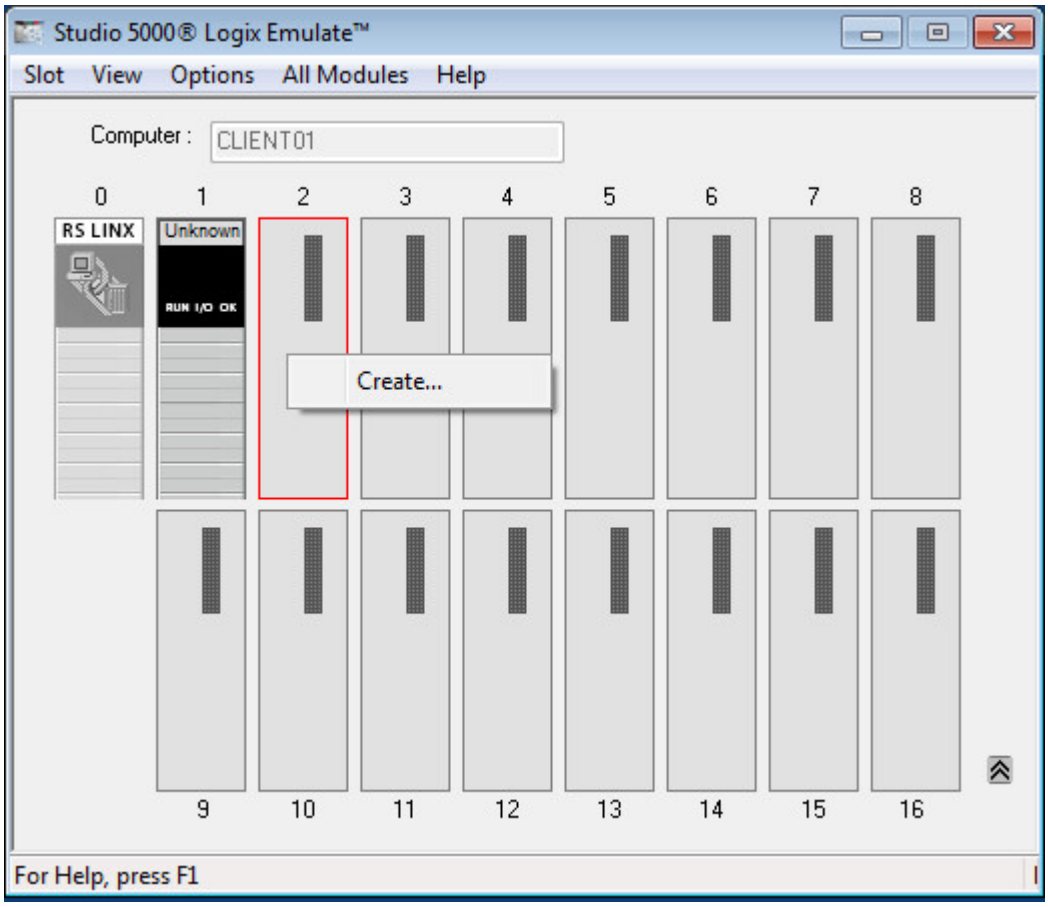
Studio 5000® Logix Emulate™ allows programmer to emulate project from version 20 to the latest version. A combination of controllers and Simulation cards can be installed in the chassis from slot **2** to**16**.

DO NOT remove cards in slot **0** & **1**. They are reserved for RSLinx Classic and RSLinx Enterprise communication.

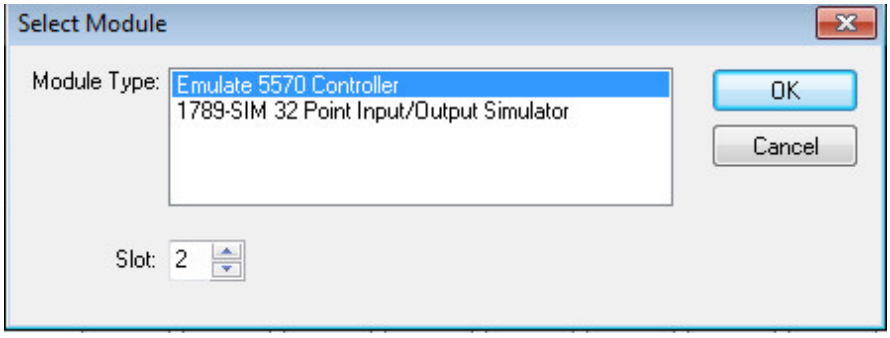


Let's add a **5570 Controller** in slot 2, and **1789-SIM 32 Point Input/Output Simulator** in slot 3.

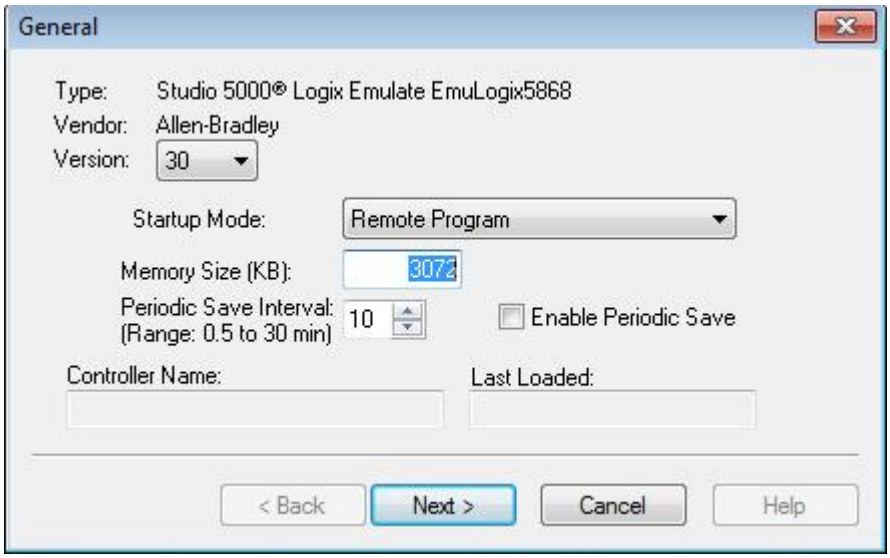
First highlight slot 2 and click the right mouse button. Select **Create....**



Select **Emulate 5570 Controller** and click the **OK** button to continue.



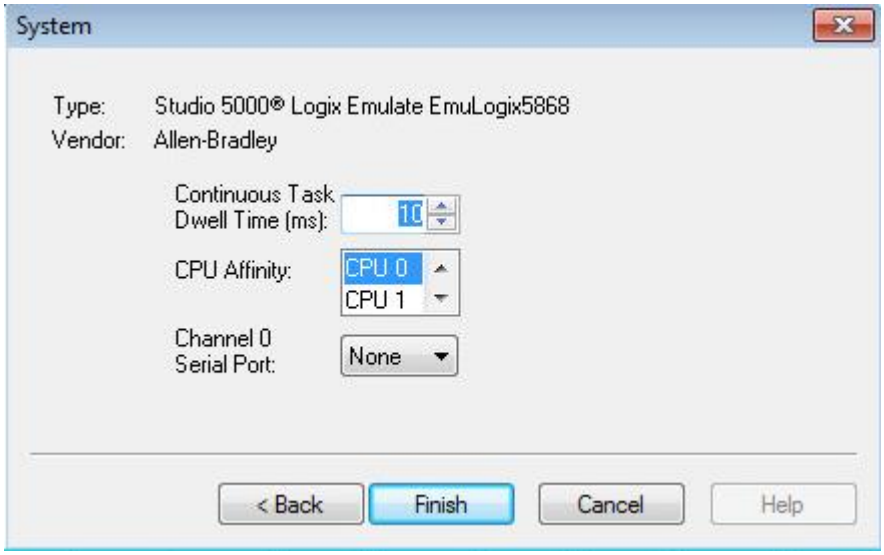
Click on the **Next** button to continue.



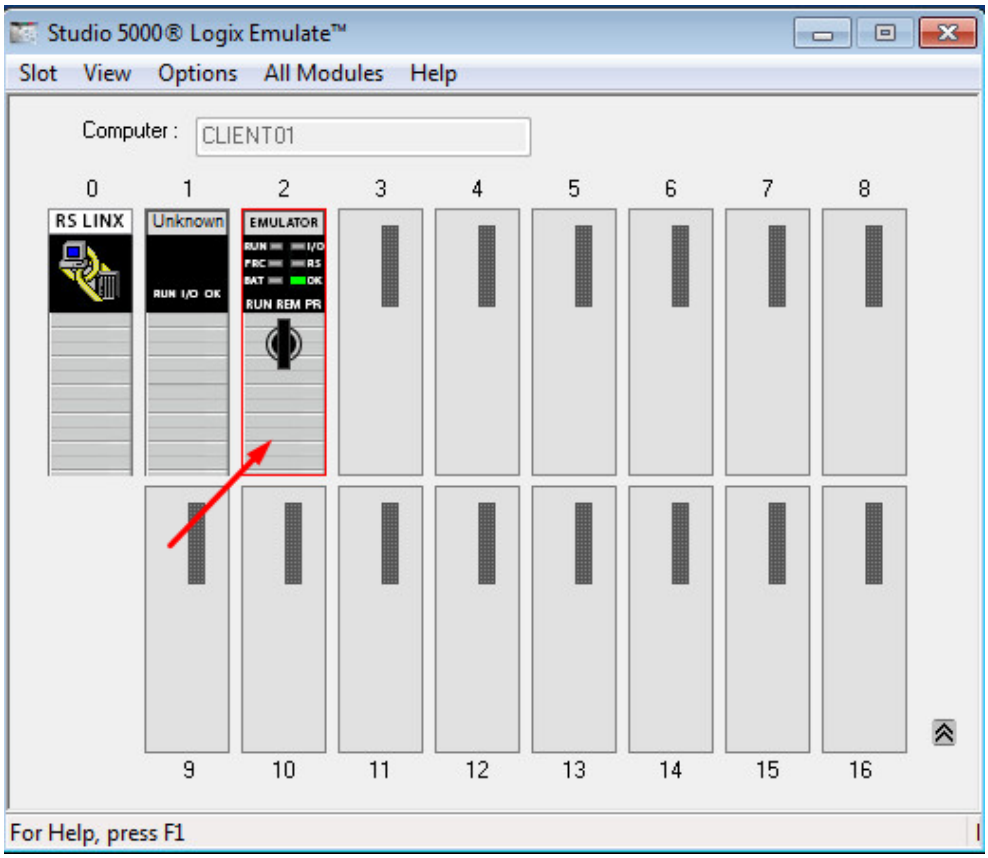
The startup Mode determines what state the controller will be in when the PLC is restarted. Most PLCs will start up in the RUN mode so you would have to select "Last Controller State" if you want that to happen. The Memory Size let you select the amount of PC memory that will be reserved for the emulator Controller.

The Periodic Save Interval determines how often the complete program is written to the hard drive of the PC if enabled.

Click on the **Finish** button to complete the setup.

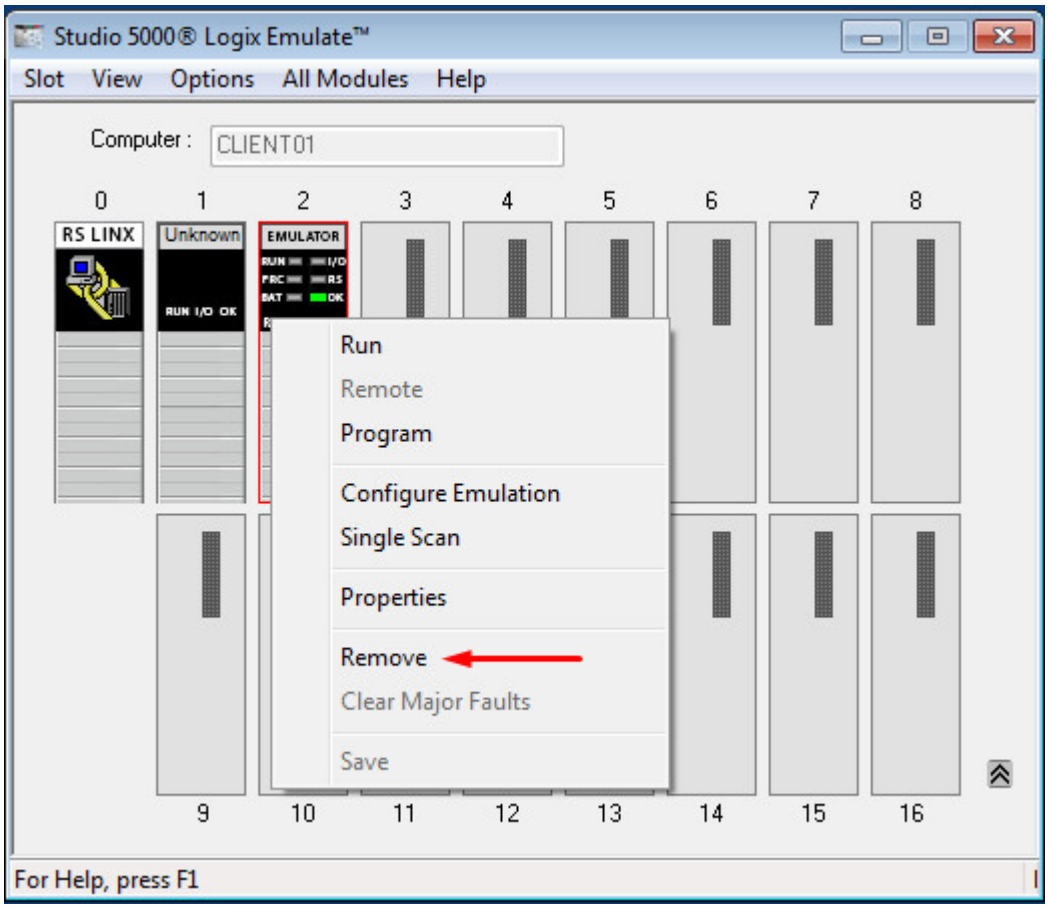


The new controller will appear in slot 2.

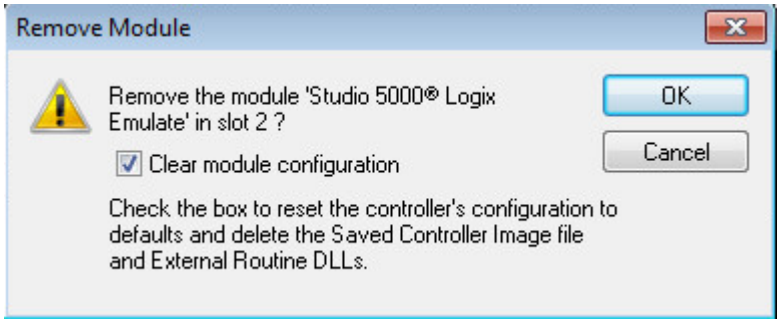


Here is how to remove a controller or simulate I/O card from the chassis.

Click the right mouse button on the controller or simulation I/O card then select **Remove**.



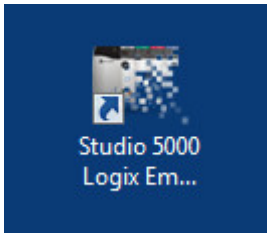
Make sure the **Clear module configuration** box is checked. Click the **OK** button to complete the removal.



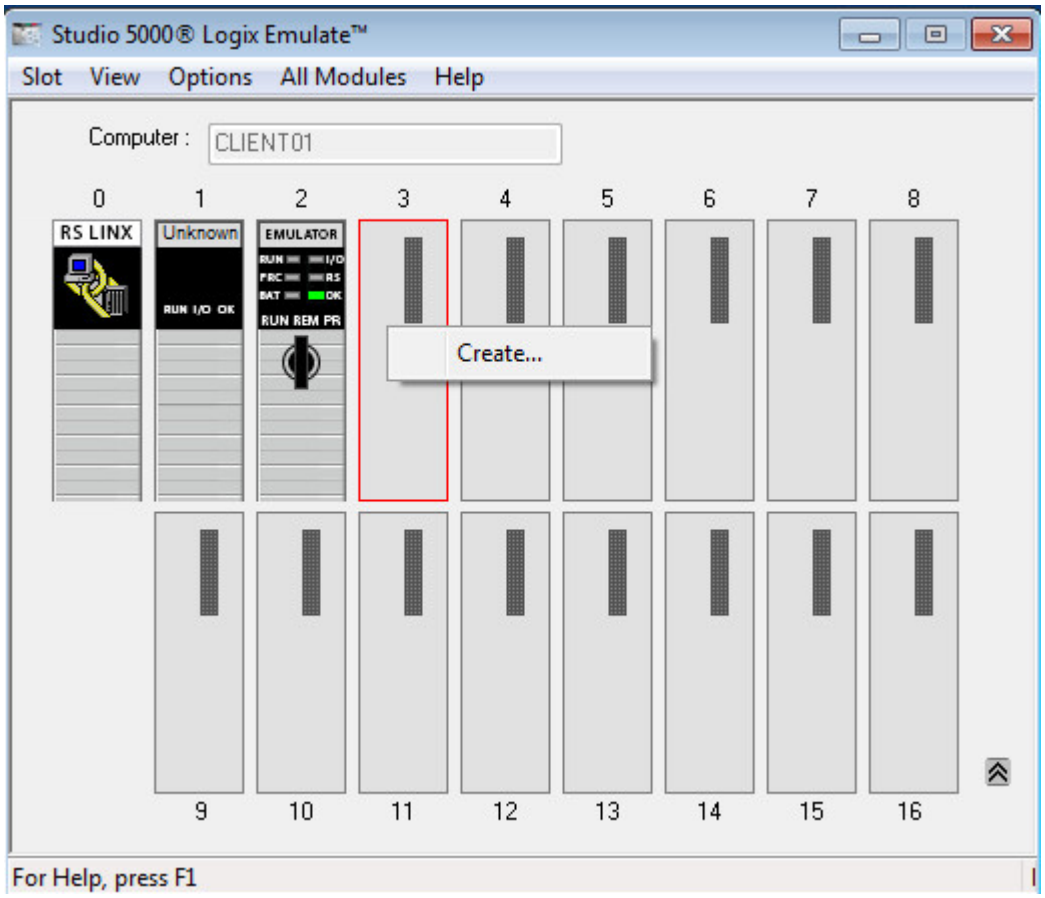
For more information: https://literature.rockwellautomation.com/idc/groups/literature/documents/gr/lgem5k-gr016_-en-p.pdf

4. INSTALL 1789-SIM 32 POINT INPUT/OUTPUT SIMULATOR

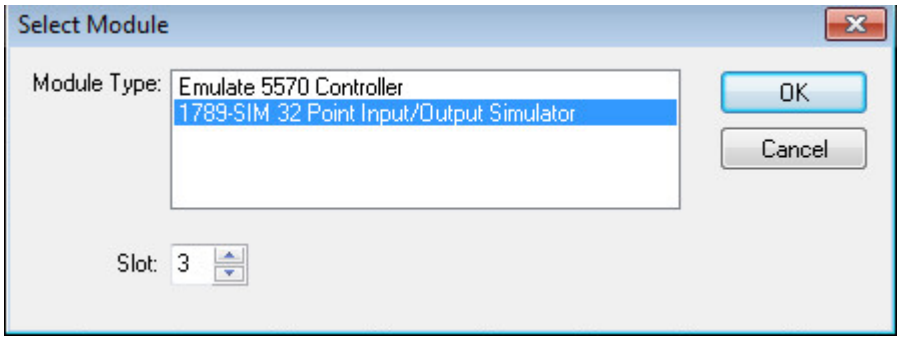
Double click on the Studio 5000 Logix Emulate icon located on the desktop.



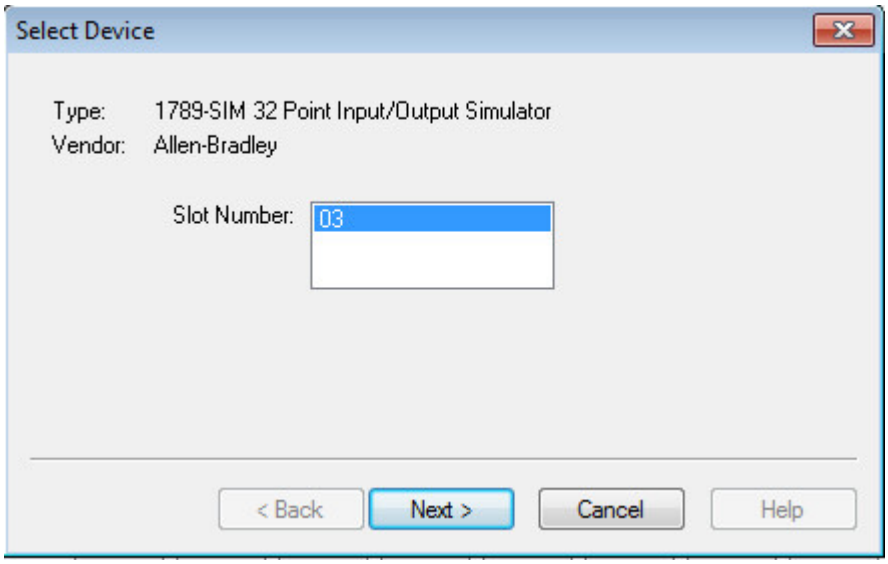
Right click on the next available slot and select **Create...**



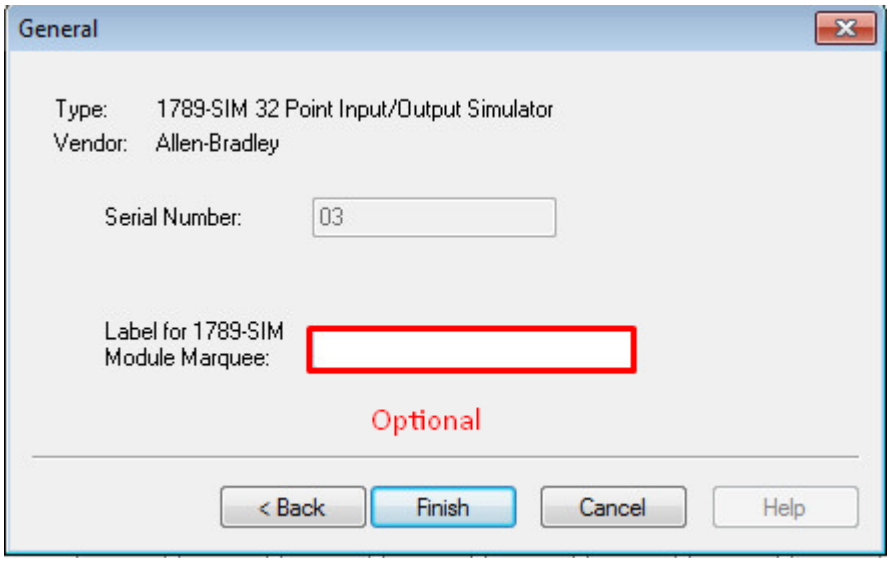
Highlight **1789-SIM 32 Point Input/Output Simulator** and click on the **OK** button to continue.

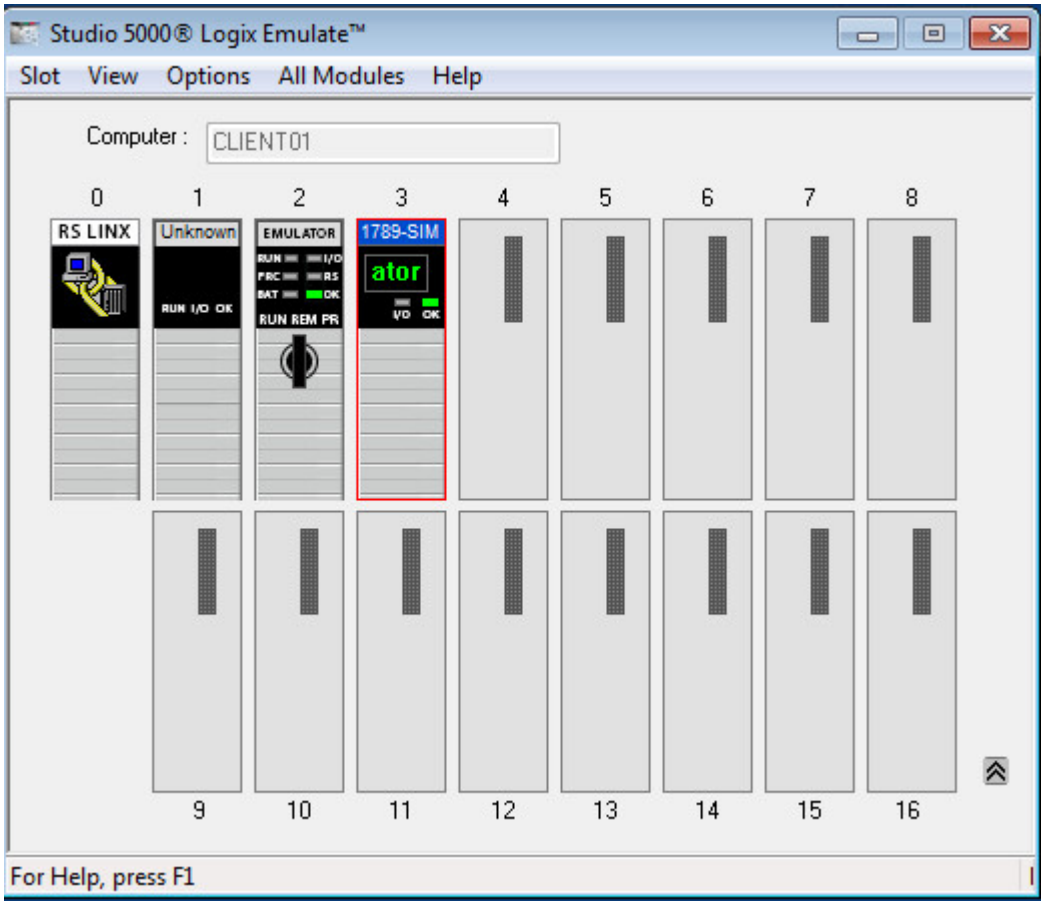


Click on the **Next** button to continue.



It's optional to enter a name in the Label for 1789-SIM Module Marquee box. Click on the **Finish** button to complete the setup.


General
Type: 1789-SIM 32 Point Input/Output Simulator
Vendor: Allen-Bradley
Serial Number: 03
Label for 1789-SIM Module Marquee:
Optional
< Back
Finish
Cancel
Help


Studio 5000® Logix Emulate™
Slot View Options All Modules Help
Computer : CLIENT01
0 1 2 3 4 5 6 7 8
RS LINX
Unknown
EMULATOR
1789-SIM
9 10 11 12 13 14 15 16
For Help, press F1

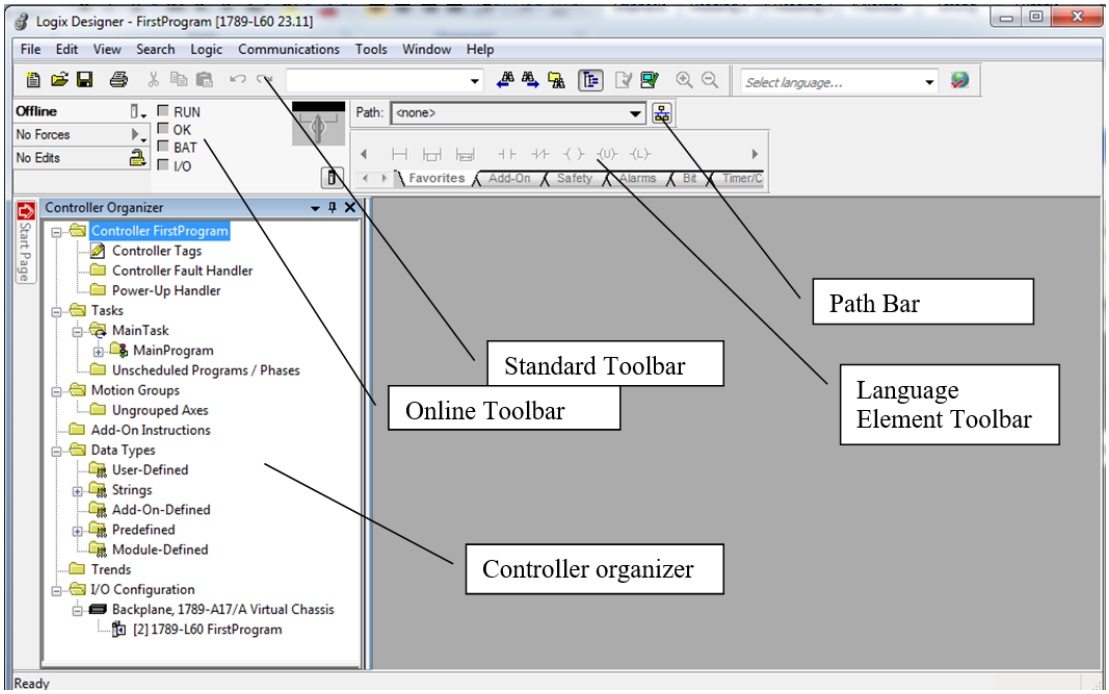
Part II Studio 5000

The Studio 5000 Automation Engineering & Design Environment® combines engineering and design elements into a common environment. The first element is the Studio 5000 Logix Designer® application. The Logix Designer application is the rebranding of RSLogix 5000® software and will continue to be the product to program Logix 5000™ controllers for discrete, process, batch, motion, safety, and drive-based solutions.

Programming languages available in Studio 5000 professional:

Ladder Logic
Sequential Function Chart
Structured Text
Function Block

Controller Organizer



Controller Folder

In this folder you can find the Controller scoped tags, Start-up handling routines and the fault handling routines. A controller scoped tag is a tag that is accessible by all programs and tasks. The Start-Up routine can be used if a separate program is needed to run on start-up. The same can be said for the Fault-Handling routine, it is used if a separate program needs to run when a fault occurs.

Task Folder

This is the folder that will be used the most. This is where you find Tasks, Programs, Routines and the program tags. A Task is something we have not dealt with before in the PLC5 part, this is not available in PLC5 or SLC's. The closest comparison would be an interrupt routine in those processors. In a Task you create Programs. A program holds a folder with Program Tags, these tags are accessible only by the program it's in. The other parts of the Program Folder are routines. One of the routines has to be designated as the main routine. Routines can be created with mixed languages. You could have a ladder file for discrete control and use a Structured Text routine if more complex calculations are needed.

Motion Group Folder

If motion is used, this folder organizes the motion axis from a single axis motion system to a coordinated motion system with multiple axis.

Trend Folder

You can create multiple Tends and store them in this folder for use when you need them. A Trend is a Graphical way of displaying data that is collected over a period of time. You can use 8 data point per Trend.

Data Types Folder

This an organizational folder that stores templates of data types. There are 4 subfolders for the different types. The User Defined folder is used to create your own data type. More about this will be handled in level 2. The String folder is for ASCII characters and the length of the

string. Predefined tags are tags that are particular to a function. For example a Timer is put together using a preset ,accumulator and status bits. This information is all stored in the pre-defined data types. The module defined data types are data structures used for particular IO cards used in a PLC system.

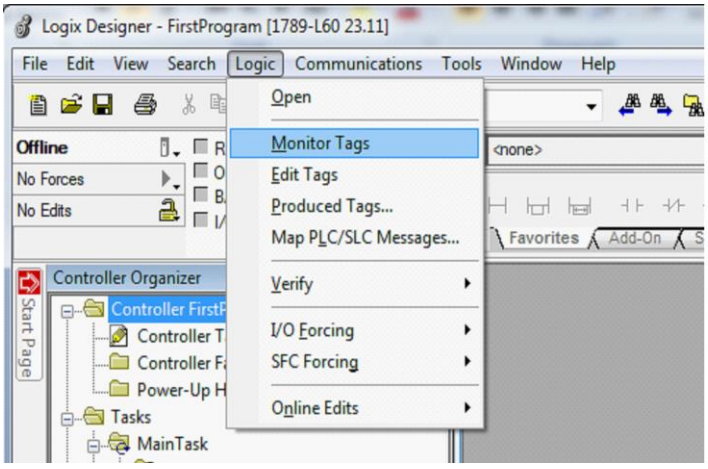
IO Configuration Folder

A Logix system needs to have the IO cards configured in the project that will be used in that project. It does not mean it needs to configure all the cards in the chassis like you have to do for a SLC system, but the cards in the chassis that will be used for the project need to be in the IO Configuration. This includes the communication cards needed to get to remote IO chassis. An Ethernet card used for programming and connecting HMI systems does not need to be in the IO Configuration.

Controller and Program Tags

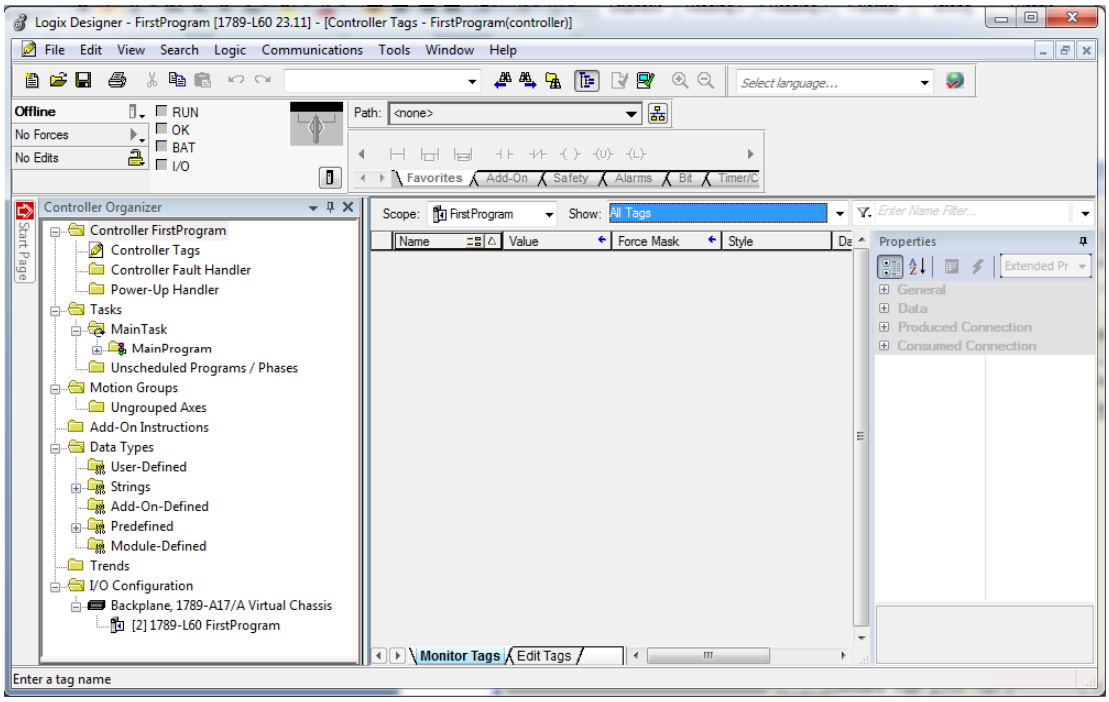
All tags in Logix 5000 controllers have to be created. This sounds like a lot of work but it is actually very easy, and as you will find out later much nicer then in traditional PLC's. Tags are created when IO cards are added to the IO Configuration folder, they can be created in the database by you or on the fly while entering ladder files.

At this point you might wonder what is meant with the term "Tag". A tag is an item in the PLC data table that can be used in a routine. A tag could be of any available data type. For example, a sensor connected to an input of the PLC has a reference to this input somewhere in the data table. You can give this input whatever name you want by creating let's say a "Tag" called "EndProx". This will now be a Boolean tag(1 or 0) to represent the status of this sensor. But you can also create a "Tag" called "MotorTemperature" and use a DINT (Double Integer) as the data type. Both of these examples are considered one tag but on consist of only one bit and the other uses 32 bits of data.



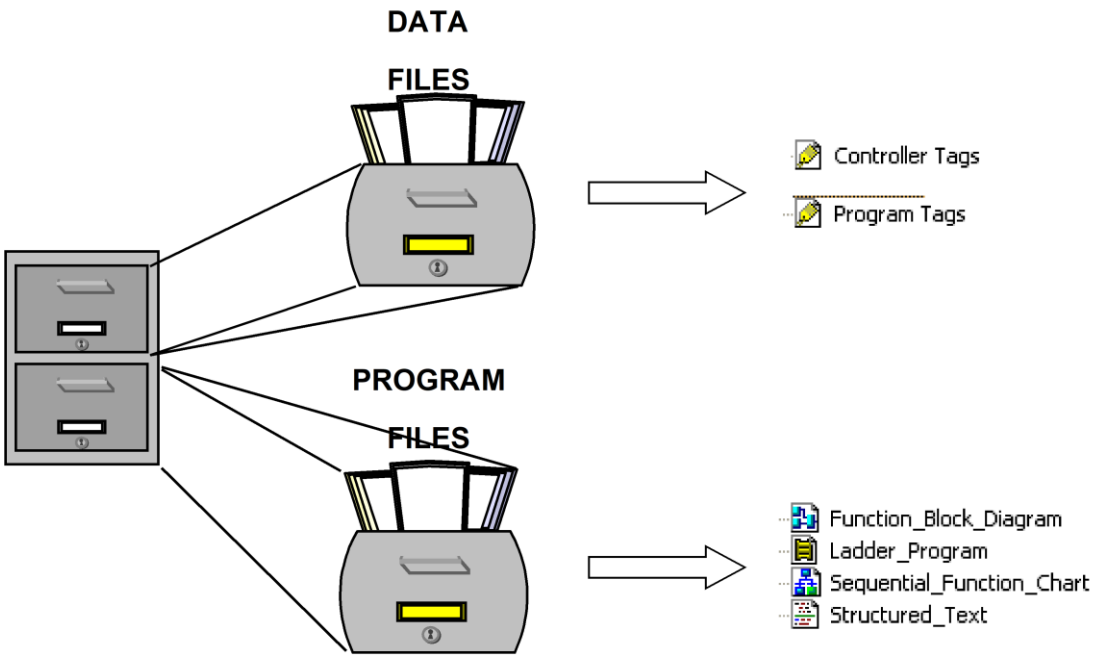
Let's take a look at the default database when programming a Logix 5000 program.

Select "*Logic*" from the toolbar and click "*Monitor Tags*".



Memory Organization

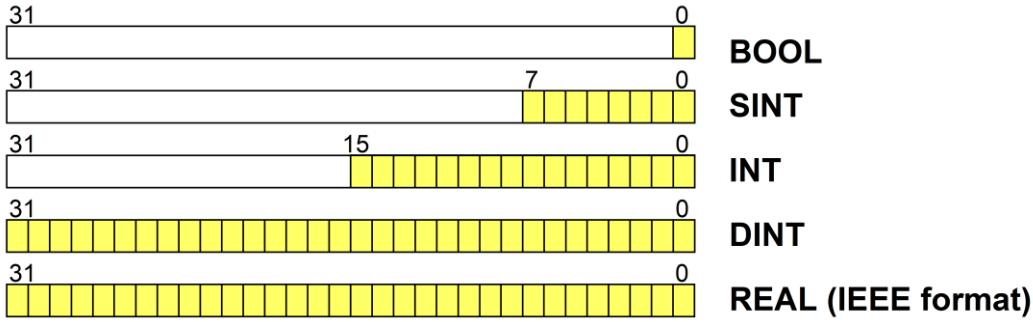
Data in PLC's can be divided in two sections. Data Files and Program Files. The instructions you enter to create your program are stored in a Folder called "Program Files". In Allen Bradley Logix PLC's they are stored in a folder hierarchy made up of "Tasks" > "Programs" > "Routines"



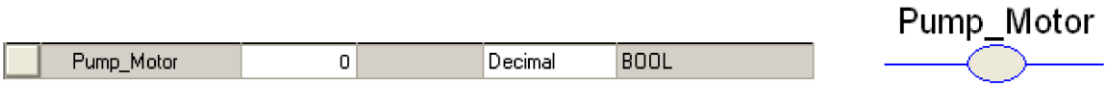
At runtime however, the instructions have a “State” that needs to be stored somewhere. A proximity sensor that is wired into an input is either true or false when the system is powered up. This information has to be stored somewhere. This is called the Tag Database. In Logix PLC’s they are either controller tags or program tags. Tags created in the “Program Tag” folder are accessible only by the “Routines” in that program file. It is therefore possible to create tags with the same name as long as they are in different programs. Tags that are created in the “Controller Tag” folder can be accessed by all programs/routines in the controller. This is referred to as “Data Scope”. So you have “Controller Scoped Tags” and “Program Scoped Tags”

Logix Data Types

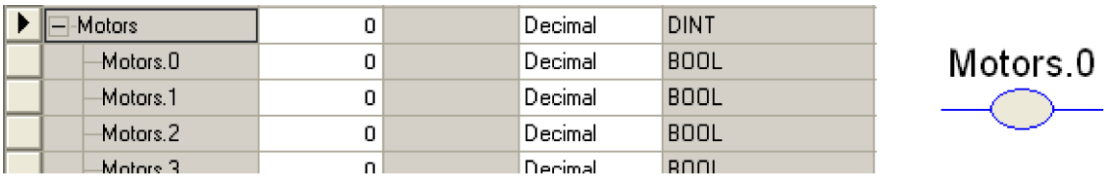
When tags are created in the data table they are based on a “Data Type”. A logix controller is a 32 bit controller so all tags created are based on 32 bits of memory. All individual tags created in a Logix controller use a 32-bit word.



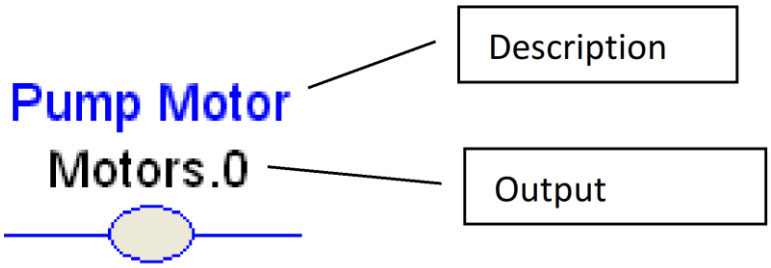
A Boolean tag called “Pump_Motor” uses all 32 bits but in a ladder program, but the status of this is still just True or False.



The advantage of the Boolean tag “Pump_Motor” is that it is self describing. It is very clear what this tag is used for. It does however take up more memory then is needed.



In the above picture you can see a tag called “Motors” and is created as a Double Integer (DINT) and the address on the output is “Motors.0”. This is much more memory friendly but is not as clear as the first option as to what the output is used for. You can of course used descriptions to make clear to the user what the output is used for but these descriptions are not downloaded to the controller. They are part of the “Off Line” file.

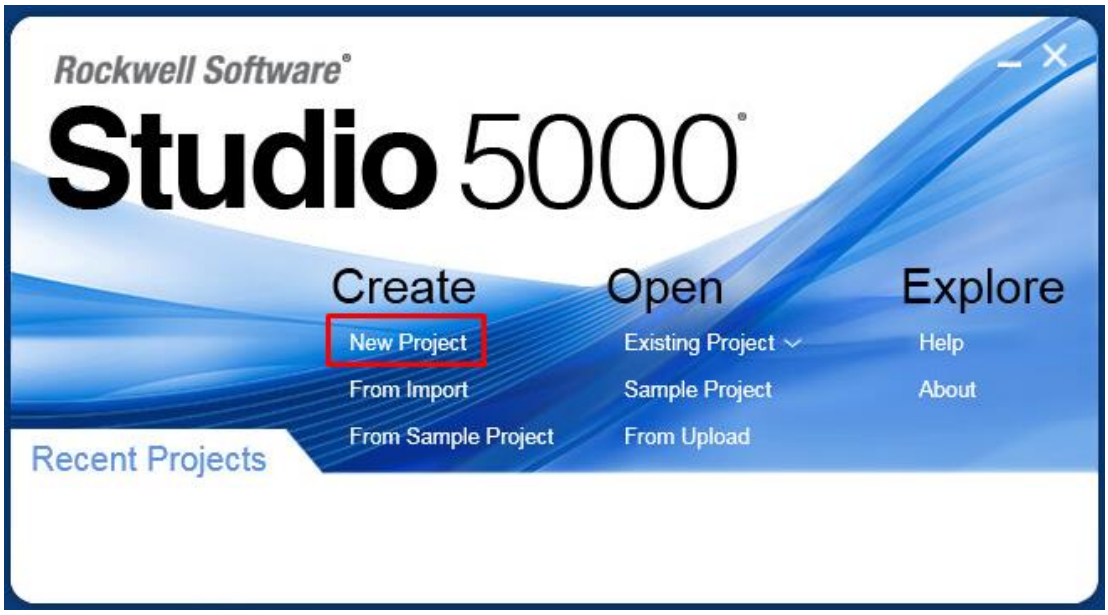


The previously created tags are still internal tags and have no output associated with it. IO tags are created automatically when the IO cards you are going to use are inserted using the IO configuration folder.

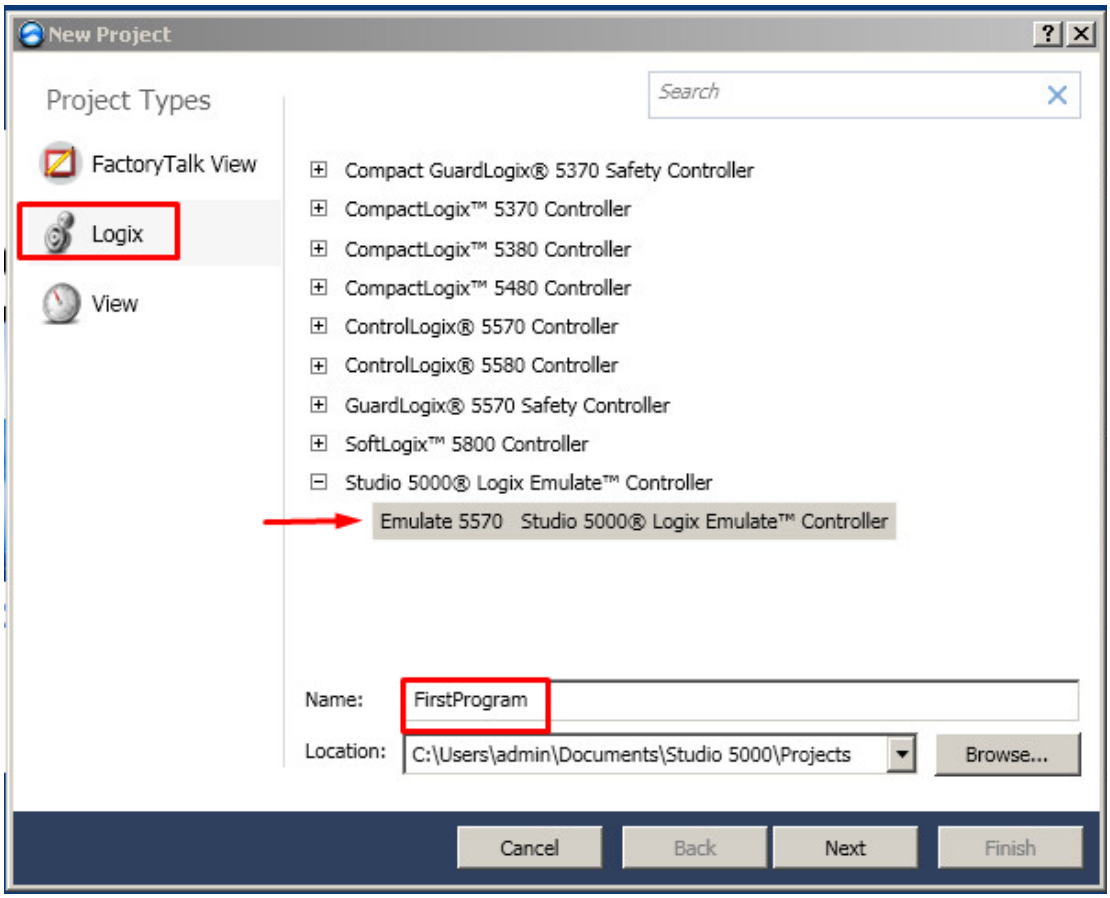
1. CREATE NEW PROJECT, PROJECT PATH, ONLINE/OFFLINE

Double click on Studio 5000 icon located on the desktop.

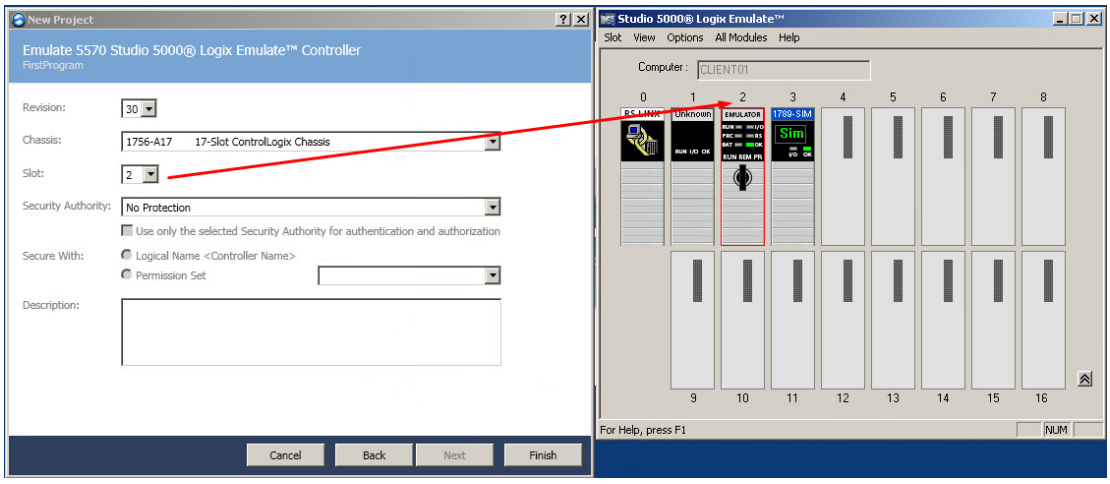
Click on the **New Project**



Click **Logix** if not already selected. Select **Studio 5000® Logix Emulate™ Controller** -> **Studio 5000® Logix Emulate™ Controller**. Enter a file name for the program and change the file location if required. click the **Next** button to continue.

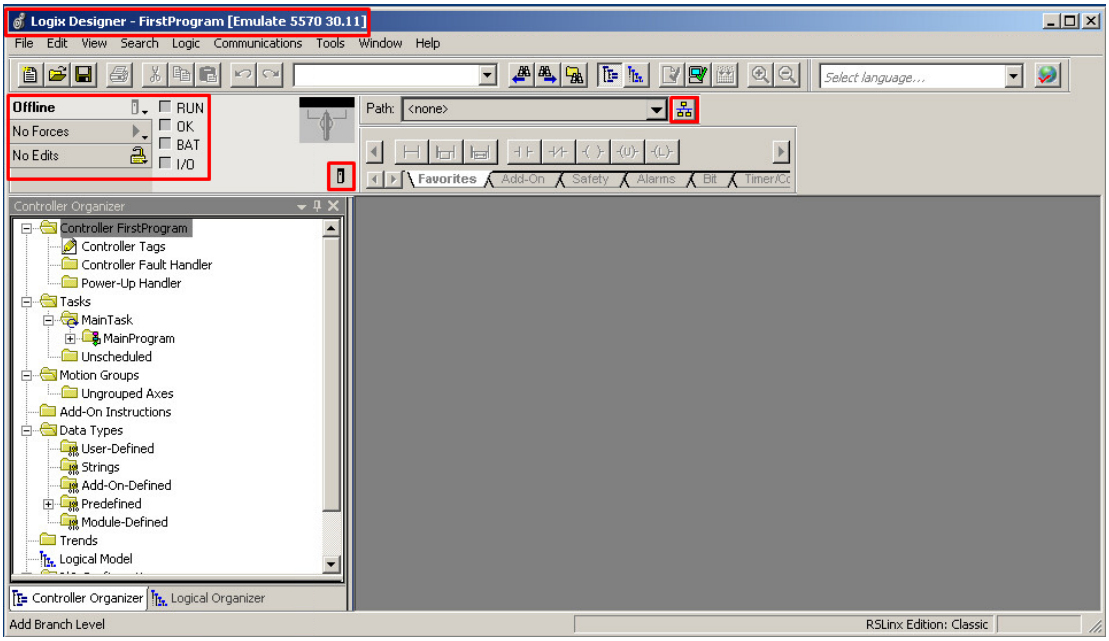


Chassis size is **17** (0-16), and the controller is located in slot **2**. Click the **Finish** button to continue.



** Don't panic if you happen to select the wrong controller or slot number in the project setup. Project settings can be modified inside the program.*

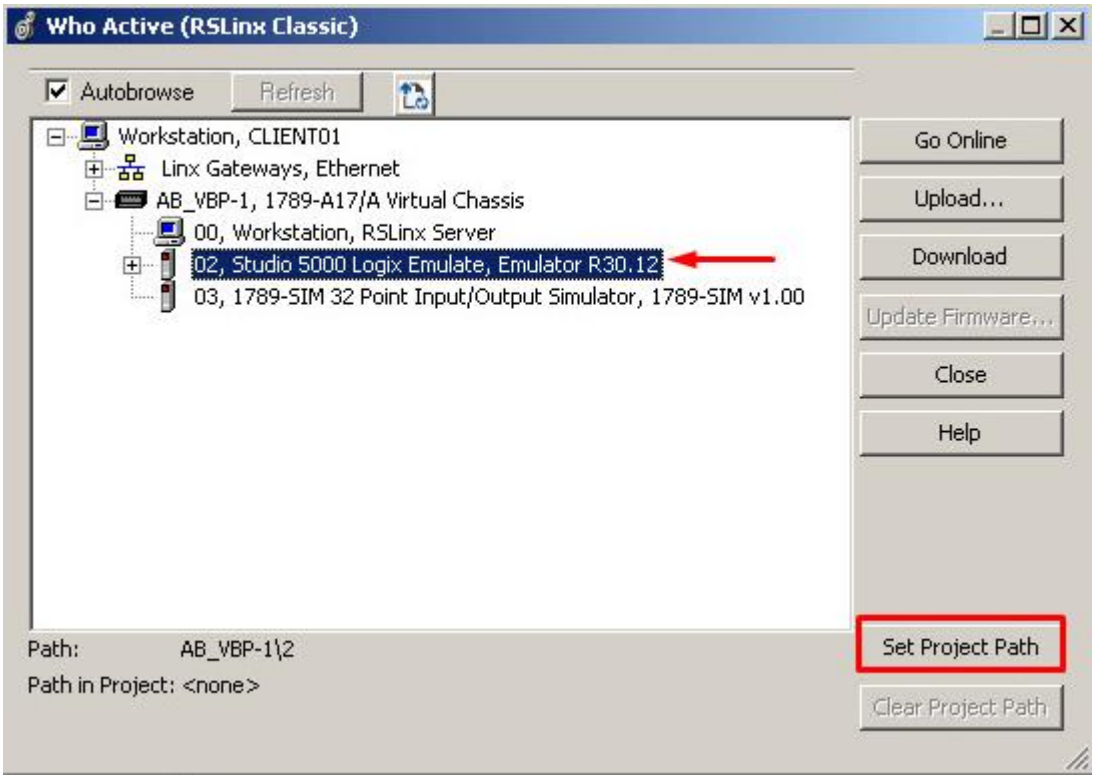
This is how the studio 5000 screen looks. The menu bar shows name of the program (FirstProgram), and the controller type and version (Emulate 5570 30.11). **Offline** menu is for download/upload and change application mode.



A path is required in order to download the project. Path is the communication between Studio 5000 to the device (controller). Click the **Who Active** icon to open the properties.

Expand the **1789-A17/A Virtual Chassis** by click on the + sign. Select **02**, **Studio 5000 Logix Emulate, Emulator R30.12**.

Click the **Set Project Path** button to set as the default path. Click the **Close** button to exit.

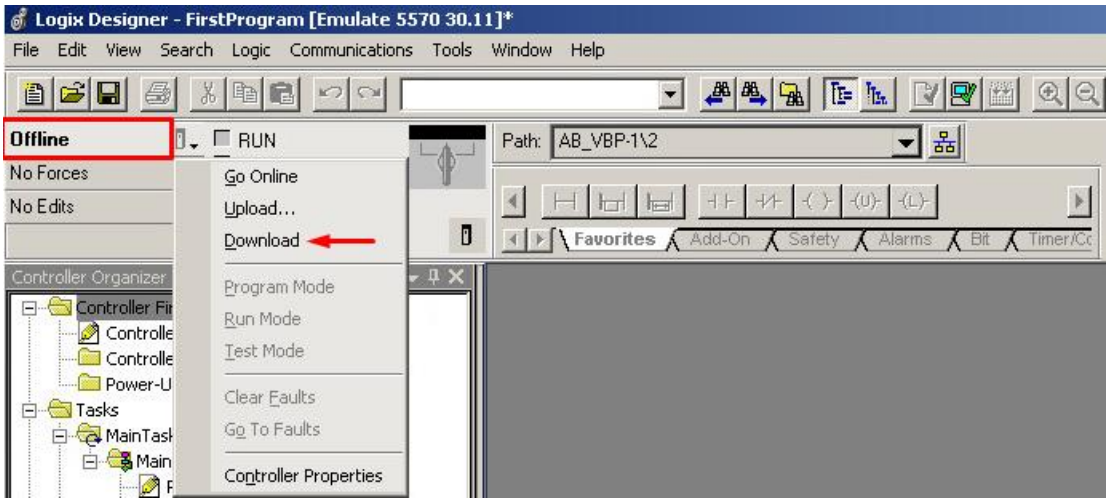


** If the **Set Project Path** button is grayed out. That means the device selected is not a controller*

Now the path should show **AB_VBP-1\2**.

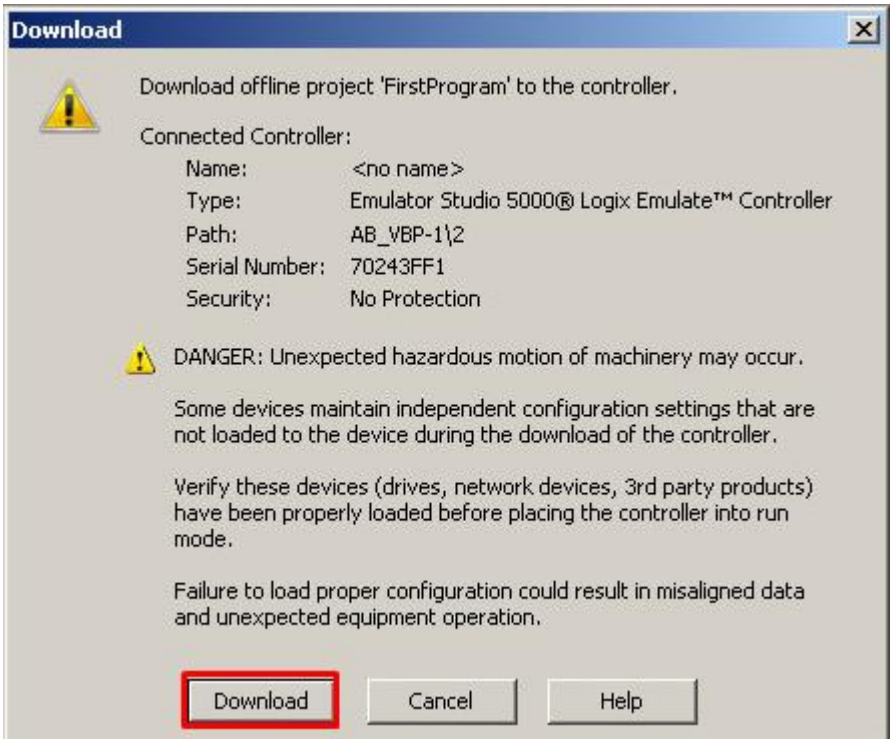
AB_VBP-1 is the driver, and \2 is the location of the controller.

Let's download a blank program to the controller. Click on the **Offline** to open the drop down menu then select **Download**.



** Download a blank program will erase application store inside a controller.*

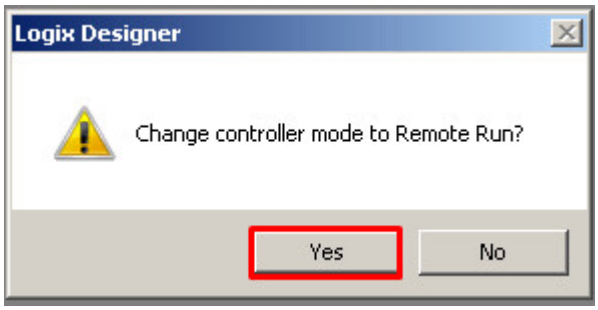
Click the **Download** button on the pop up window.



Now the program is transferred in the controller, but not yet executed. **BLUE** color indicate the program is REM Prog (remote program) mode.

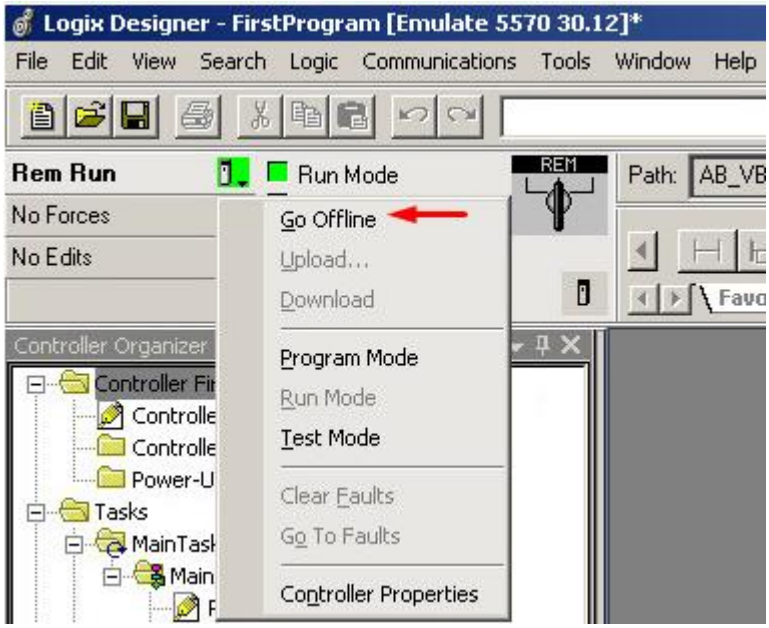
Click on the **Rem Prog** and change to **Run Mode**.

Click the **Yes** button to switch to Remote Run.



GREEN color indicate the program is now in Run Mode.

To Modify the program, switch from **Rem Run** to **Go Offline**. You can modify a program online, but it's dangerous if you make a mistake. Because the program will become active after your finalize the changes.


Logix Designer - FirstProgram [Emulate 5570 30.12]*
File Edit View Search Logic Communications Tools Window Help
Rem Run
Run Mode
No Forces
No Edits
REM
Path:
AB_VB
Go Offline
Upload...
Download
Program Mode
Run Mode
Test Mode
Clear Faults
Go To Faults
Controller Properties
Controller Organizer
Controller Fi
Controlle
Controlle
Power-U
Tasks
MainTask
Main

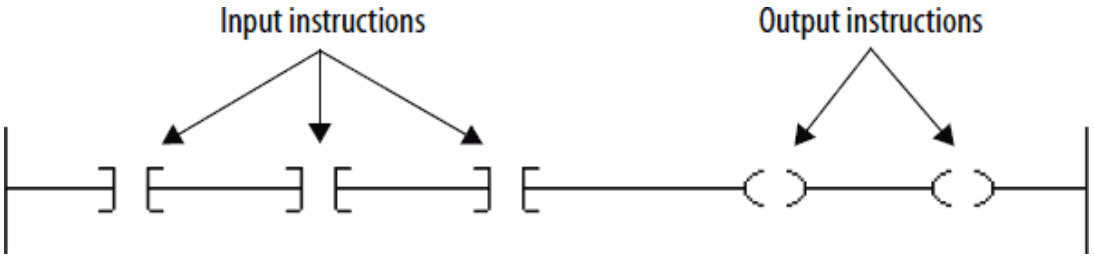
2. PROGRAM LADDER DIAGRAM

Instruction

You organize ladder diagram as rungs on a ladder and put instructions on each rung. There are two basic types of instructions:

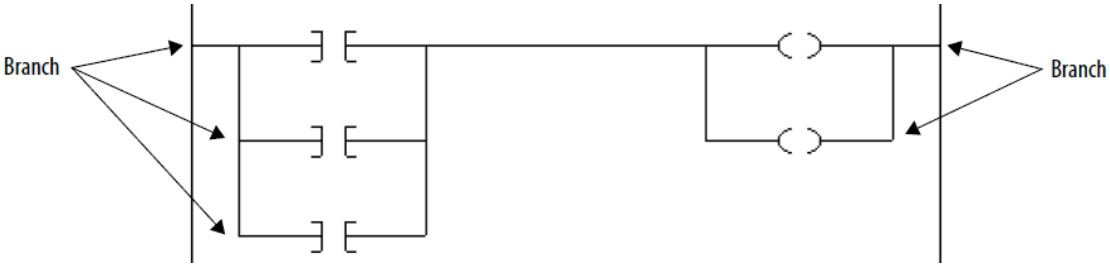
• **Input instruction**: An instruction that checks, compares, or examines specific conditions in your machine or process.

• **Output instruction**: An instruction that takes some action, such as turn on a device, turn off a device, copy data, or calculate a value.

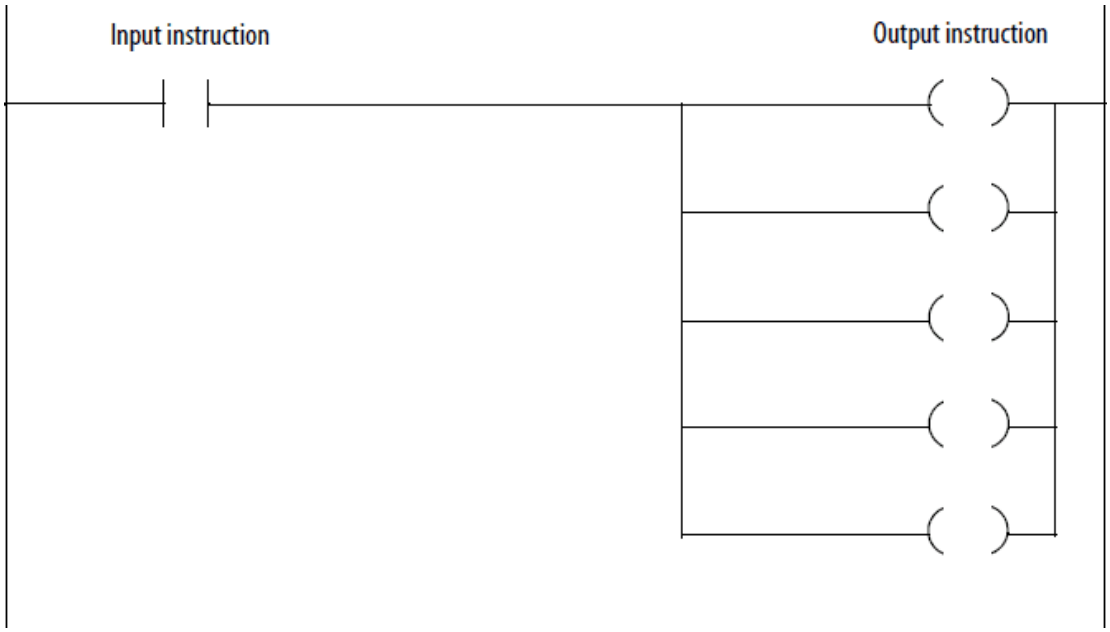


Branch

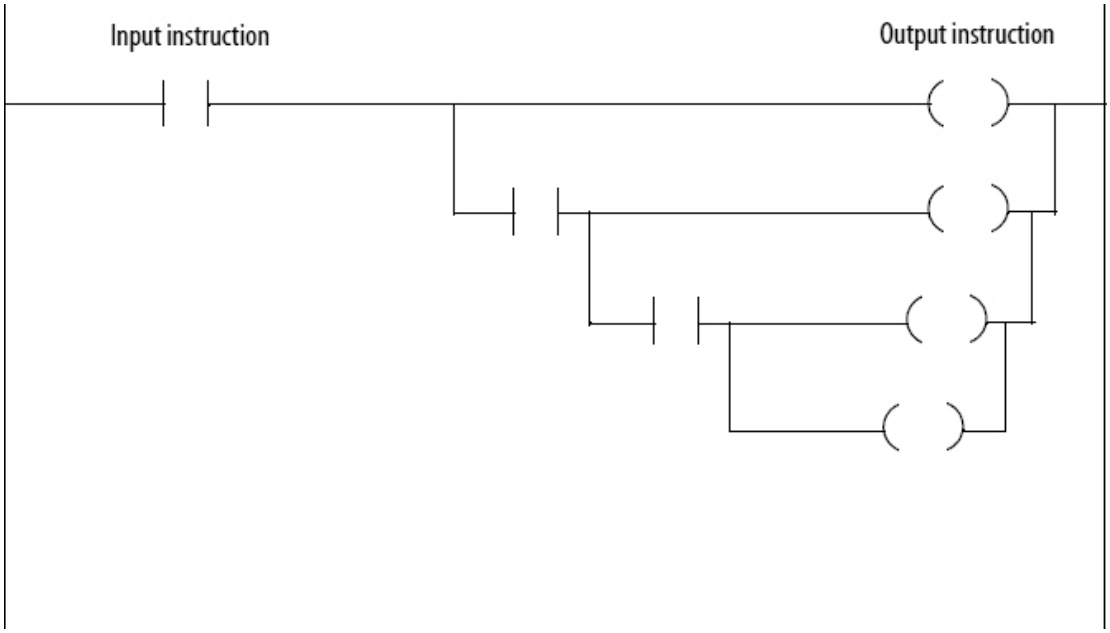
A branch is two or more instructions in parallel.



There is no limit to the number of parallel branch levels that you can enter. This example shows a parallel branch with five levels. The main rung is the first branch level, followed by four additional branches.



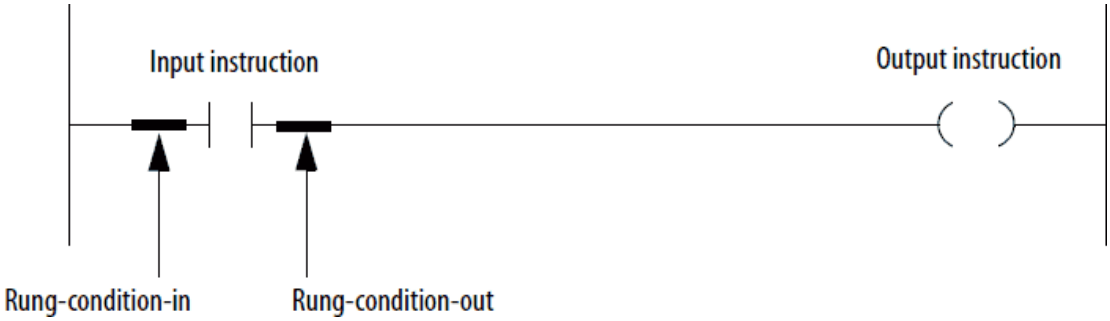
You can nest branches to as many as 6 levels. This example shows a nested branch. The bottom output instruction is on a nested branch that is three levels deep.



Large rungs with complex, nested branches result in having to scroll through the ladder editor and may end up spanning multiple pages when you print the logic. To make it easier to maintain, divide the logic into multiple smaller rungs.

Rung condition

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in).



Only create instructions that affect the rung-condition-in of subsequent instructions on the rung.

If the rung-condition-in to an input instruction is true, the controller evaluates the instruction and sets the rung-condition-out to match the results of the evaluation.

- If the instruction evaluates to true, the rung-condition-out is true.
- If the instruction evaluates to false, the rung-condition-out is false.

An output instruction does not change the rung-condition-out.

- If the rung-condition-in to an output instruction is true, the rung-condition-out is set to true.
- If the rung-condition-in to an output instruction is false, the rung-condition-out is set to false.

Write ladder logic

Writing ladder logic requires that you choose the input and output instructions, and choose the tag names for operands.

Choose the required instructions

1. Identify the conditions to check and separate them from the action to take for the rung.

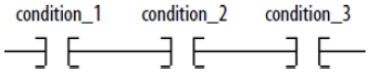
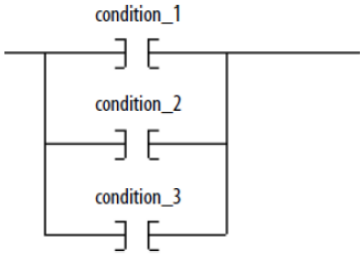
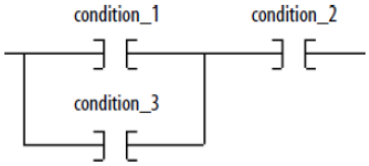
2. Choose the appropriate input instruction for each condition and the appropriate output instruction for each action.

The examples in this chapter use two simple instructions to help you learn how to write ladder diagram logic. The rules that you learn for these instructions apply to all other instructions.

Symbol	Name	Mnemonic	Description	
	Examine If Closed	XIC	An input instruction that looks at one bit of data.	
			If the bit is	**Then the instruction (rung-condition-out) is**
			On (1)	True
			Off (0)	False
	Output Energize	OTE	An output instruction that controls one bit of data.	
			If the instructions to the left (rung-condition-in) are	**Then the instruction turns the bit**
			True	On (1)
			False	Off (0)

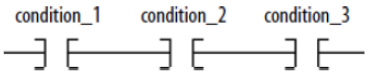
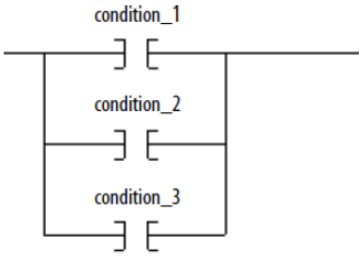
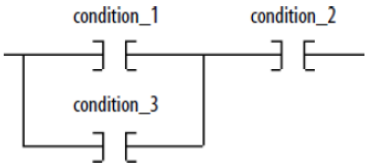
Arrange the input instructions

Determine how to arrange the input instructions on the rung, as shown below.

To check multiple input conditions when:	Arrange the input instructions:
• All conditions must be met in order to take action. For example, If condition_1 AND condition_2 AND condition_3…	In series: 
• Any one of several conditions must be met in order to take action. For example, If condition_1 OR condition_2 OR condition_3…	In parallel: 
• There is a combination of the above. For example: If condition_1 AND condition_2… OR If condition_3 AND condition_2…	In combination: 

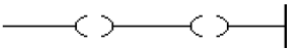
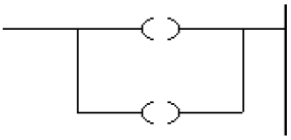
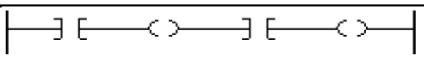
Arrange the input instructions

Determine how to arrange the input instructions on the rung, as shown below.

To check multiple input conditions when:	Arrange the input instructions:
• All conditions must be met in order to take action. For example, If condition_1 AND condition_2 AND condition_3…	In series: 
• Any one of several conditions must be met in order to take action. For example, If condition_1 OR condition_2 OR condition_3…	In parallel: 
• There is a combination of the above. For example: If condition_1 AND condition_2… OR If condition_3 AND condition_2…	In combination: 

Arrange the output instructions

Place at least one output instruction to the right of the input instructions. You can enter multiple output instructions on a rung of logic:

Option	Example
Place the output instructions in sequence on the rung (serial).	
Place the output instructions in branches (parallel).	
Place the output instructions between input instructions. The last instruction on the rung must be an output instruction.	

Choose a tag name for an operand

Tag names follow these formats:

For a:	Specify:
Tag	*tag_name*
Bit number of a larger data type	*tag_name.bit_number*
Member of a structure	*tag_name.member_name*
Element of a one dimension array	*tag_name[x]*
Element of a two dimension array	*tag_name[x,y]*
Element of a three dimension array	*tag_name[x,y,z]*
Element of an array within a structure	*tag_name.member_name[x]*
Member of an element of an array	*tag_name[x,y,z].member_name*

where:

- *x* is the location of the element in the first dimension.
- *y* is the location of the element in the second dimension.
- *z* is the location of the element in the third dimension.

For a structure within a structure, add *.member_name*.

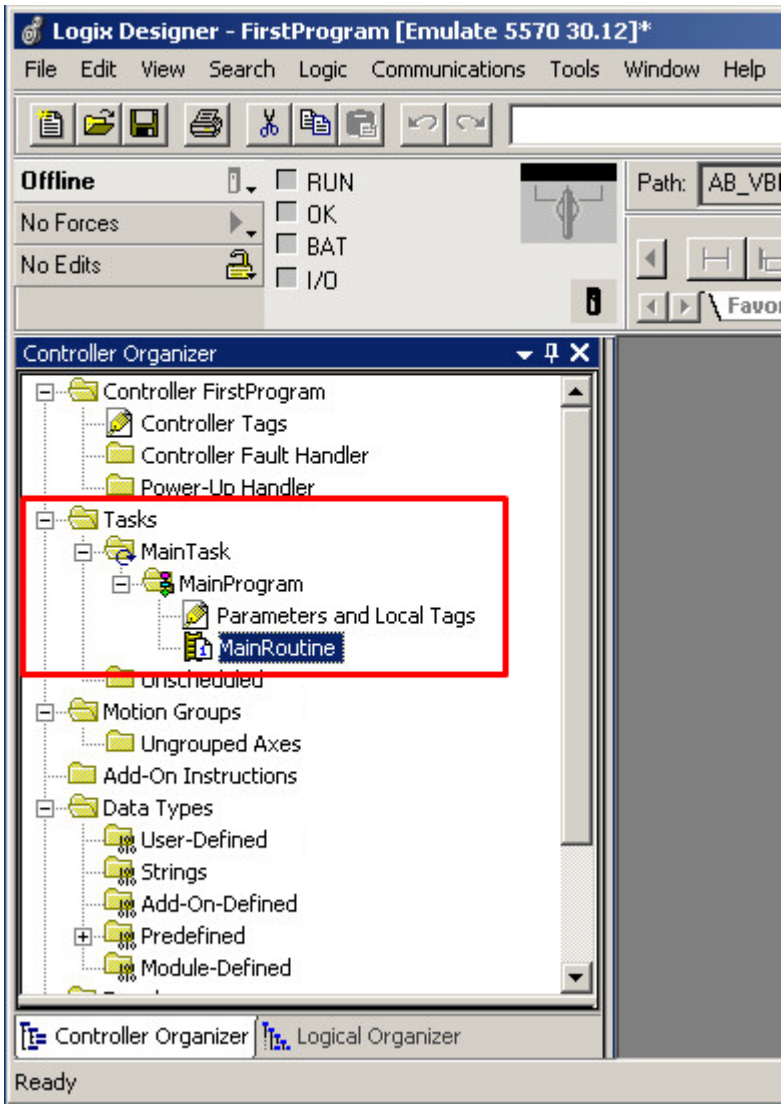
Example: **Choose a Tag Name for an Operand**

To Access:	The tag name looks like this:
machine_on tag	machine_on
bit number 1 of the one_shots tag	one_shots.1
DN member (bit) of the running_seconds timer	running_seconds.DN
mix member of the north_tank tag	north_tank.mix
element 2 in the recipe array and element 1,1 in the tanks array	COP Copy File Source recipe[2] Dest tanks[1,1] Length 1
element 2 in the preset array within the north_tank tag	CLR Clear Dest north_tank.preset[2] 0
part_advance member of element 1 in the drill array	drill[1].part_advance

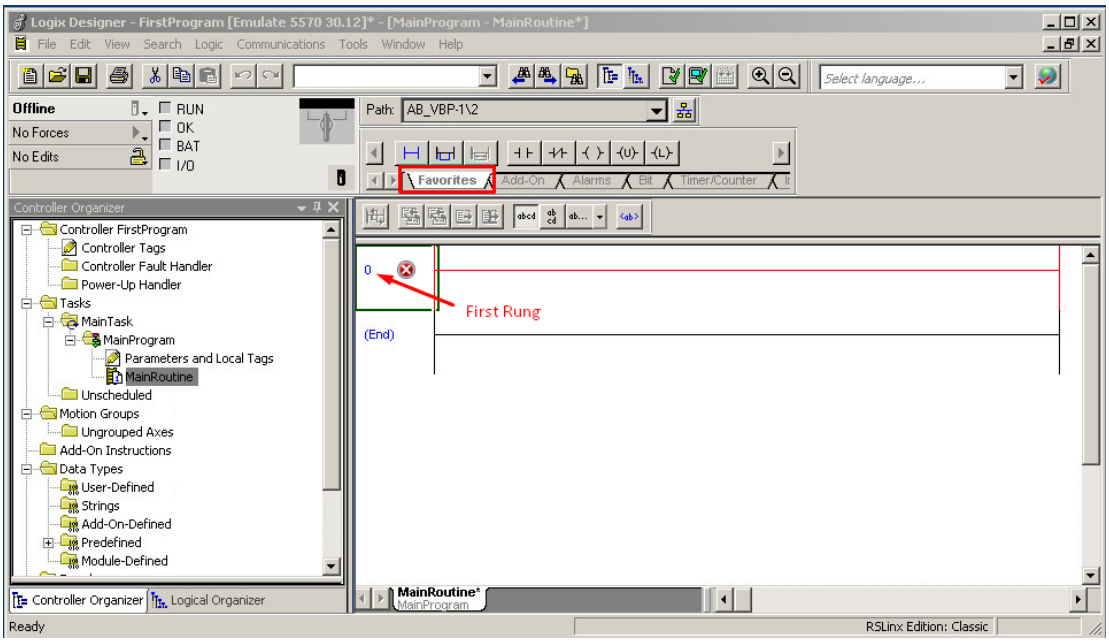
Enter ladder logic

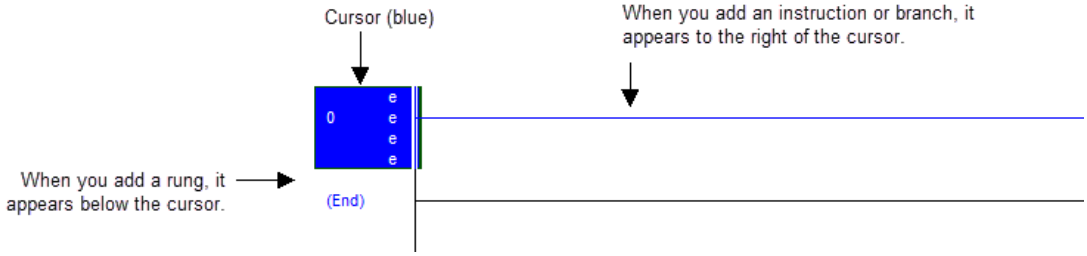
In a new project, a continuous task, a main program, and a main routine is created by default.

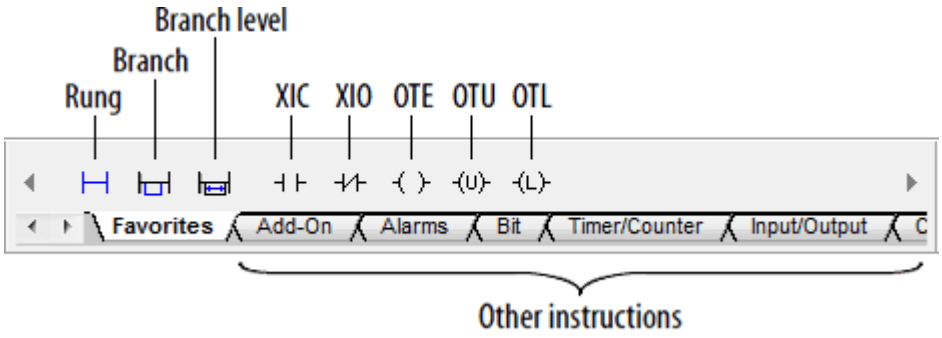
Tasks -> **MainTask** -> **MainProgram** -> **MainRoutine**.



Double click the **MainRoutine** to open the main routine page. Rung **0** is the first line in the ladder that is ready for instructions. You will find common instructions are located in the Favorites tab.







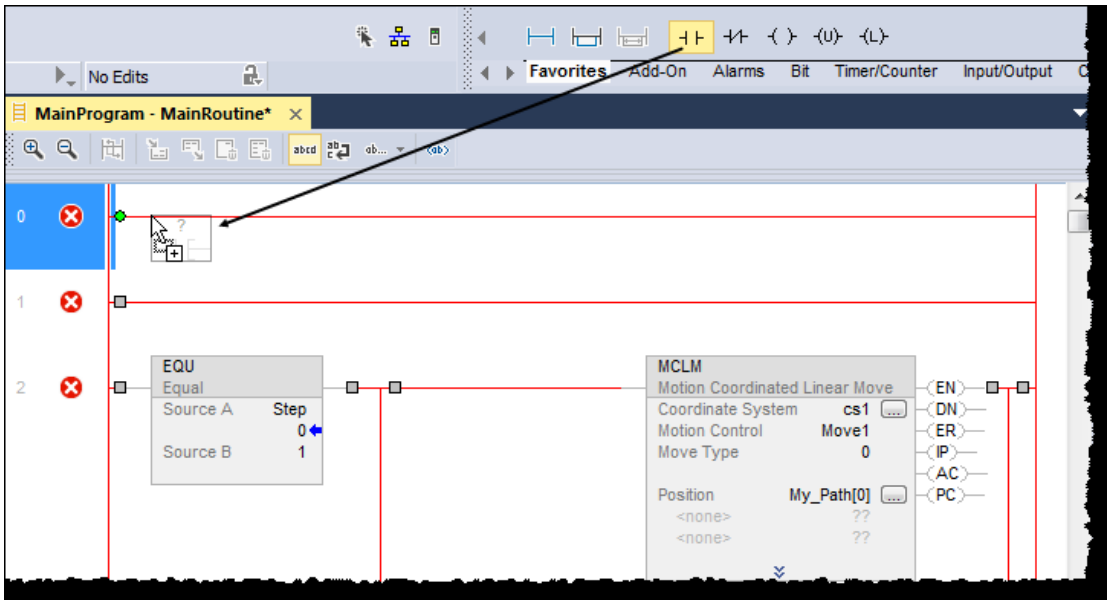
Append an element to the cursor location

1. Click to select the instruction, branch, or rung that is above or to the left of where you want to add an element.

2. On the **Language Element** toolbar, click the button for the element that you want to add.

Drag and drop an element

Drag the button for the element directly to the desired location. A green dot shows a valid placement location (drop point).

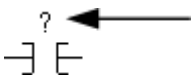


Assign instruction operands

After you add an instruction to a ladder rung, you assign tags to the instruction operands. You can create a new tag, use an existing tag, or assign a constant value.

Create and assign a new tag

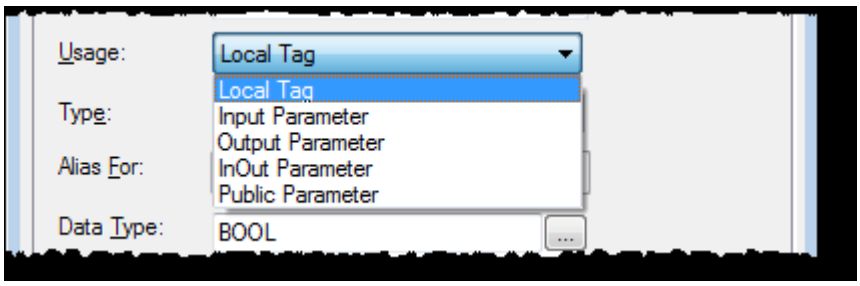
1. Click the operand area of the instruction.



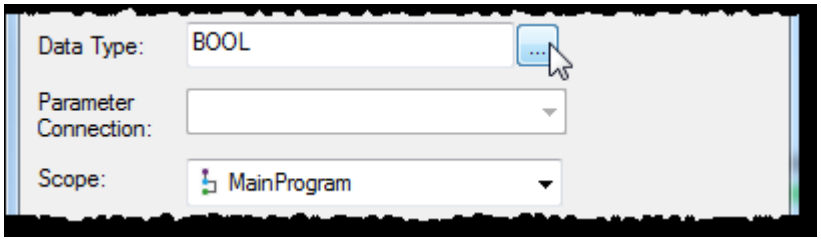
2. Type a name for the tag and press the **Enter** key.

3. Right-click the tag name and then click **New** "tag_name".

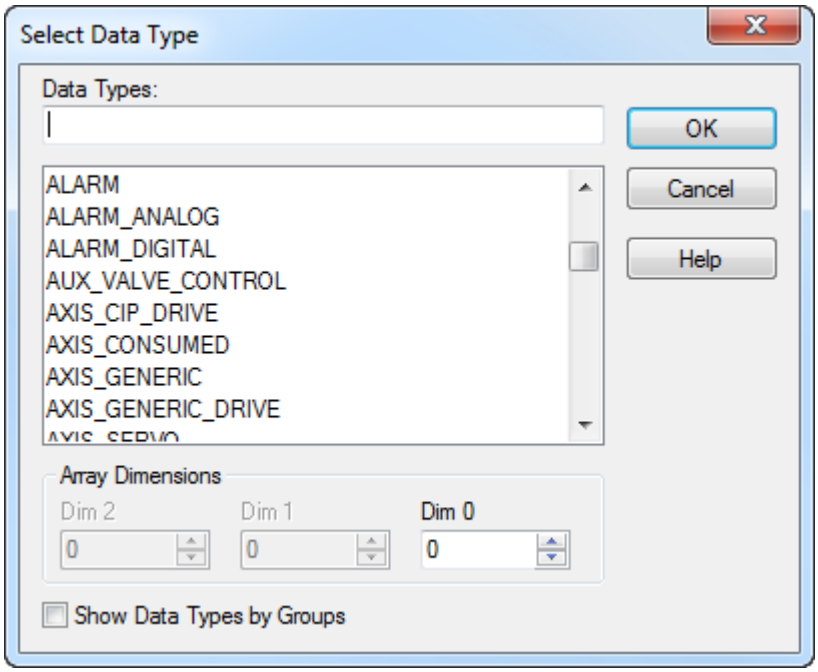
4. In the **New Parameter or Tag** dialog box, in the **Usage** box, choose the usage.



5. In the **New Parameter or Tag** dialog box, in the **Data Type** box, click the button.



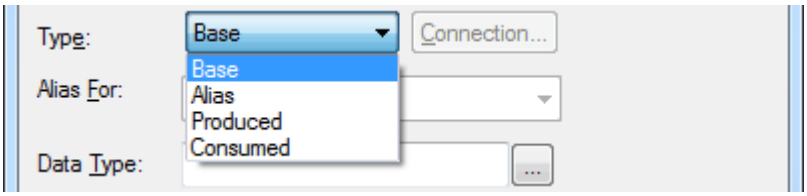
6. In the **Select Data Type** dialog box, choose the data type for the tag.



If you want to define the tag as an array, in the **Array Dimensions** boxes, enter the number of elements in each dimension.

7. Click **OK**.

8. In the **New Parameter or Tag** dialog box, choose the scope for the tag.



9. Click **OK**.

Choose a name or an existing tag

1. Double-click the operand area, and then click  . The Tag Browser window appears.

2. Select the name or tag:

To select a:	Do this:
Label, routine name, or similar type of name	Click the name.
Tag	Double-click the tag name.
Bit number	A. Click the tag name. B. To the right of the tag name, click  . C. Click the required bit.

3. Press the **Enter** key or click a different spot on the ladder diagram to close the Tag Browser.

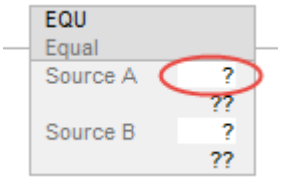
Drag and drop a tag from the Tags window

1. Find the tag in the **Controller Tags** or the **Program Parameters and Local Tags** window.

2. Double-click the tag to select it.

3. Click and drag the tag to its location on the instruction. A green dot appears to show you where you can drop the tag.

Assign an immediate (constant) value



1. Click the operand area of the instruction.

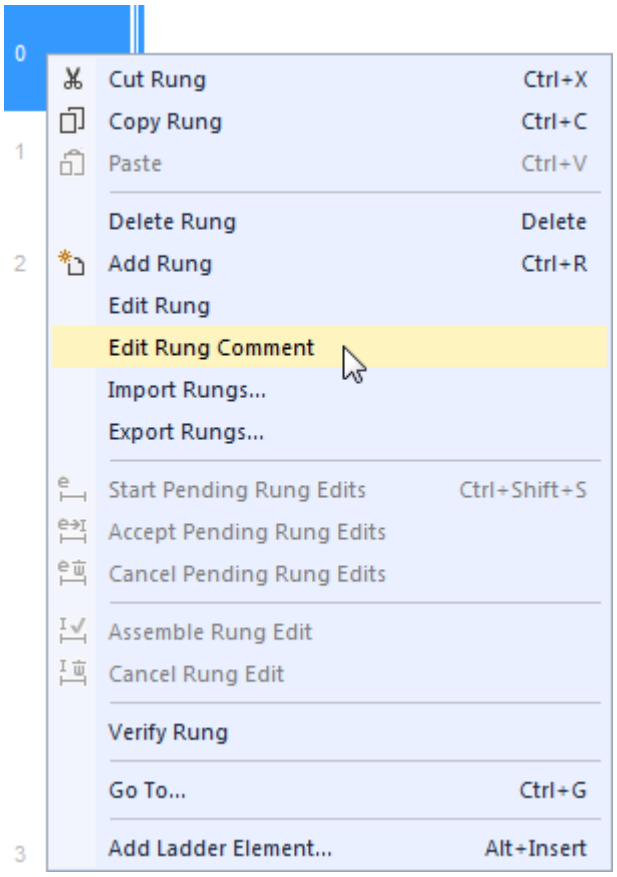
2. Type the value and press the **Enter** key.

Enter a rung comment

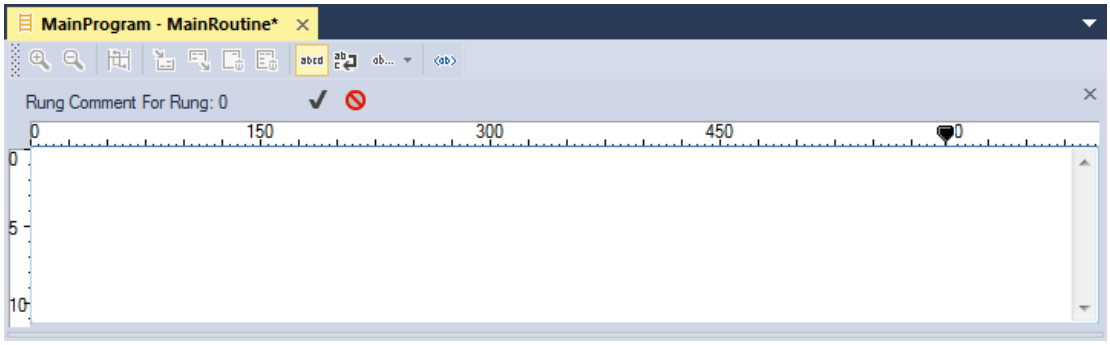
When entering a rung of ladder logic, you can add comments that explain the purpose of your rung.

To enter a rung comment, perform this procedure.

1. Right-click the rung number of your ladder logic and then click **Edit Rung Comment**.



The **Rung Comment** dialog box appears.



2. Type your rung comment, and then click the green check to save your changes or click the red X to discard your changes.

Language switching

With version 17 and later of the application, you have the option to display project documentation, such as tag descriptions and rung comments for any supported localized language. You can store project documentation for multiple languages in a single project file rather than

in language-specific project files. You define all the localized languages that the project supports and set the current, default, and optional custom localized language. The application uses the default language if the current language's content is blank for a particular component of the project. However, you can use a custom language to tailor documentation to a specific type of project file user.
Enter the localized descriptions in your project, either when programming in that language or by using the import/export utility to translate the documentation off-line and then import it back into the project. Once you enable language switching, you can dynamically switch between languages.

• Component descriptions in tags, routines, programs, user-defined data types, and Add-On Instructions.
• Equipment phases.
• Trends.
• Controllers.
• Alarm Messages (in ALARM_ANALOG and ALARM_DIGITAL configuration).
• Tasks.
• Property descriptions for modules in the Controller Organizer.
• Rung comments, SFC text boxes, and FBD text boxes.

For more information on enabling a project to support multiple translations of project documentation, see the online help.

Verify the routine

1. In the **Standard** toolbar click the  **Verify** icon.

2. Errors are listed in the **Output** window on the **Errors** tab at the bottom of the application.

   a. To go to the first error or warning, press the **F4** key.

   b. Correct the error according to the description in the **Errors** tab.

   c. Repeat until you have corrected all errors.

3. To close the **Output** window, press the **Alt+1** keys.

3. BIT INSTRUCTIONS

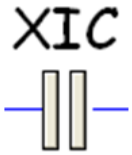
Programming basic bit Instructions

You have to learn how to use software like Studio 5000 and RSLinx in order to do anything with a PLC. You need to know about setting up communication and up/download or just go online with a PLC. But this course is about making programs so the PLC does what you want it to do. So we will start with the basic and most used instructions in a PLC program.

These instructions are called bit instructions because they are working at the "Bit Level" in a PLC program. This means the instruction is either True or False. It could also be called 0 or 1.

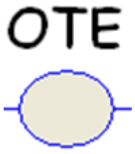
Input Instructions:

XIO – Examine If Open



XIC – Examine If Closed

Output Instructions:

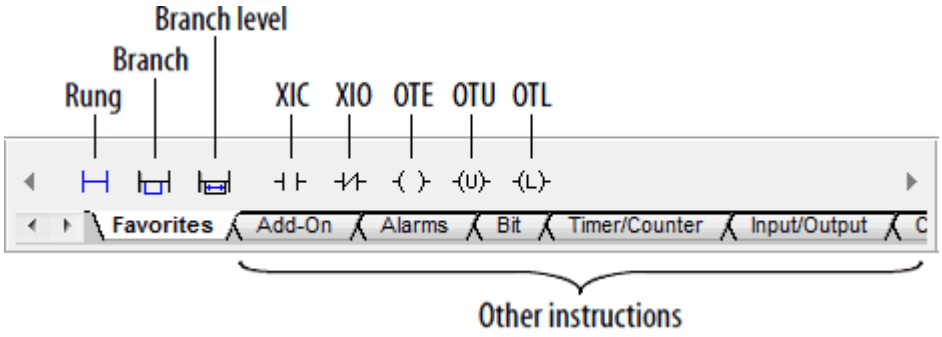


OTE – Output Energize

OTU – Output Unlatch

OTL – Output Latch

Instructions under Favorites tab in Studio 5000



Although these instructions work fine with fine with internal bit addresses in memory, there needs to be information going into the PLC from field devices wired into I/O (Input/Output) cards. Input and Output devices come in so many different formats it would almost be impossible to describe here. But the end result of most of these devices is the same; they result into a 1 or 0 in the PLC data table. For example, a pushbutton is either pushed in or not resulting in 1 or 0. It could also be a photo-eye with it's beam broken or not giving the same result in the PLC. Usually we refer to the state of an Input or Output device as either True or False.

If you want to:	**Use this instruction:**
Enable outputs when a bit is set	XIC
Enable outputs when a bit is cleared	XIO
set a bit	OTE
set a bit (retentive)	OTL
clear bit (retentive)	OTU

I/O (Inputs & Outputs)

I/O (Inputs & Outputs)

Input devices are transducers that change events into electrical signals. Input devices are wired into an Input Module.

Output devices are transducers that change

AC Input device

Photo Switch

DC Input device

Temperature sensor
Analog Input device

POT

Analog device

INPUT DEVICES

- Pushbuttons
- Selector Switches
- Limit Switches
- Level Switches
- Photoelectric Sensors
- Proximity Sensors
- Motor Starter Contacts
- Relay Contacts
- Thumbwheel Switches

OUTPUT DEVICES

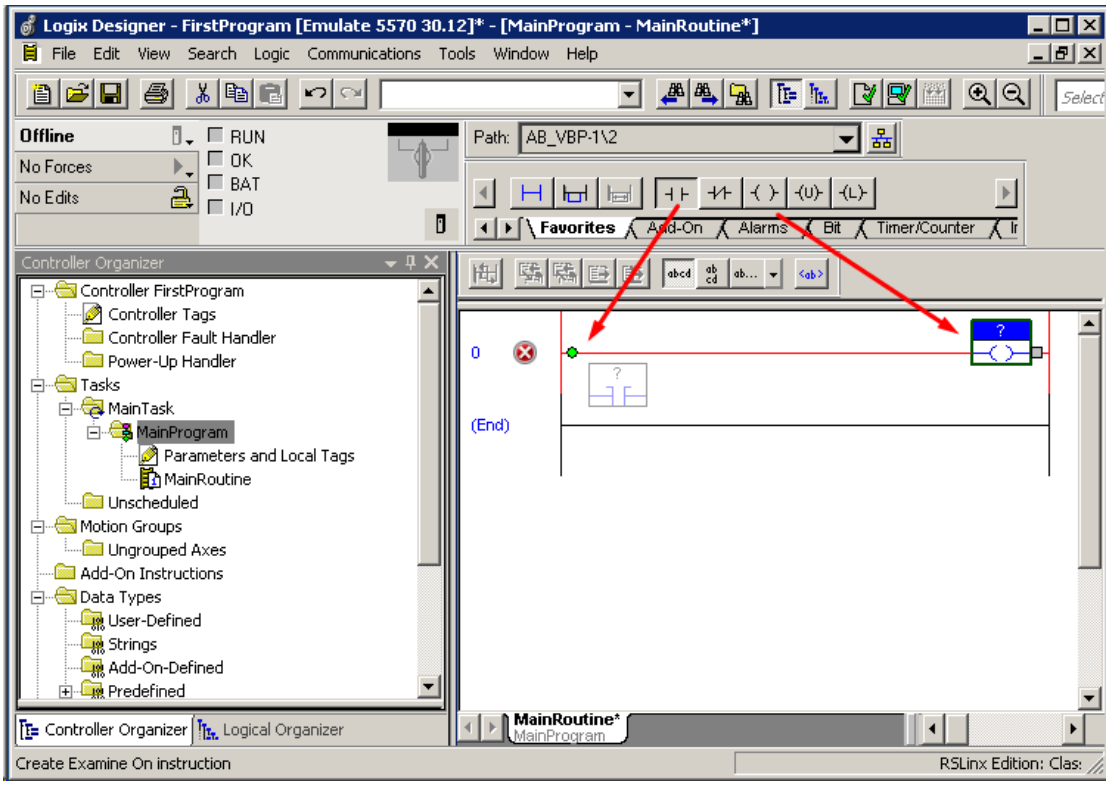
- Valves
- Motor Starters
- Solenoids
- Control Relays
- Alarms
- Lights
- Fans
- Horns

4. EXERCISE 1 - SIMPLE LOGIC

Simple Logic

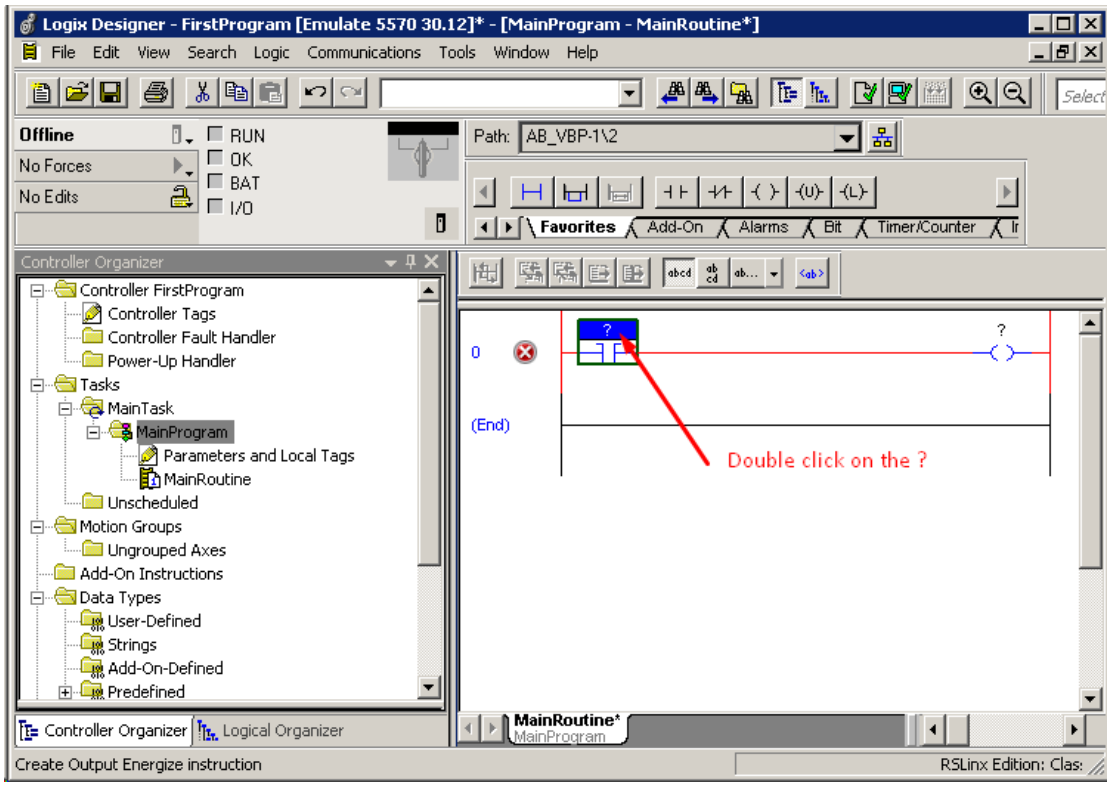
1. Create an Input instruction "**PB1**" and an Output instruction "**MOTOR**".

Drag the instruction from the Favorites tab and bring it to the first rung. A **green** dot will appear when the cursor nears the rung. Release the mouse button on the **green** dot.

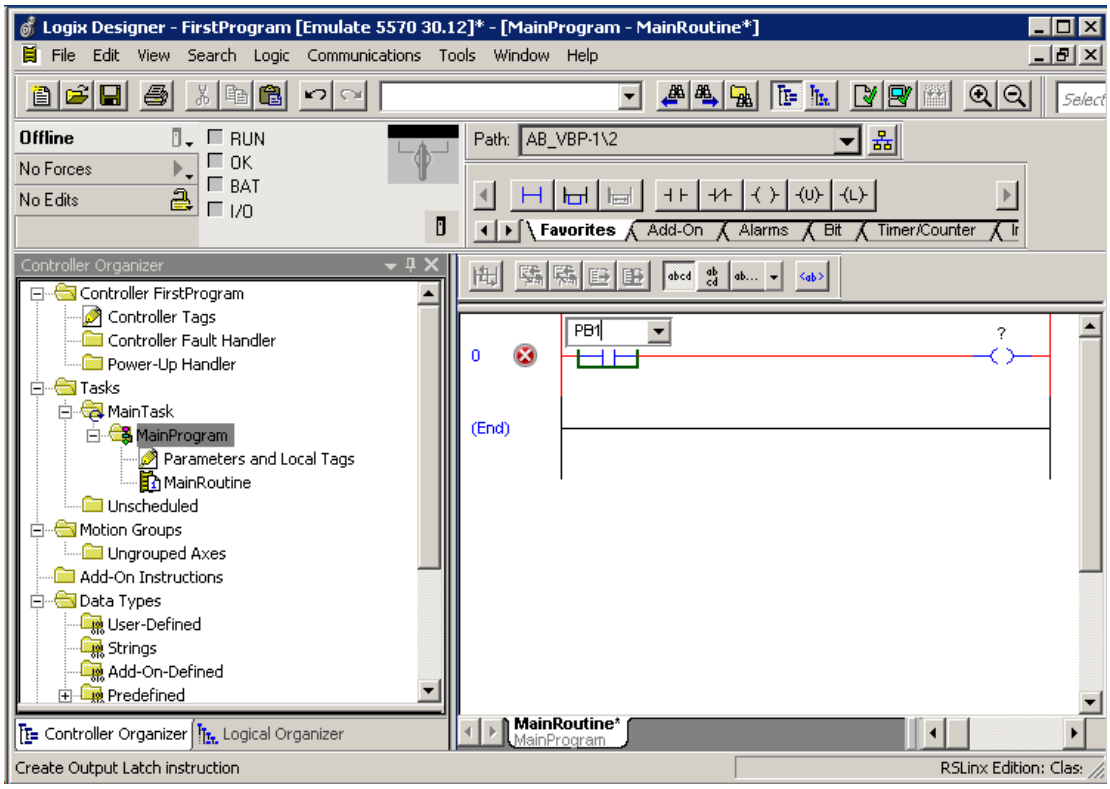


Remember: Input instruction on the left side, and output instruction on the right side.

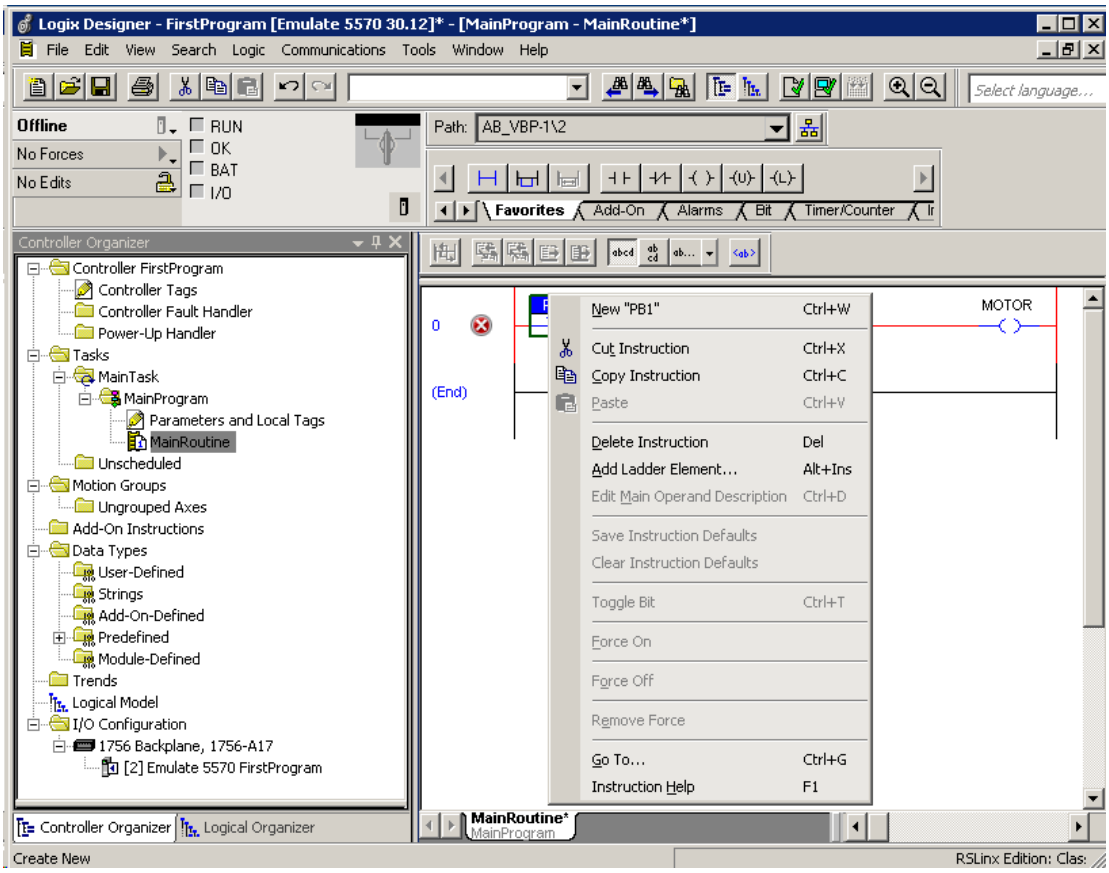
Double click on the **?** mark. Enter a tag name in the box. No space allow for the tag name.



There are a few naming conventions. For example, PushButton1, Push_Button_1, PB1.

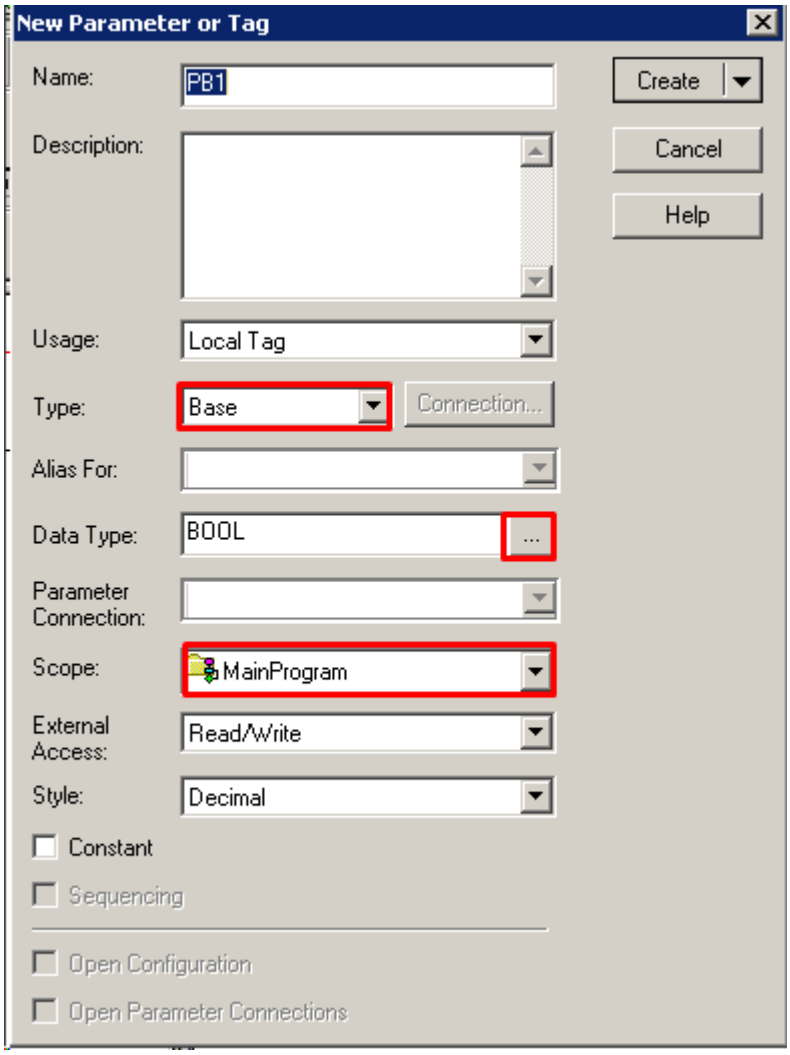


After create the tags, next step is to define the tags. Right click on a new tag "PB1", select **New "PB1" Ctrl+W**.



Base on the instruction type, a default Data Type is assigned to the tag. You can change the data type by click on the button at the end of the Data Type. Select controller scope or one of the existing programs. Controller tag will be equivalent to global a value and program tag is local to a program only.

Click **Create** button to continue. Repeat the same step for the "MOTOR" tag.



New Parameter or Tag dialog box parameters

Use the New Parameter or Tag dialog box to create program parameters or tags. The following table describes the parameters in the dialog box.

Parameter	Description
Name	Used to entera name for the tag. If you are creating a new program parameter from a language editor, and the parameter is from another program, the tag name appears in the Name box.
Description	Used to enter a description for the tag.

Usage

The Usage value for the tag or program parameter.

- The usage value options are: Local Parameter, Input Parameter, Output Parameter, InOut Parameter, <controller> for controller scope tags, or Public Parameter. Public Parameter is not a usage option for Add-On Instructions.
- The default usage value is Local Parameter. Usage is unavailable for members of tags.
- Only Input Parameter, Output Parameter, and Public Parameter are referenced by a parameter from another program.
- Only a Local tag can be an alias tag for an Input Parameter, Output Parameter, and Public Parameter from another program.

Type

Used to select an Alias, Base, Produced, or Consumed type tag. There are additional considerations for produced and consumed tags.

- Keep the following considerations in mind for produced tags:
  - Choose Produced as the tag type if you want to make this tag available to remote controllers through controller-to-controller messaging. This parameter is unavailable when online; you must be offline to choose to produce a tag. This parameter is also unavailable for tags that cannot be used as producers.

There are several tag requirements for producing data.

- You can include the connection status information with the produced tag in a standard controller. In order to do this, you must create a user defined type whose first member is a CONNECTION_STATUS type variable. Then, the remaining members in the user defined type contain the data to be produced.
- You can only produce a tag with data types the size of DINTs (32 bits) or larger.
- You cannot produce I/O tags in the form of <adapterName>:<slot>:<i/o/c>.
- You must first create an alias to the I/O tag and then produce that alias (for example, you cannot produce MyAdapter:3:O, but you can produce alias, MyOutput, that points to MyAdapter:3:O). This reduces name length issues that result from consuming to the produced tag.
- You cannot produce Motion, Axis, or Message tags, because no data object exists underneath them.
- Click Connection to set the maximum number of consumers or connections allowed for this tag. The default number of consumers is 2.
- If you are producing a Safety tag, these requirements are mandatory. You must create a user defined type whose first member is a CONNECTION_STATUS type variable. The remaining members in the user defined type contain the data to be produced. Then use the user defined type in the Data Type field (the Class field must be set to Safety).
- Consumed Safety tags have the same requirements for the data type field as produced tags do.

Connection

Connection is available when you set Type to Produced or Consumed. Connection defines the consumed or produced tag.

Important: Set up only one consumed tag to get data from the same producing tag in another controller. Setting up more than one consumed tag can result in unpredictable controller-to-controller behavior.

Alias For

Used to select the base tag and qualifier that the tag references. For an Add-On Instruction definition, only Input and Output tags can alias Local Tags; Local Tags and InOut tags cannot alias another tag. The controller must be offline before you can make changes to this box.

Data Type

Used to select the type of tag to create. Used the to open the Select Data Type dialog box, and select a data type.

The following data types require additional configuration:

- Message
- AXIS_CONSUMED
- AXIS_GENERIC
- AXIS_GENERIC_DRIVE
- AXIS_CIP_DRIVE
- AXIS_VIRTUAL
- AXIS_SERVO
- AXIS_SERVO_DRIVE
- Motion Group
- CAM array
- CAM PROFILE array
- PID

For alias tags, this box is read-only because the data type of the alias tag is defined by the base tag.

The following tags can be used only in controller scope and do not support arrays:

- AXIS_CIP_DRIVE
- AXIS_CONSUMED
- AXIS_GENERIC
- AXIS_GENERIC_DRIVE
- AXIS_SERVO, AXIS_SERVO_DRIVE
- AXIS_VIRTUAL
- COORDINATE_SYSTEM
- MESSAGE
- MOTION_GROUP

There are additional considerations for data types that are used with safety tags.

The following data types cannot be used when creating Safety tags:

AXIS_CONSUMED, AXIS_GENERIC, AXIS_GENERIC_DRIVE, AXIS_CIP_DRIVE, AXIS_SERVO, AXIS_SERVO_DRIVE, AXIS_VIRTUAL, MOTION_GROUP, MESSAGE, COORDINATE_SYSTEM, ALARM_ANALOG, ALARM_DIGITAL, and MODULE data types.

The software prevents the direct creation of invalid tags in a safety program. In the event that invalid tags are imported, they cannot be verified.

In Safety programs, the Logix Designer application also prevents:

- The creation of a Safety tag using a structured type (User-Defined, Add-On Defined, predefined) when they contain one or more members of type REAL. This limitation includes nested structures.
- The modification of a User Defined or Add-On Defined type that would cause an invalid data type to be included when the User Defined or Add-On Defined type is already referenced directly or indirectly by a Safety tag. This limitation includes nested structures.
- Invalid tags from being created using either the New Program Parameter or Tag dialog box or Tag Properties dialog box.

Parameter Connection	Used to select a select a single connection for the parameter. If you need to make multiple connections, select the Open Parameter Connections check box. When you click Create, the Configuration Connection dialog box opens and you can view and configure connections for the parameter.
Scope	Used to select the scope in which to create the tag. Select controller scope or one of the existing programs on the controller. If you are creating a new program parameter from a language editor, and the parameter is from another program, the program name appears in the Scope box.
Class	Indicates whether the tag is a Standard tag or a Safety tag. This value appears for Safety applications only.
External Access	Used to select the access allowed to the tag from external applications such as HMIs. The available options include: • Read/Write, • Read Only, • None (no access)
Style	The display format for this data type. The following settings are available: • Binary, Decimal, Hex, Octal, ASCII, or Exponential • Date/Time - Displays the tag value as date and time with millisecond precision. Using this style, the displayed tag value might look like the following example: DT#2014-05-28-19:00:01.200_000 (UTC-05:00)

- Date/Time (ns) - Displays the tag value as date and time with nanosecond precision. Using this style, the displayed tag value might look like the following example:

  LDT#2014-05-28-19:00:01.200_000_000 (UTC-05:00)

Constant	Select the Constant check box topreventexecuting logic from writing values to the associated tag or parameter. If a logic routine tries to write to a constant tag, the routine will not verify. The Constant check box is not available for alltag types. Tip: You can use the Data Monitor dialog box to write values to the parameter. The Value boxes in all other browsers and language editors are disabled when the tag is a Constant tag. Changes made to the values of constant tags are recorded in the Controller Log for future reference.
Sequencing	Enables the tag or parameter visible on the FactoryTalk Batch server to be visible. Only tags and parameters on which Usage is set to Input or Output can be used by a sequence.
Open Configuration	Enables a configuration wizard or dialog box to open after you click Create. The Open Configuration check box is available for only specific data types and is not available for program scope tags that are set to InOut for Usage.
Open Parameter Connections	Enables the Configuration Connection dialog box to open after you click Create.

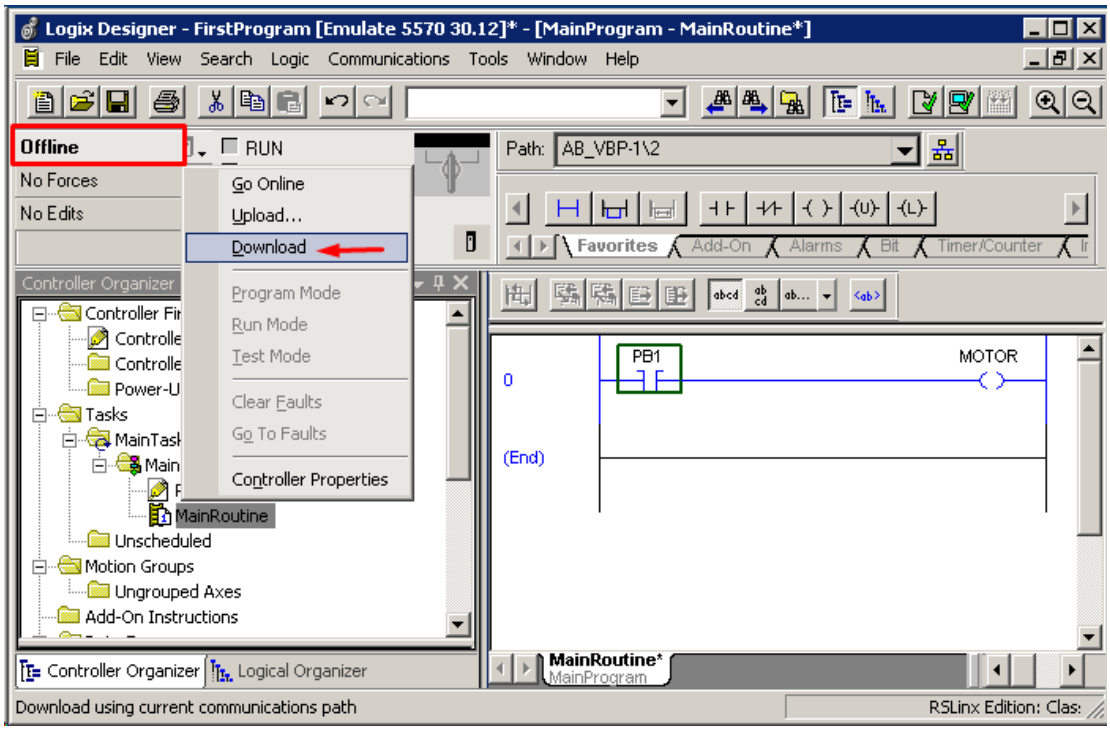
After define the tags, a valid rung should look like this. Input instruction will be on the left side of the run, and output instruction will be on the right.

Correct:

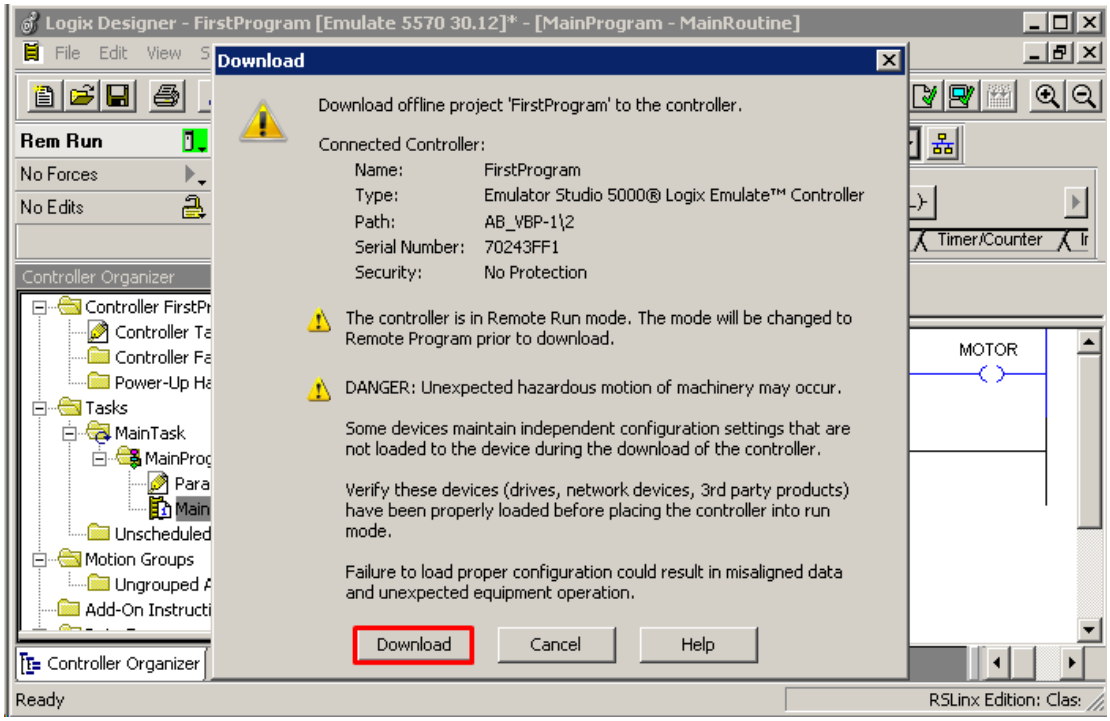
Wrong:

The **RED** color indicate an error on the rung. You can verify the routine by click on the Verify icon on the menu bar. Logix Designer will no allow program to download if error is not clear.

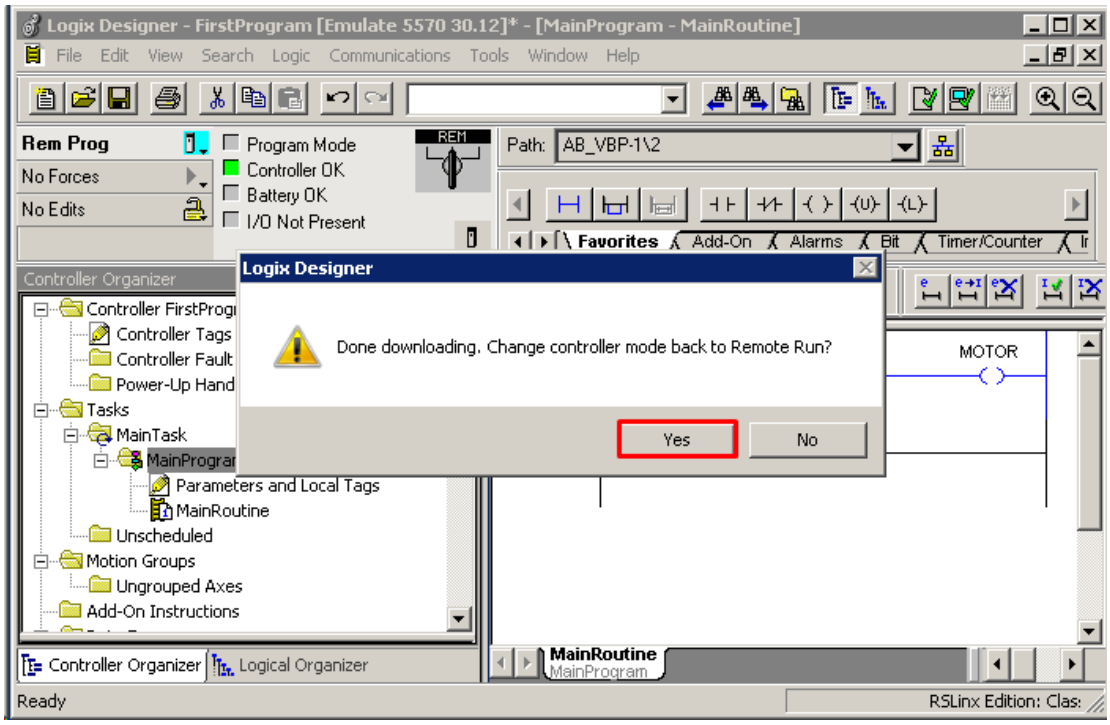
Click the **Online** menu and select **Download** from the drop down.



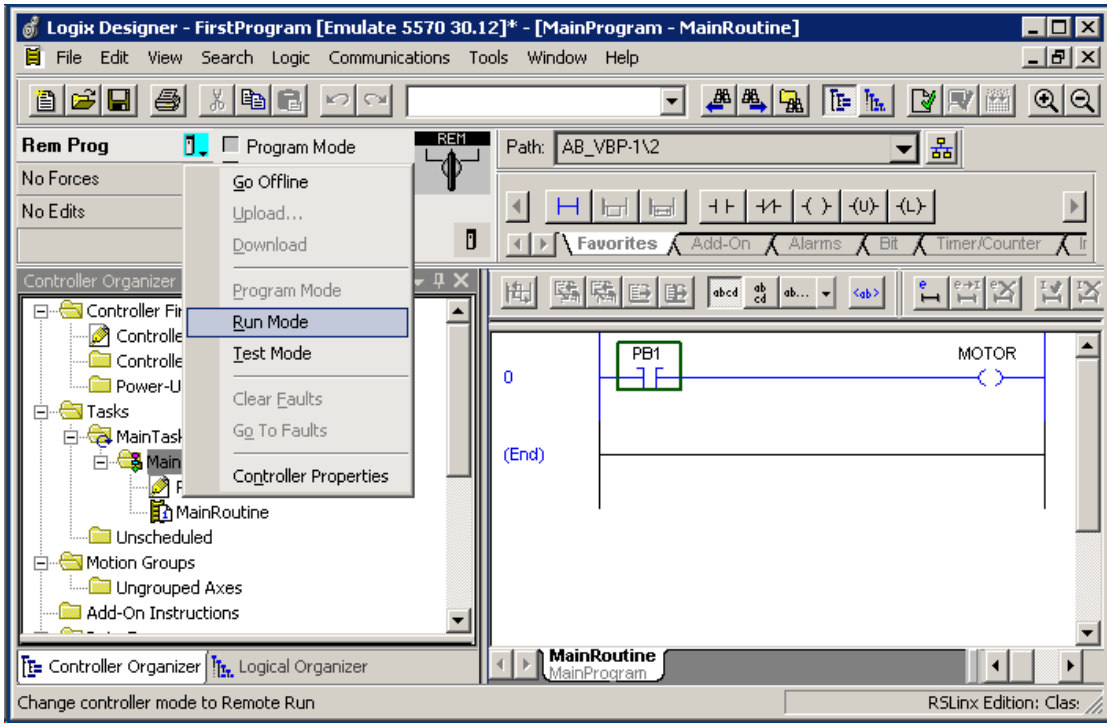
Click the **Download** button when prompted.



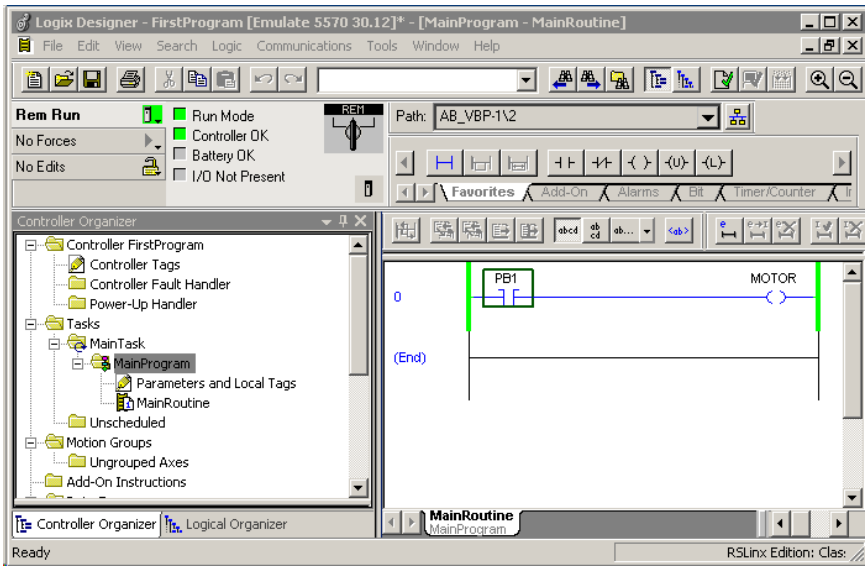
Click the **Yes** button to continue.



Switch the program to Run Mode. If the program is already in Run Mode then no action is required.

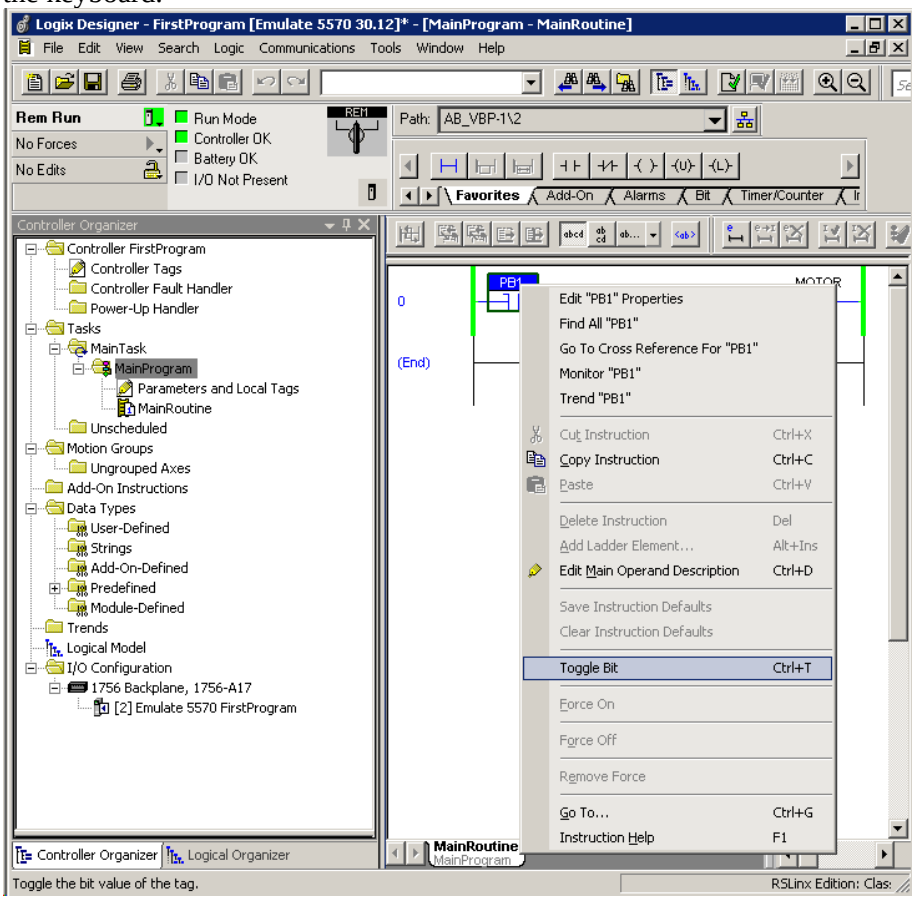


Green color indicate the controller is in run mode.

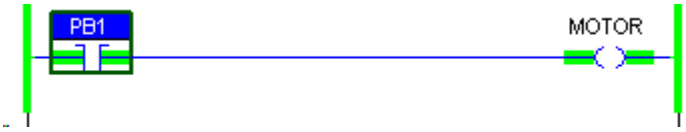


Since no input or output module installed. We need to manually toggle the input to test the logic.

Right click on the "PB1" and select **Toggle Bit (CTRL +T)** or press **CTRL + T** shortcut on the keyboard.



Now the "PB1" is TRUE, output "MOTOR" will become energized.



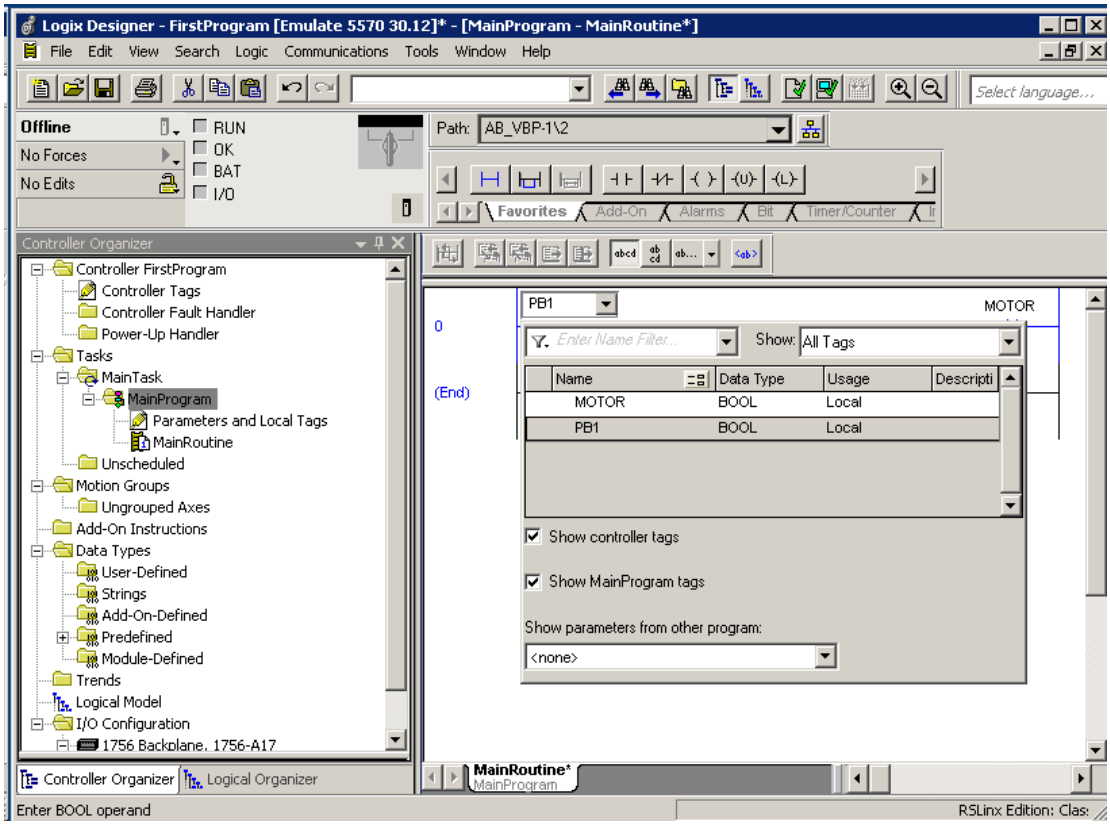
5. EXERCISE 2 - NOT LOGIC

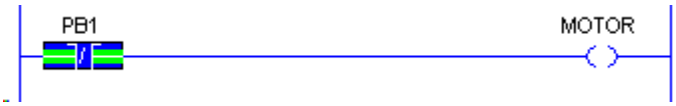
NOT Logic (Inverted)

Modify the program from exercise 1 by delete the **XIC** instruction. Drag the **XIO** instruction from the Favorites tab.

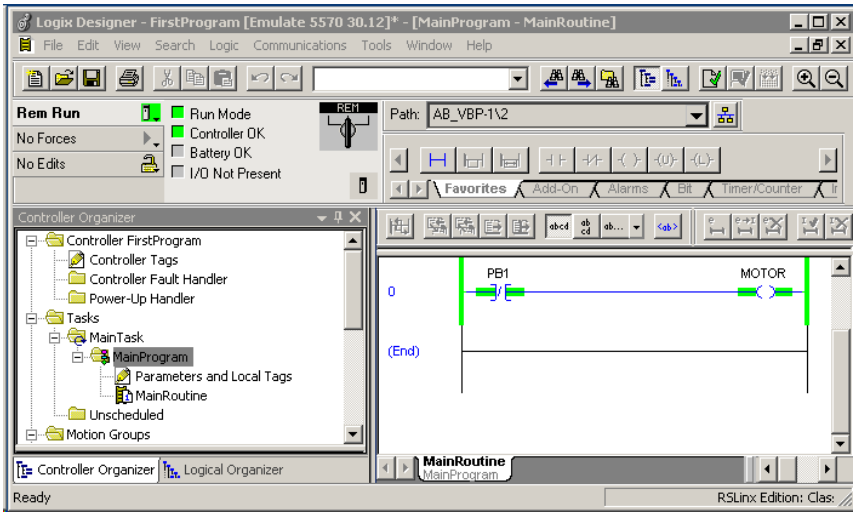
Since you already created the "PB1" from the previous exercise. You can reuse the "PB1" tag by double click on the **?** mark.

Select the "PB1" from the drop down.



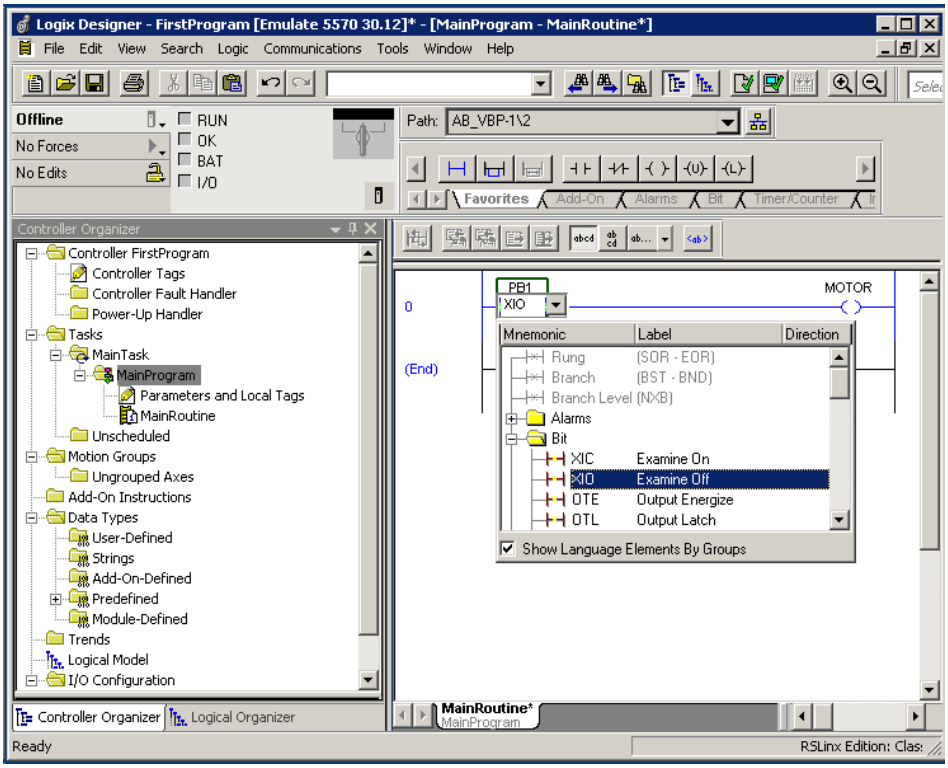


Download the program and observe the outcome of the logic. Output "MOTOR" is ON without needing to toggle "PB1" because the **XIO** instruction acts like a NOT logic.



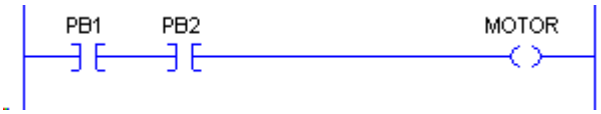
Information:

Instead of delete and recreate the instruction. An easy way to modify the instruction is by double click on the instruction itself. A drop down menu will list all the available bit instructions. Double click on the **XIO** Examine Off instruction to change from XIC to XIO.

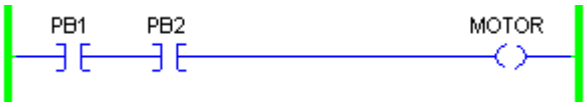


AND Logic

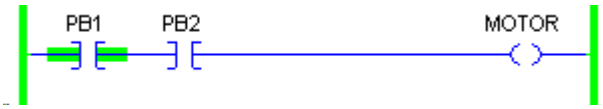
Create and define a new tag call "**PB2**".

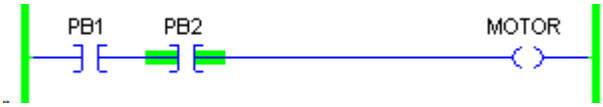


Download and observe the outcome. The logic has two **XIC** instructions connected in series.

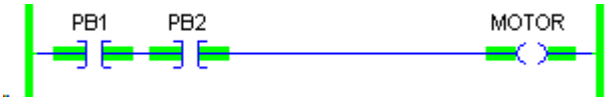


Toggle either "PB1" or "PB2" only will not energize the output "MOTOR" because "PB1" and "PB2" are connected in series.





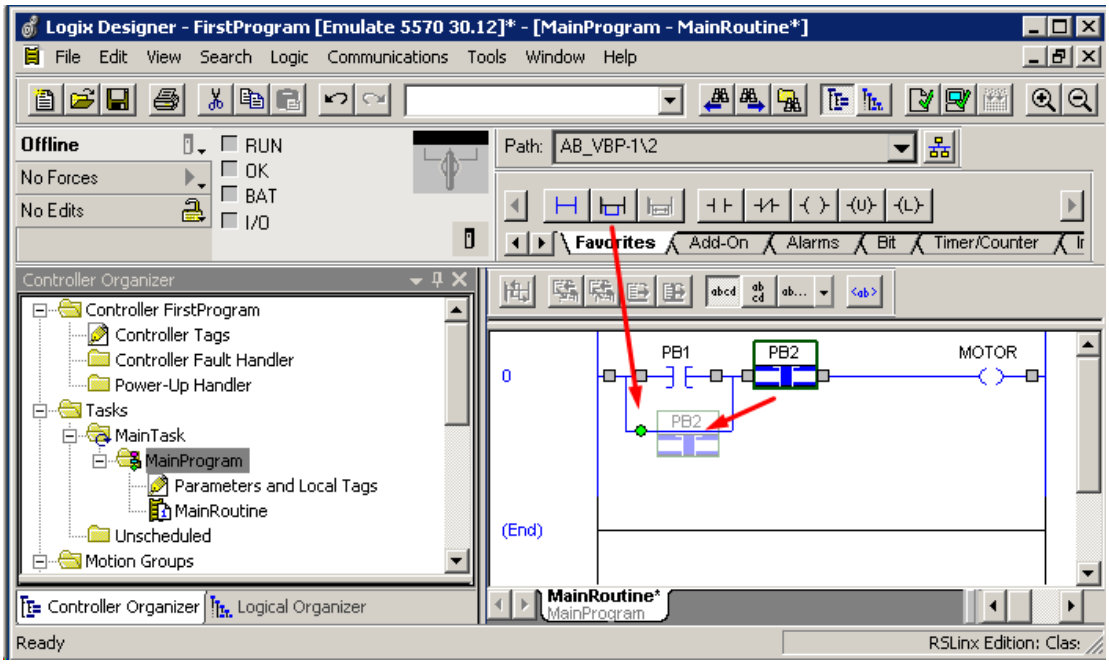
Output "MOTOR" only energize when both buttons ("PB1", "PB2") are TRUE.

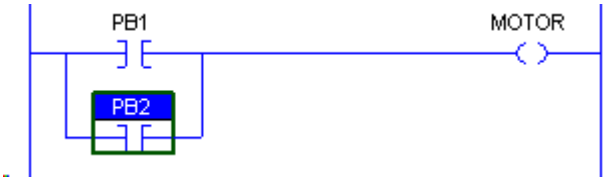


6. EXERCISE 4 - OR LOGIC

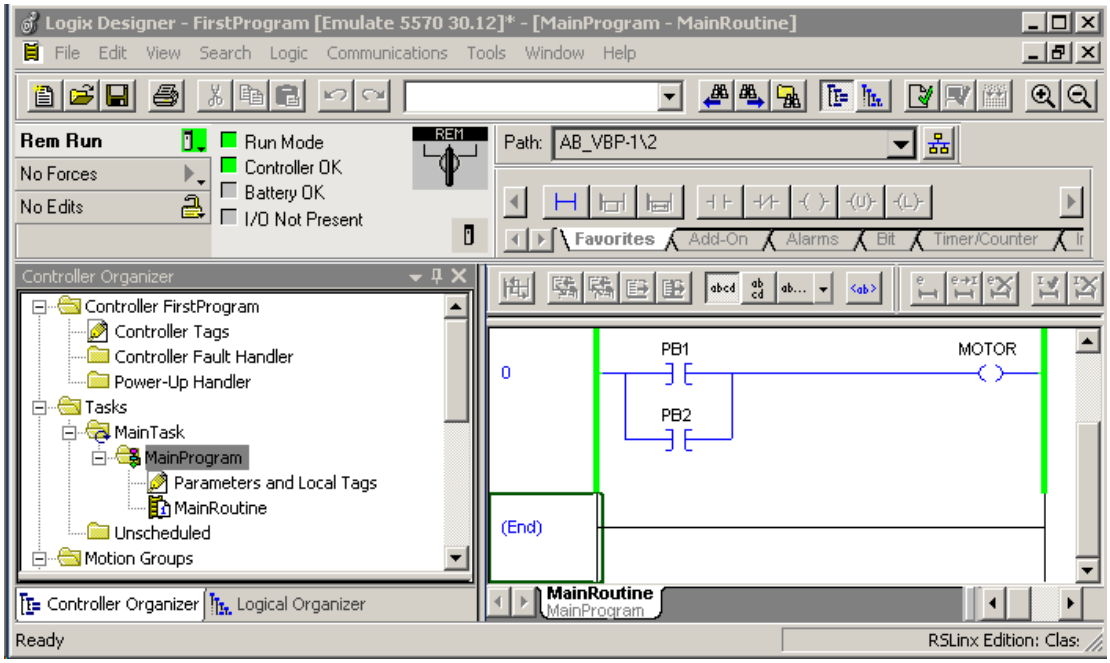
OR Logic

Drag and drop a branch from the Favorites tab to rung 0. Highlight the bit instruction, use drag and drop to move the button to the correct location.

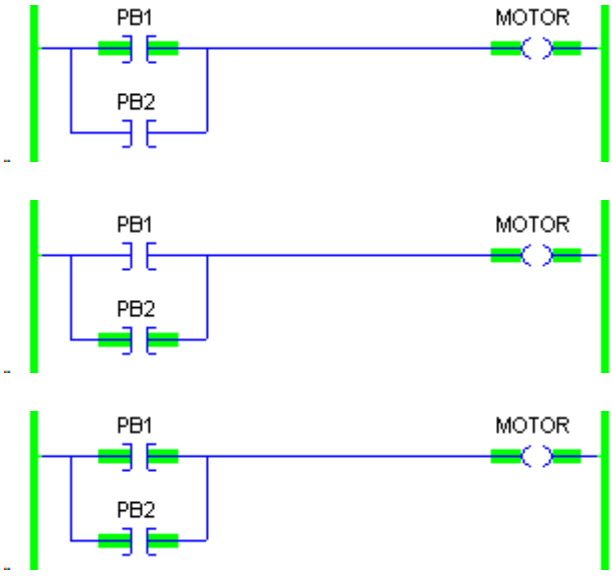




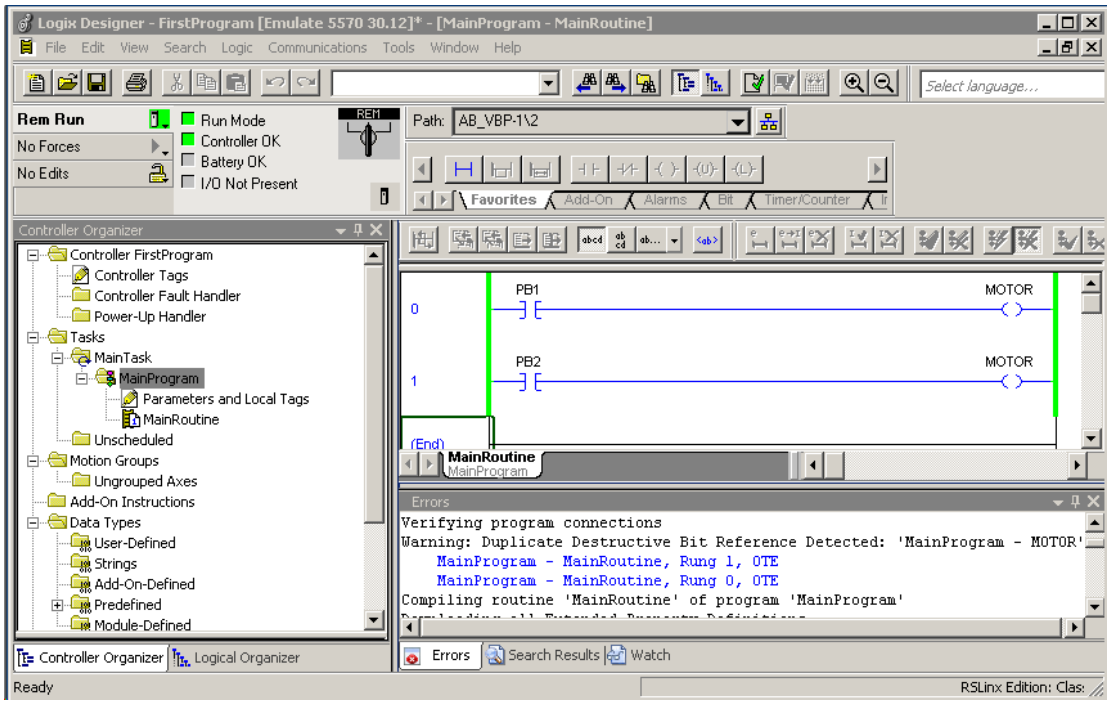
Download the program and switch to Run Mode.



Toggle either "PB1" or "PB2" will energize the "MOTOR" because "PB1" and "PB2" are connected in parallel.

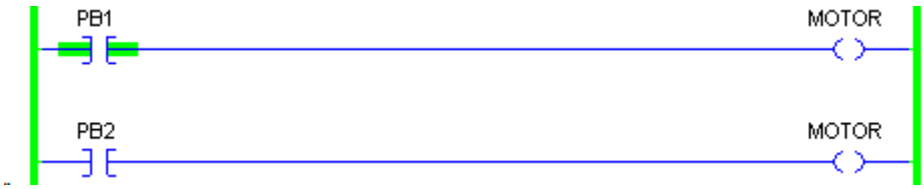


RULE: NEVER duplicate an output except when use latch and unlatch instructions. Even thought the program can be downloaded successfully, but a warning message will indicate "Duplicate Destructive Bit Reference Detected".

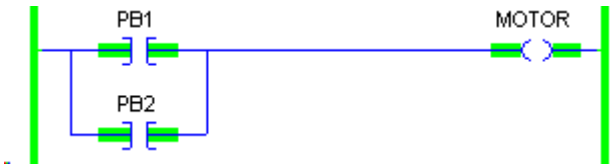


"MOTOR" should turn ON according to the first rung, but second rung contradict with the first rung because "PB2" is OFF.

WRONG



CORRECT

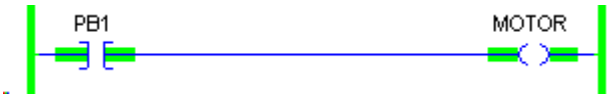


7. EXERCISE 5 - COMPLEX LOGIC

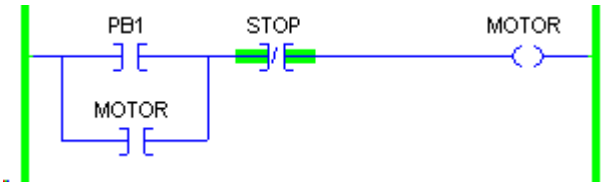
Complex Logic

If "PB1" is a momentary push button, by Press and Hold the "PB1", "MOTOR" stays ON.

Release push button "PB1" will reset the bit which turns OFF the "MOTOR".

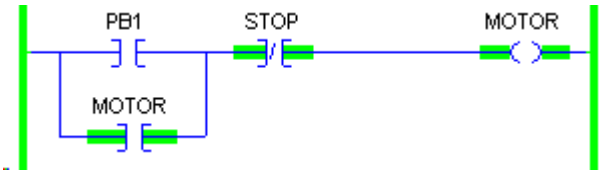


Let's create a start-stop logic that allows the "MOTOR" to stay on even after released the push button "PB1". Also implement a "STOP" button to turn OFF the "MOTOR" on demand.

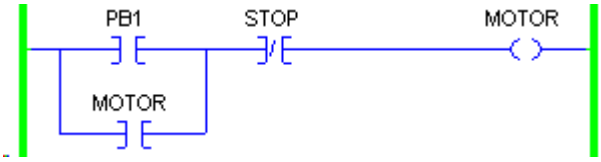


To simulate a momentary button for "PB1". Highlight the "PB1" and press **CTRL+T** twice on the keyboard.

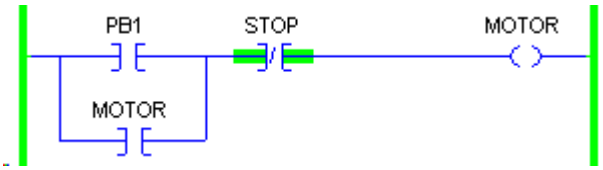
The reason output "MOTOR" stays on because tag "MOTOR" is being used as an input condition in an OR scenario. When "PB1" is TRUE, output "MOTOR" and input "MOTOR" become energized at the same time.



Toggle the "STOP" button will turn OFF the "MOTOR"



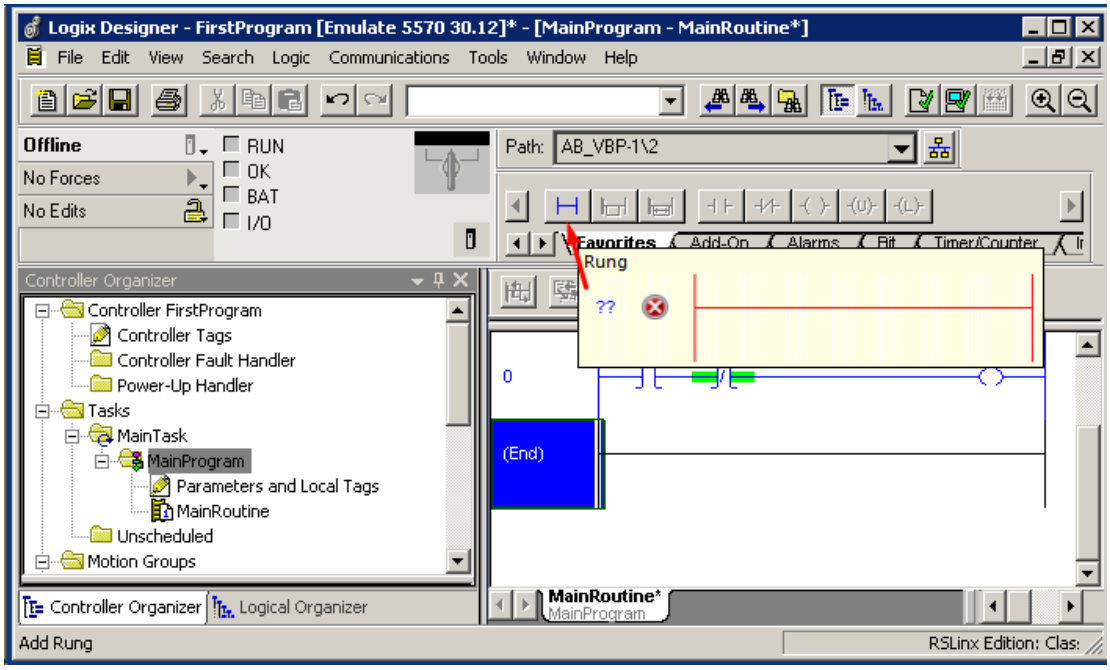
Remember to toggle the "STOP" button again otherwise "MOTOR" can not be started.


PB1
STOP
MOTOR
MOTOR

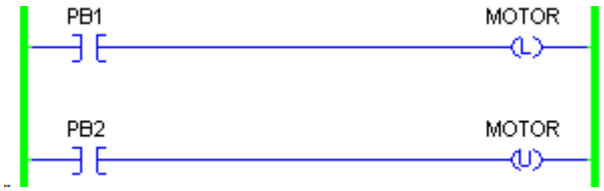
8. EXERCISE 6 - LATCH AND UNLATCH

Latch and Unlatch

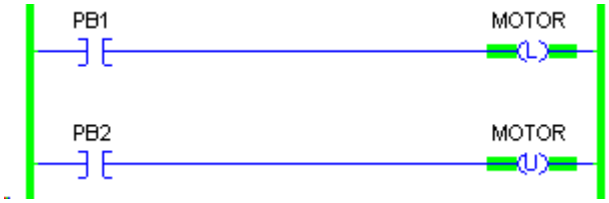
Go to the Favorites tab and insert a new rung.



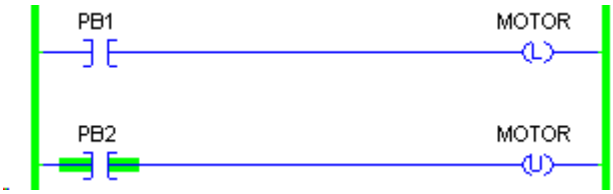
Create the logic below using **OTL** and **OTU** instructions. When "PB1" is TRUE then **OTL** instruction sets the data bit to 1 for "MOTOR". The data bit remains TRUE until it is cleared by "PB2" with the **OTU** instruction.



Press "PB1" to set the data bit.



Press "PB2" to clear the data bit.



Note: OTL and OTU MUST use in pair and both instructions use the same tag.

Information:

Output Latch (OTL)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, CompactLogix 5480, and ControlLogix 5580 controllers.

The OTL instruction sets (latches) the data bit.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

Ladder Diagram

Operand	Type	Format	Description
Data bit	BOOL	tag	Bit to be modified. There are various operand addressing modes possible for 'Data bit', see common attributes.

Description

When true, the OTL instruction sets the data bit. The data bit remains true until it is cleared, typically by an OTU instruction. When disabled, the OTL instruction does not change the status of the data bit.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See common attributes section for operand related faults.

Execution

Ladder Diagram

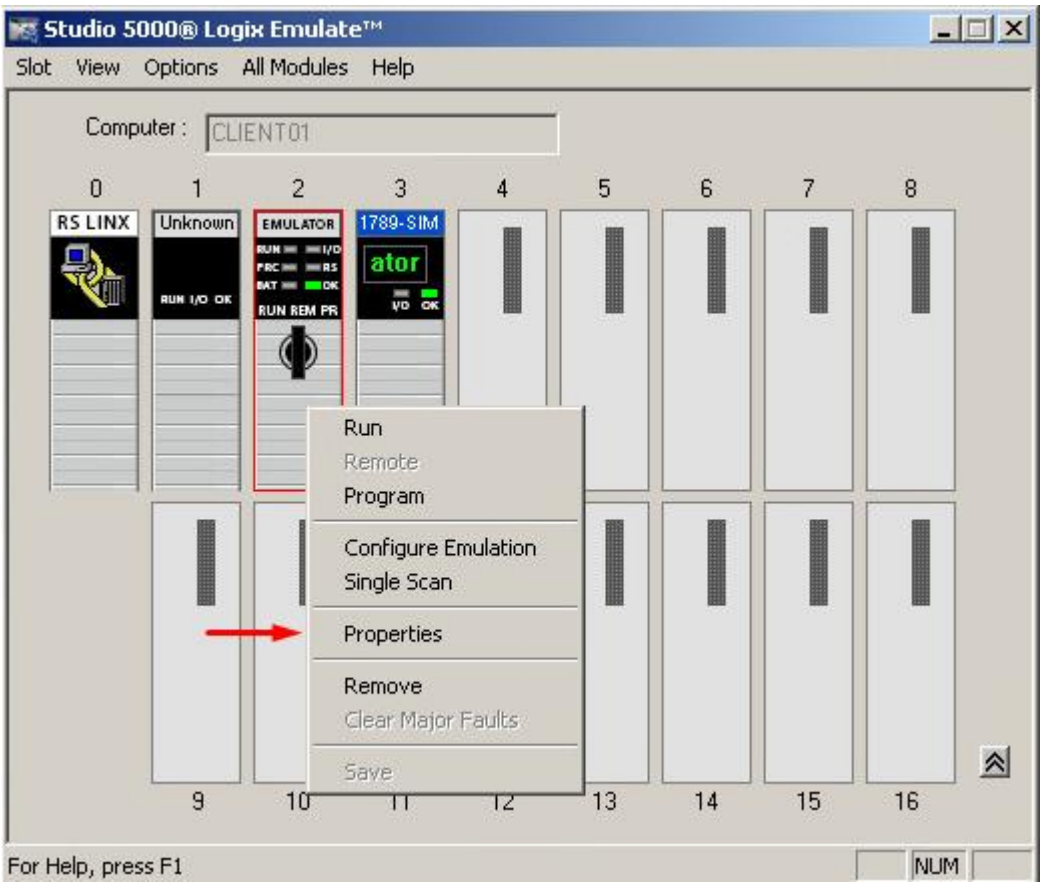
Condition/State	Action Taken
Prescan	N/A.
Rung-condition-in is false	N/A
Rung-condition-in is true	The data bit is set to true.
Postscan	N/A

For CompactLogix 5380, CompactLogix 5480, and ControlLogix 5580 controllers only, if the operand is an indirect array reference and the subscript is out of range, then the controller does not generate a major fault when the OTL instruction is false.

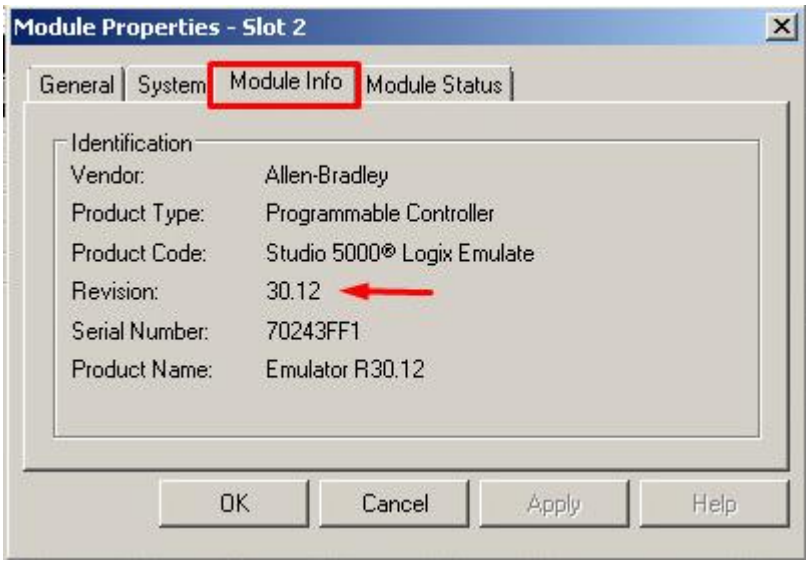
9. Q&A

1. Revision of the program MUST match with the controller version. See instruction below how to find out the revision of the controller.

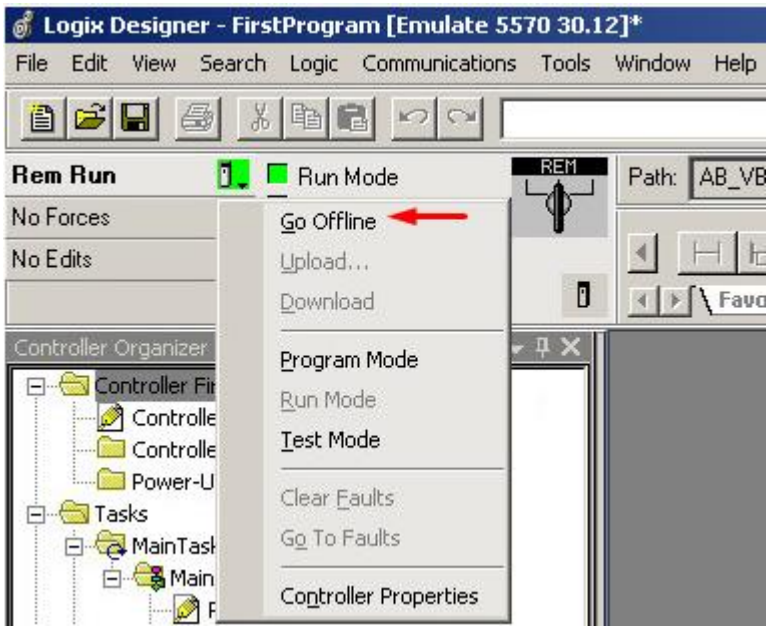
To find out the version of the Logix Emulate™ Controller installed in the chassis. Go to Studio 5000® Logix Emulate™ chassis, select slot 2 and right click on the mouse the select **Properties**.

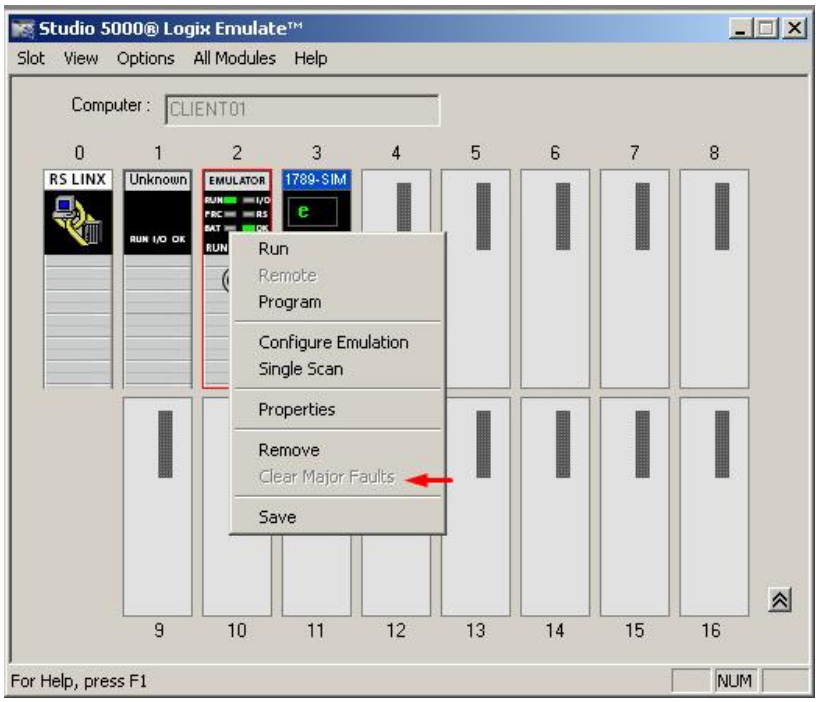


Select the Module information tab. Revision shows 30.12, ignore the minor revision .12.

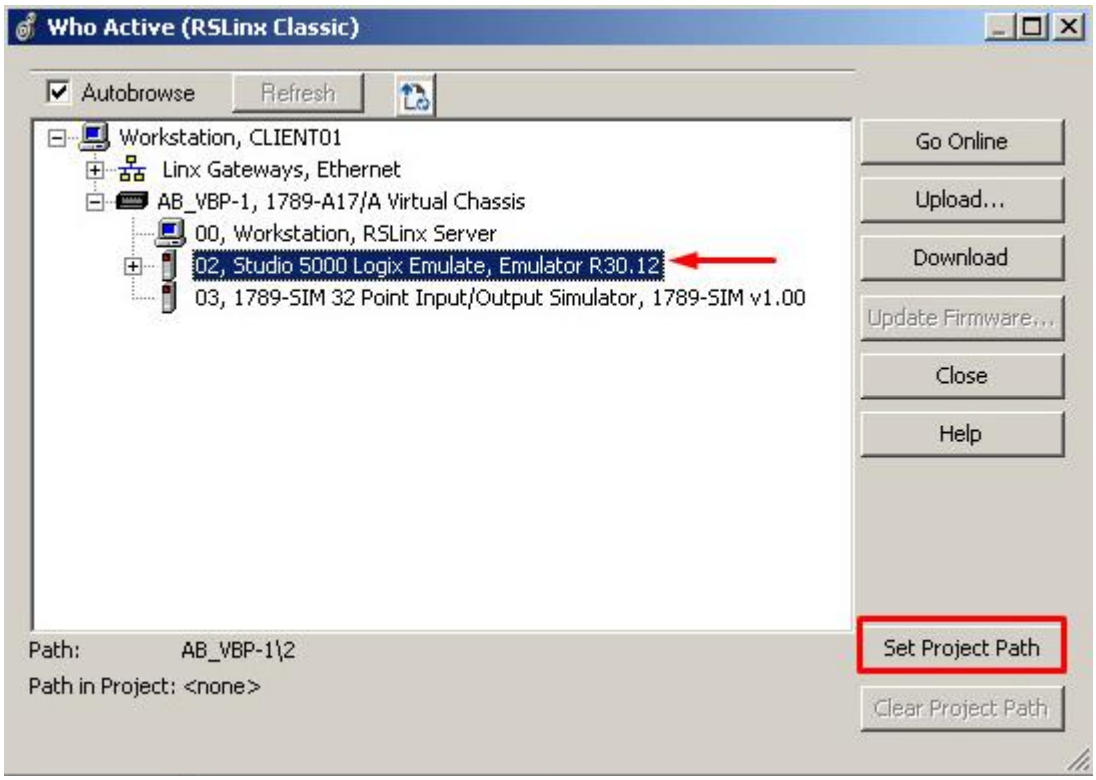


2. **RED** color indicate the controller faulted. To reset the the controller fault, go to Studio 5000 chassis. Right click on the faulted controller and select Clear Major Fault then switch the controller back to Remote.

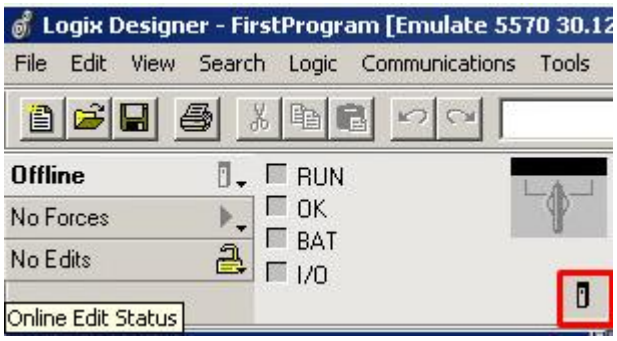


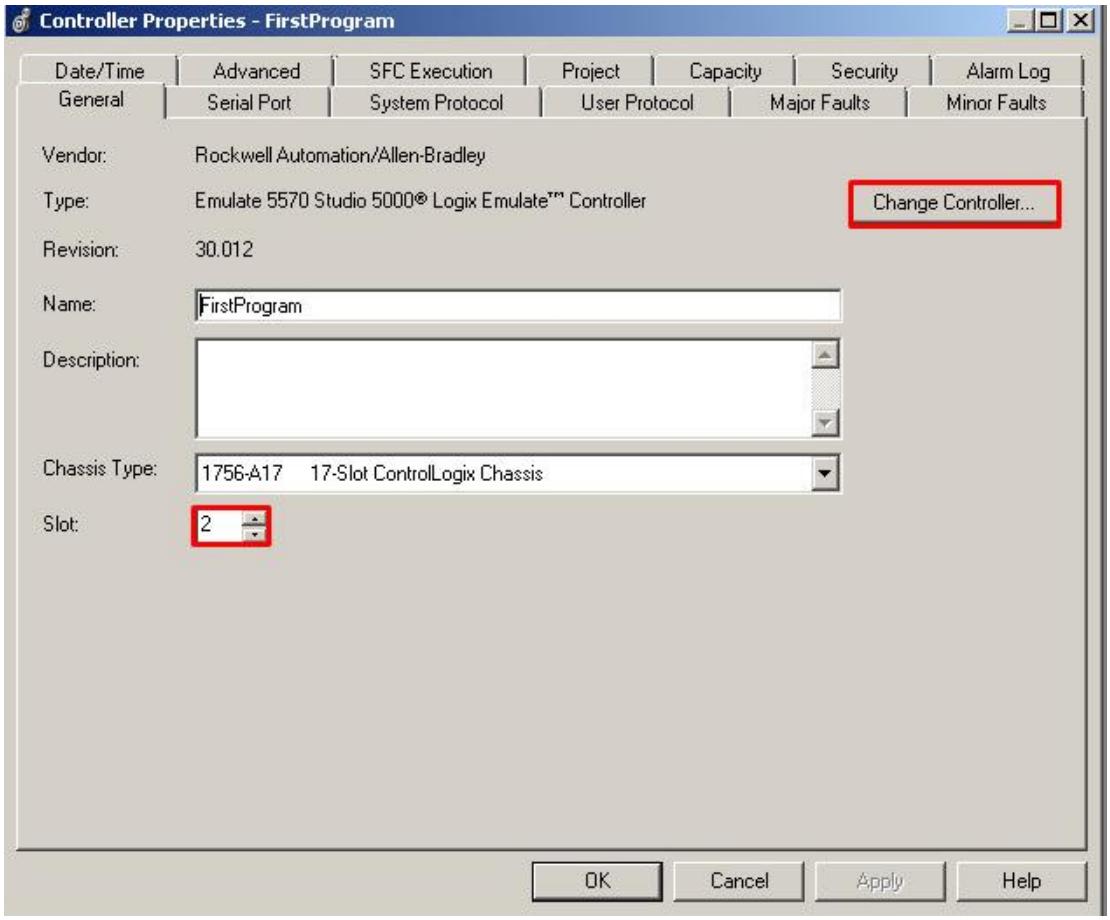


3. **Set Project Path** button is grayed out. That means the device selected is not a controller.



4. Change the controller type or slot number. Click on this controller properties icon to enter the properties menu. Make to to click on the **Apply** button to save the changes.





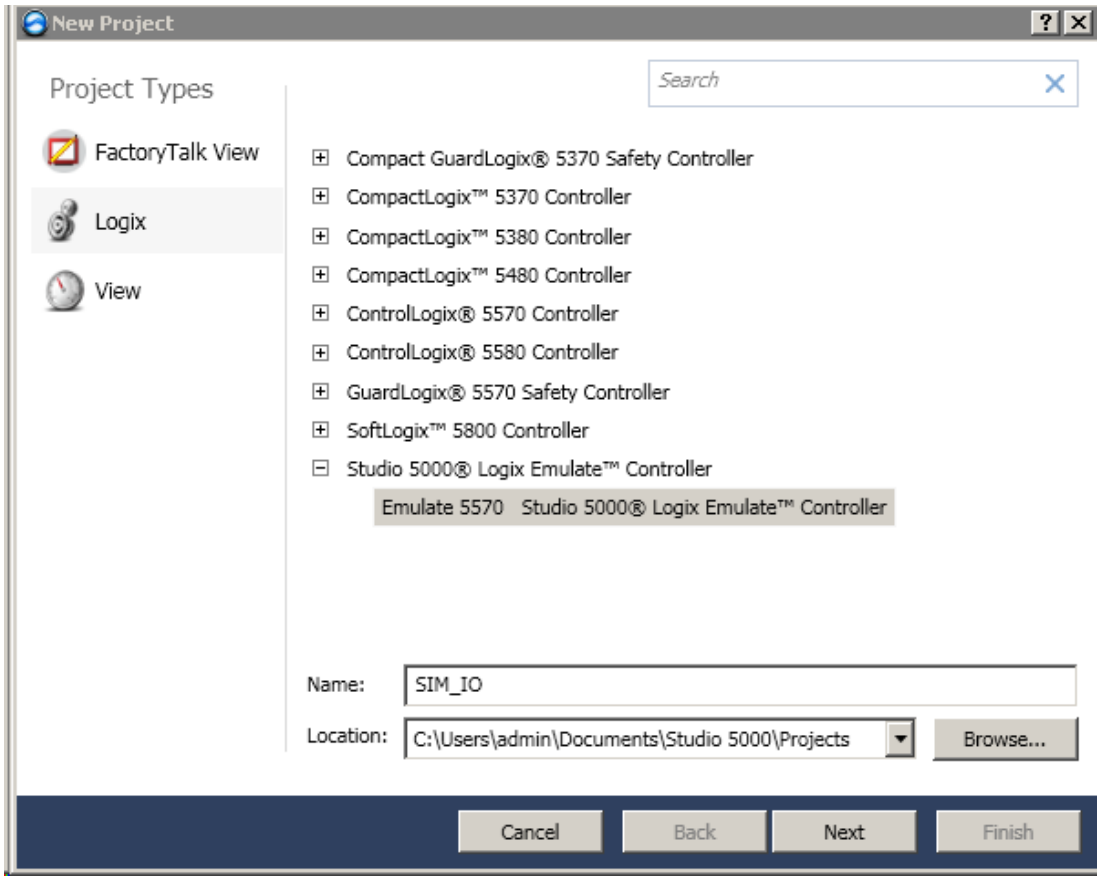
Part III 1789-SIM 32 Point Input/Output Simulator

Open Studio 5000® Logix Emulate™ chassis and install 1789-SIM 32 Point Input/Output Simulator card in slot 3 . Click here for the steps how to Install 1789-SIM 32 Point Input/Output Simulator.

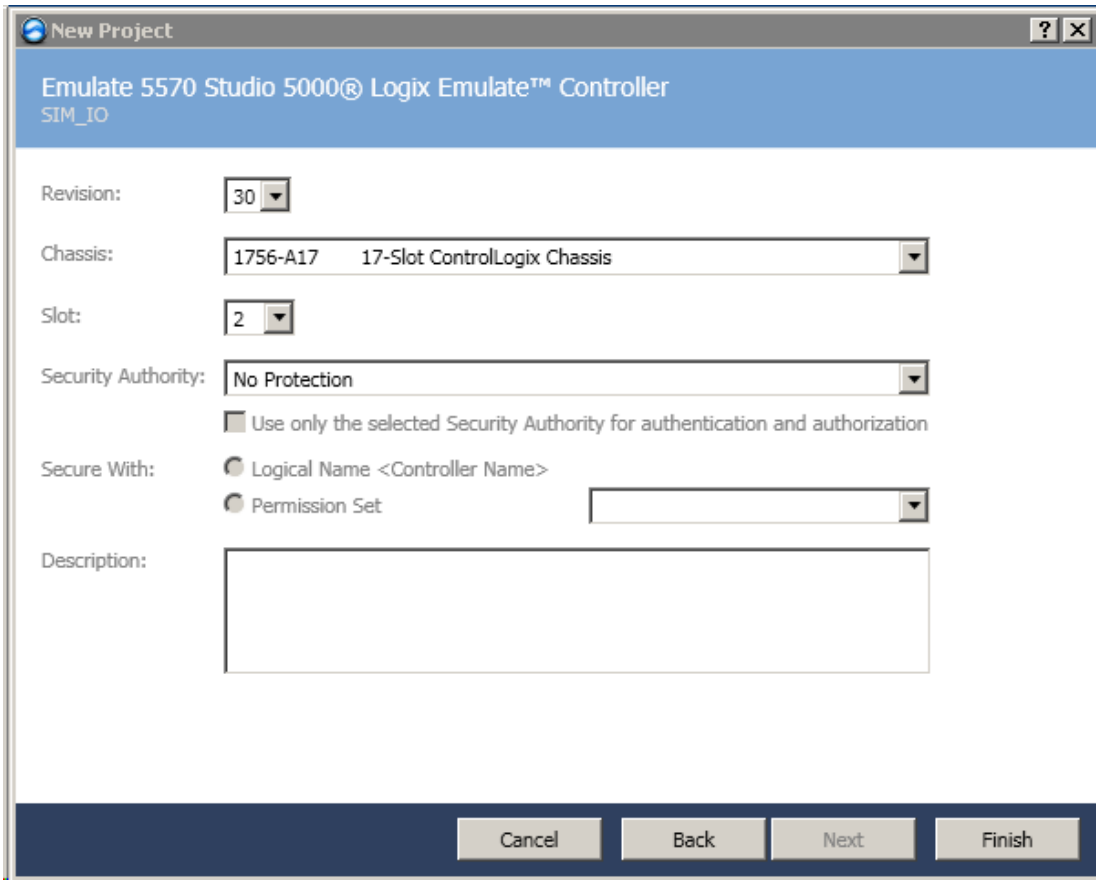
Double click on the Studio 5000 icon located on the desktop.

Create a new project

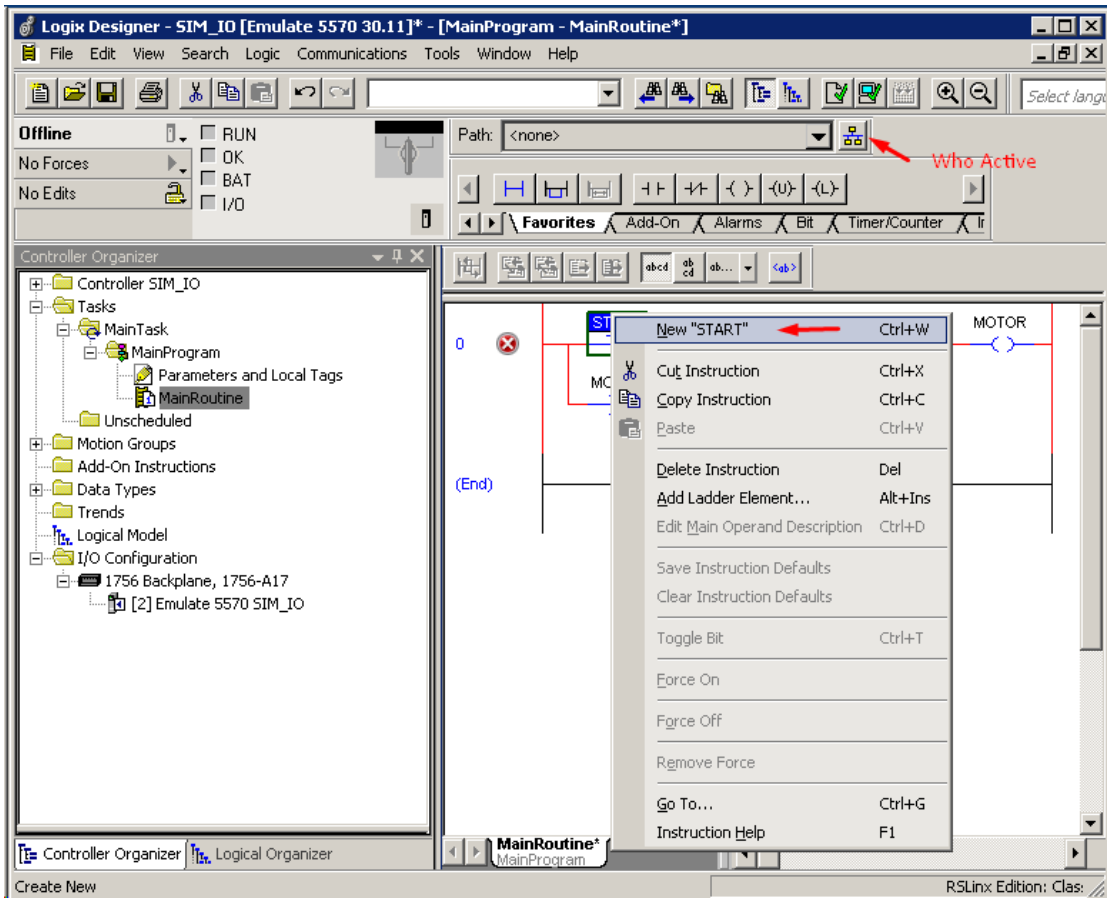
Enter a Name for the project SIM_IO. Click **Next** button to continue.



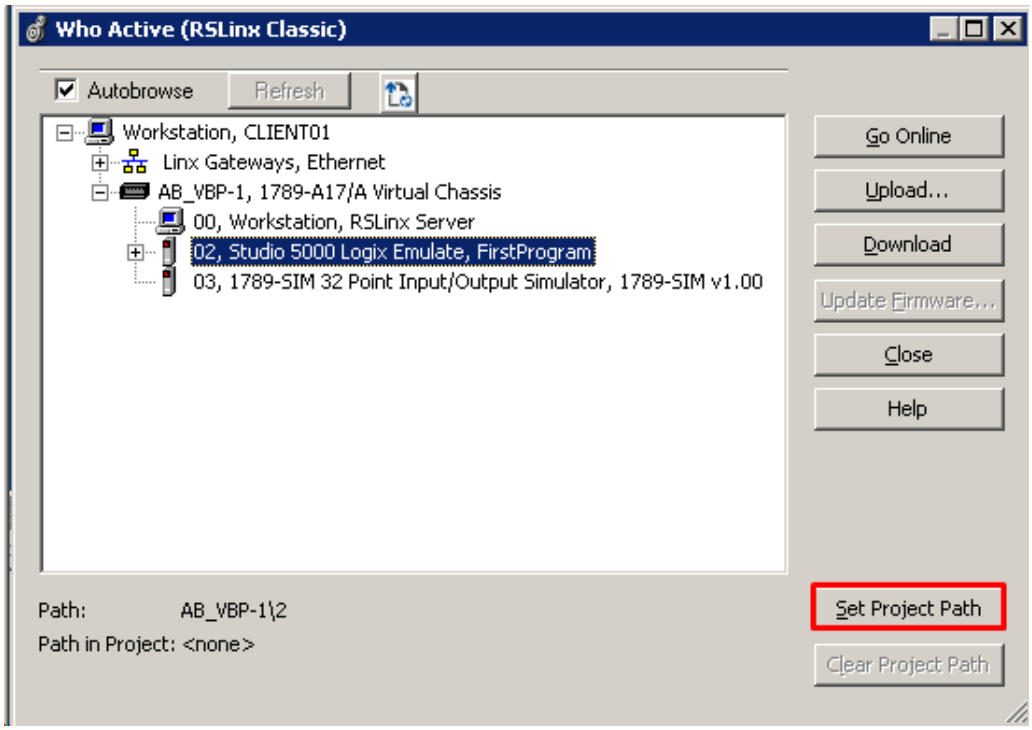
Use revision **30**, chassis **1756-A17**, and slot **2**. Click on the **Finish** button to continue.



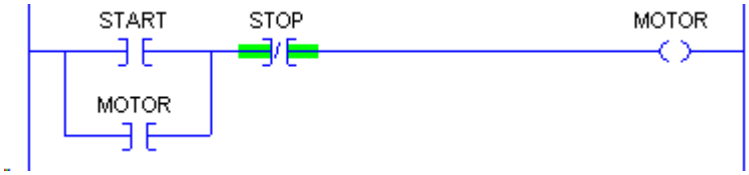
Set the Project path by click on the **Who Active** icon.



Who Active (RSLinx Classic, select the 02, **Studio 5000 Emulate**. Click the **Set Project Path** button then click the **Close** button to exit.

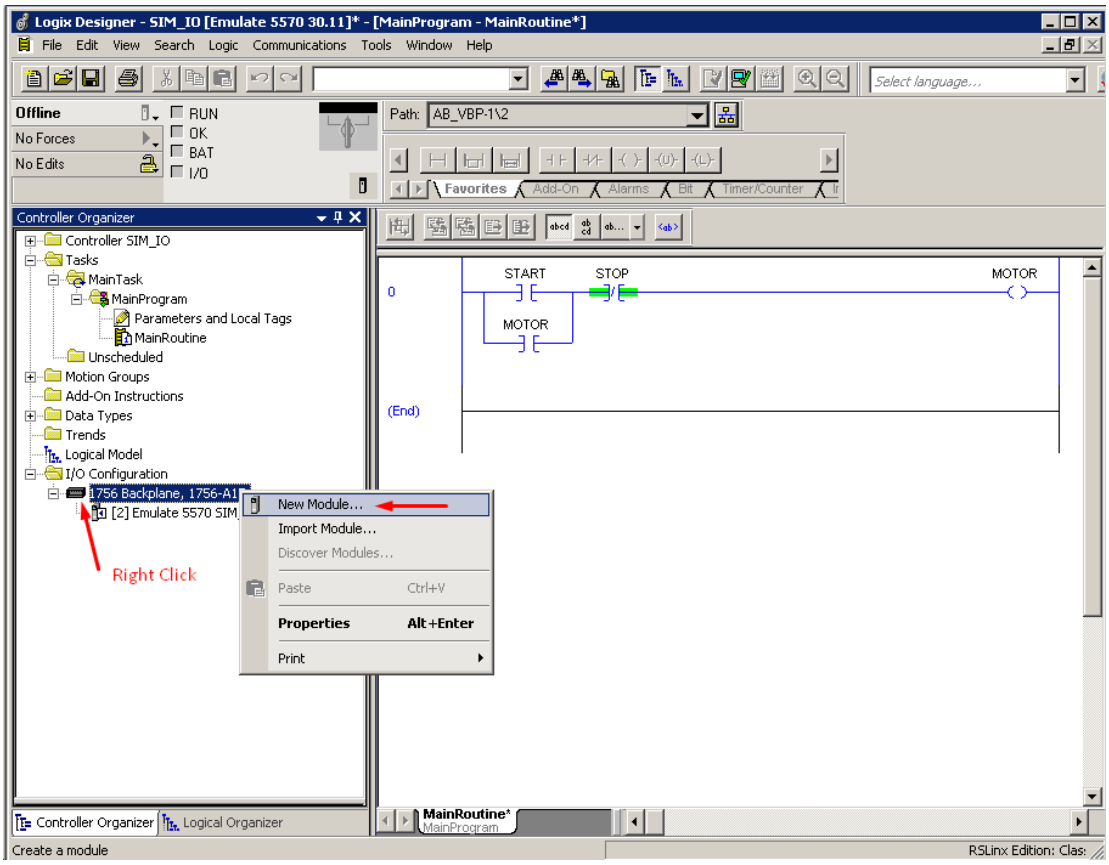


Create a start-stop logic in the MainRoutine. Remember to define the new tags by right click on the tag then select New "tag name"

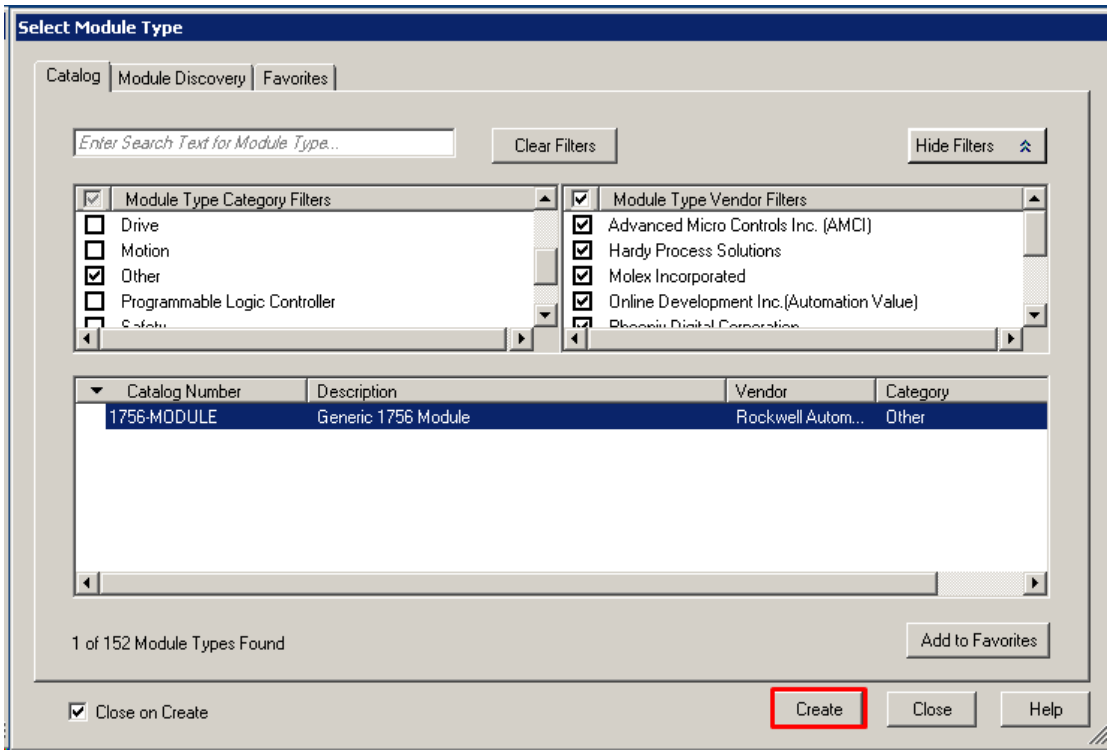


Let's start with adding the simulator card to the IO Configuration in RSLogix 5000. Right Click on the "Backplane" icon in the IO Configuration folder and select "New Module"

The window below shows the available IO cards for a SoftLogix Controller. This list will of course show many more cards when you are programming a ControlLogix Controller. A SoftLogix Controller is limited to the cards you can get that fit in a PCI slot of a PC. You can still access different types of IO modules in your application through a network. You would need to select a communications card first and then built the "Path" to the distributed IO chassis.



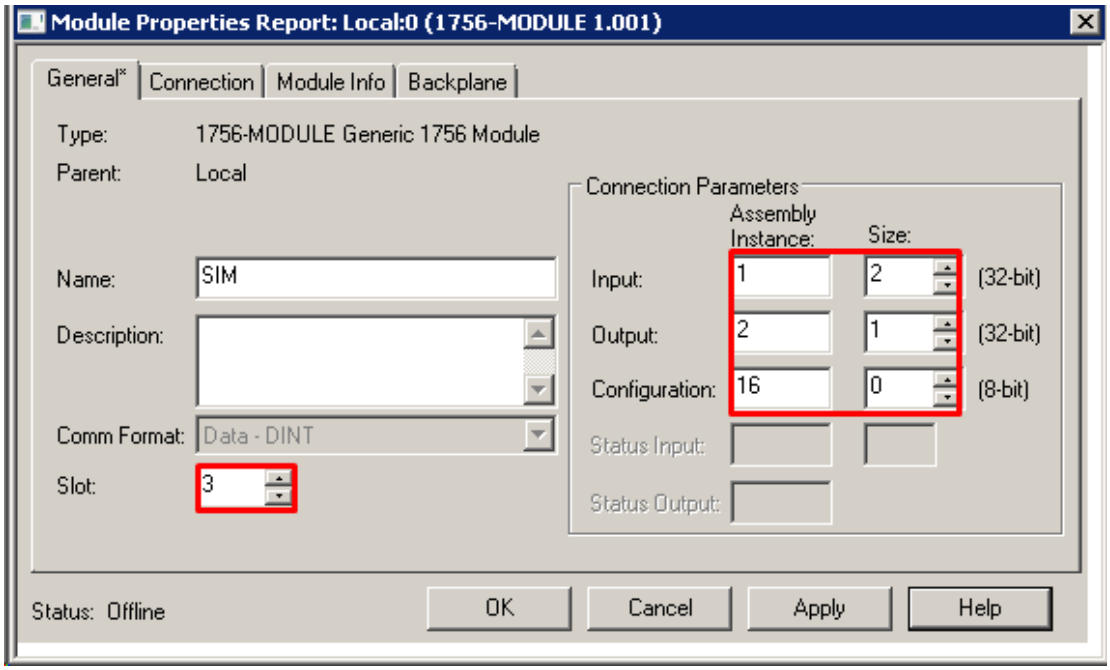
Go to Other under category, select **1756-MODULE Generic 1756 Module**. Click the **Create** button to continue.



Change the Slot number to **3**, and populate the input/output/configuration with the given values. Click **OK** button to continue.

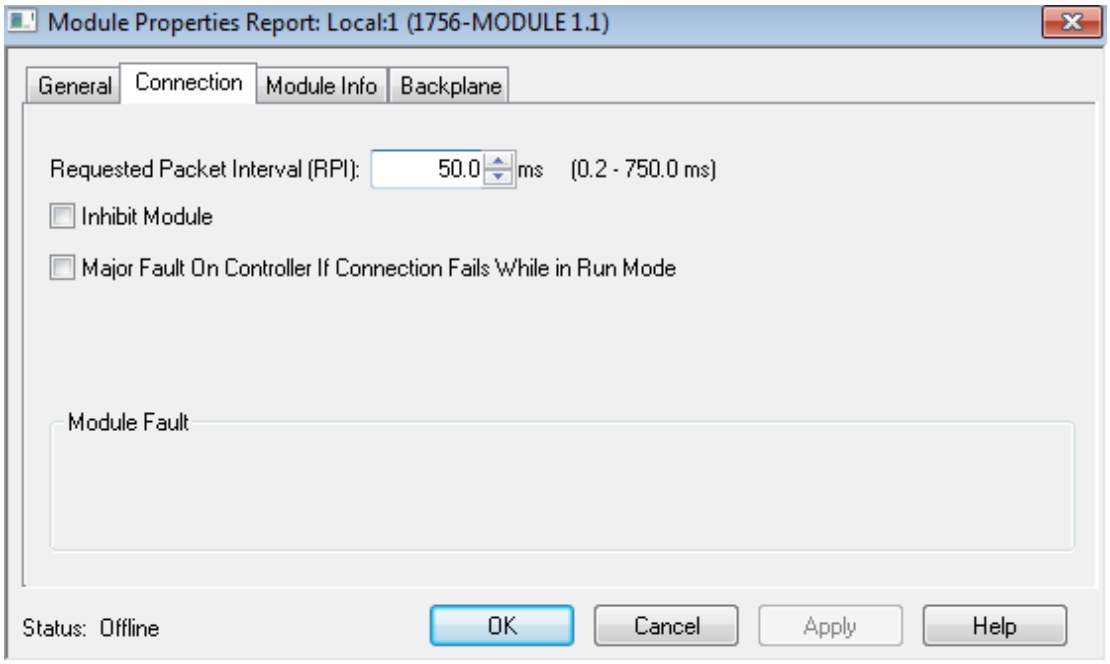
Unfortunately, the 1756-SIM module does not have a profile loaded in the module database. Most modules do and will make it much easier to create meaningful IO tags. In this situation you need to add a generic 1756 Module and fill in the details of the card. This needs to be done also if you use cards from other vendors to work with Allen Bradley. The vendor will provide all the details you need to install the card into your project.

In the “New Module” window you need to fill in the Input, Output and Configuration Assembly Instances and their sizes. This will create the DINT’S (Double Integers) that are needed to send data from the card into the processors memory. You can find this information in the SoftLogix 5800 user manual. If you have a card from a different vendor, their manual will have this information for their particular card.



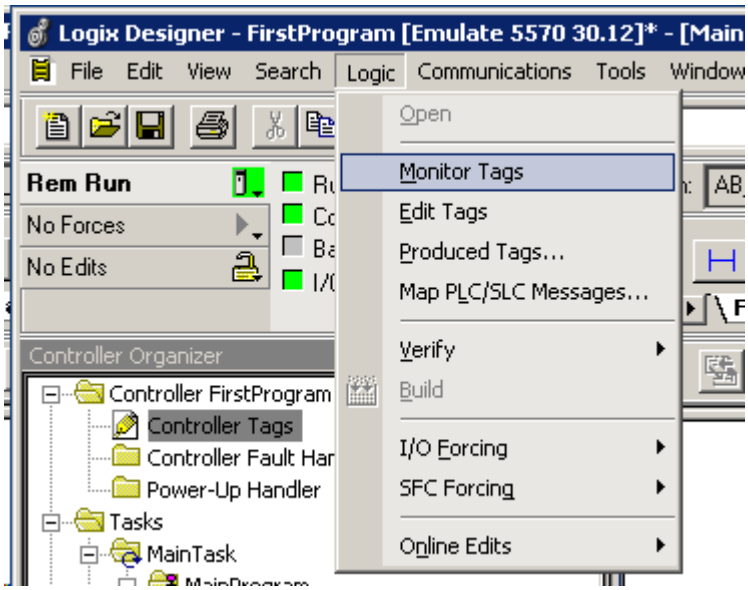
Go to Connection tab and change the Requested Packet Interval to **50** ms or more.

Because you selected a generic module in the "Select Module" window, a RPI setting of 5 ms was the default setting. The 1756-SIM card will not work at this setting and needs to be changed to 50ms. Click **Apply** and **OK** button to close the window.

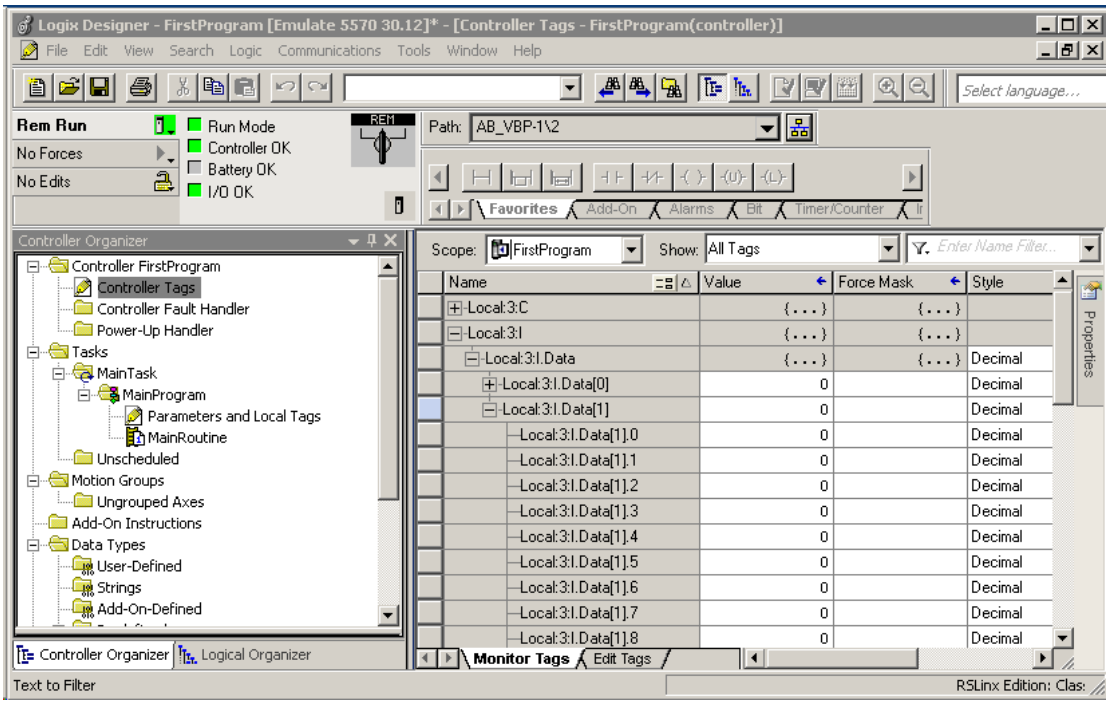


Most modules for a Logix PLC system have a factory default setting. This does not mean you always have to leave it at that setting.

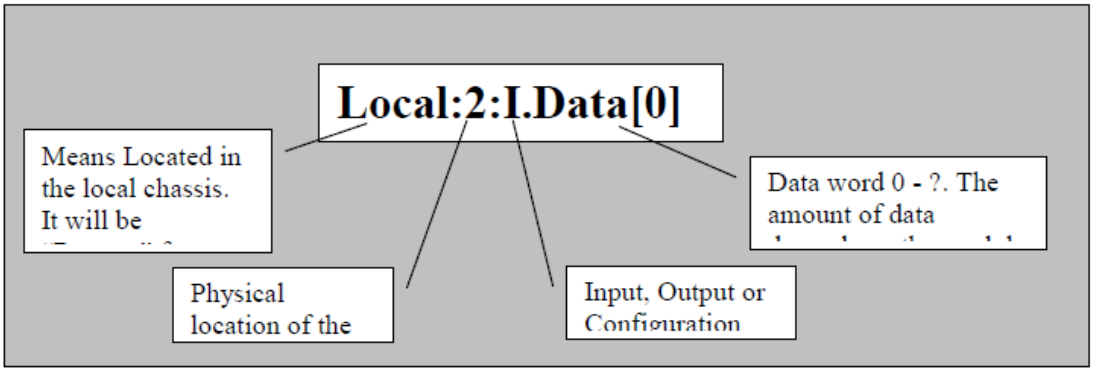
Adding an IO card in the IO configuration creates the tags for you in the tag database. Lets have a look at the IO database.



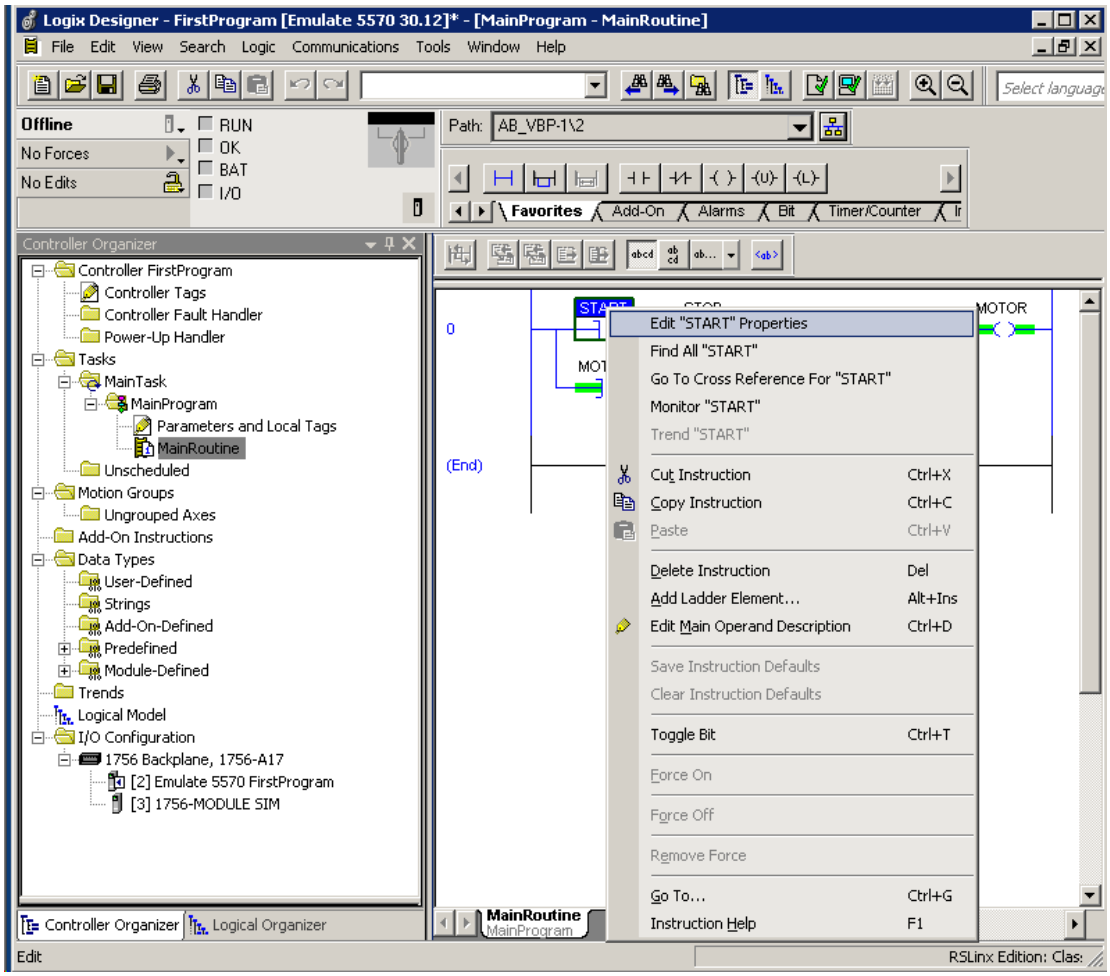
In the Logix platform PLC's you can create tags with names of your own choice. The IO cards however still need to have a tag that points to the physical location of the IO module. In a ControlLogix PLC you are allowed to put the processor in any slot you choose. You can also have multiple processors in a chassis. So there is no slot dependency anymore for any module. The CompactLogix PLC is a bit different. The CPU has to be the left-most module and IO cards to the right of it. For addressing, you still count from left to right starting at 0. So the addressing in a ControlLogix and mostly the SoftLogix is as follows.



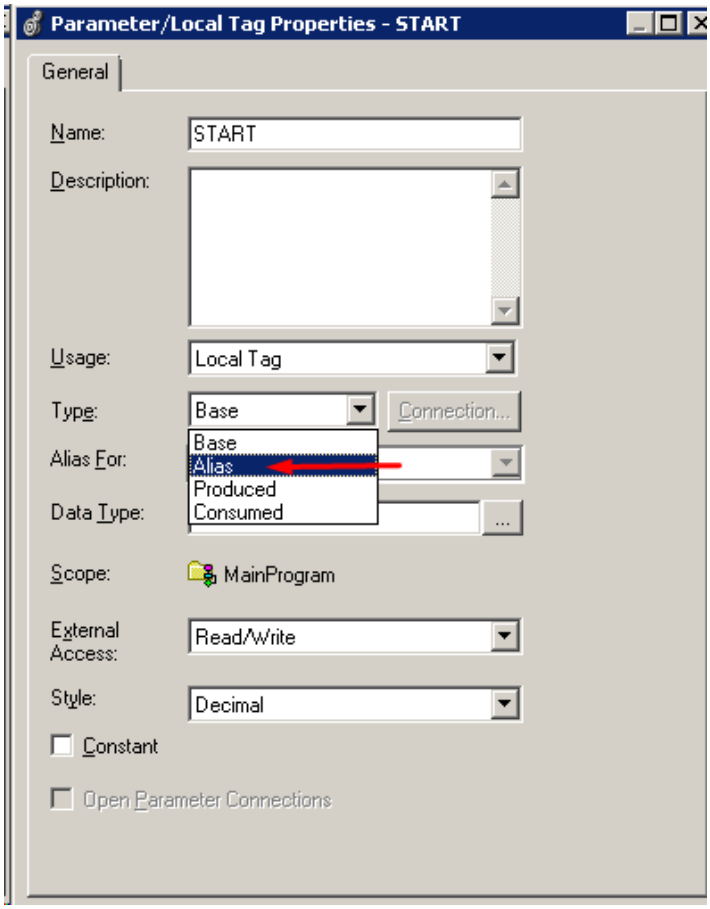
The tags described above are automatically created. Some tags however are internal tags and are not pointing to any IO point.



Right click on the "START" tag and select Edit "START" Properties

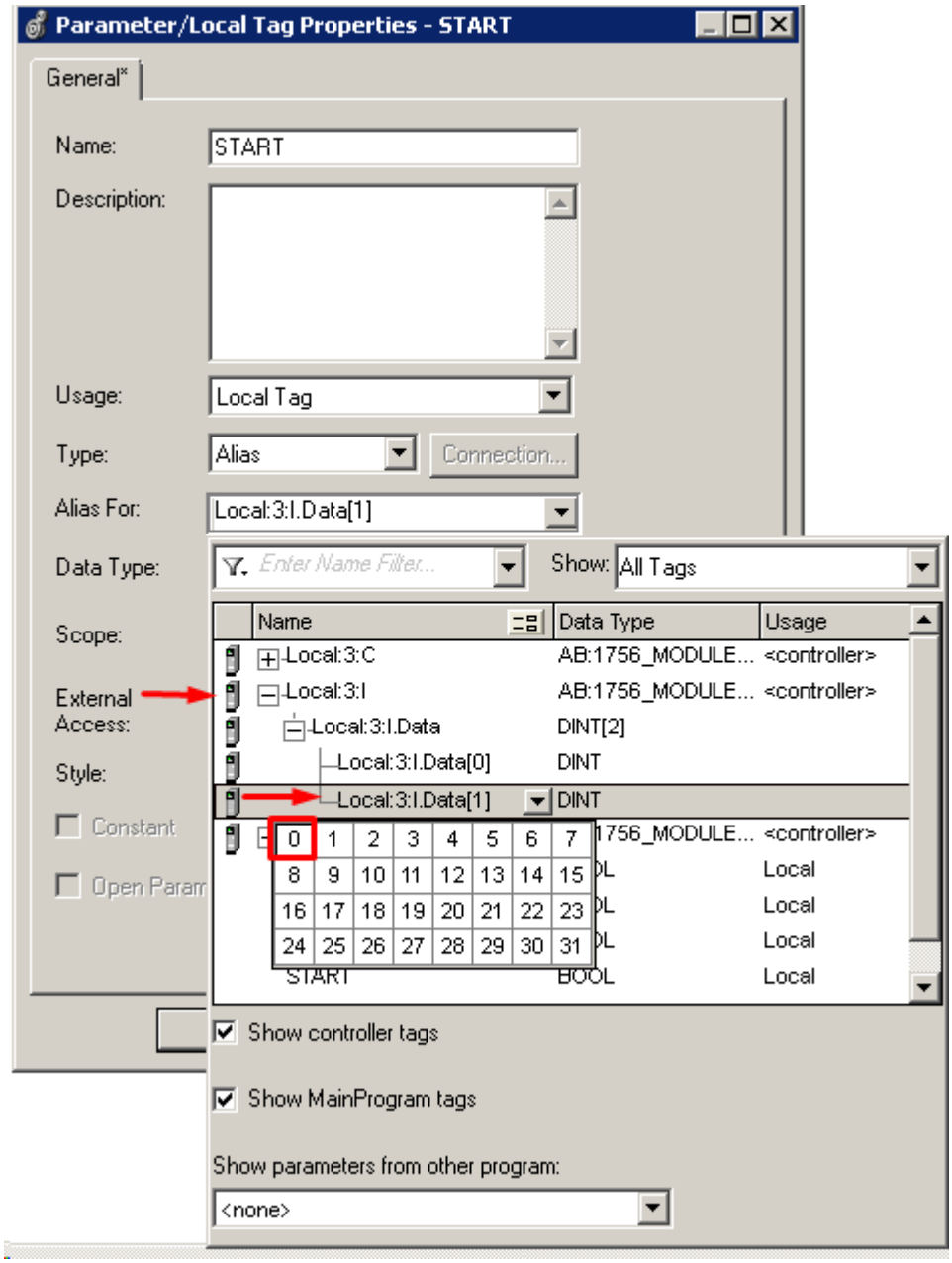


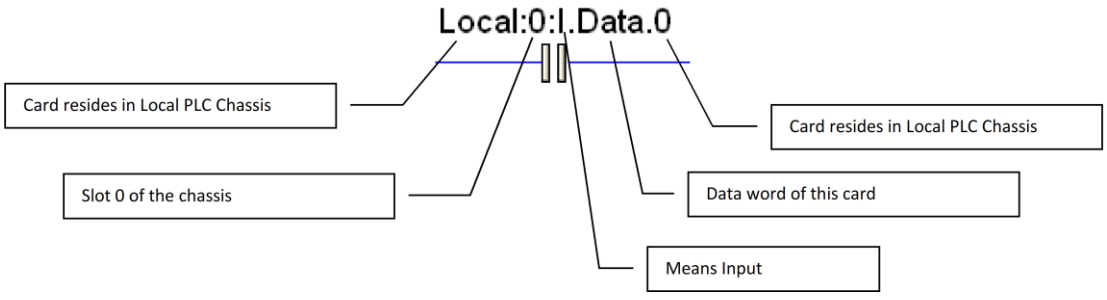
Change Type from Base to Alias.



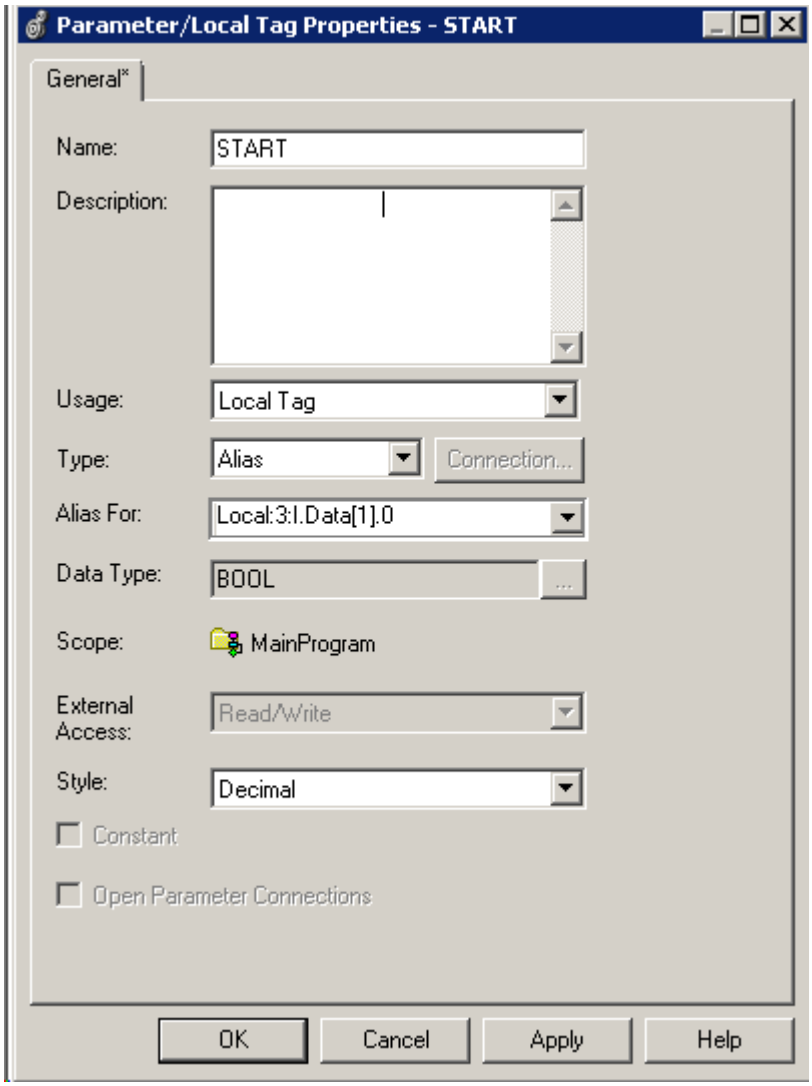
Change to Alias in the drop down box. Go to **Local3:I.[Data]** -> **Local3:I.Data[1]**. Select the first bit **0**.

The input data will appear in the second DINT tag, for example: **Local:1:I.Data[1].0** (module in slot 1, first bit of data).

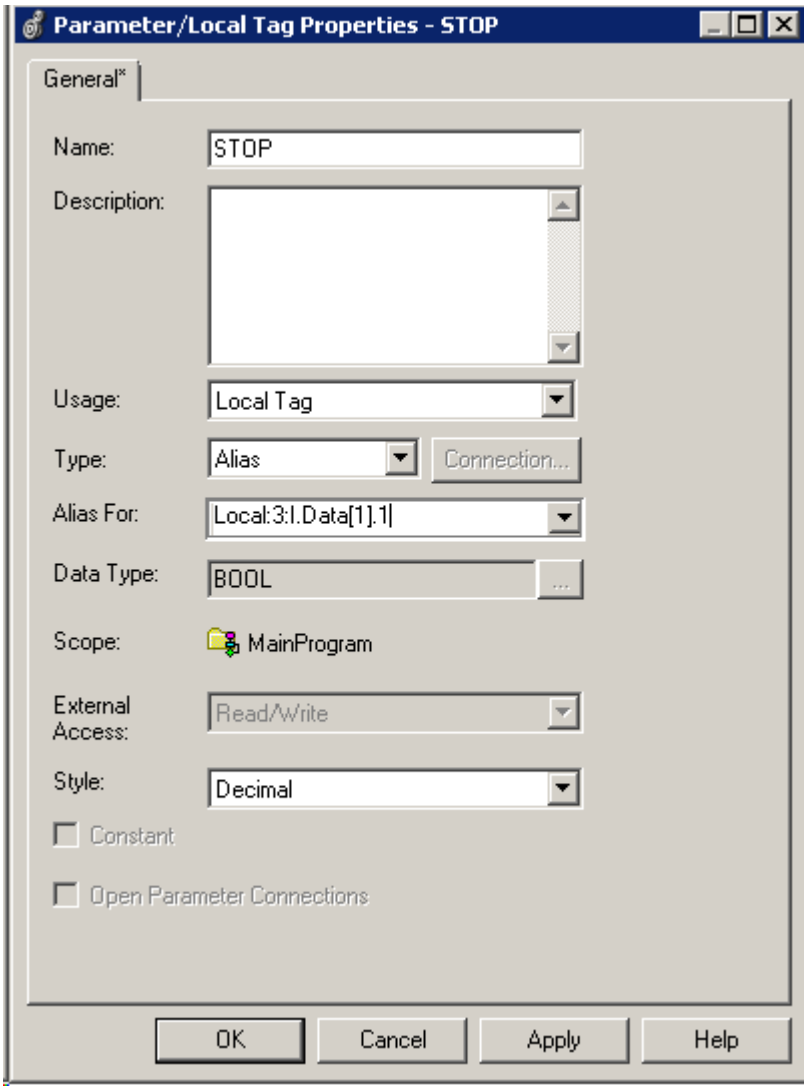




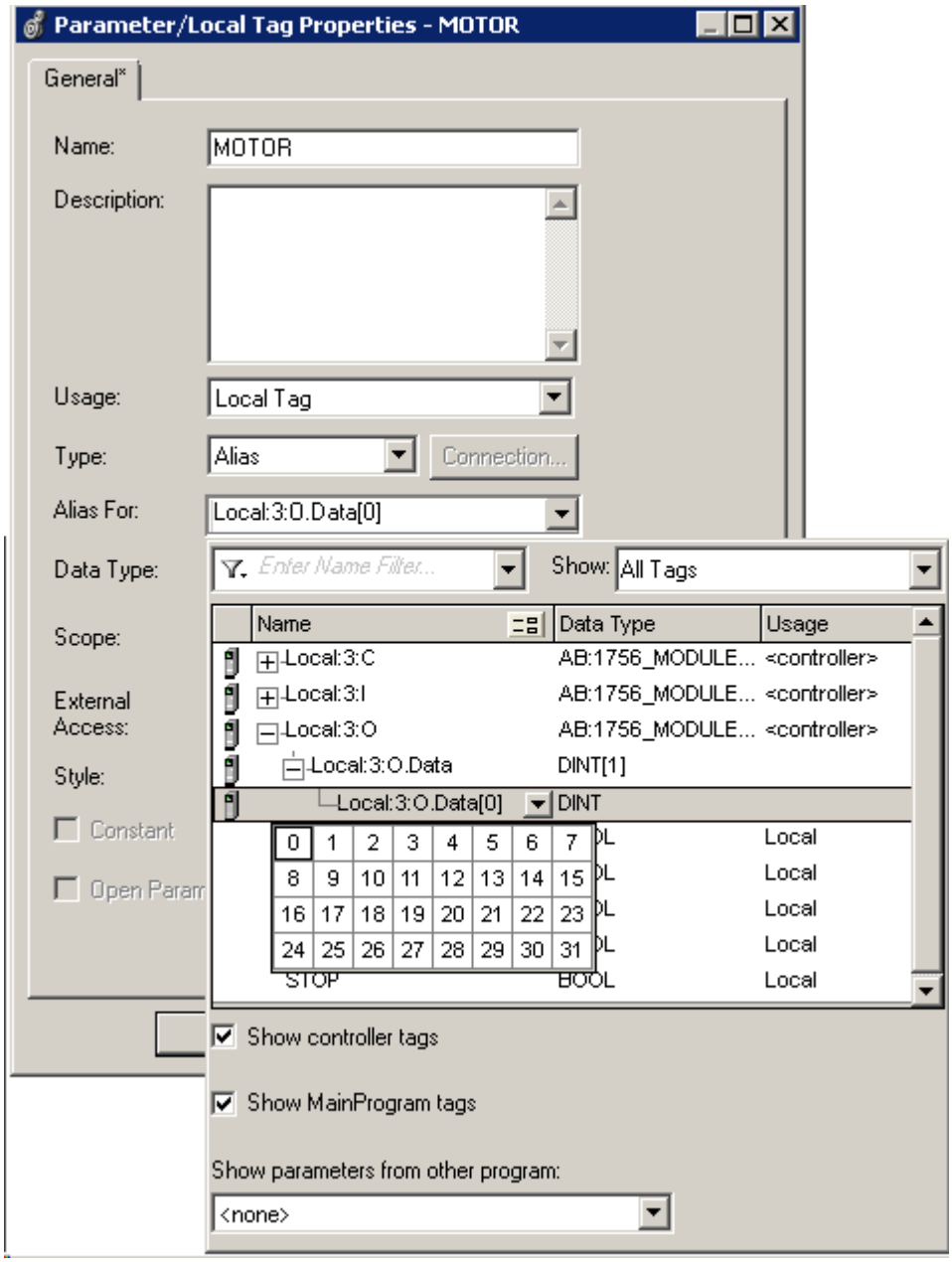
Click **OK** button to apply the changes and exit the parameter properties page.



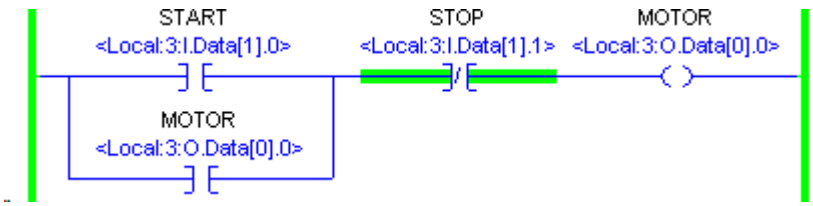
Repeat the same procedure for the "STOP" button.



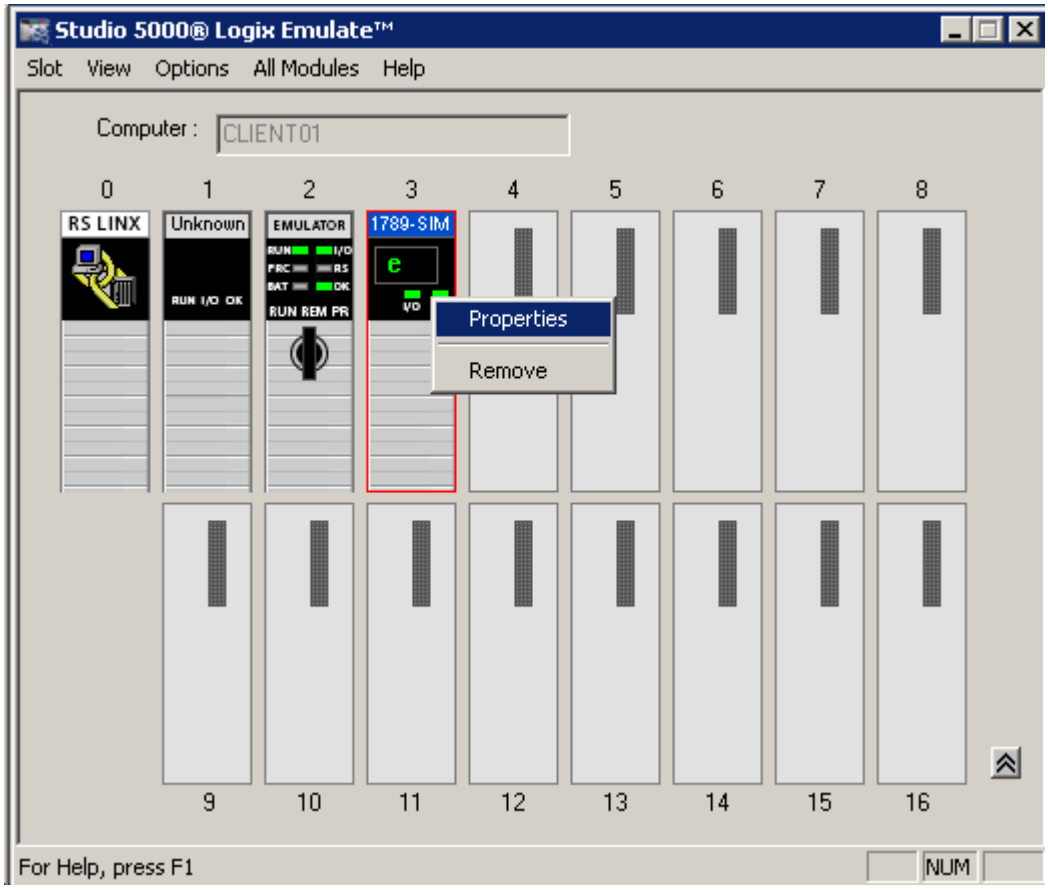
Assign the output address for the "MOTOR". Select **LOCAL3:O->Local3:O.[Data]** -> **Local3:O.Data[0]** and click on the bit **0**.



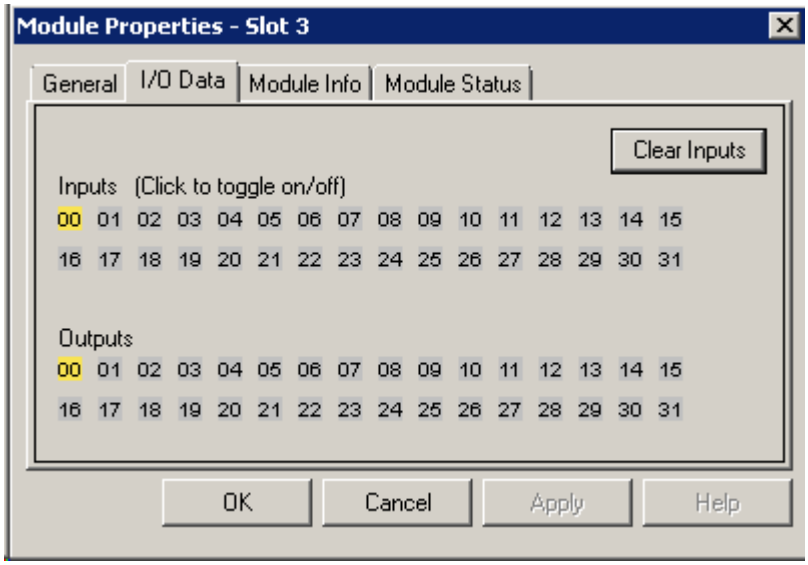
Download the project and switch to Run Mode.



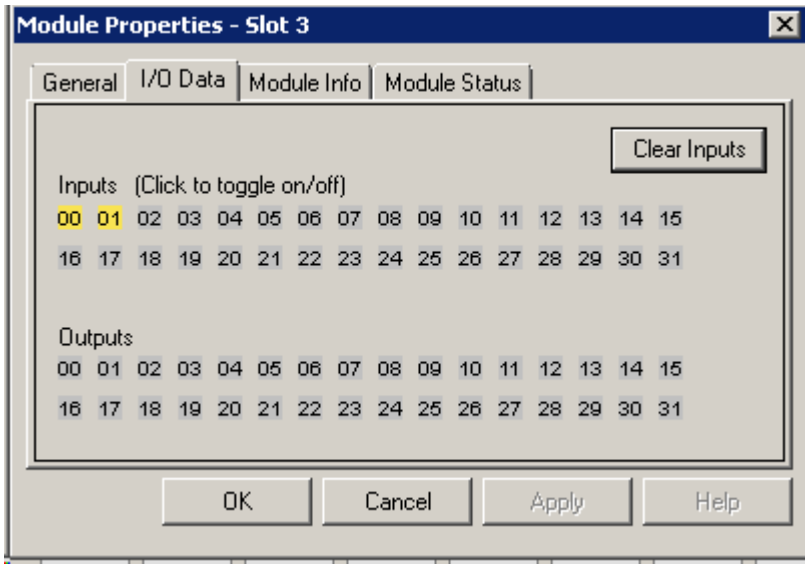
Open Studio 5000® Logix Emulate™ chassis. Right click on the 1789-SIM card and select Properties



Go to the I/O Data tab. Select bit **00** ("START") for the input. You should see input bit **00** and output **00** both turn ON in the Studio 5000. Yellow color indicate the bit is set.



Click on the input **01** ("STOP") will turn OFF the "MOTOR"



Click on the **Clear Inputs** button to clear the inputs.

** If the 1789-SIM 32 Point Input/Output Simulator is NOT responding. Remove the module and reinstall it again.*

Part IV Timers

Timer Instructions

The TON instruction is a non-retentive timer that accumulates time when the instruction is enabled (rung-condition-in is true).

Ladder Diagram

TON - time how long a timer is enabled

TOF - time how long a timer is disabled

RTO - accumulate time

RES - reset a timer or counter

Function Block

TONR - time how long a timer is enabled with built-in reset in function block

TOFR - time how long a timer is disabled with built-in reset in function block

RTOR - accumulate time with built-in reset in function block

1. TIMER ON DELAY (TON)

The TON instruction is a non-retentive timer that accumulates time when the instruction is enabled (rung-condition-in is true).

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

Ladder Diagram

Operand	Type	Format	Description
Timer	TIMER	tag	Timer structure
Preset	DINT	immediate	How long to delay (accumulate time)
Accum	DINT	immediate	Total msec the timer has counted

TIMER Structure

Mnemonic	Data Type	Description
.EN	BOOL	The .EN bit stores rung-condition-in.
.TT	BOOL	The timing bit indicates that a timing operation is in process
.DN	BOOL	The done bit indicates that .ACC >= .PRE.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of milliseconds that have elapsed since the TON instruction was enabled.

Description

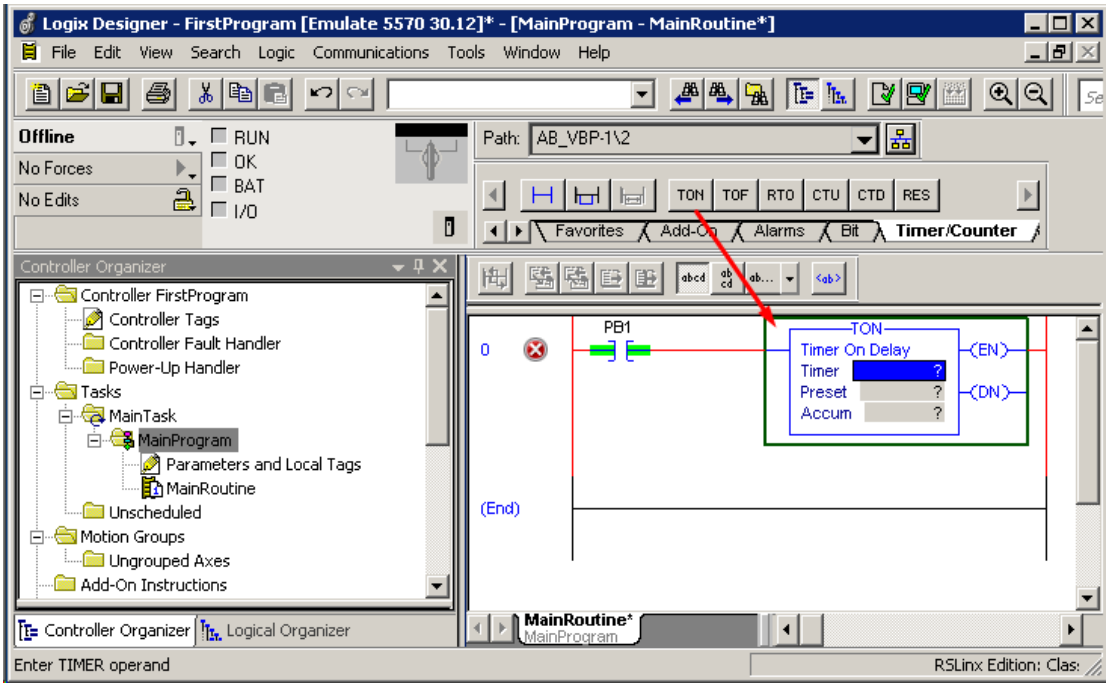
The TON instruction accumulates time from the time it is enabled until:

- The TON instruction is disabled
- The .DN bit is set to true

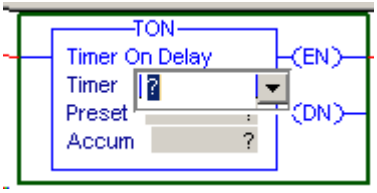
The time base is always 1 msec. For example, for a 2 second timer, enter 2000 for the .PRE value.

Example:

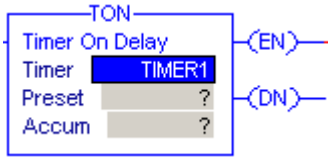
Go to the Timer/Counter tab. Drag the **TON** instruction and drop on rung 0.



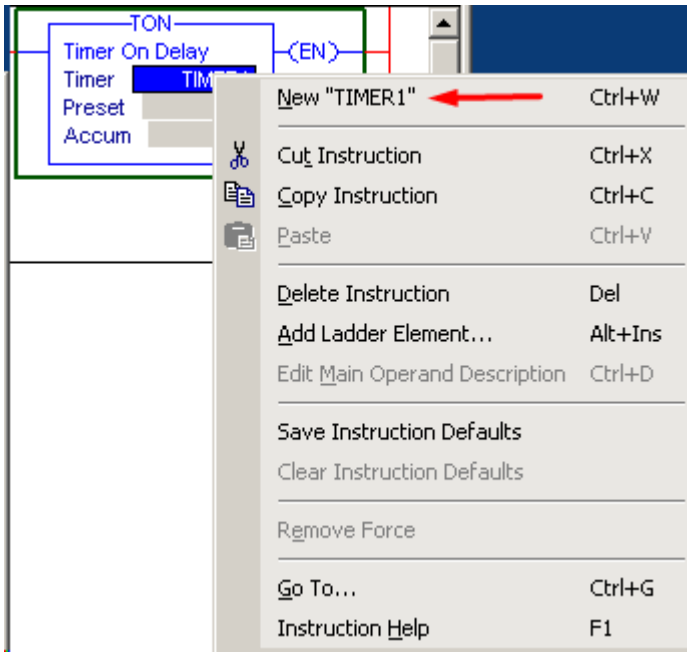
Double click on the **?** mark inside the TON instruction.



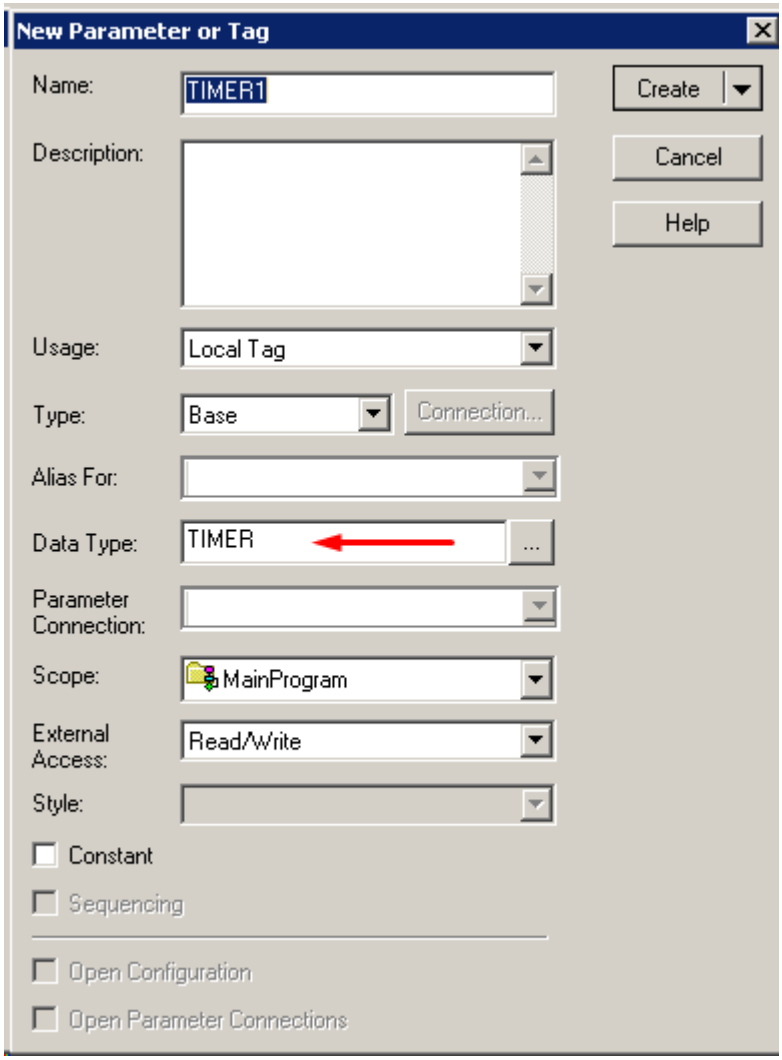
Enter "TIMER1" in the timer box. Not space allowed for the tag name.



Define the new tag name "TIMER1"

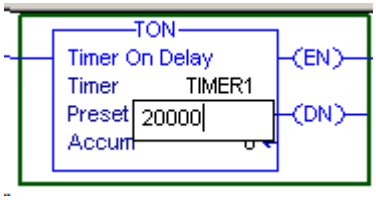


Make sure the data type is Timer. Click Create button to continue.

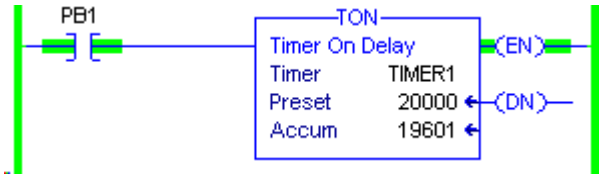


Next enter a preset value by double click on the Preset box and enter 20000 (time base is in millisecond). The 20000 millisecond will be equivalent to 20 seconds.

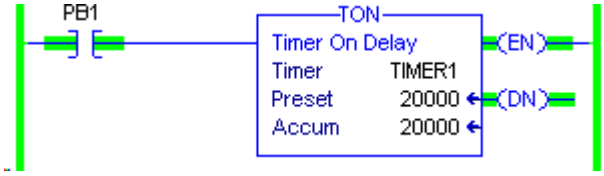
Keep the Accum with default value 0. Consider the Accum is the starting value, and Preset is the ending value.



Download and test the timer. Press the "PB1" to enable the timer. Notice the (DN) bit is OFF while the timer is timing.

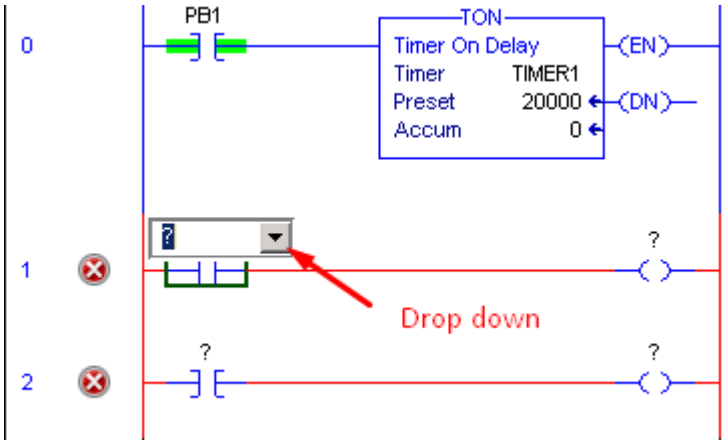


When the Preset value equals to the Accum value, (DN) is set to 1. Even though the timer still enabled, but the timer stops timing unless is it reset.

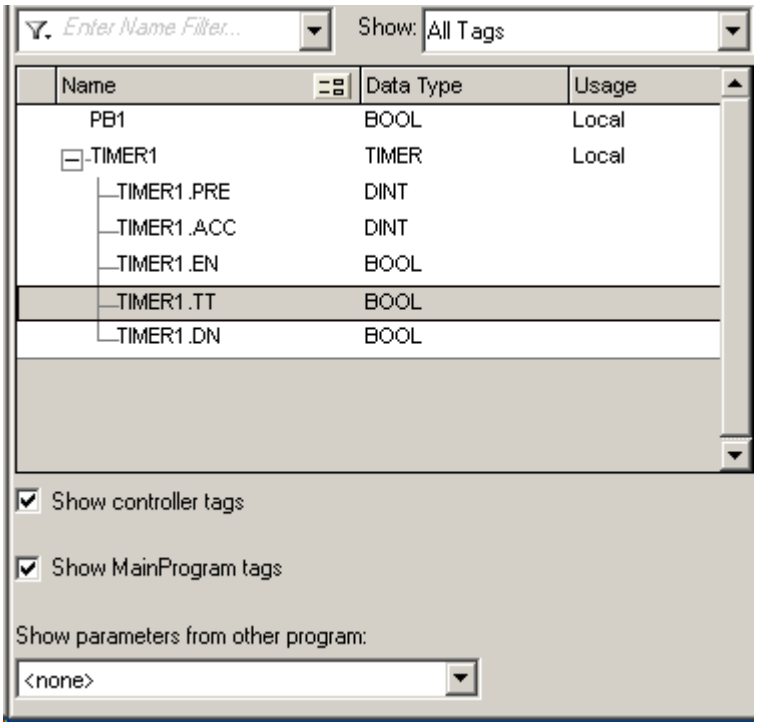


Let's learn how to use Done Bit (DN), and Timer Timing Bit (TT) in an application.

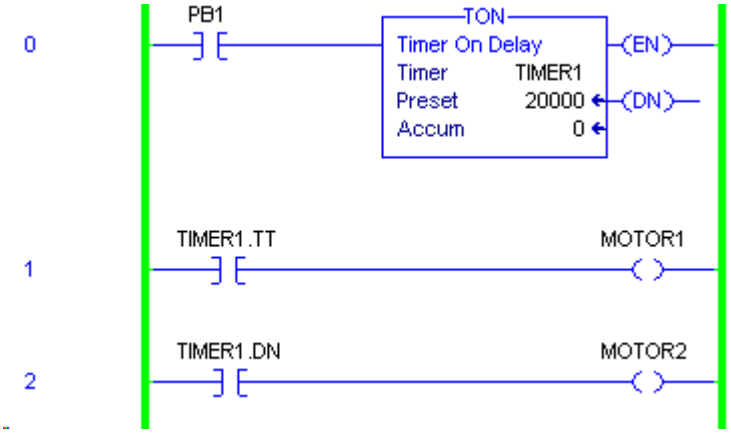
Create the logic below in MainRoutine. Double click on the **?** mark, and click on the drop down menu.



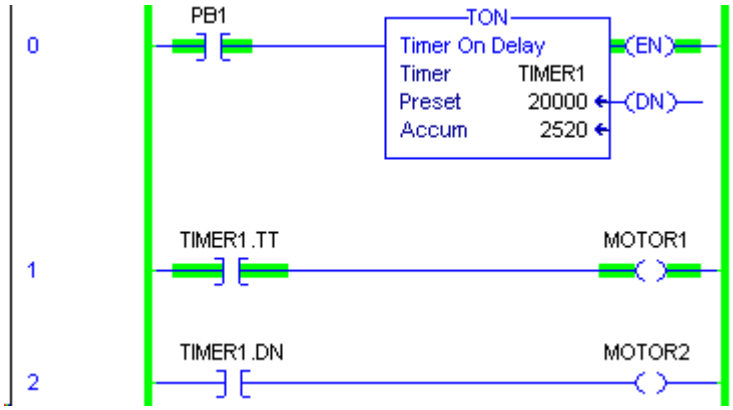
Click on the + sign next to the "TIMER1" to expand the folder. Highlight TIMER1.TT and press Enter key to select.



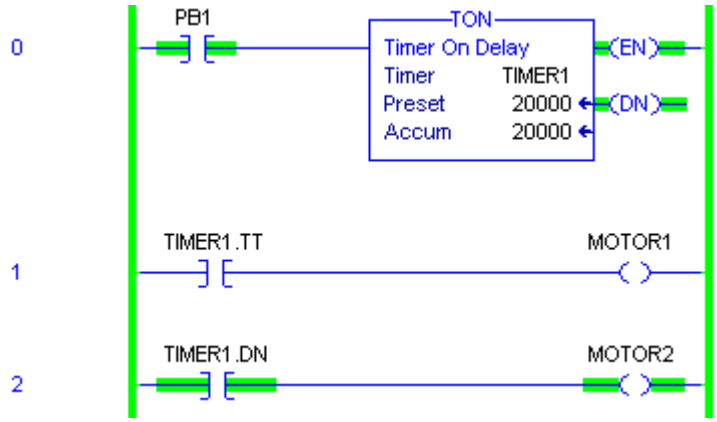
Create two new tags "MOTOR1" and "MOTOR2". Download the program and switch to Run Mode.



Observe the outputs. TIMER1.TT is TRUE while the timer is timing. Therefore "MOTOR1" turns ON, and TIMER1.DN remains FALSE.



When TIMER1.DN is TRUE, "MOTOR2" turns ON.



Note: timer timing (TT) and timer done (DN) bit is opposite to each other.

2. TIMER OFF DELAY (TOF)

The TOF instruction is a non-retentive timer that accumulates time when the instruction is enabled (rung-condition-in is false).

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

Ladder Diagram

Operand	Type	Format	Description
Timer	TIMER	tag	Timer structure
Preset	DINT	immediate	How long to delay (accumulate time)
Accum	DINT	immediate	Total msec the timer has counted

TIMER Structure

Mnemonic	Data Type	Description
.EN	BOOL	The .EN bit stores rung-condition-in.
.TT	BOOL	The timing bit indicates that a timing operation is in process.
.DN	BOOL	The done bit indicates that .ACC > or = .PRE.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated value must reach before the instruction clears the .DN bit.
.ACC	DINT	The accumulated value specifies the number of milliseconds that have elapsed since the TOF instruction was enabled.

Description

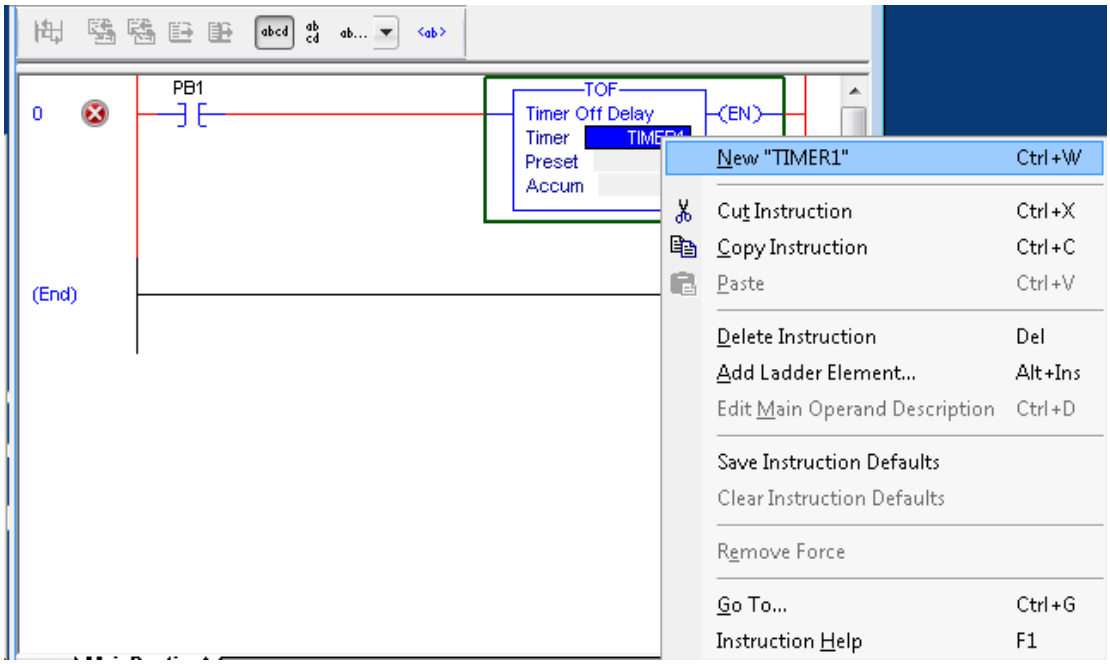
The TOF instruction accumulates time until:

- The TOF instruction is disabled
- The .DN bit is cleared to false

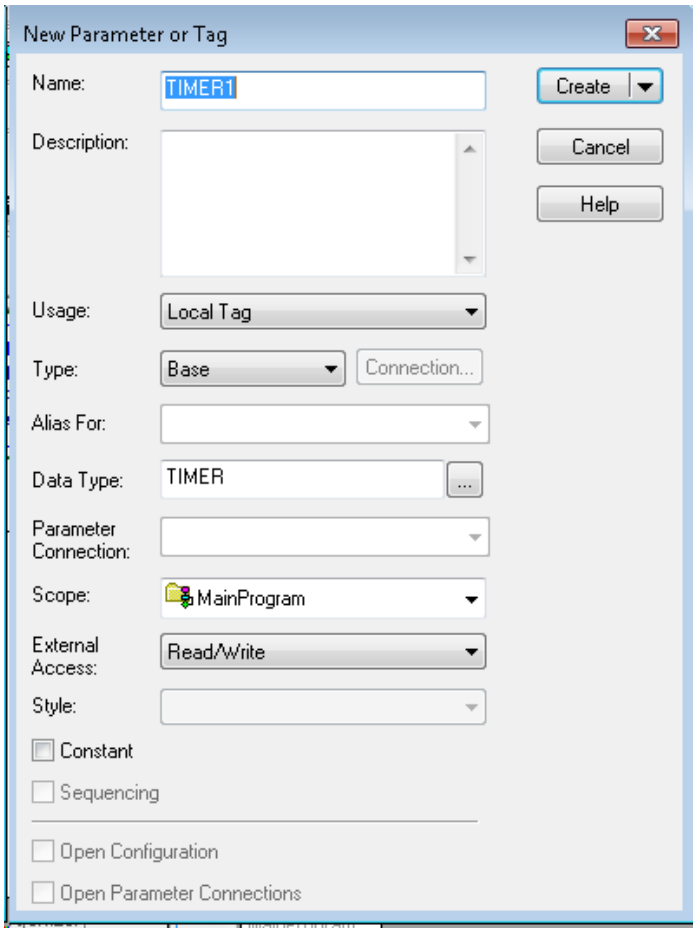
The time base is always 1 msec. For example, for a 2 second timer, enter 2000 for the .PRE value.

Example:

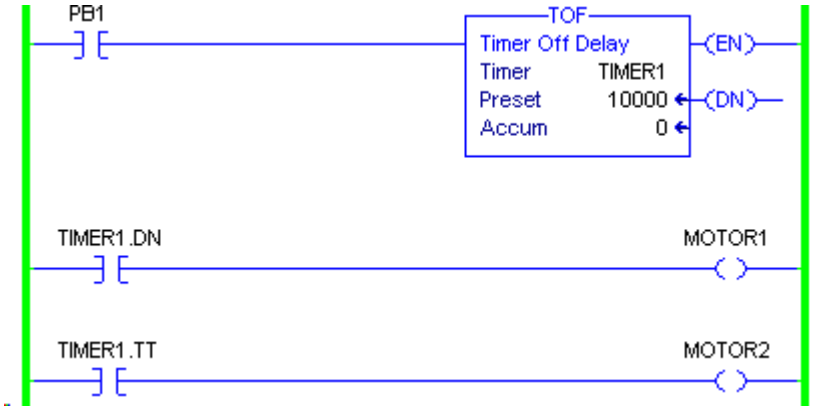
Create a TOF instruction and a input push button. Enter "TIMER1" in the Timer box, right click and select New "TIMER1".



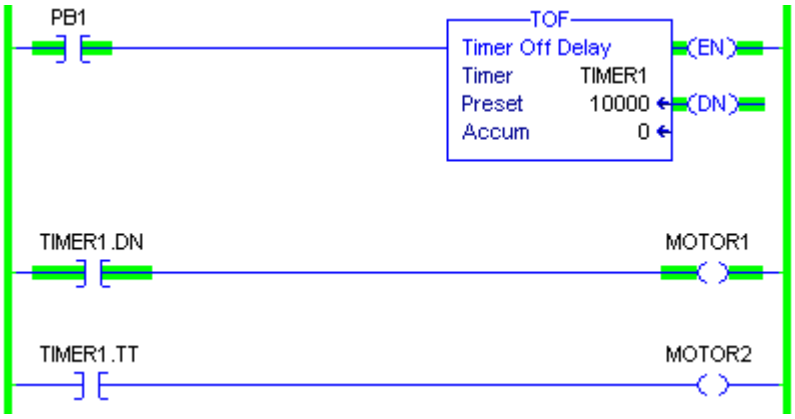
click **Create** button to continue.



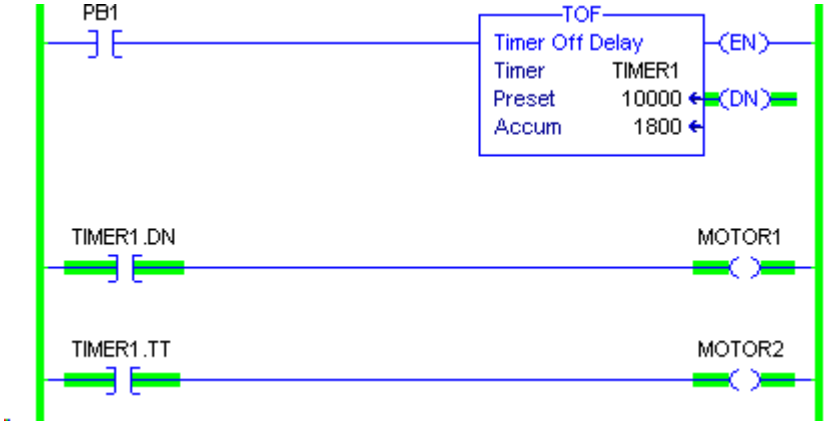
Create the logic below and download. Toggle "PB1" and observe the TIMER1.DN and TIMER1.TT bit.



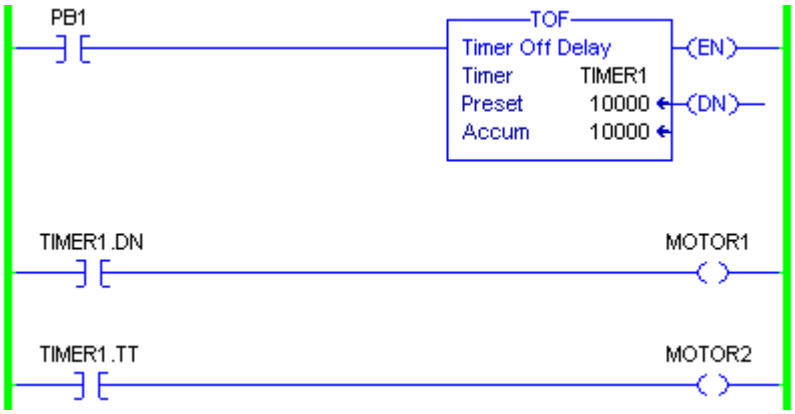
When "PB1" is ON, DN bit is set to 1 and Accum does not increment.



When "PB1" is OFF, DN bit remains on and Accum starts to increment.

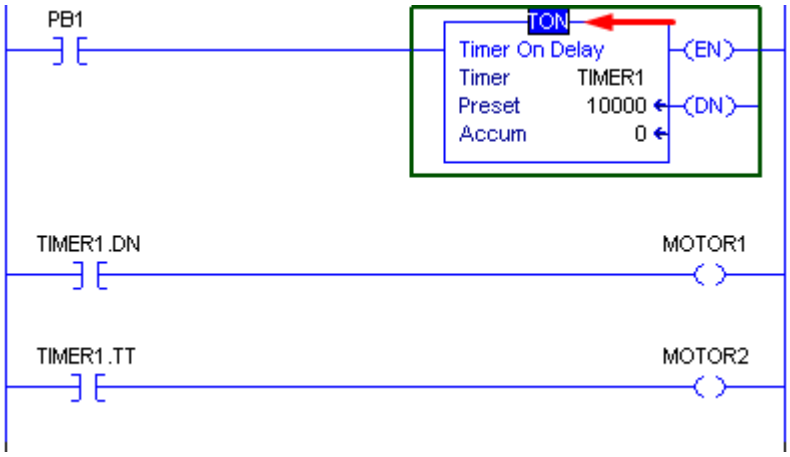


Timer stops when Accum reaches the Preset value, and both DN and TT bit turns OFF.

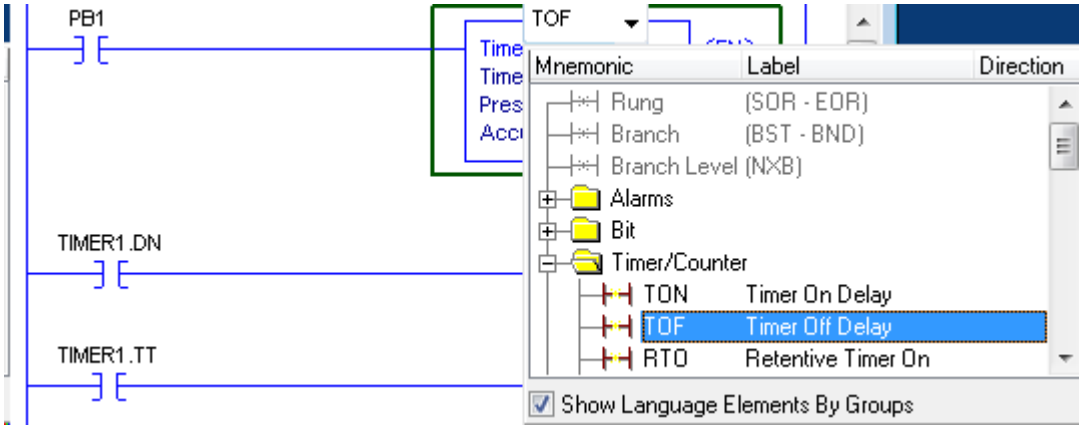


Tips:

An easy way to change the instruction type is by click on the Instruction text.



Click on the drop down menu then double click on a different timer from the list.



3. RETENTIVE TIMER ON(RTO)

The RTO instruction is a retentive timer that accumulates time when the instruction is enabled.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

Ladder Diagram

Operand	Type	Format	Description
Timer	TIMER	tag	Timer structure
Preset	DINT	immediate	How long to delay (accumulate time)
Accum	DINT	immediate	Number of msec the timer has counted

TIMER Structure

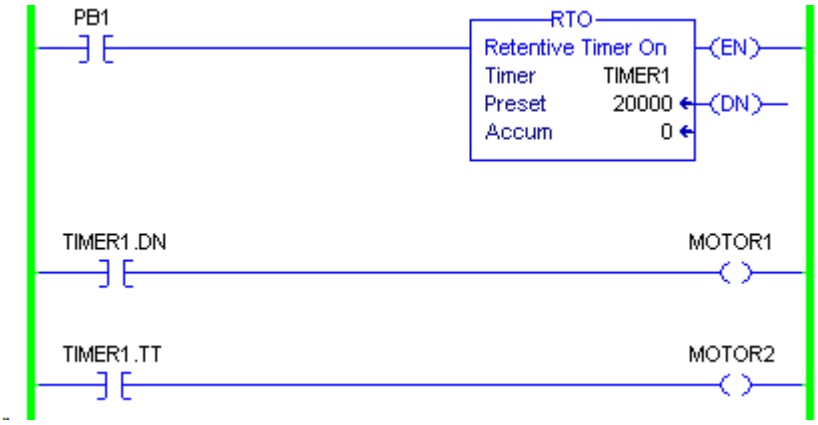
Mnemonic	Data Type	Description
.EN	BOOL	The .EN bit stores rung-condition-in.
.TT	BOOL	The timing bit indicates that a timing operation is in process
.DN	BOOL	The done bit indicates that .ACC > or = to .PRE.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of milliseconds that have elapsed since the RTO instruction was enabled.

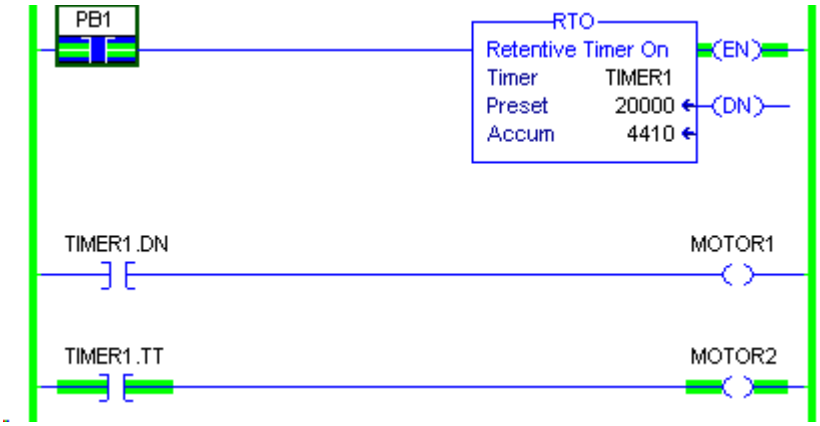
Description

The RTO instruction accumulates time until RTO instruction is disabled.

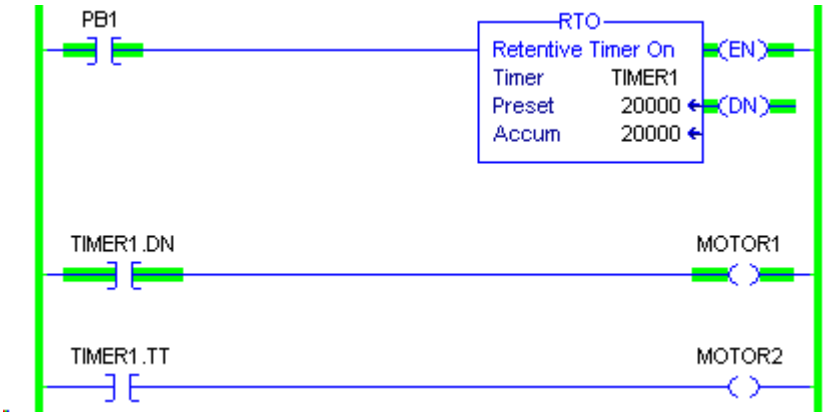
Example:

Create the logic below with RTO instruction. Download the program, toggle the "PB1", wait for 5 seconds then toggle again to turn OFF "PB1".

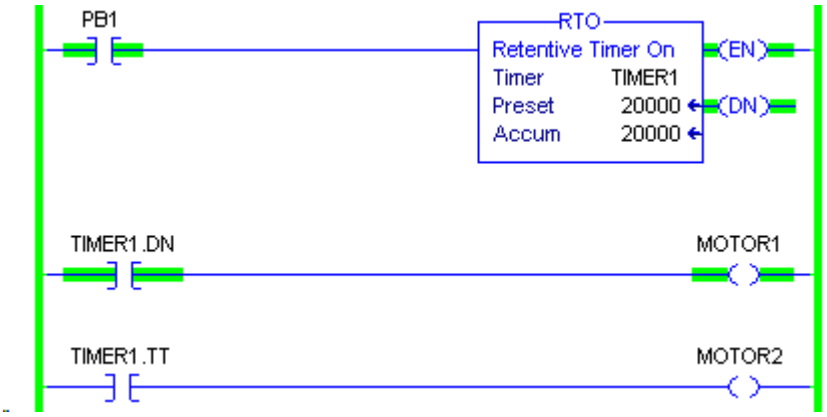




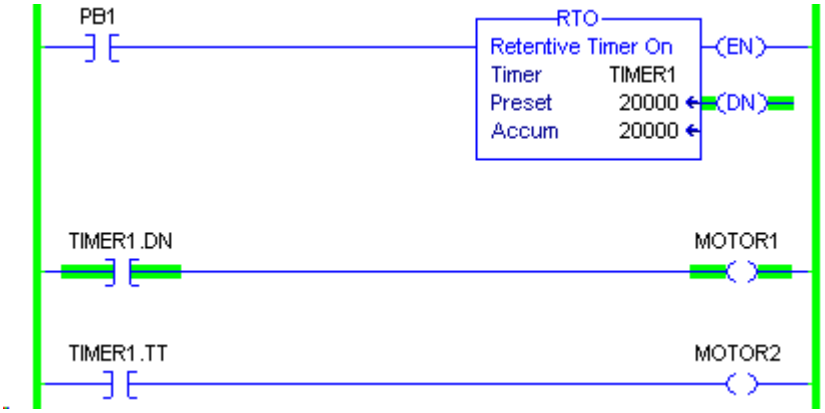
RTO is similar to TON timer, but Accum does not reset when "PB1" is OFF.



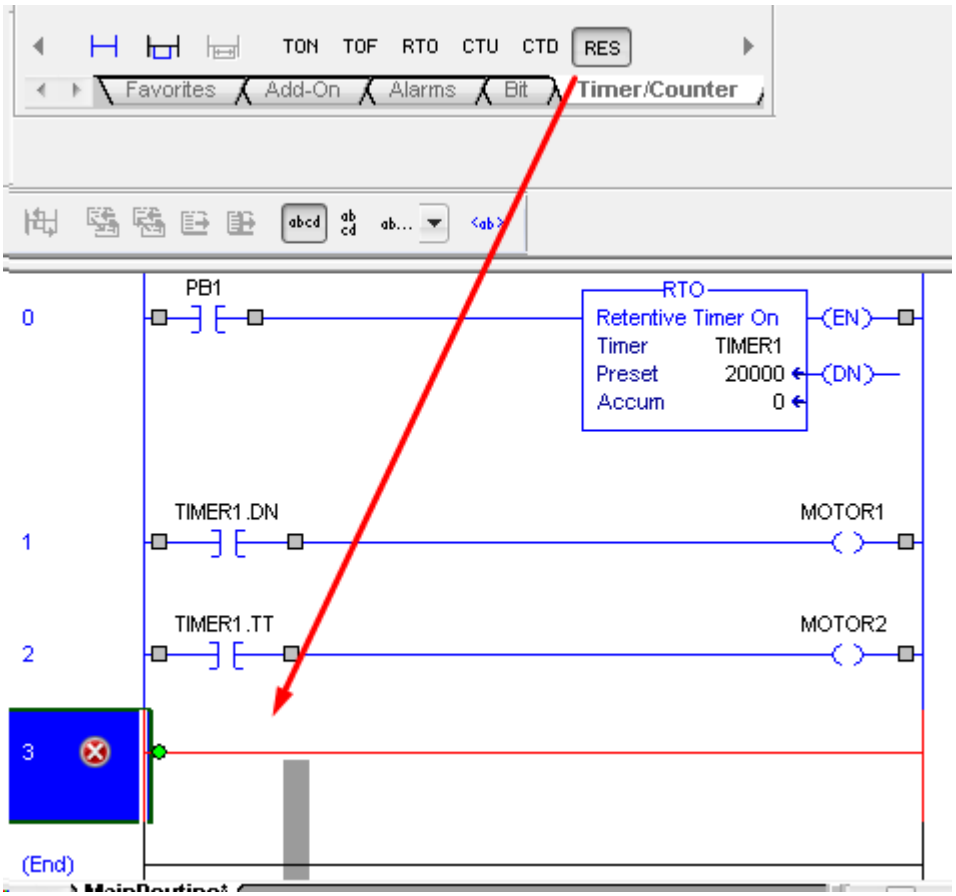
Toggle the "PB1" to ON again and wait until the Accum reaches Preset value.



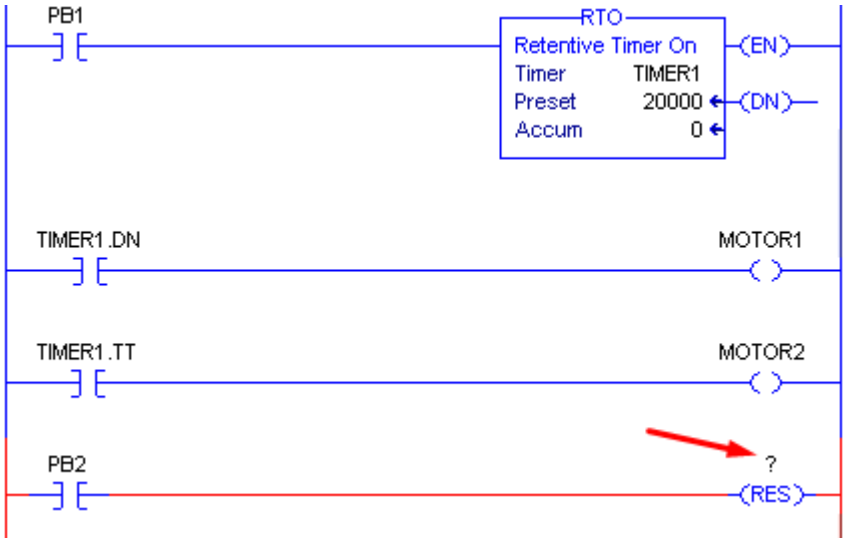
Toggle the "PB1" to OFF does not reset the Accum even though Accum reaches the Preset value.



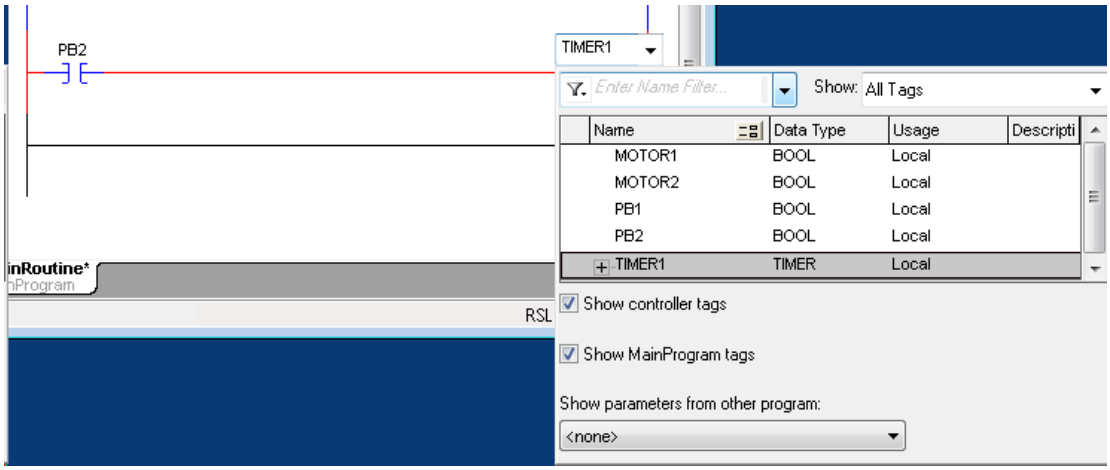
In order to reset the Accum, a **RES** instruction is required. Insert a new Rung then go the Timer/Counter tab and select the **RES** instruction.



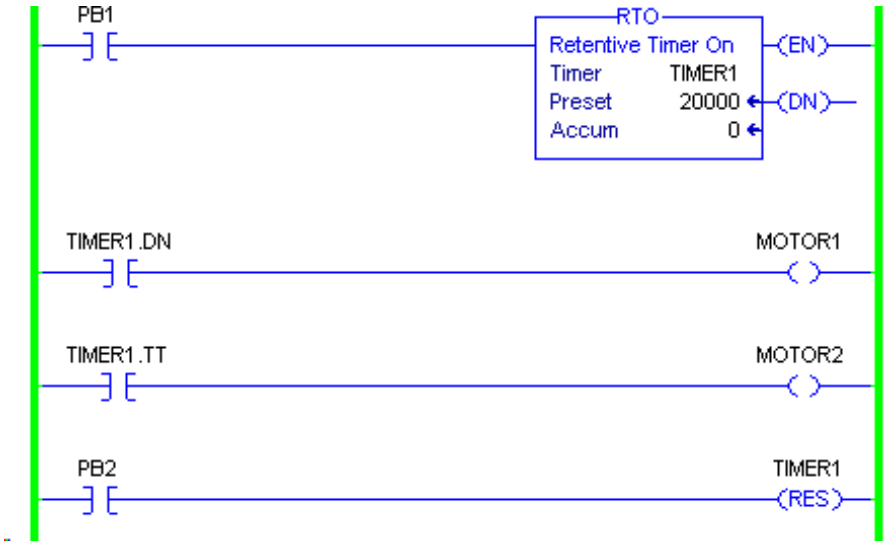
Create another input call "PB2", Double click on the **?** mark to display the drop down menu.



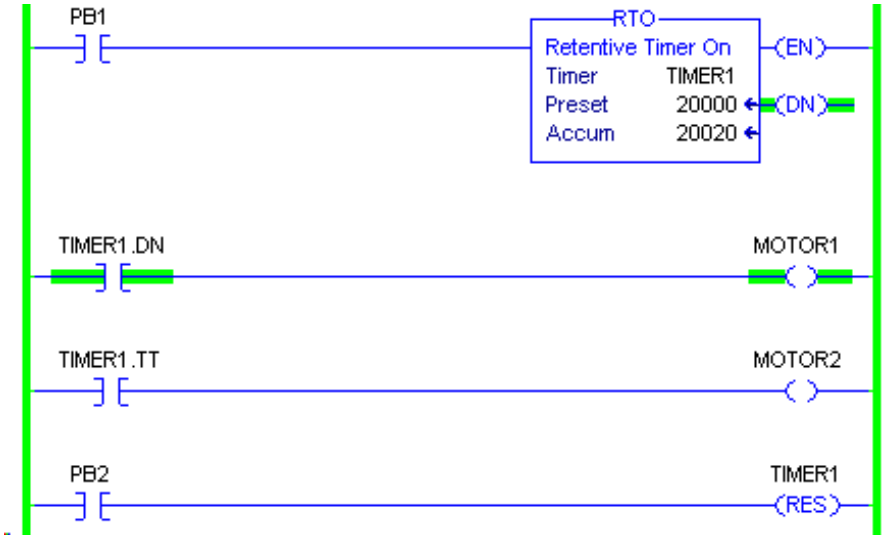
Select "TIMER1" from the list.



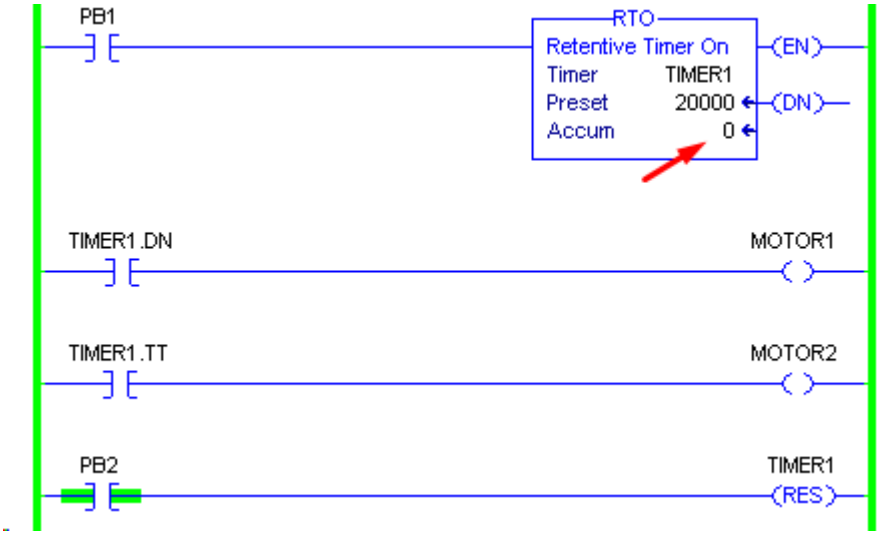
Download the program and toggle "PB1". Wait until Accum reaches the Preset value.



Toggle "PB1" again to turn OFF.



Now toggle "PB2" to ON will reset the Accum. Remember to toggle "PB2" back to OFF, otherwise the timer will be reset constantly.



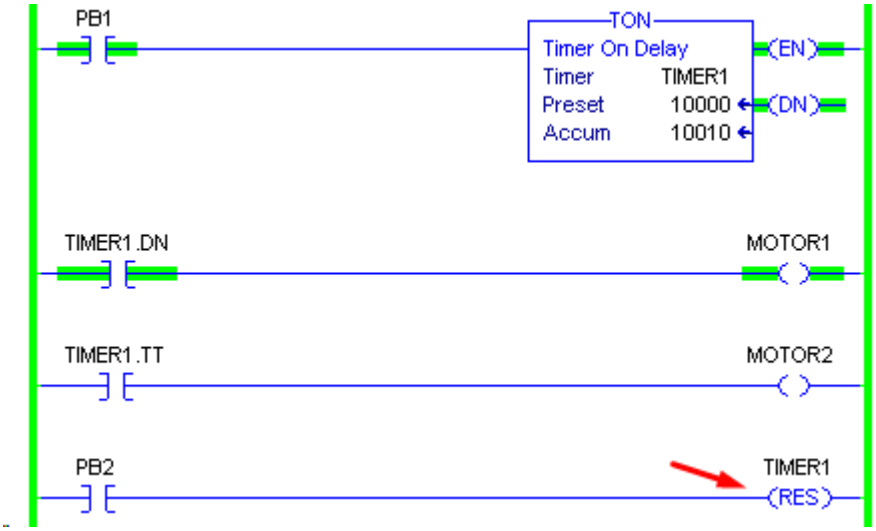
4. EXERCISE 1 - RESET TIMER

Reset Timer Methods

There are multiple ways of resetting a timer. **RES** instruction can be use to reset both timers and counters, but use done bit method only apply to TON instruction.

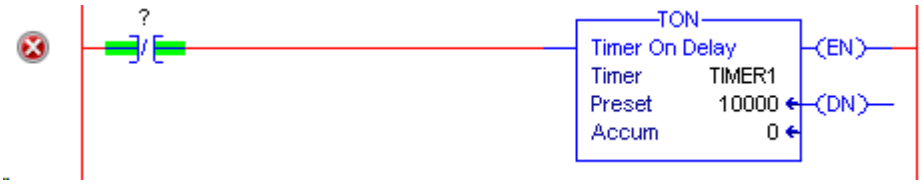
1. Use the **(RES)** instruction, toggle "PB2" ON will reset the Accum. Remember to turn "PB2" OFF after reset, otherwise the timer will reset on every scan.

(RES) instruction can be used for TON, TOF, RTO, also for counters as well.

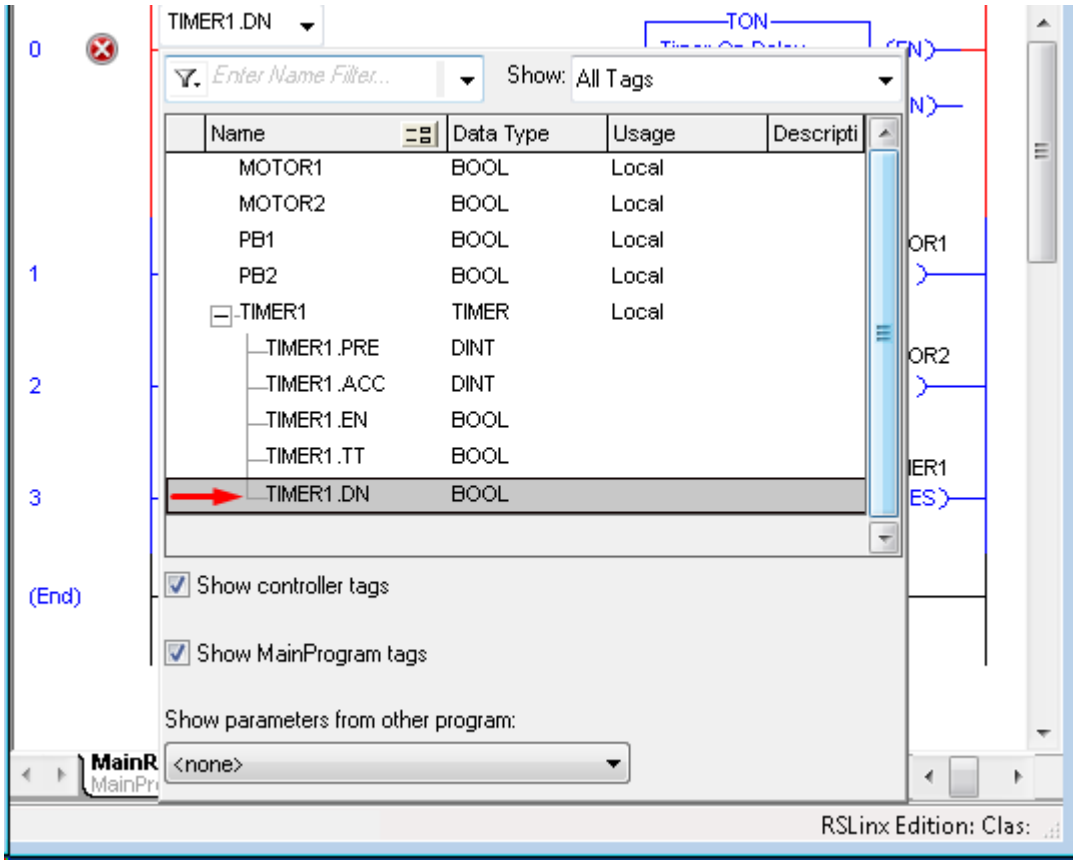


2. Use the Done bit, but this method only works for the **TON** instruction.

Insert an XIO instruction, and double click on the question mark to bring up drop down menu.

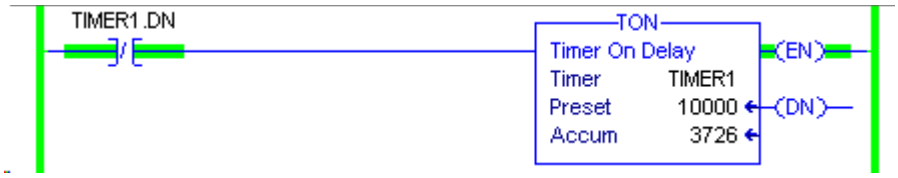


Navigate to the TIMER1 tag, and click on the + sign to expand the folder. Select the TIMER1.DN bit by double click on it.



The logic behind this method is use the XIO instruction to change TIMER1.DN from FALSE to TRUE. When the Accum reaches the Preset value then Done bit will becomes TRUE.

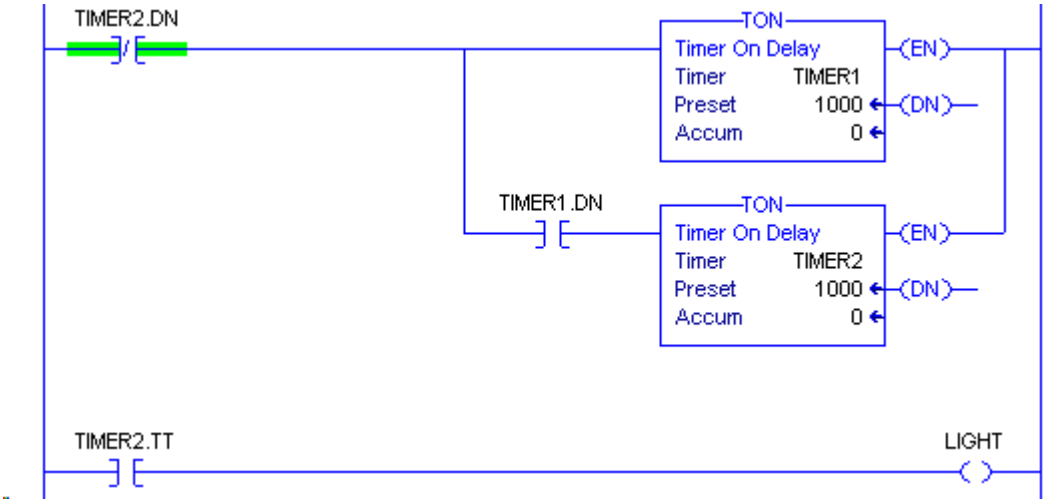
When Done bit becomes TRUE, XIO TIMER1.DN will be OFF momentary which indirectly resets the timer. After timer is reset, XIO TIMER1.DN becomes TRUE again.



5. EXERCISE 2 - FLASHING LIGHT

There are many methods of create a flashing light. Here is one method using two timers.

The two timers interlocks each other with the done bit. TIMER2.DN is used as an input to energize TIMER1, but also used to reset TIMER1 when the done bit is TRUE. When TIMER1 is done then TIMER2 starts timing. TIMER2.TT is being used to control the light ON/OFF at 1 second time interval



Part V Counters

Counter up/down Instructions

1. COUNT UP (CTU)

The CTU instruction counts upward

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

Ladder Diagram

Operand	Type	Format	Description
Counter	COUNTER	tag	Counter structure
Preset	DINT	immediate	How high to count
Accum	DINT	immediate	Accumulated value of the counter

COUNTER Structure

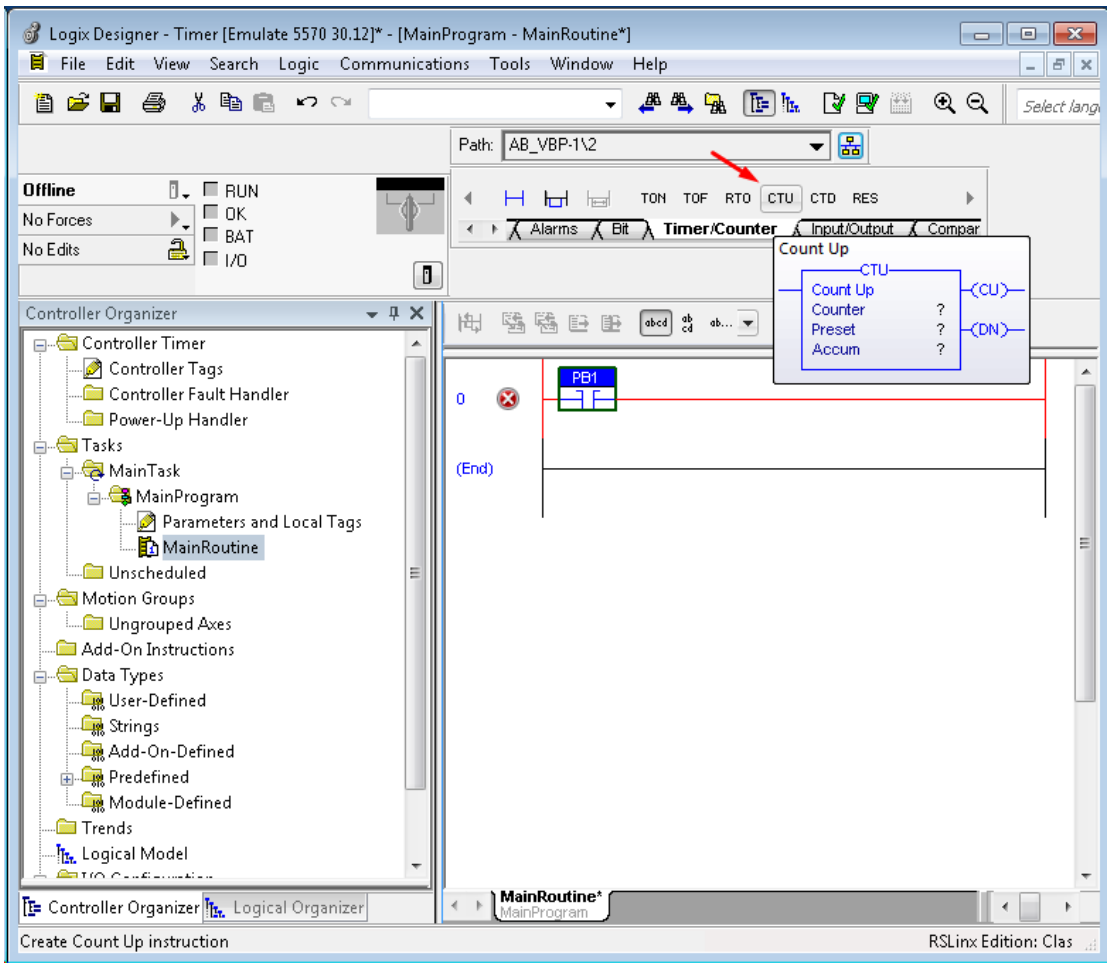
Mnemonic	Data Type	Description
.CU	BOOL	The count up enable bit indicates the CTU instruction was executed when Rung-condition-in was true.
.DN	BOOL	The done bit indicates that .ACC > or = to .PRE.
.OV	BOOL	The overflow bit is set by the CTU instruction when the counter increments past the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and continues counting up again.
.UN	BOOL	The underflow bit is set by the CTD when the counter decrements past the lower limit of -2,147,483,647. The counter then rolls over to 2,147,483,647 and begins counting down again.
.PRE	DINT	The preset value specifies the value which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of transitions the instruction has counted.

Description

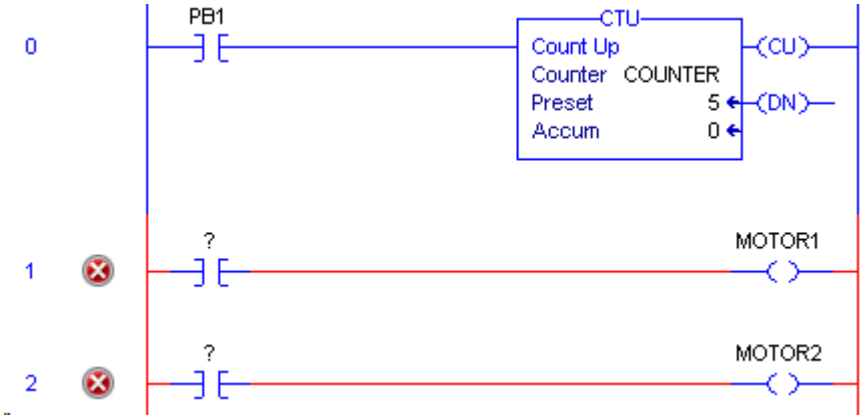
When rung-condition-in is true and .CU is false, ACC will be incremented by one. When rung-condition-in is false, .CU will be cleared to false.

Example:

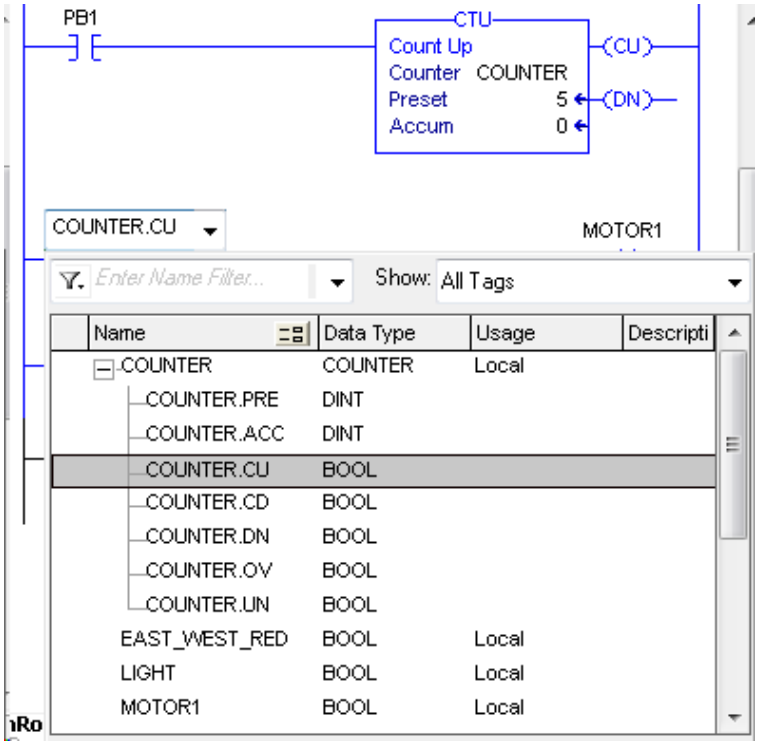
Go to Timer/Counter tab and drag the **CTU** instruction to rung 0.

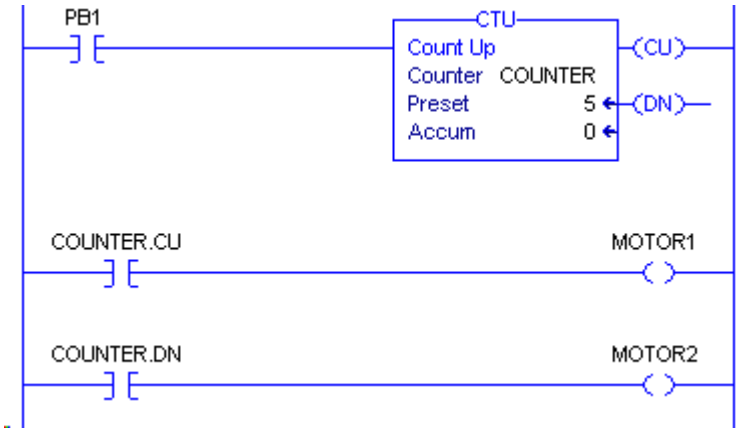


Create an input "PB1" tag name "COUNTER" for the **CTU** instruction. Define the new tag "COUNTER" and enter 5 for the preset value.

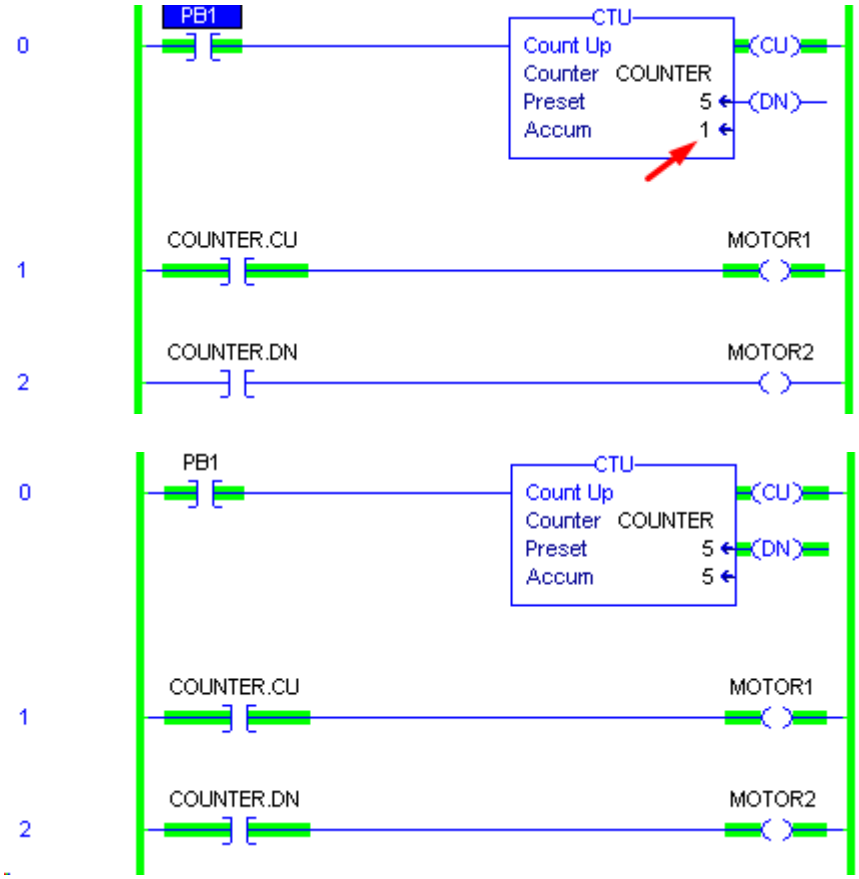


Double click on the question mark (**?**). Go to tag "COUNTER", expand the folder then select the COUNTER.CU. repeat the same steps for COUNTER.DN.

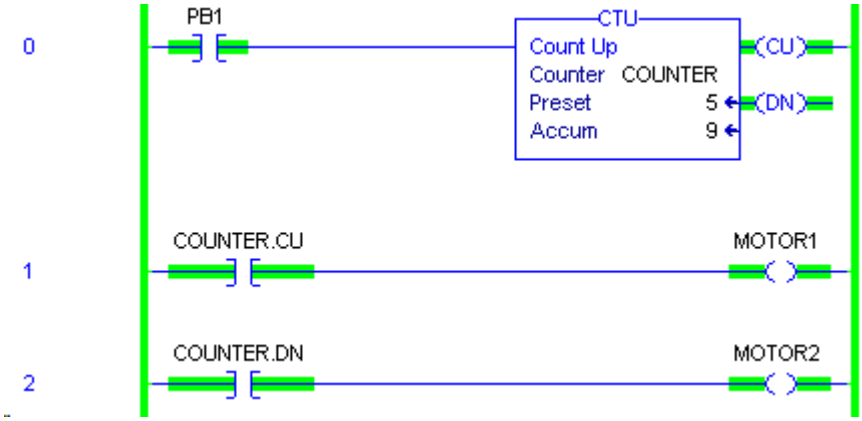




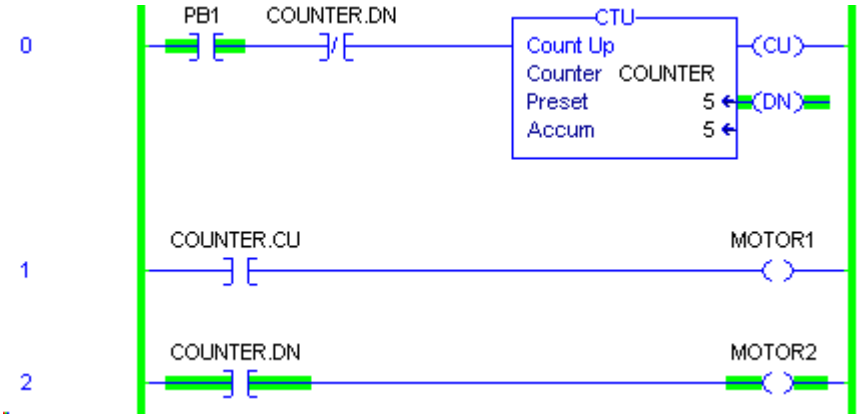
Download the program and toggle the "PB1". When "PB1" is TRUE, counter will increment by 1. Output "MOTOR1" is energized because the COUNTER.CU is TRUE. Continue to toggle the "PB1" until counter's Accum equals the Preset.



Even though the done bit is ON, but accum value will continue to increment belong the preset.



To prevent the accum exceed the preset value. Place the "PB1" in series with the done bit **XIO** instruction.


0
PB1
COUNTER.DN
CTU
Count Up
Counter COUNTER
Preset 5
Accum 5
(CU)
(DN)
1
COUNTER.CU
MOTOR1
2
COUNTER.DN
MOTOR2

2. COUNT DOWN (CTD)

The CTD instruction counts downward

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

Ladder Diagram

Operand	Type	Format	Description
Counter	COUNTER	tag	Counter structure
Preset	DINT	immediate	How high to count
Accum	DINT	immediate	Accumulated value of the counter

COUNTER Structure

Mnemonic	Data Type	Description
.CD	BOOL	The countdown enable bit indicates the CTD instruction is executed when rung-condition-in was true.
.DN	BOOL	The done bit indicates that .ACC > or = to .PRE.
.OV	BOOL	The overflow bit is set by the CTU instruction when the counter increments past the upper limit of 2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting up again.
.UN	BOOL	The underflow bit is set by the CTD when the counter decrements past the lower limit of -2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting down again.
.PRE	DINT	The preset value specifies the value which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of transitions the instruction has counted.

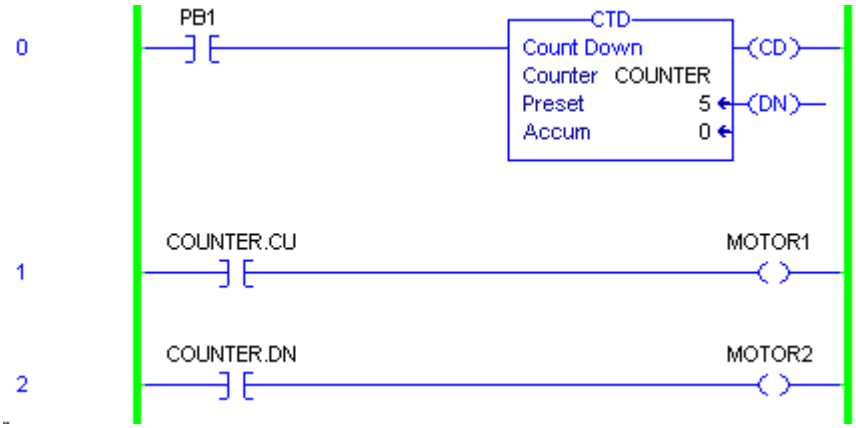
Description

The **CTD** instruction is typically used with a **CTU** instruction that references the same counter structure.

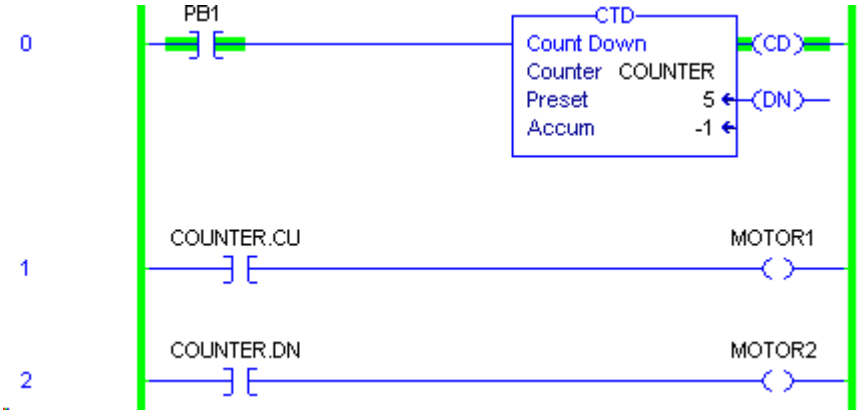
When rung-condition-in is true and .CD is false, .ACC will be decremented by one. When rung-condition-in is false, .CD will be cleared to false.

Example:

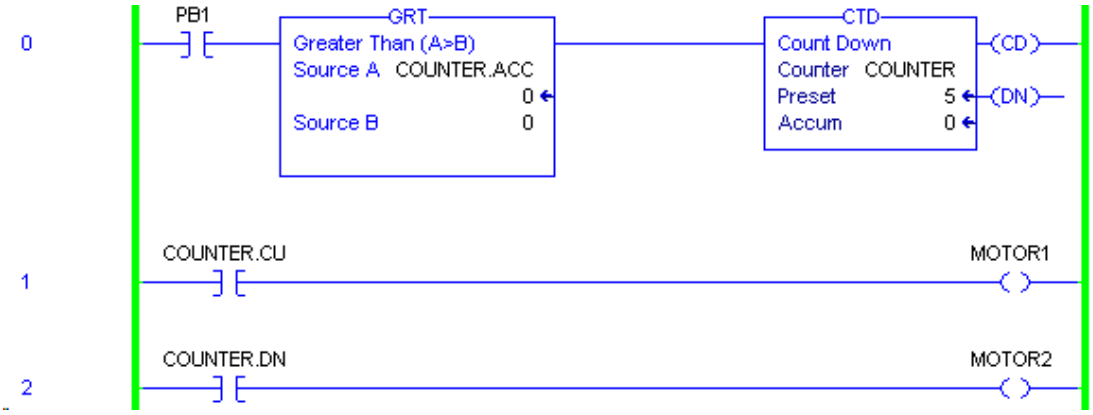
Create the logic below with counter down instruction **CTD**. Toggle the "PB1" to enable the counter down.



Notice the Accum is set to zero. When toggle the input "PB1", the Accum will become -1 from the current value 0. In this case, the done bit can not be use to prevent the accum moving toward a negative value.

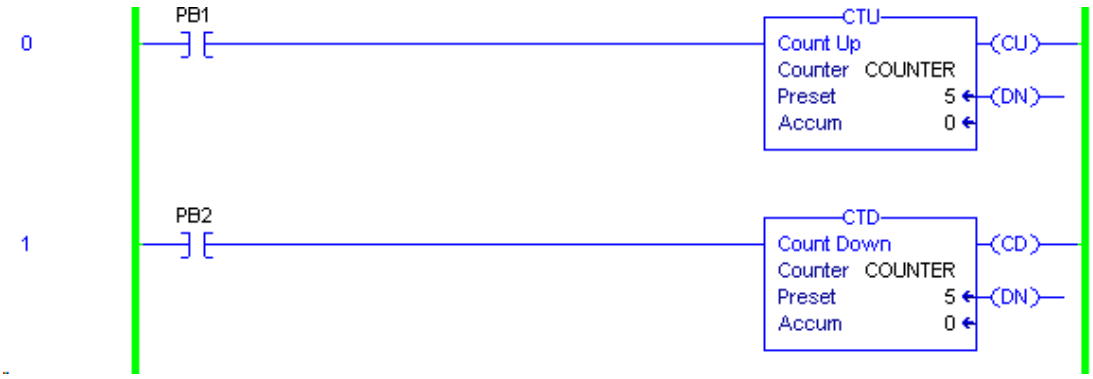


One method is to use a comparison instruction which will be covered in other comparison module.

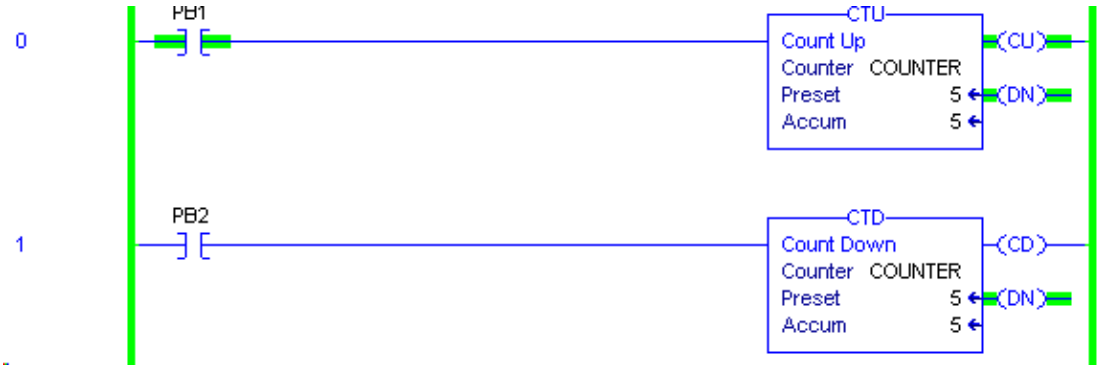

0
PB1
GRT
Greater Than (A>B)
Source A COUNTER.ACC
0
Source B 0
CTD
Count Down
Counter COUNTER
Preset 5
Accum 0
(CD)
(DN)
1
COUNTER.CU
MOTOR1
2
COUNTER.DN
MOTOR2

3. EXERCISE - COUNTER UP AND DOWN

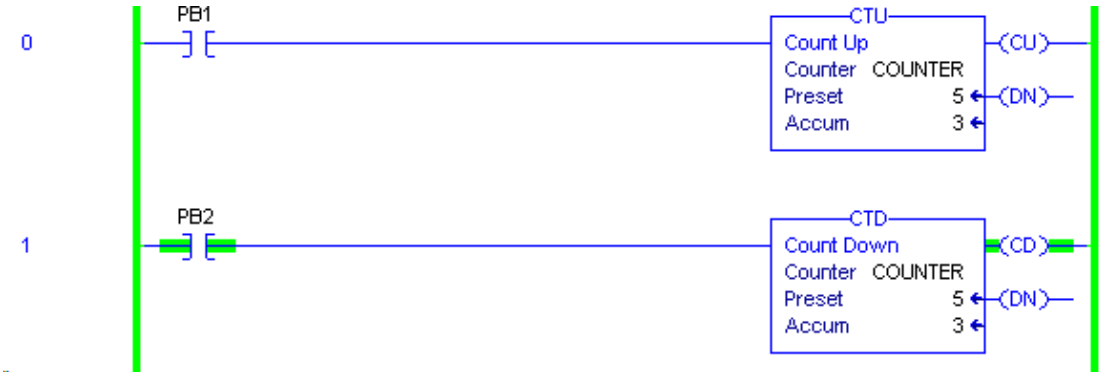
Create a logic using counter up and down instruction. Both instructions will share the same tag "COUNTER"



Toggle the "PB1" to increment accum to 5. Both CTU and CTD done bit is TRUE because they shared the same tag "COUNTER"



Toggle the "PB2" will decrease the accum value. Apply this logic to a parking lot application. Entrance gate is control by **CTU** instruction witch keeps track number of cars entered. CTD instruction is the exit gate, that subtract number of cars leaving the parking lot.



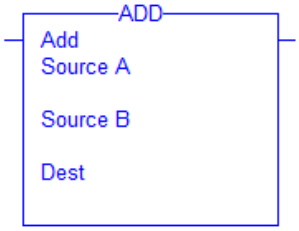
Part VI Math and Compute Instructions

Add (ADD)

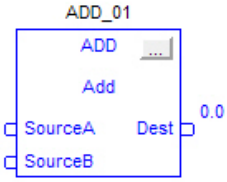
The ADD instruction adds Source A to Source B and places the result in the Destination.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Operands

There are data conversion rules for mixed data types within an instruction. See Data Conversion.

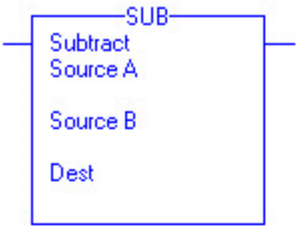
Ladder Diagram

Operand	Type	Format	Description
Source A	SINT INT DINT REAL	Immediate tag	value to add to Source B
Source B	SINT INT DINT REAL	Immediate tag	value to add to Source A
Destination	SINT INT DINT REAL	tag	tag to store the result

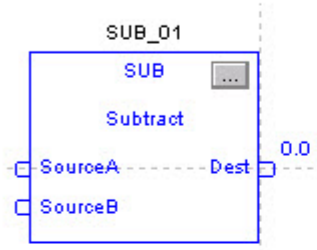
Subtract (SUB)

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Operands

There are data conversion rules for mixed data types within an instruction. See Data Conversion.

Ladder Diagram

Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	value from which to subtract Source B
Source B	SINT INT DINT REAL	immediate tag	value to subtract from Source A
Destination	SINT INT DINT REAL	tag	tag to store the result

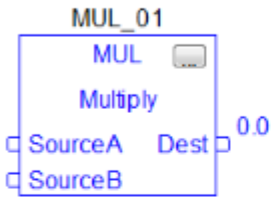
Multiply (MUL)

The MUL instruction multiplies Source A with Source B and places the result in the Destination.

Available Languages

Ladder Diagram

Function Block



Structured Text

This instruction is not available in structured text.

Operands

There are data conversion rules for mixed data types within an instruction. See Data Conversion.

Ladder Diagram

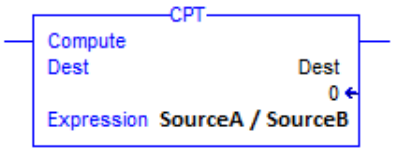
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	Immediate tag	Value of the multiplicand
Source B	SINT INT DINT REAL	immediate tag	Value of the multiplier
Destination	SINT INT DINT REAL	tag	Tag to store the result.

Divide (DIV)

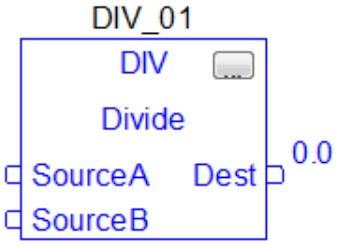
The DIV instruction and the operator '/' divides Source A by Source B and places the result in the Destination.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversion.

Ladder Diagram

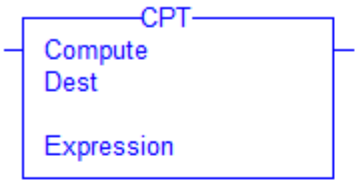
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value of the dividend.
Source B	SINT INT DINT REAL	immediate tag	Value of the divisor.
Destination	SINT INT DINT REAL	tag	Tag to store the result.

Compute (CPT)

The CPT instruction performs the math operations you define in the expression.

Available Languages

Ladder Diagram



Function Block

This instruction is not available in function block.

Structured Text

This instruction is not available in structured text.

Operands

There are data conversion rules for mixed data types within an instruction. See Data Conversion.

Ladder Diagram

Operand	Type	Format	Description
Destination	SINT INT DINT REAL	tag	Tag to store the result
Expression	SINT INT DINT REAL	immediate tag	An expression consisting of tags and/or immediate values separated by operators

Description

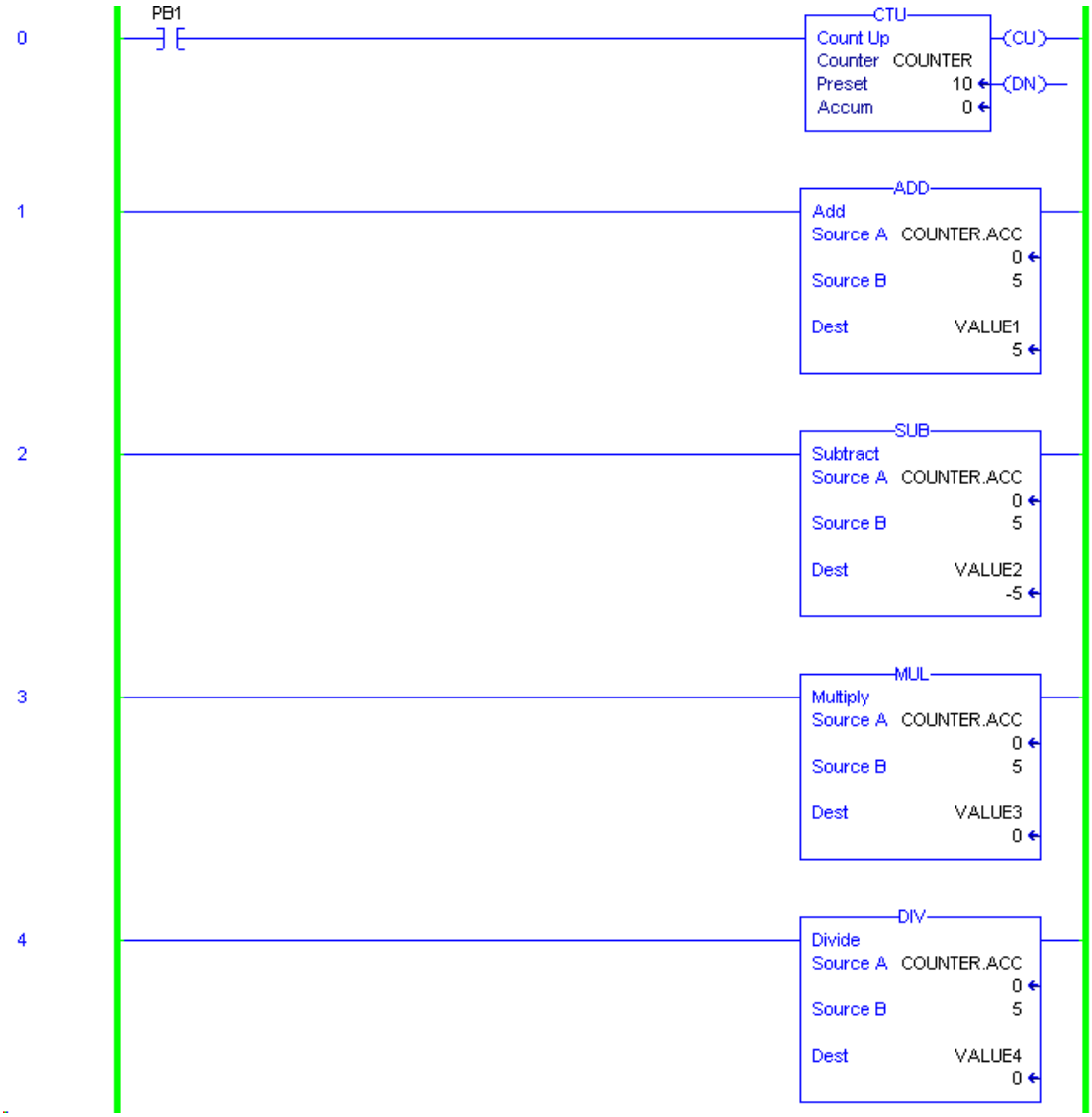
The CPT instruction performs the math operations you define in the expression. When true, the CPT instruction evaluates the expression and places the result in the Destination. The advantage of the CPT instruction is that it allows you to enter complex expressions in one instruction.

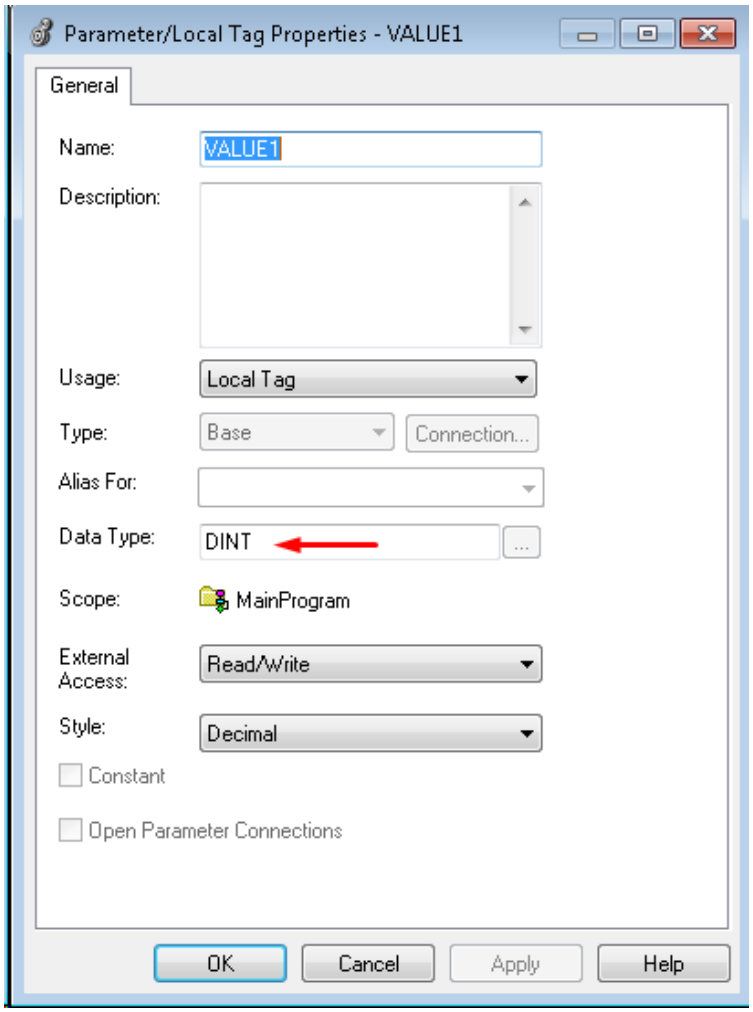
1. EXERCISE 1 - MATH INSTRUCTIONS

Create the logic below in Studio 5000. Use the COUNTER.ACC as source A for the math instructions. Define the Destination "VALUE1", "VALUE2", "VALUE3", and "VALUE4" as double integer. Assign value 5 for source B.

Right now the accumulator of the counter is at 0.

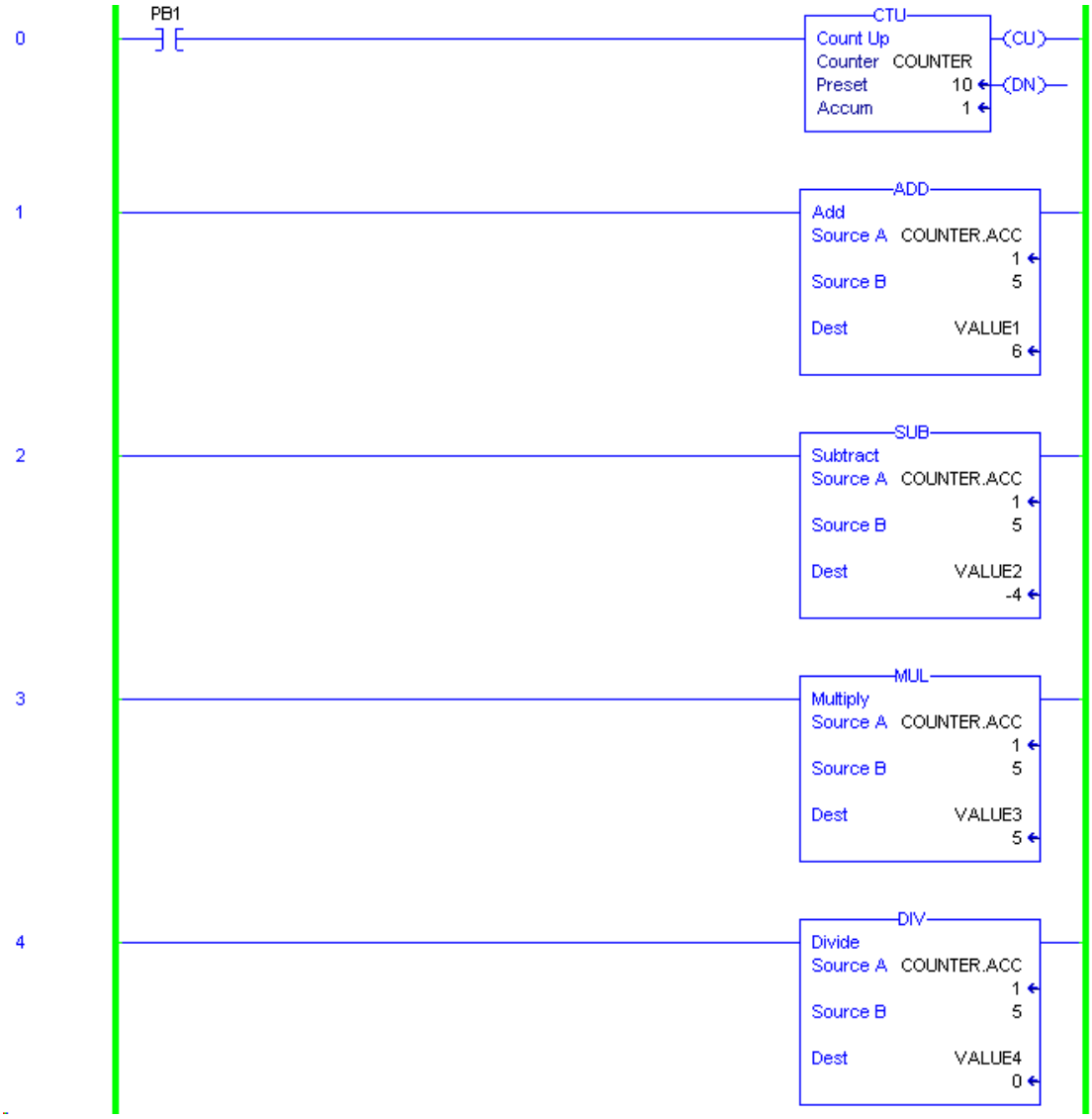
ADD instruction source A = 0, source B = 5, destination = 5
SUB instruction source A = 0, source B = 5, destination = -5
MUL instruction source A = 0, source B = 5, destination = 0
DIV instruction source A = 0, source B = 5, destination = 0

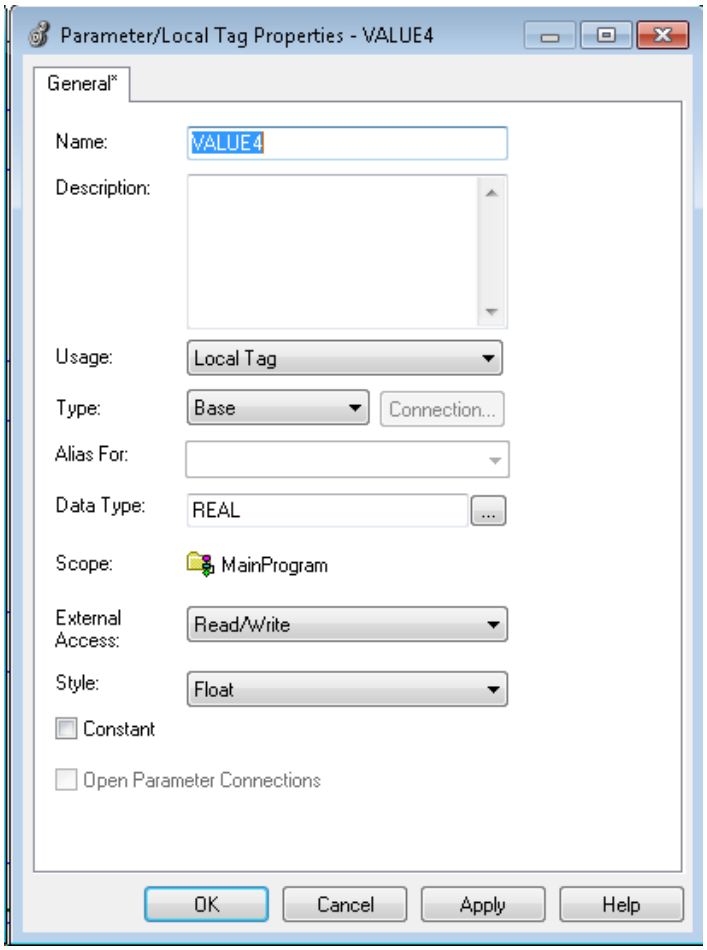




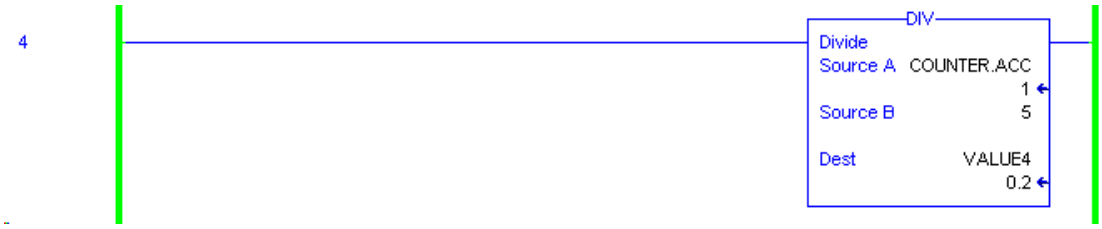
Toggle the "PB1" to increment the accumulator to 1

ADD instruction source A = 1, source B = 5, destination = 6
SUB instruction source A = 1, source B = 5, destination = -4
MUL instruction source A = 1, source B = 5, destination = 5
DIV instruction source A = 1, source B = 5, destination = 0 (the reason destination = 0 due to the data type incorrect, change the data type from integer to real)


0
PB1
CTU
Count Up
Counter COUNTER
Preset 10
Accum 1
CU
DN
1
ADD
Add
Source A COUNTER.ACC
1
Source B 5
Dest VALUE1
6
2
SUB
Subtract
Source A COUNTER.ACC
1
Source B 5
Dest VALUE2
-4
3
MUL
Multiply
Source A COUNTER.ACC
1
Source B 5
Dest VALUE3
5
4
DIV
Divide
Source A COUNTER.ACC
1
Source B 5
Dest VALUE4
0



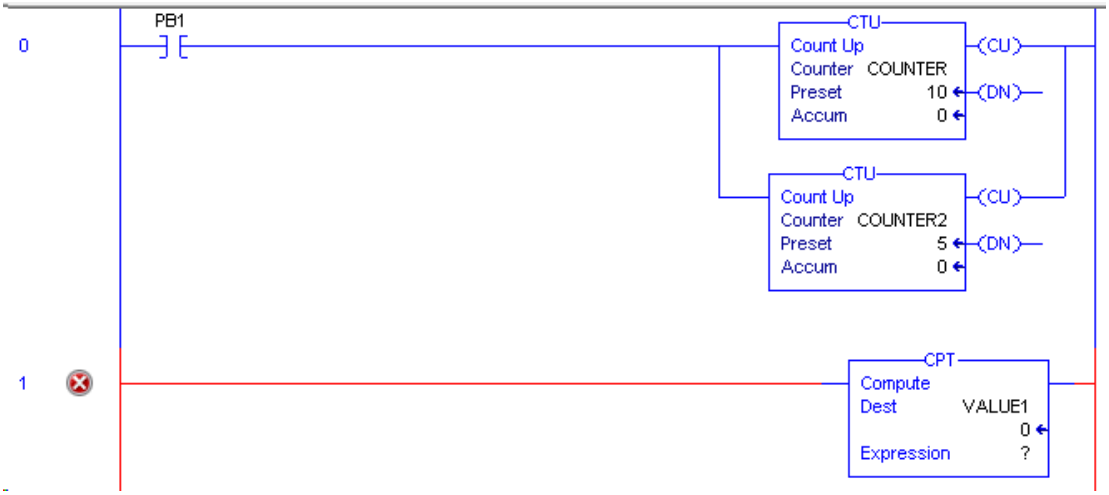
Now the destination for **DIV** instruction should show 0.2 instead of 0.



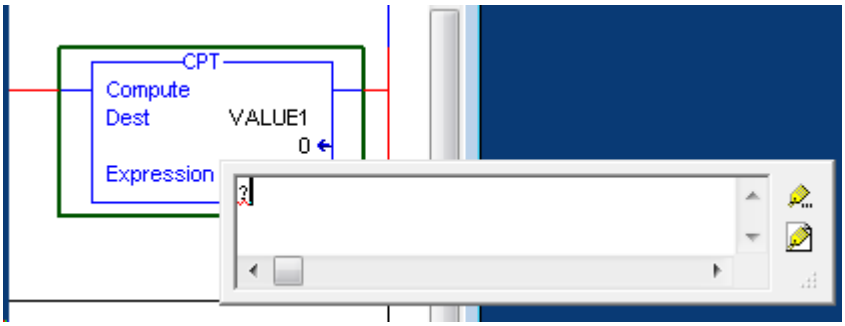
Continue to increment the accumulator and observe the changes.

2. EXERCISE 2 - COMPUTE (CPT)

Create the logic below using **CPT** instruction with two counters to simulate source A and B.

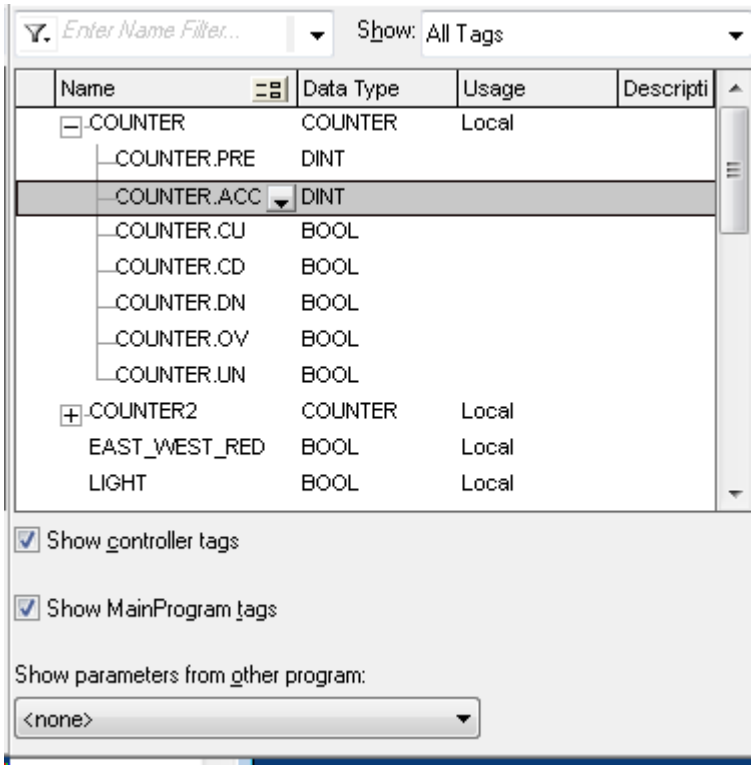


Double click on **?** mark to bring out the expression window. Delete the **?** in the expression window.

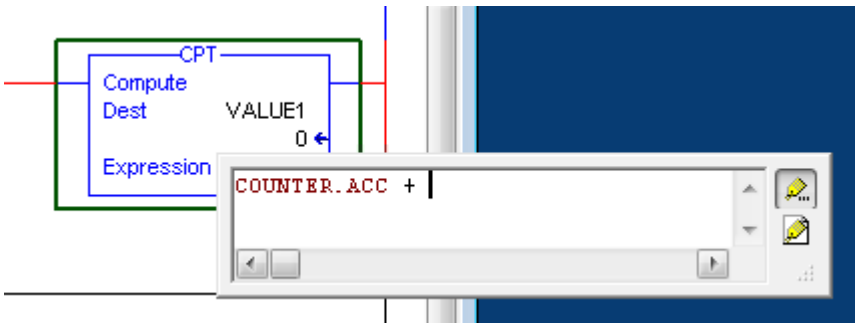


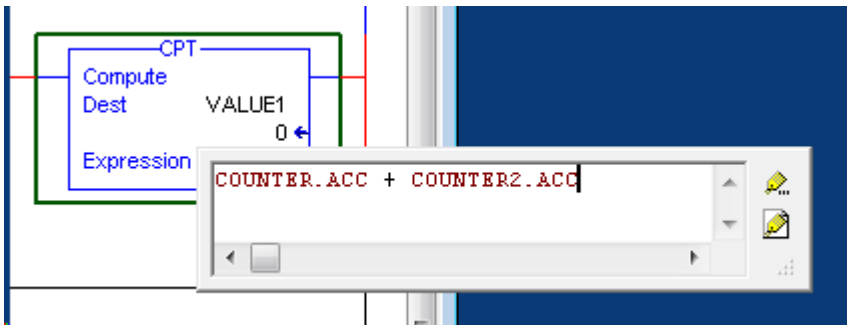
Click on the Browse Tags.

Select COUNTER.ACC from the drop down window.

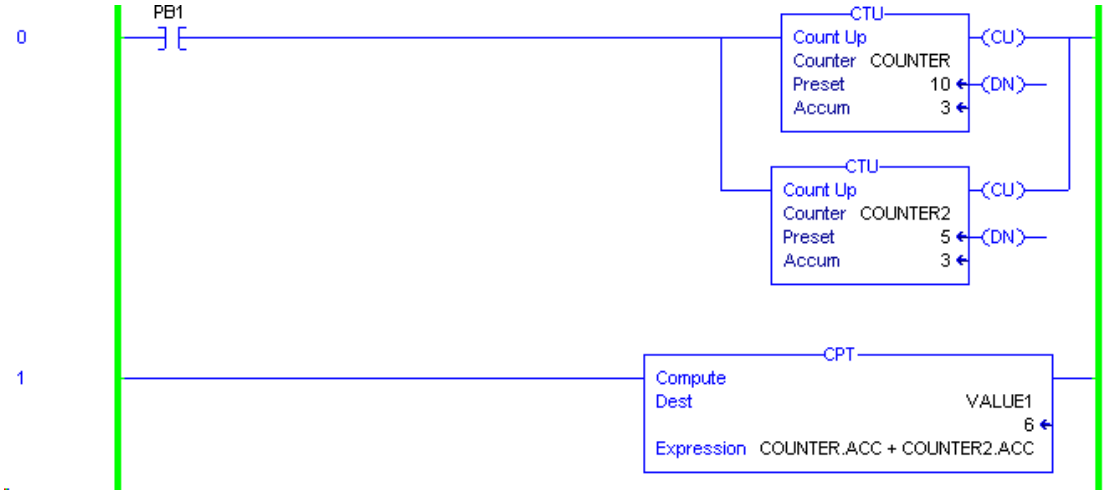


Enter COUNTER.ACC + COUNTER2.ACC into the expression window. Destination will be double integer "VALUE1"





Download the program and toggle the "PB1".


0
PB1
CTU
Count Up
Counter COUNTER
Preset 10
Accum 3
(CU)
(DN)
CTU
Count Up
Counter COUNTER2
Preset 5
Accum 3
(CU)
(DN)
1
CPT
Compute
Dest VALUE1
6
Expression COUNTER.ACC + COUNTER2.ACC

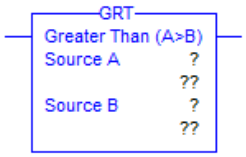
Part VII Compare Instructions

Greater Than (GRT)

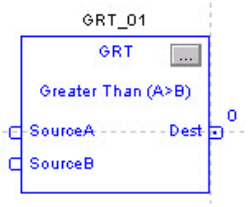
The **GRT** instruction tests whether Source A is greater than Source B.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Tip: Use the operator '>' with an expression to achieve the same result. Refer to Structured Text Syntax for more information on the syntax of expressions and assignments within structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversions.

Ladder Diagram

Numeric Comparison

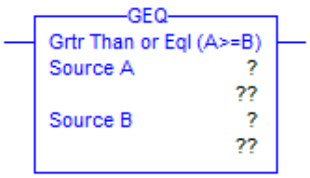
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value to test against Source B
Source B	SINT INT DINT REAL	immediate tag	Value to test against Source A

Greater Than or Equal To (GEQ)

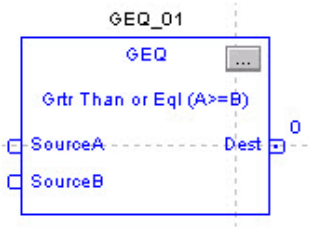
The **GEQ** instruction tests whether Source A is greater than or equal to Source B.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Tip: Use the operator '>=' with an expression to achieve the same result. Refer to Structured Text Syntax for more information on the syntax of expressions and assignments within structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversions.

Ladder Diagram

Numeric Comparison

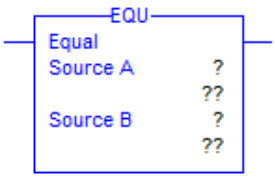
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value to test against Source B
Source B	SINT INT DINT REAL	immediate tag	Value to test against Source A

Equal To (EQU)

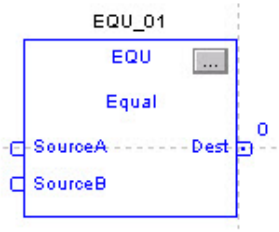
The **EQU** instruction tests whether Source A is equal to Source B.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Tip: Use the operator '=' with an expression to achieve the same result. Refer to Structured Text Syntax for more information on the syntax of expressions and assignments within structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversions.

Ladder Diagram

Numeric Comparison

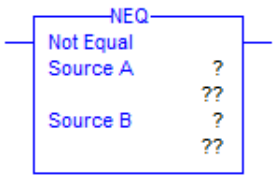
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value to test against Source B
Source B	SINT INT DINT REAL	immediate tag	Value to test against Source A

Not Equal To (NEQ)

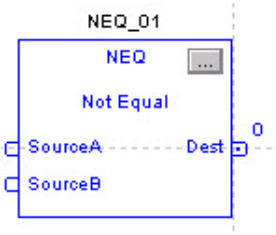
The **NEQ** instruction tests whether Source A is not equal to Source B.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Tip: Use the operator '<>' with an expression to achieve the same result. Refer to Structured Text Syntax for more information on the syntax of expressions and assignments within structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversions.

Ladder Diagram

Numeric Comparison

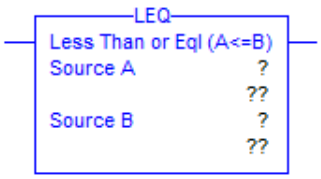
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value to test against Source B
Source B	SINT INT DINT REAL	immediate tag	Value to test against Source A

Less Than or Equal To (LEQ)

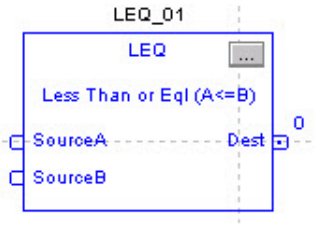
The **LEQ** instruction tests whether Source A is less than or equal to Source B.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Tip: Use the operator '<=' with an expression to achieve the same result. Refer to Structured Text Syntax for more information on the syntax of expressions and assignments within structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversions.

Ladder Diagram

Numeric Comparison

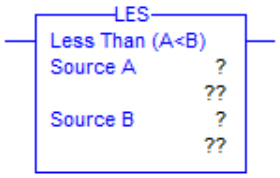
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value to test against Source B
Source B	SINT INT DINT REAL	immediate tag	Value to test against Source A

Less Than (LES)

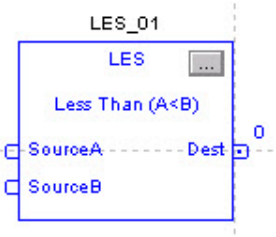
The **LES** instruction tests whether Source A is less than Source B.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Tip: Use the operator '<' with an expression to achieve the same result. Refer to Structured Text Syntax for more information on the syntax of expressions and assignments within structured text.

Operands

There are data conversion rules for mixing numeric data types within an instruction. See Data Conversions.

Ladder Diagram

Numeric Comparison

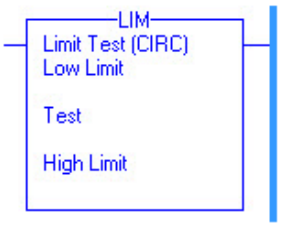
Operand	Type	Format	Description
Source A	SINT INT DINT REAL	immediate tag	Value to test against Source B
Source B	SINT INT DINT REAL	immediate tag	Value to test against Source A

Limit (LIM)

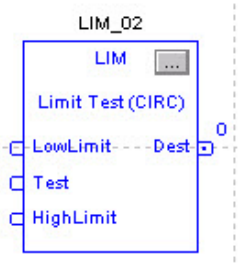
The **LIM** instruction tests whether the Test value is within the range of Low Limit to the High Limit.

Available Languages

Ladder Diagram



Function Block



Structured Text

This instruction is not available in structured text.

Operands

There are data conversion rules for mixed data types within an instruction. See Data Conversion.

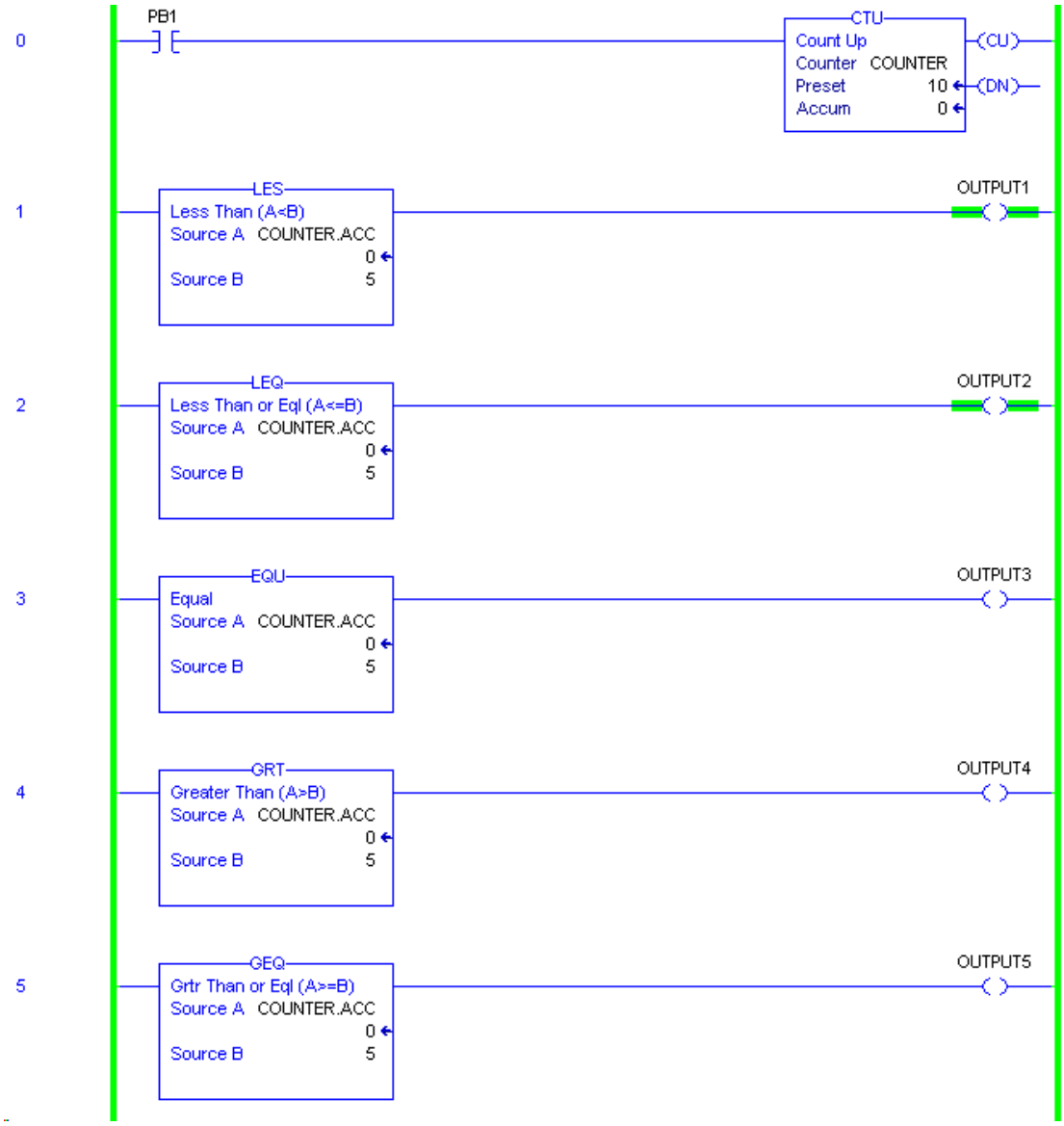
Ladder Diagram

Operand	Type	Format	Description
Low limit	SINT INT DINT REAL	immediate tag	Value of lower limit
Test	SINT INT DINT REAL	immediate tag	Value to test
High limit	SINT INT DINT REAL	immediate tag	Value of upper limit
	A SINT or INT tag converts to a DINT value by sign-extension.		

1. EXERCISE 1 - COMPARE INSTRUCTIONS

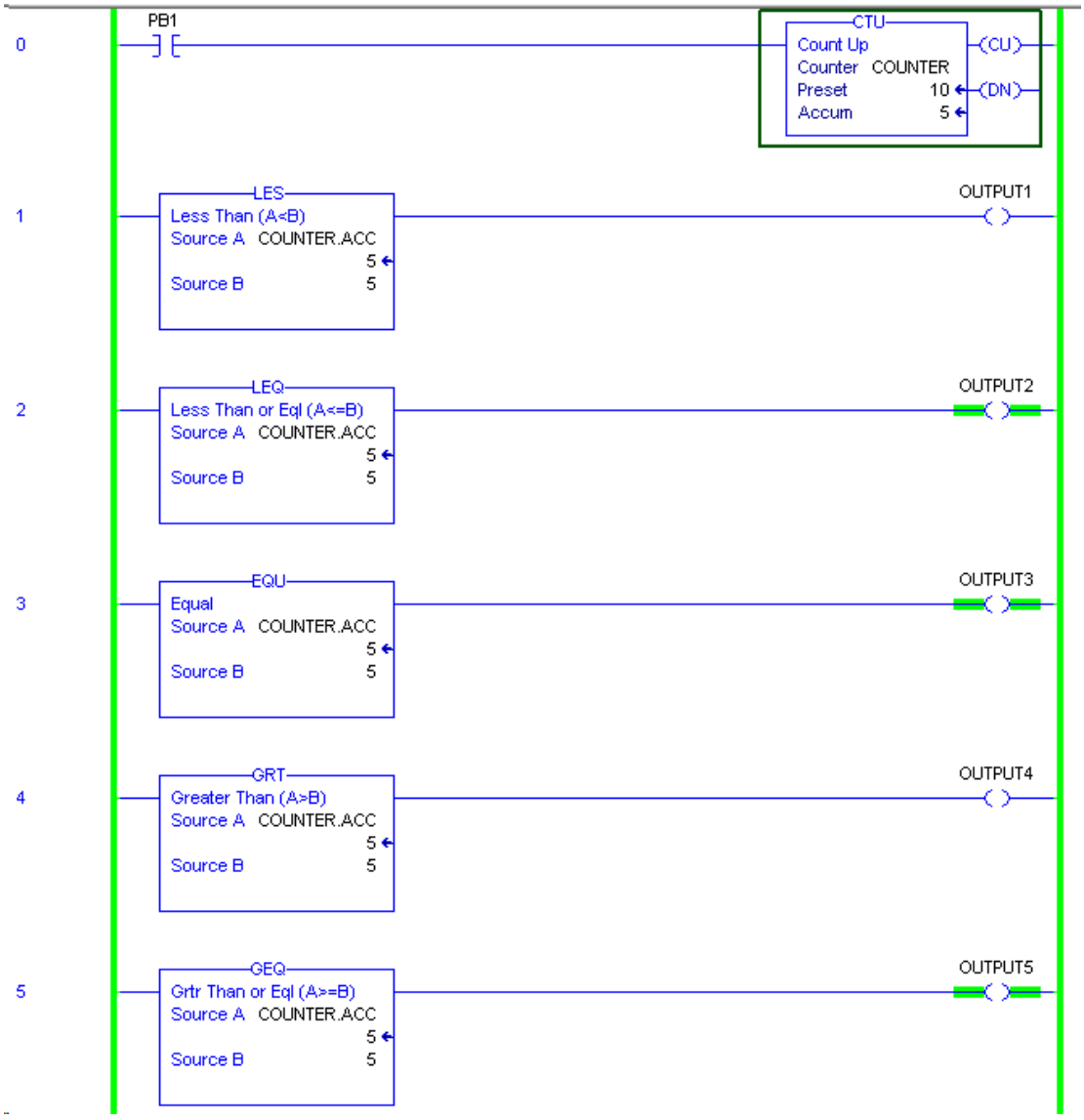
Create the logic below with compare instructions. Use a counter to simulate source A, and value 5 for source B.

LES - source A = 0 & source B = 5, source A is less than B, therefore "OUTPUT1" is true.
LEQ - source A = 0 & source B = 5, source A is less than or equal to B, therefore "OUTPUT2" is true.
EQU - source A = 0 & source B = 5, source A is not equal to source B, therefore "OUTPUT3" is false.
GRT - source A = 0 & source B = 5, source A is not greater than source B, therefore "OUTPUT4" is false.
GEQ - source A = 0 & source B = 5, source A is not greater or equal to source B, therefore "OUTPUT5" is false.



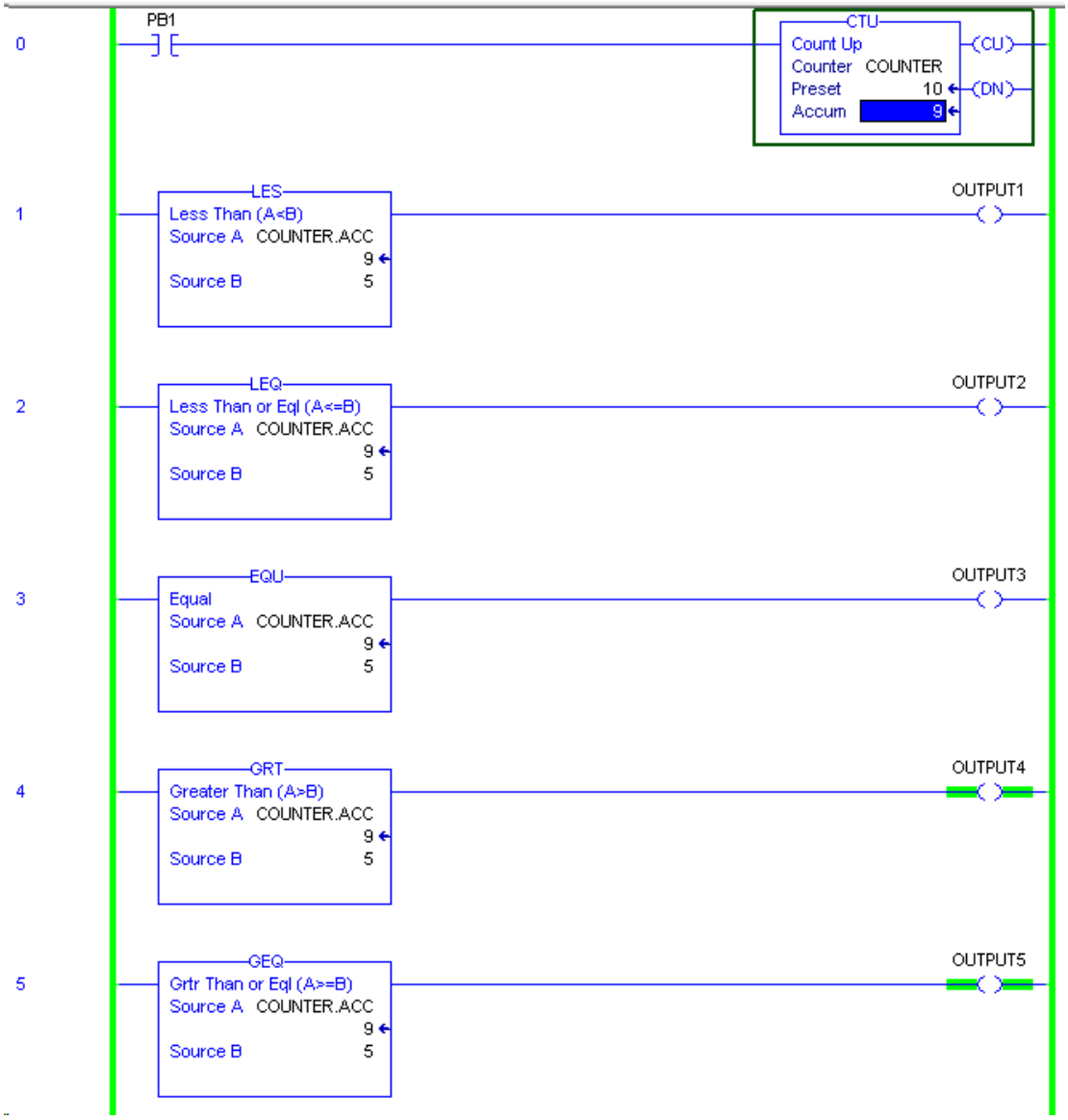
Change the counter's accumulator to 5.

LES - source A = 5 & source B = 5, source A is not less than B, therefore "OUTPUT1" is false.
LEQ - source A = 5 & source B = 5, source A is not less but equal to B, therefore "OUT-PUT2" is true.
EQU - source A = 5 & source B = 5, source A is not equal to source B, therefore "OUT-PUT3" is true.
GRT - source A = 5 & source B = 5, source A is not greater than source B, therefore "OUT-PUT4" is false.
GEQ - source A = 5 & source B = 5, source A is not greater but equal to source B, therefore "OUTPUT5" is true.



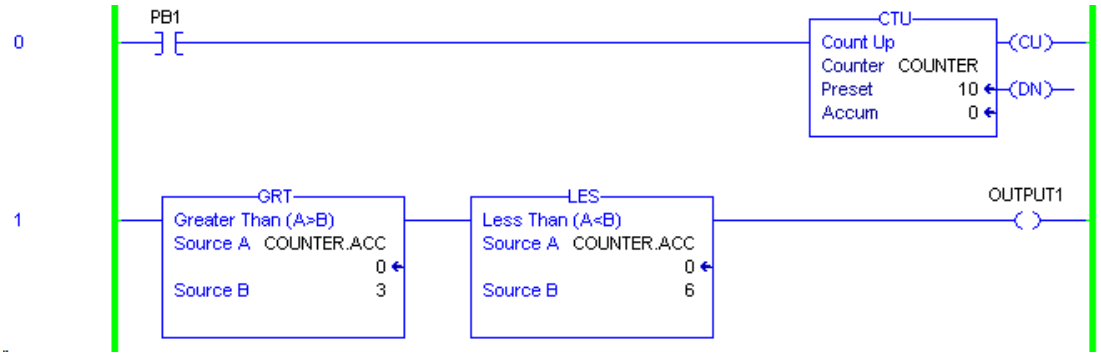
Change the counter's accumulator to 9.

LES - source A = 9 & source B = 5, source A is not less than B, therefore "OUTPUT1" is false.
LEQ - source A = 9 & source B = 5, source A is not less or equal to B, therefore "OUT-PUT2" is false.
EQU - source A = 9 & source B = 5, source A is not equal to source B, therefore "OUT-PUT3" is false.
GRT - source A = 9 & source B = 5, source A is greater than source B, therefore "OUT-PUT4" is true.
GEQ - source A = 9 & source B = 5, source A is greater or equal to source B, therefore "OUTPUT5" is true.

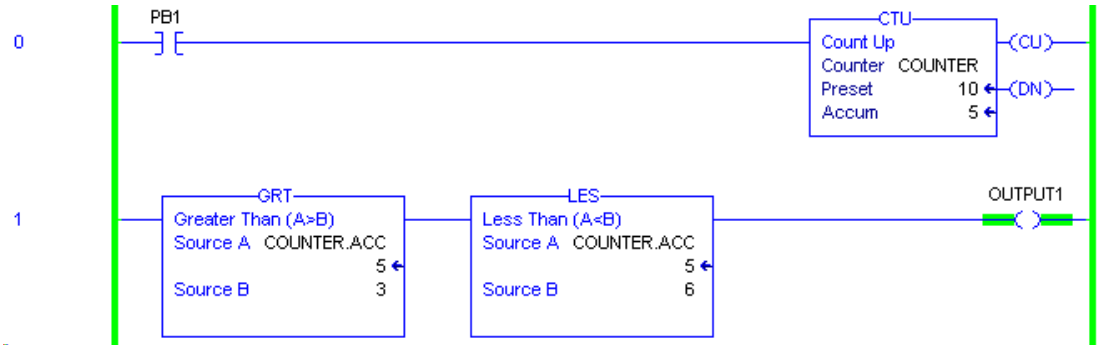

0
PB1
CTU
Count Up
Counter COUNTER
Preset 10
Accum 9
(CU)
(DN)
1
LES
Less Than (A<B)
Source A COUNTER.ACC
9
Source B 5
OUTPUT1
2
LEQ
Less Than or Eql (A<=B)
Source A COUNTER.ACC
9
Source B 5
OUTPUT2
3
EQU
Equal
Source A COUNTER.ACC
9
Source B 5
OUTPUT3
4
GRT
Greater Than (A>B)
Source A COUNTER.ACC
9
Source B 5
OUTPUT4
5
GEQ
Grtr Than or Eql (A>=B)
Source A COUNTER.ACC
9
Source B 5
OUTPUT5

2. EXERCISE 2 - AND CONDITION

You can specify a condition for a range of values. Example below will become true when accumulator is greater than 3 and less than 6.

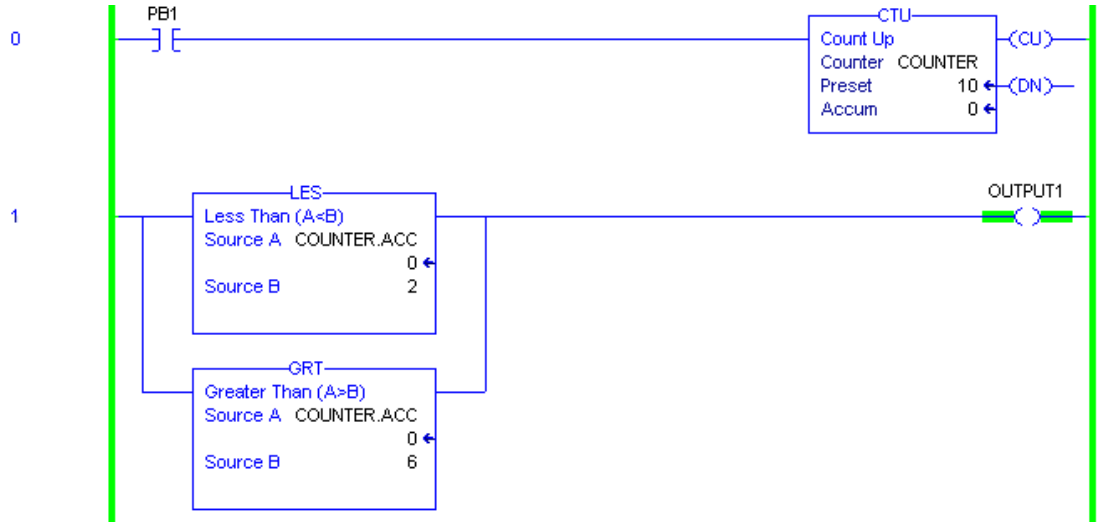


Change the accumulator by toggle the "PB1" to 5 for the counter and observe the outcome.

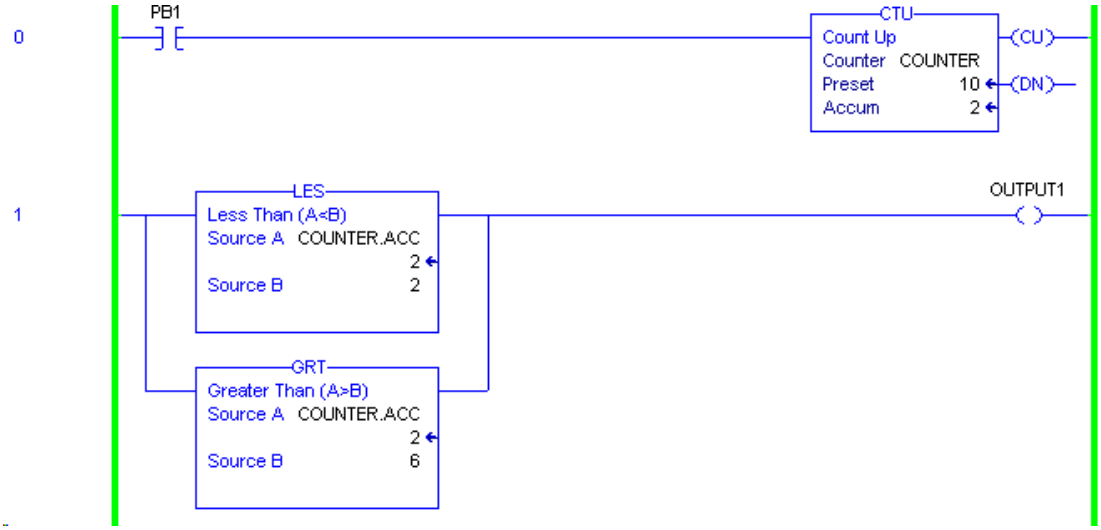


3. EXERCISE 3 - OR CONDITION

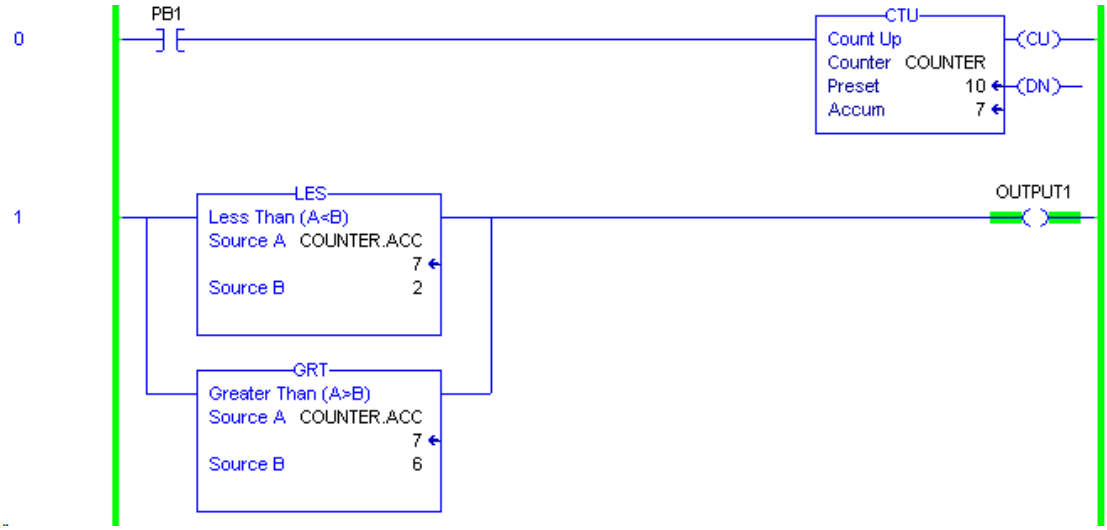
Create the logic below. Because the accumulator is less than 2 therefore the "OUTPUT1" is TRUE.



Change the accumulator of the counter to 2. Now source A is no longer less than source B therefore the "OUTPUT1" is FALSE.



Change the accumulator of the counter to 7. Now source A is greater than source B therefore the "OUTPUT1" is TRUE.


0
PB1
CTU
Count Up
Counter COUNTER
Preset 10
Accum 7
(CU)
(DN)
1
LES
Less Than (A<B)
Source A COUNTER.ACC
7
Source B 2
GRT
Greater Than (A>B)
Source A COUNTER.ACC
7
Source B 6
OUTPUT1

Part VIII FactoryTalk View Studio

INTRODUCTION TO FACTORYTALK VIEW STUDIO

There are 2 versions of this development software available, FactoryTalk View Studio for ME or FactoryTalk View Studio Enterprise. The full version of FactoryTalk View Studio can be used for Supervisory Edition (SE) applications or stand alone PC based HMI applications. You can also built Machine Edition (ME) applications with this software although you only need the FactoryTalk View Studio for Machine addition to do that.

We have the full version so if time permits we will set up a small distributed HMI system in the classroom. The FactoryTalk View Studio for ME is used mostly to program applications for PanelView Plus operator interfaces. (See picture below). If you are familiar with RSView32 you will see that FactoryTalk View Studio has many similarities.

FactoryTalk® View Machine Edition (ME) software is a versatile HMI application that provides a dedicated and powerful solution for machine-level operator interface devices. As an integral element of the Rockwell Automation visualization solution, FactoryTalk View Machine Edition provides superior graphics, run-time user management, language switching and faster commissioning time through a common development environment.

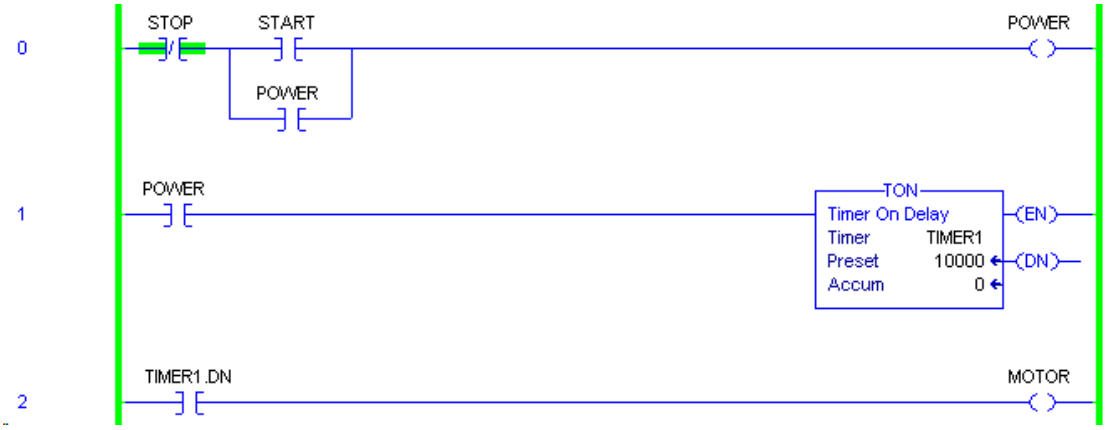
FactoryTalk® View Site Edition (SE) is a supervisory-level HMI software for monitoring and controlling distributed-server/multi-user applications. It provides a comprehensive and accurate picture of operations, meeting the demands of multiple stakeholders including engineering, maintenance, operations, and production Information Technology (IT). FactoryTalk View SE provides robust and reliable functionality in a single software package that scales from a standalone HMI system to a distributed visualization solution. As our customers have experienced over the past few years, FactoryTalk View SE enables you to take advantage of mobility, virtualization, and other new features.


RSMalthouse Overview
RSView ME Station
Germination Machine

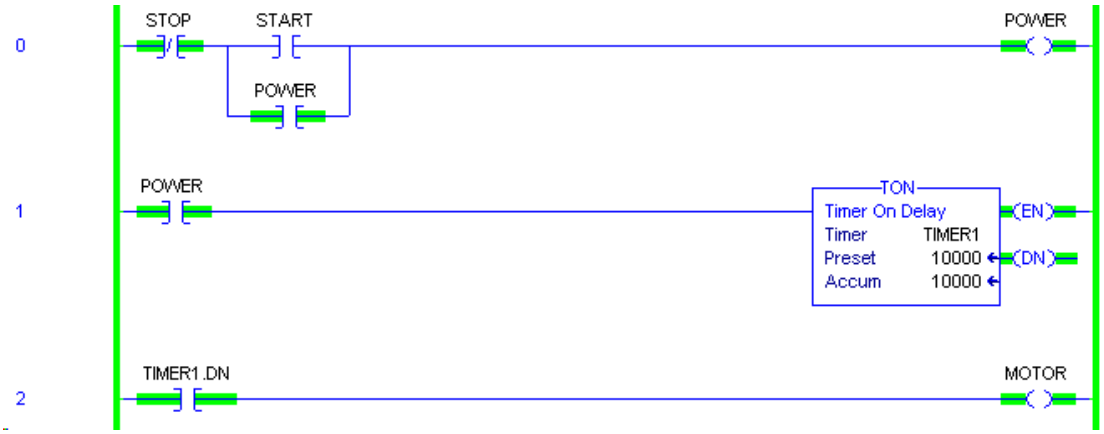
1. EXERCISE 1 - RSLINX ENTERPRISE COMMUNICATION SETUP

Create a logic that use a timer to delay the motor for 10 seconds before turn off.

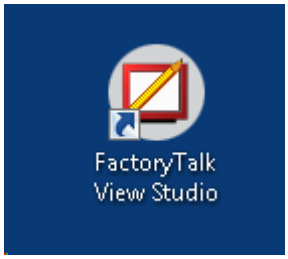
Download and make sure the program is in Run Mode.



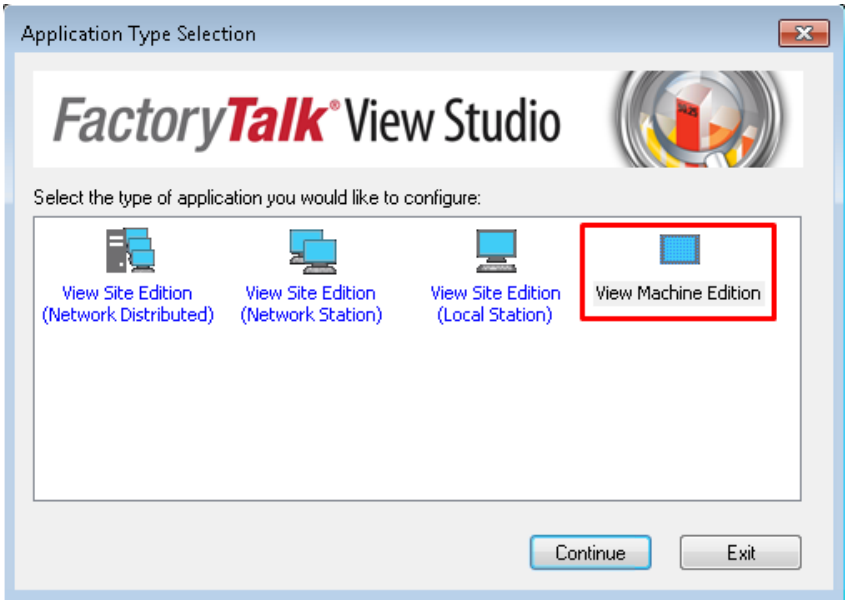
Toggle the "START" button to enable the timer, and "STOP" button to disable and reset the timer.



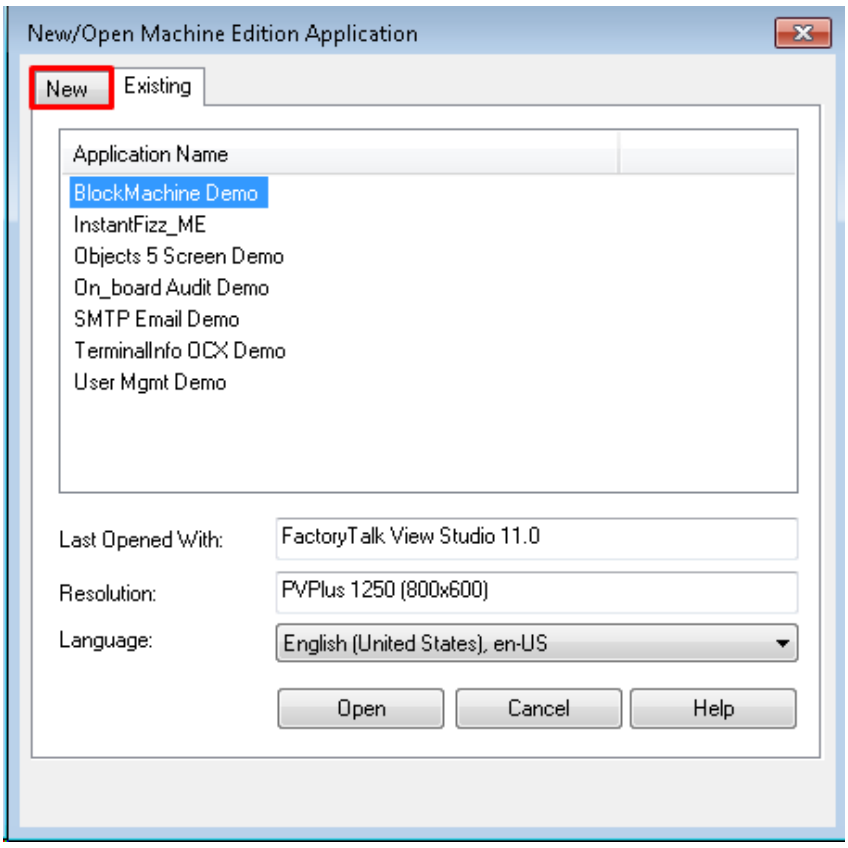
Let's open FactoryTalk View Studio by double click on this icon located on the desktop.

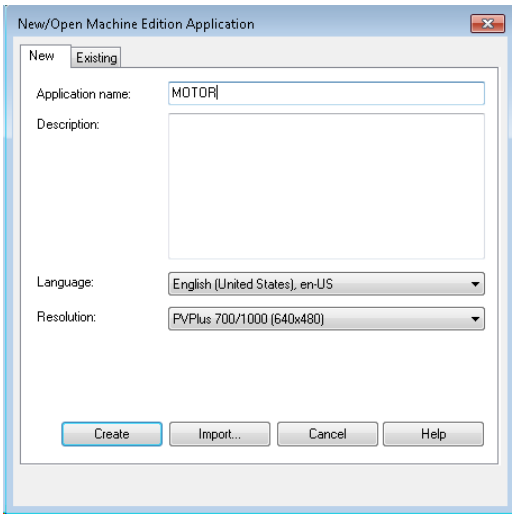


Select the View Machine Edition then click the **Continue** button.

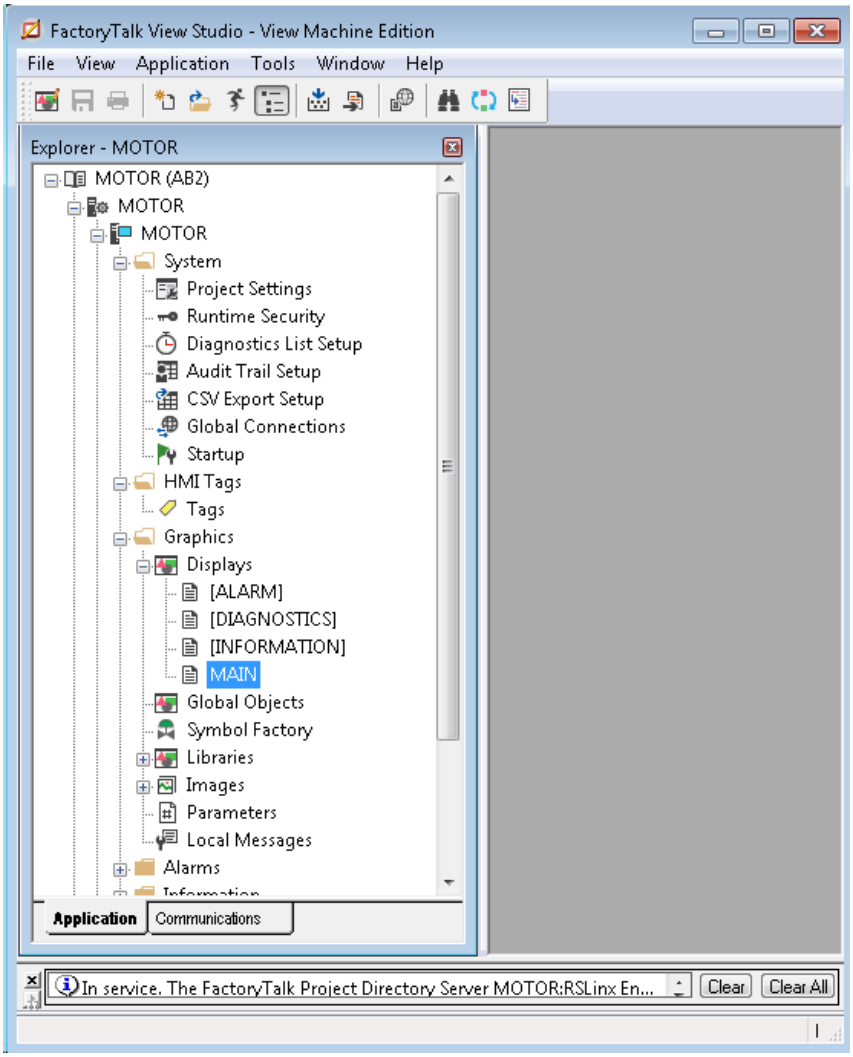


In the "**New/Open Machine Edition Application**" select the "**New**" tab and fill in a name for your application. Select the correct language and click the "**Create**" button.





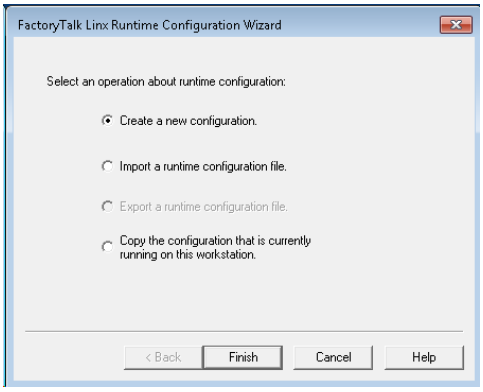
After FactoryTalk View Studio launched, you will see “Explorer - MOTOR” window on the left. There are folders with tools and features for create an application. There are two important steps, first is to set up the communication then build a graphic display (HMI).



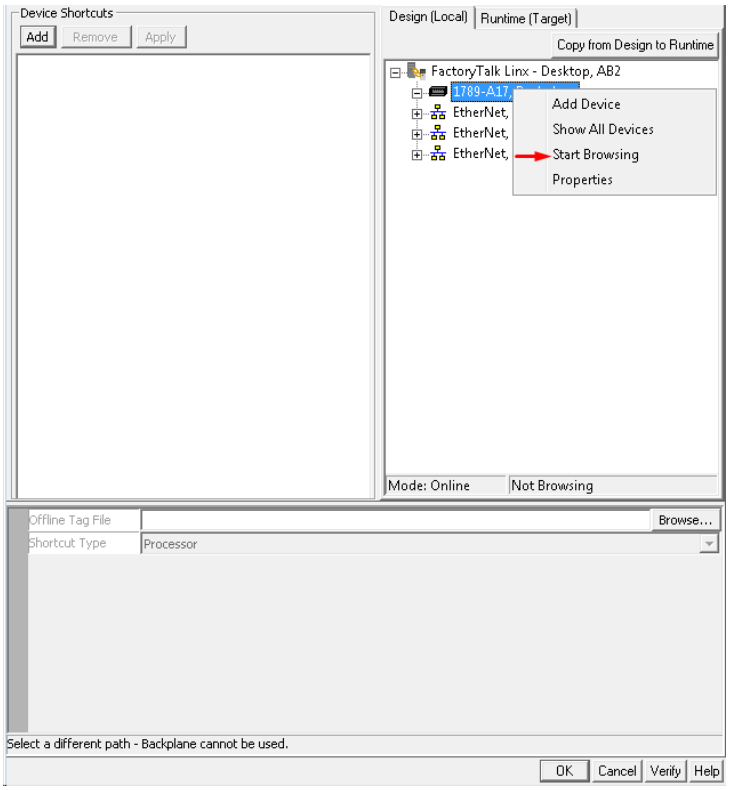
There are a few different ways you can communicate with PLC's. The primary communication method for FactoryTalk View Studio is with an application called **"RSLinx Enterprise".** This application is the next generation of the RSLinx you have used in previous labs. RSLinx Enterprise is the preferred method of communication, but it does have limitations in some cases. It can be used to set up communication with the SoftLogix Controller on your PC, but it could not be used to communicate with legacy products like PLC5 or SLC500. However you can use **"RSLinx Classic"** for setup OPC connection for the legacy products.

Scroll to the lower bottom of explore. Expand the FactoryTalk Linx folder then double click on the Communication Setup to open the FactoryTalk Linx Runtime Configuration Wizard.

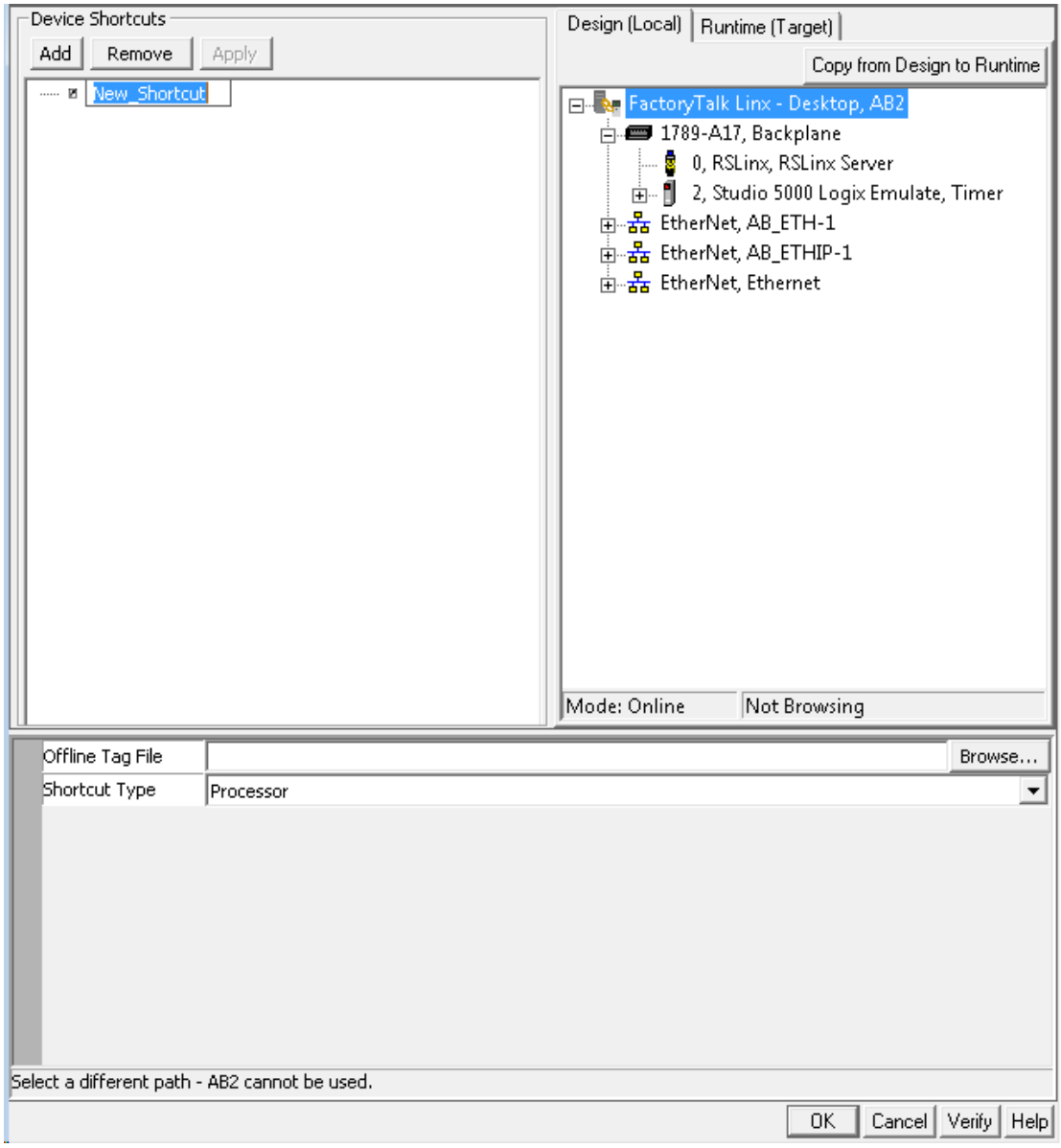
Select Create a new configuration, click the **Finish** button to continue.



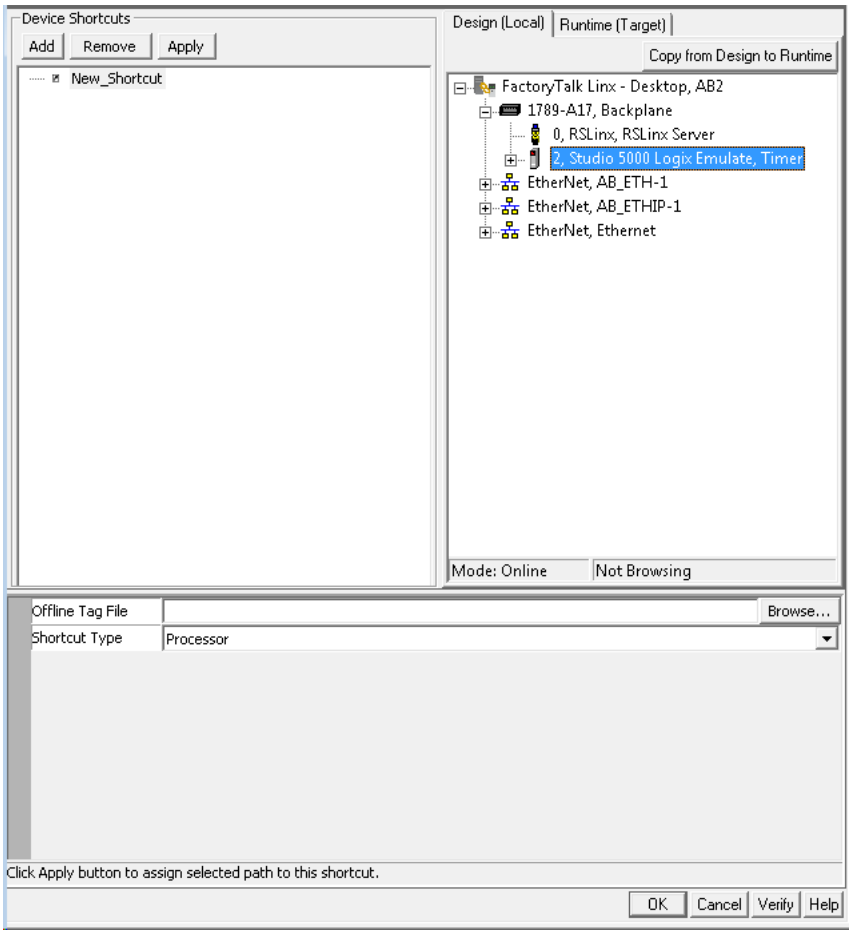
On the right window, right click on the 1789-A17, Backplane driver. Select Start Browsing to refresh the list.



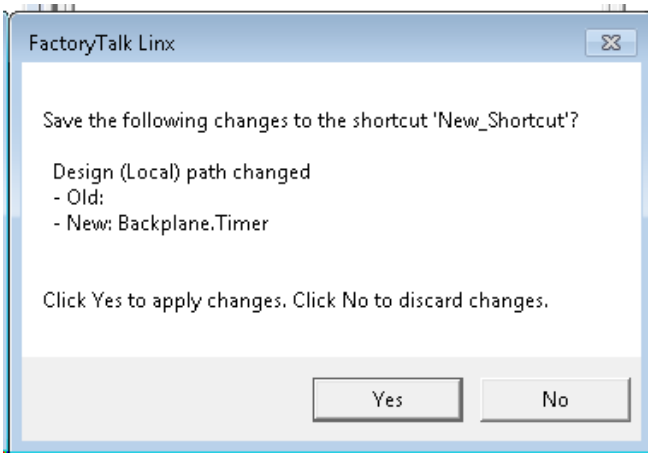
Click the **Add** button to create a new shortcut to the controller. Rename New_Shortcut if required. At this point the **Apply** button should be grayed out.



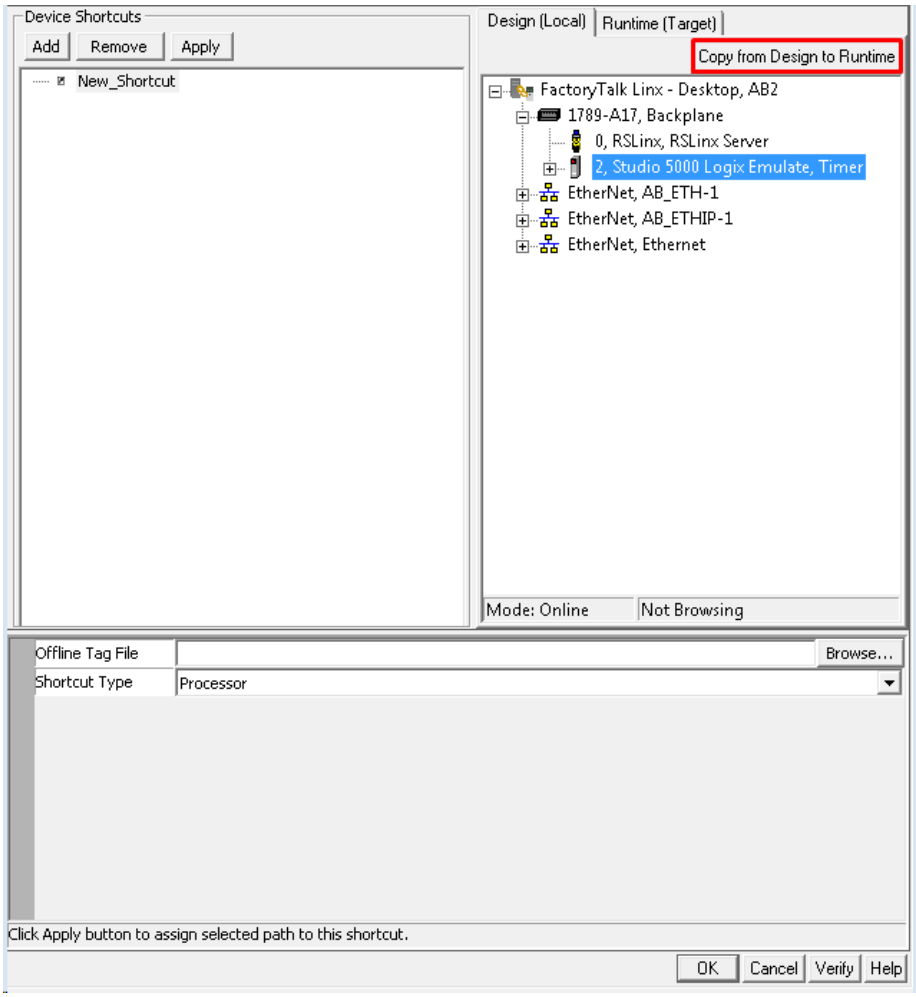
Highlight the New_Shortcut on the left window then select **1789-A17, Backplane** -> **2, Studio 5000 Logix Emulate**. Now the **Apply** button should be enabled. Click the **Apply** button to complete the setup.



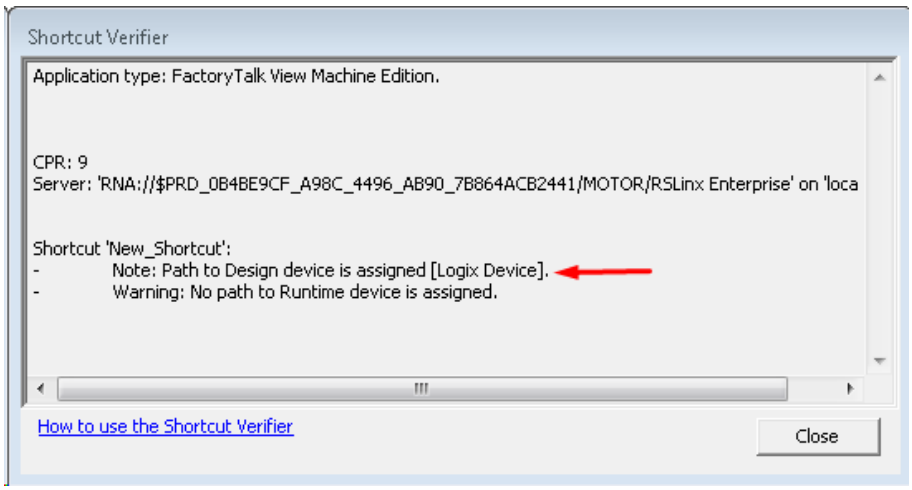
Click the **Yes** button to continue.



Click the **Copy from Design to Runtime** button in case you to compile to a run-time application later.

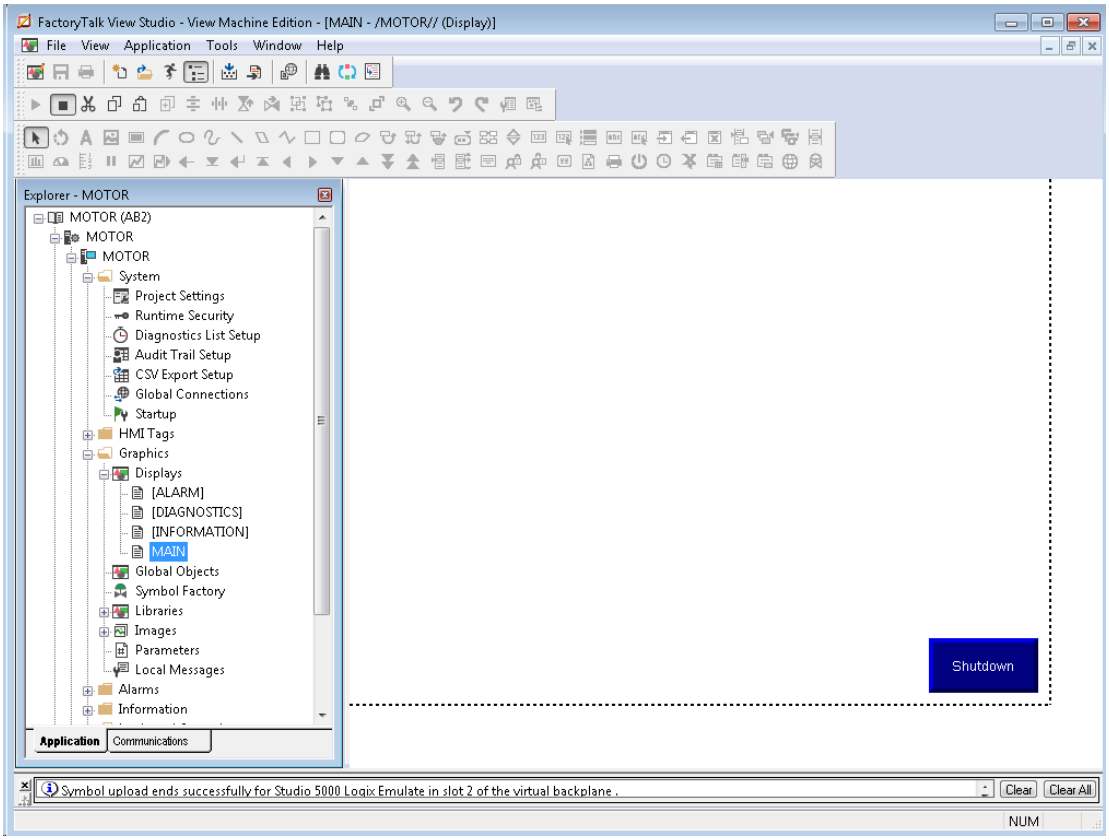


Click the **Verify** button to verify the communication is setup properly. Close the pop window and click the **OK** button to complete the communication setup.



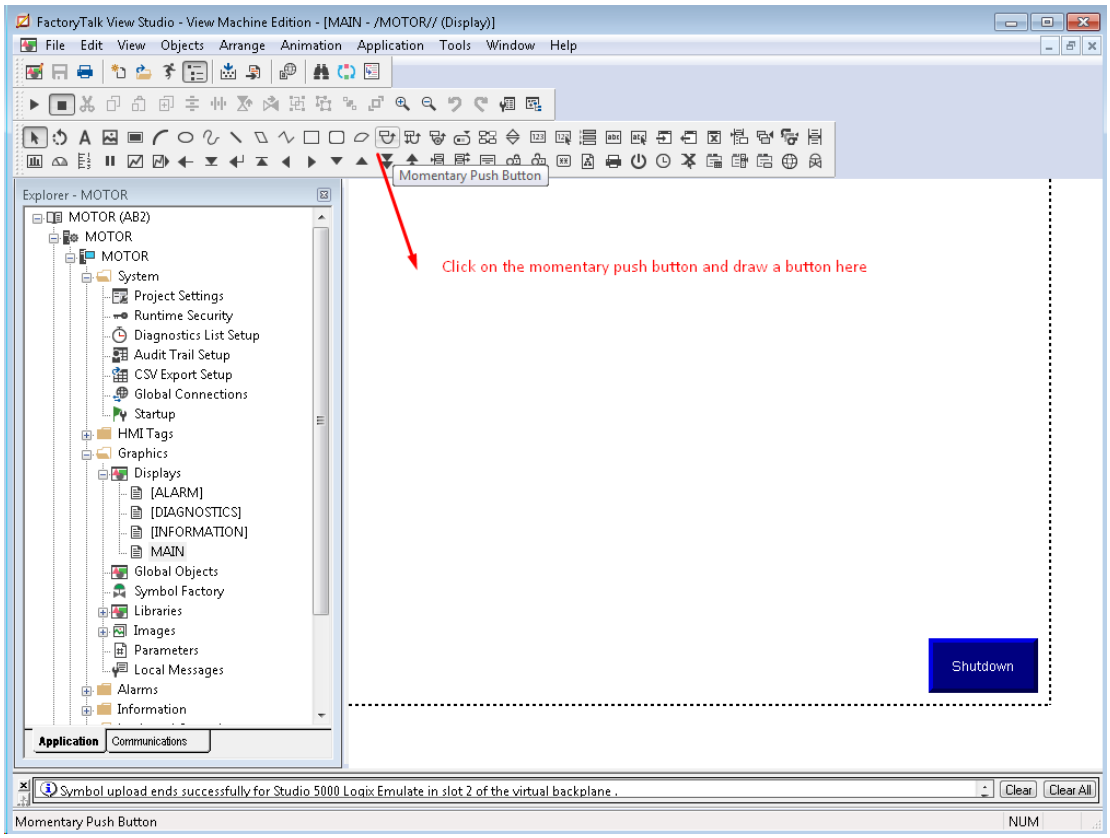
2. EXERCISE 2 - GRAPHIC

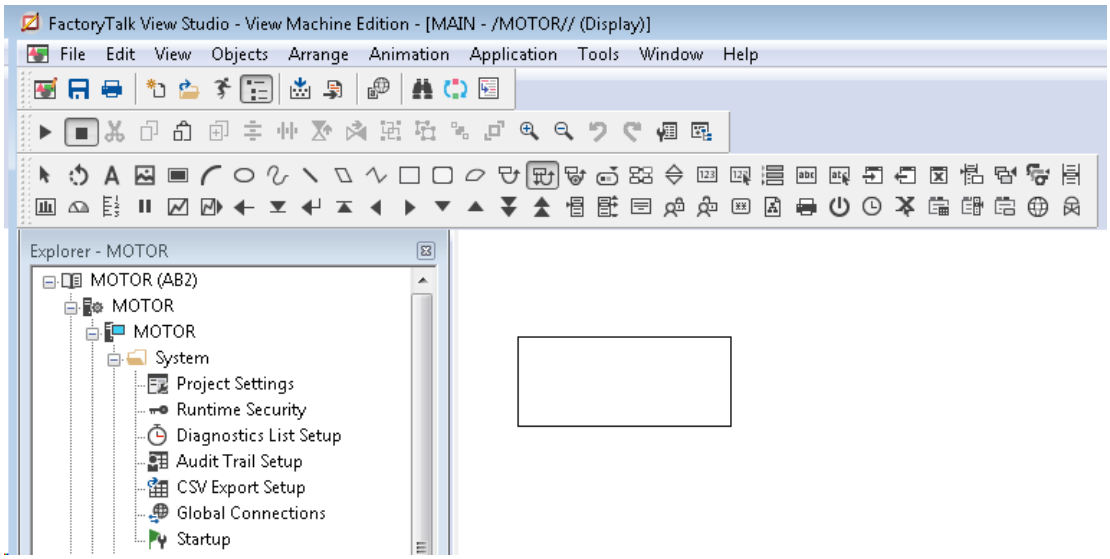
Expand the Graphics tree then select Display . Double click the Main folder to open the default page. Click anywhere in the blank space to enable the tool bar.



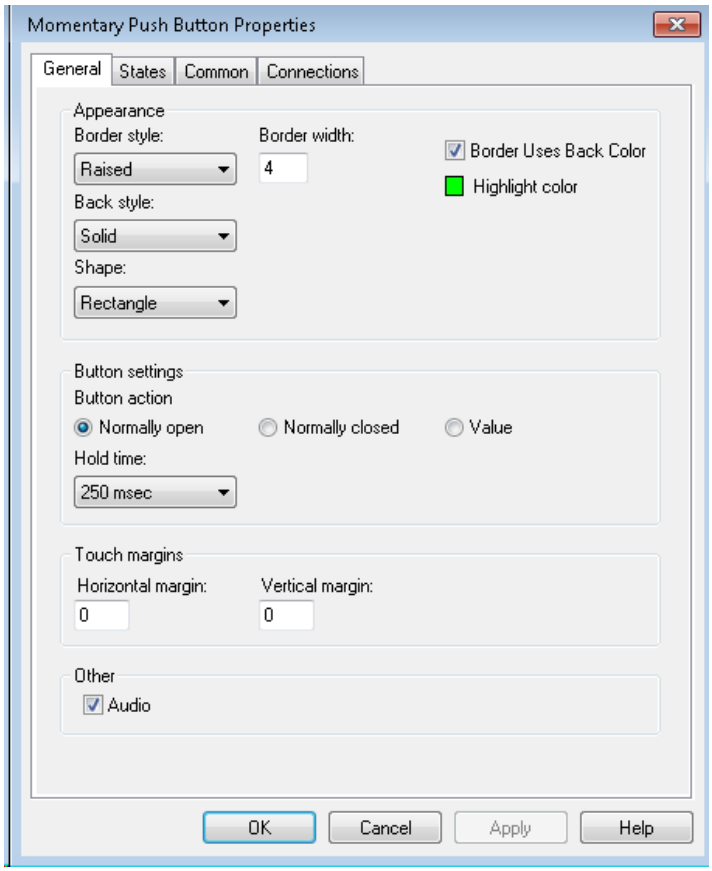
You are now ready to add objects to your display. You can have objects on the display for explanatory purpose to show an operator how the process works. These objects are usually stationary and can be made part of the background what we sometimes call “Wallpaper”. Other object can be used to interact with the PLC. This could be done by reading data from the PLC or writing to it. You can use the many standard objects available from the toolbar, or use pictures from the library. You can also use the drawing tools available in FactoryTalk View Studio and even animate the graphics you create.

Select a momentary push button from the menu bar. Draw a button by holding the left mouse button and drag.

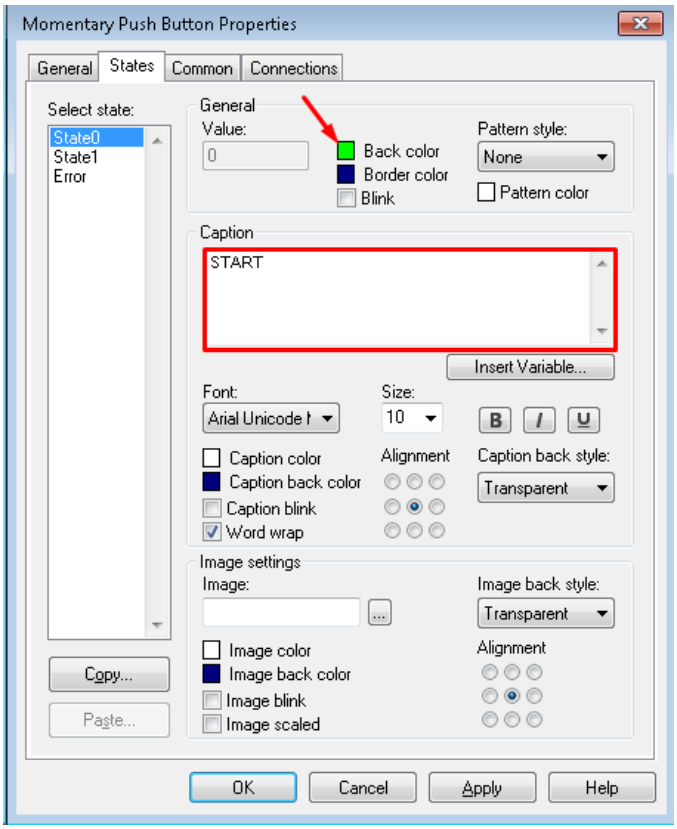




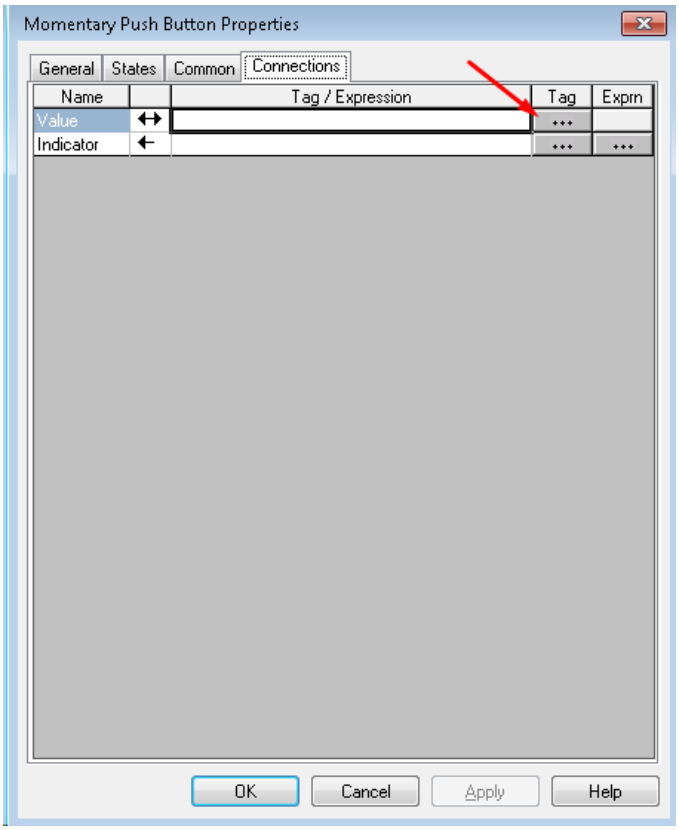
You can change the button properties like style, color, shape, state settings....



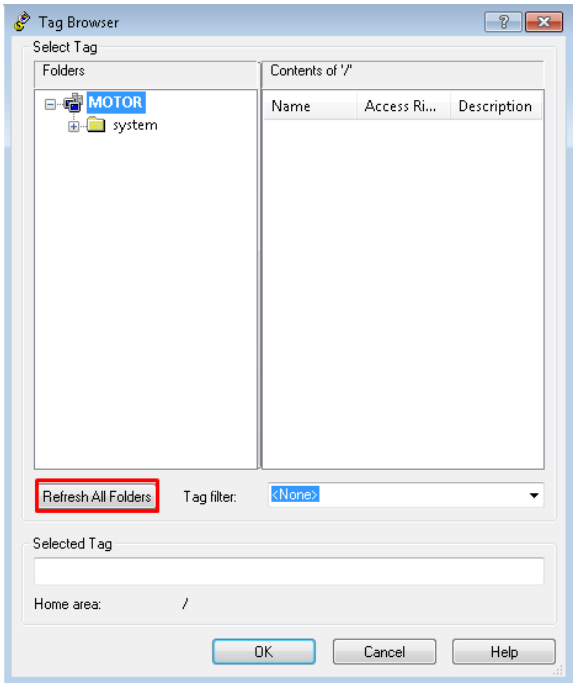
Keep the default settings in the General tab. Click on the States tab then change the Back color to green and enter START in the capture box. Click the **Apply** button to save the settings. Adjust the settings according to your design requirement.



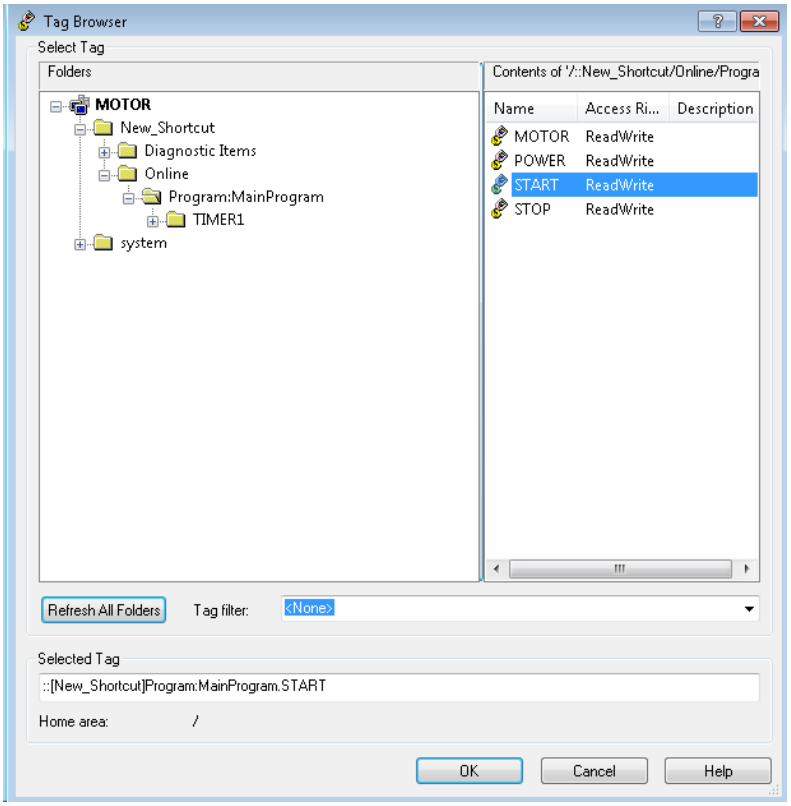
Go to the Connections tab then click on the button below Tag to enter the tag browser



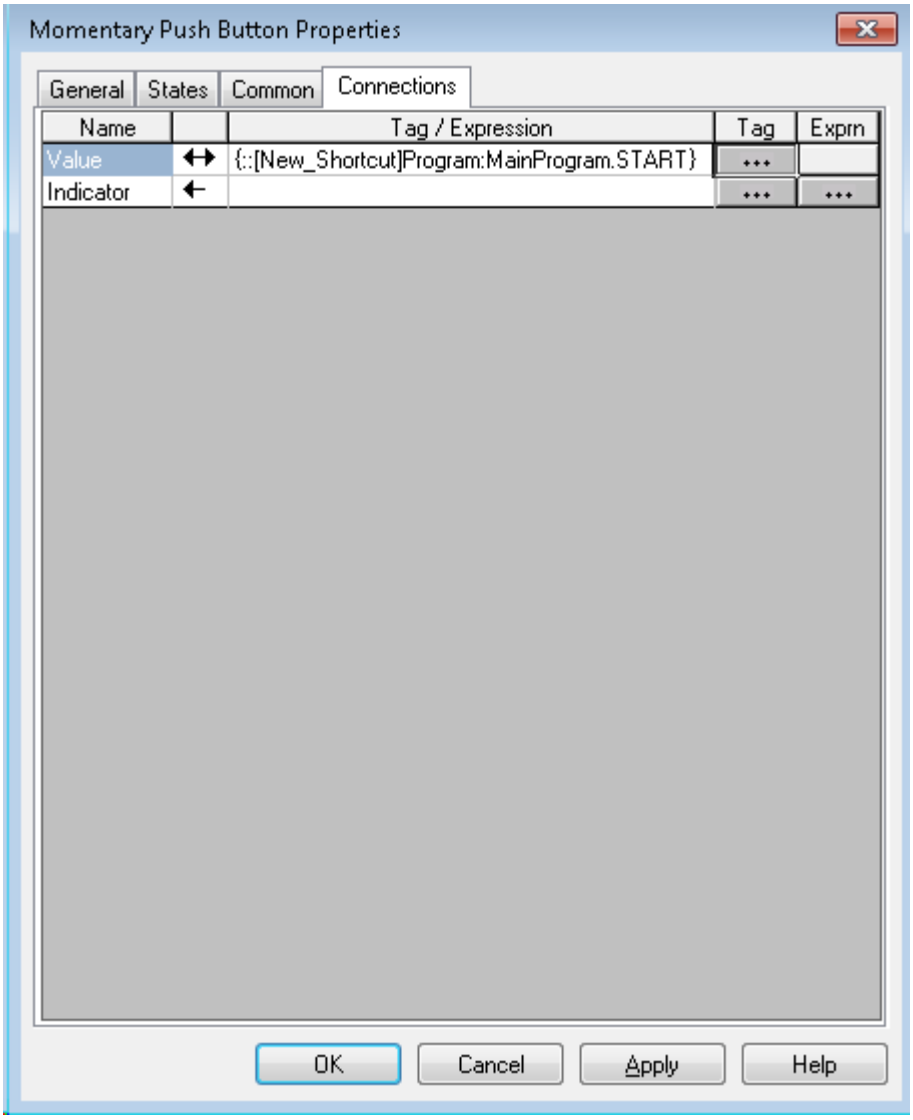
Click the **Refresh All Folders** button.



Go to New_Shortcut -> Online -> Program:MainProgram -> "START" tag. Click **OK** button to assign the "START" tag to the start button.

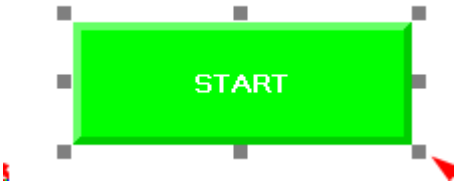


Click **OK** button to exit the momentary Push Button Properties.

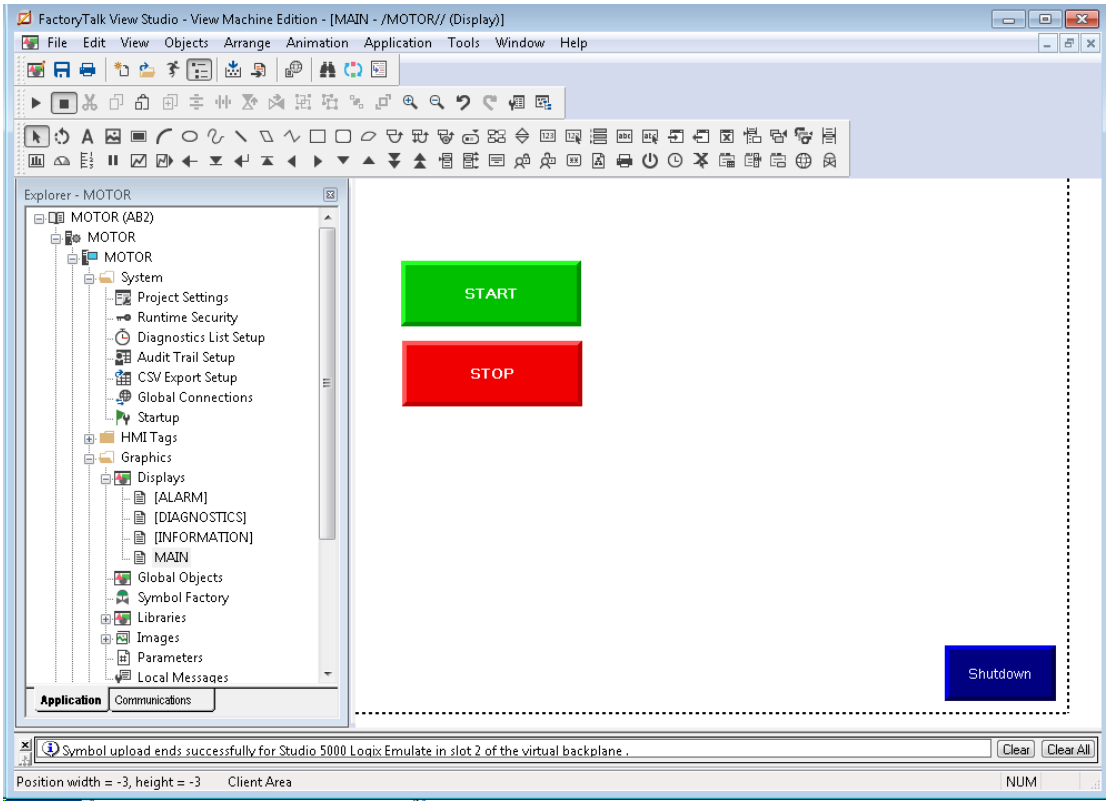


If you need to modify the setting, double click on the button will open the properties.

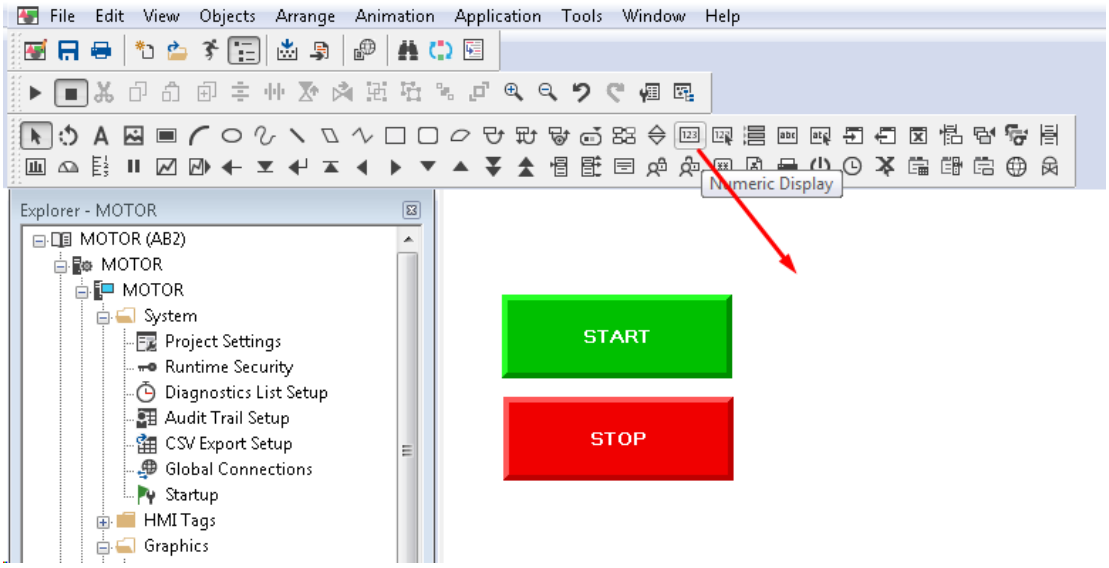
You can resize the button by select and drag on the edge of the button while holding left mouse button.



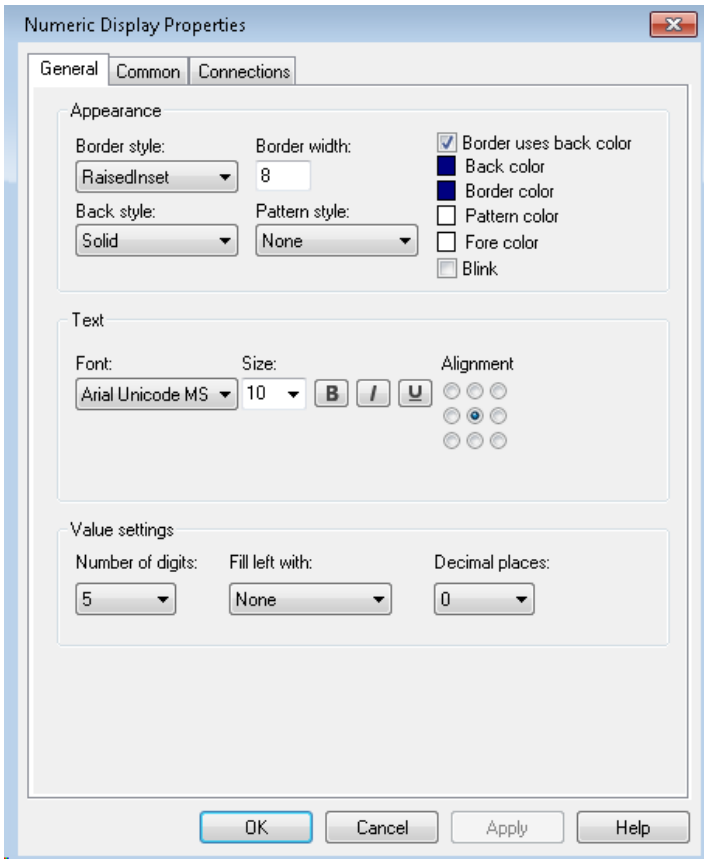
Repeat the steps above to create another button call stop. You can relocate the button by highlight it then drag the button while holding the left mouse button.

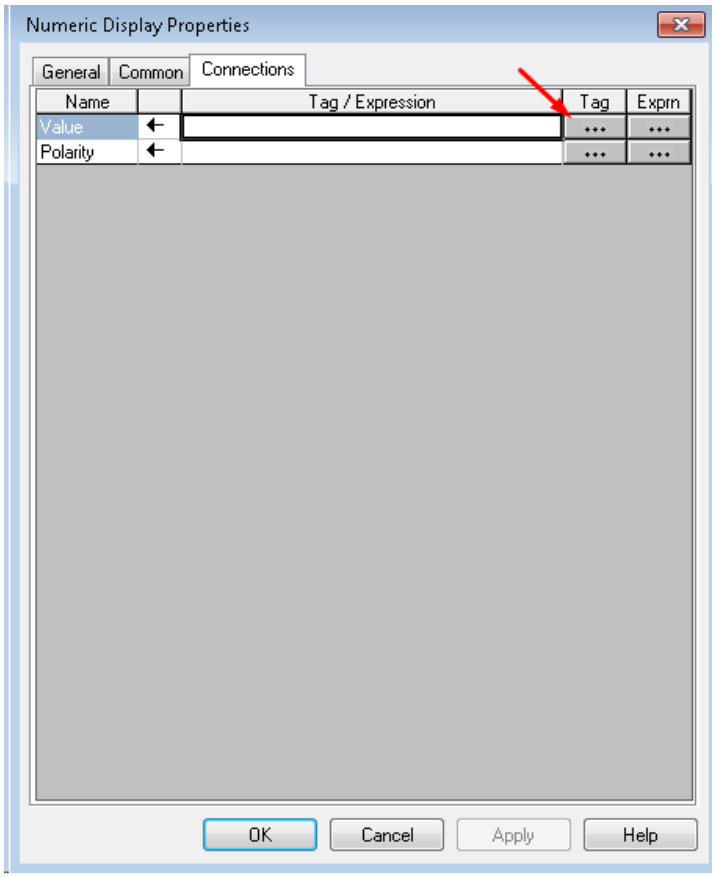


Draw a numerical display and connect the timer's accumulator to it.

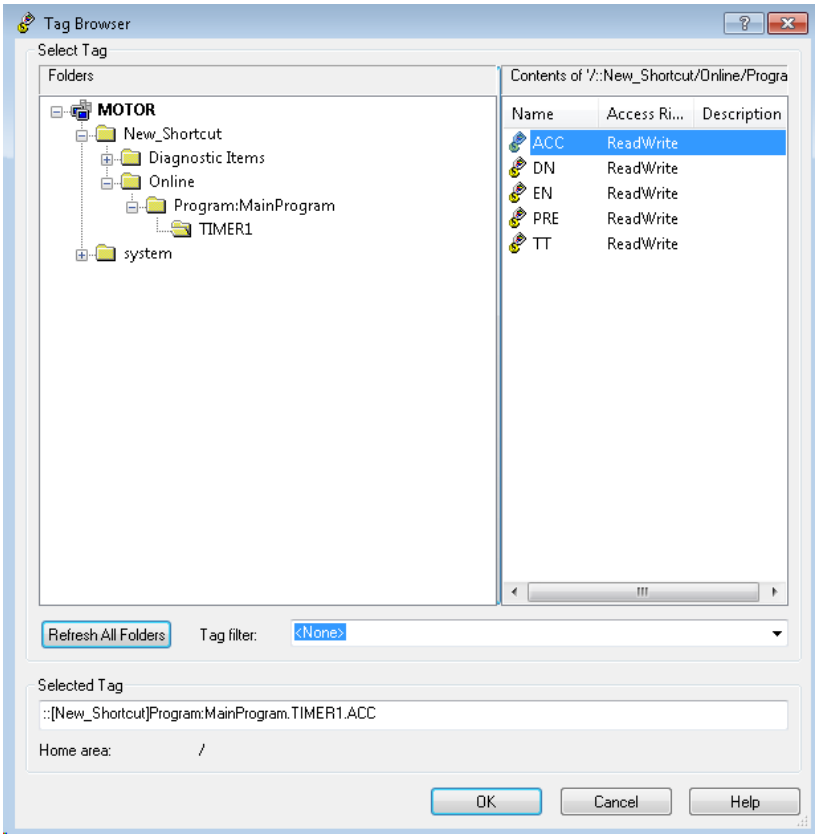


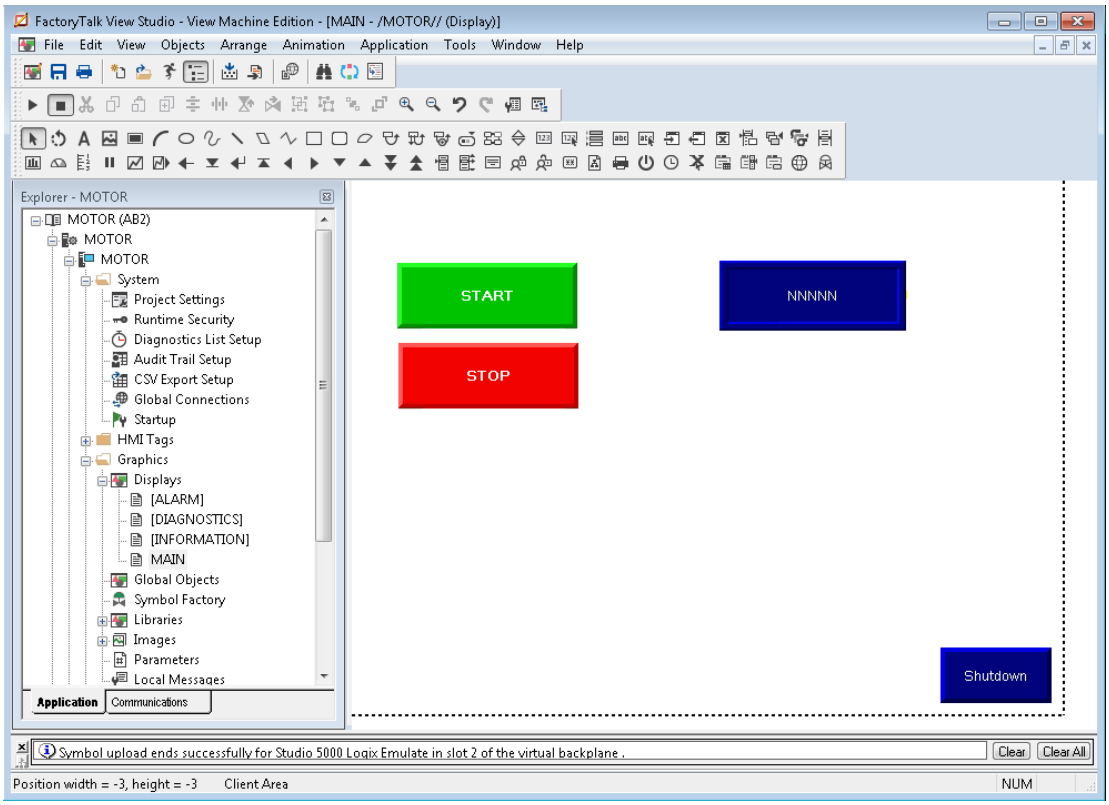
Change the number of digits if the value exceeds 5 digits. Go to connections tab then the button to enter the tag browser.


Numeric Display Properties
General
Common
Connections
Appearance
Border style:
RaisedInset
Border width:
8
Back style:
Solid
Pattern style:
None
Border uses back color
Back color
Border color
Pattern color
Fore color
Blink
Text
Font:
Arial Unicode MS
Size:
10
B
I
U
Alignment
Value settings
Number of digits:
5
Fill left with:
None
Decimal places:
0
OK
Cancel
Apply
Help

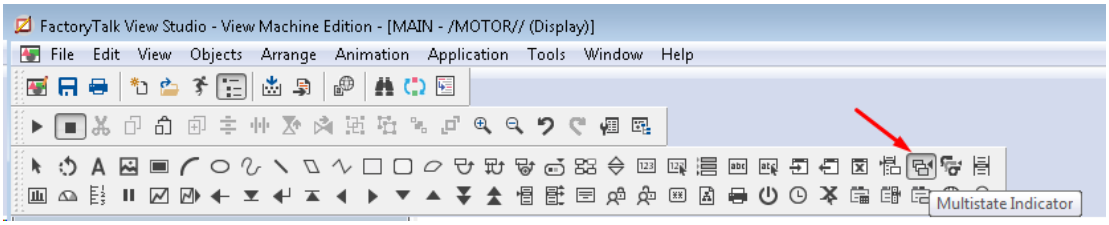

Numeric Display Properties
General
Common
Connections
Name
Tag / Expression
Tag
Exprn
Value
Polarity
OK
Cancel
Apply
Help

Select TIMER1.ACC the click **OK** button continue. click **OK** button on the next screen to exit the Numeric Display Properties.

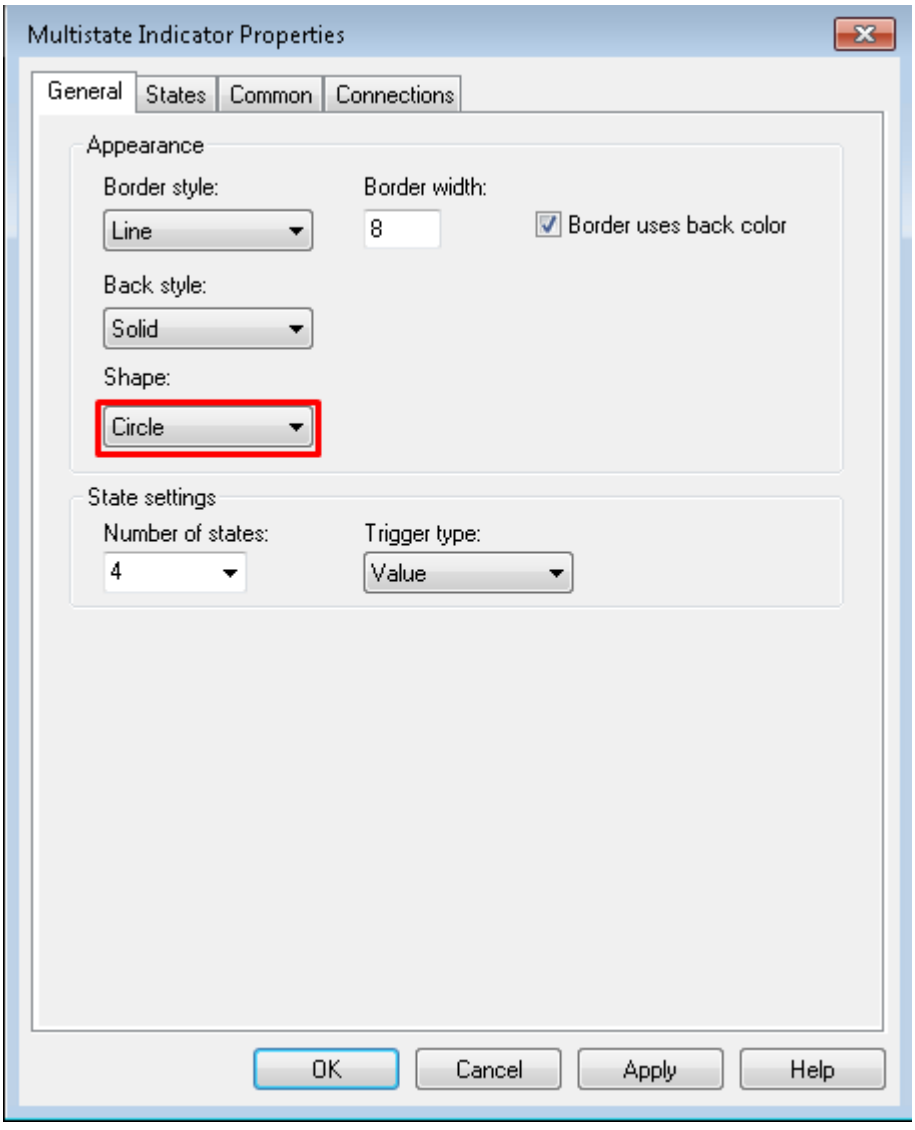




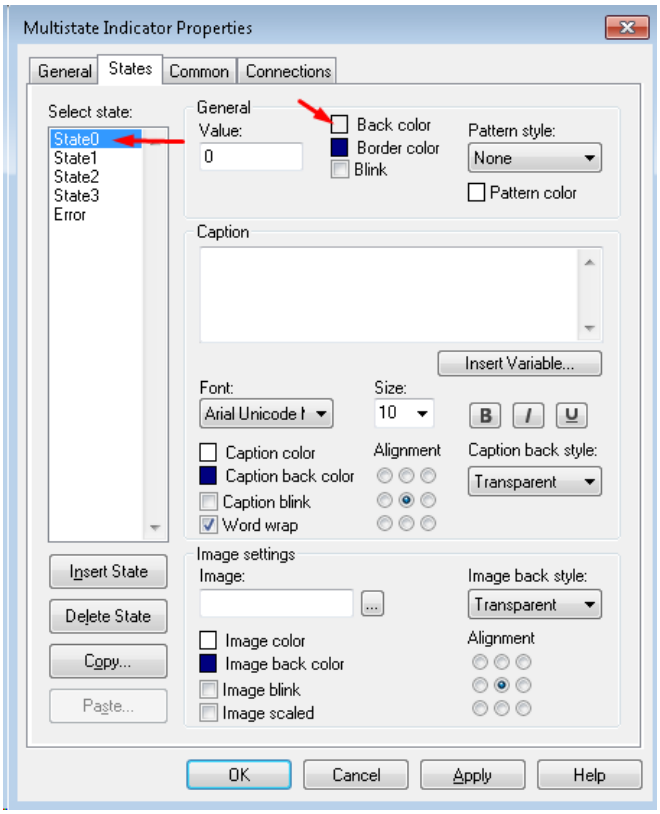
Select a multistate indicator from the menu bar.



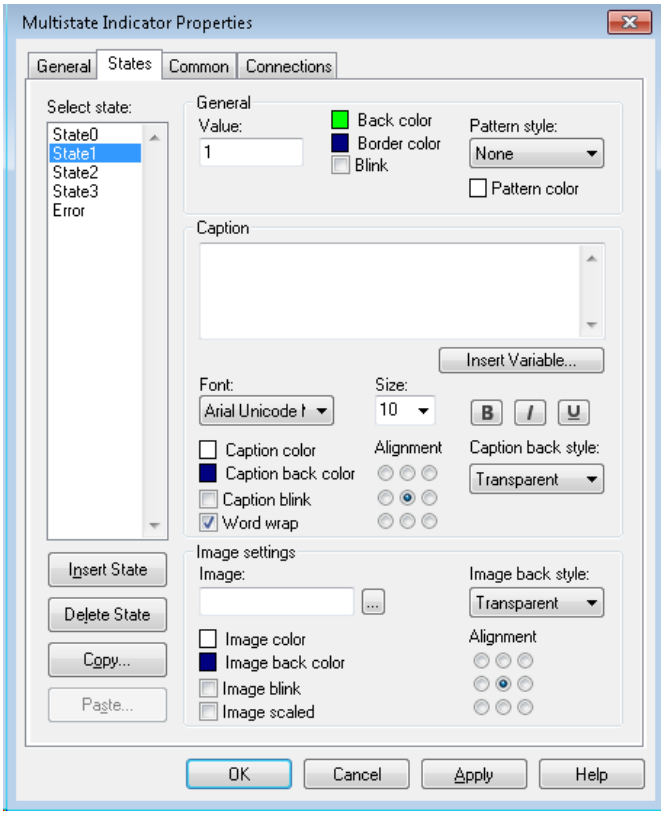
Change the Shape to circle then go to the States tab.



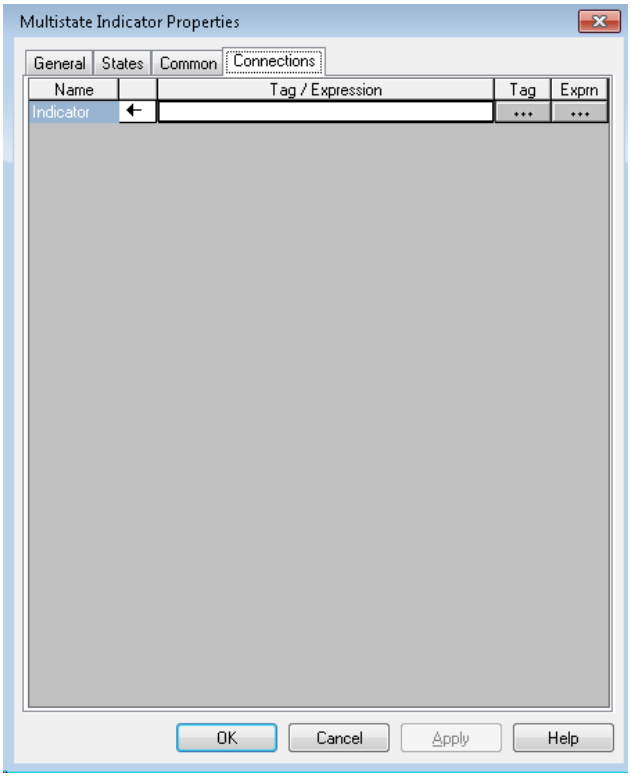
Change the background color for State0 to gray.



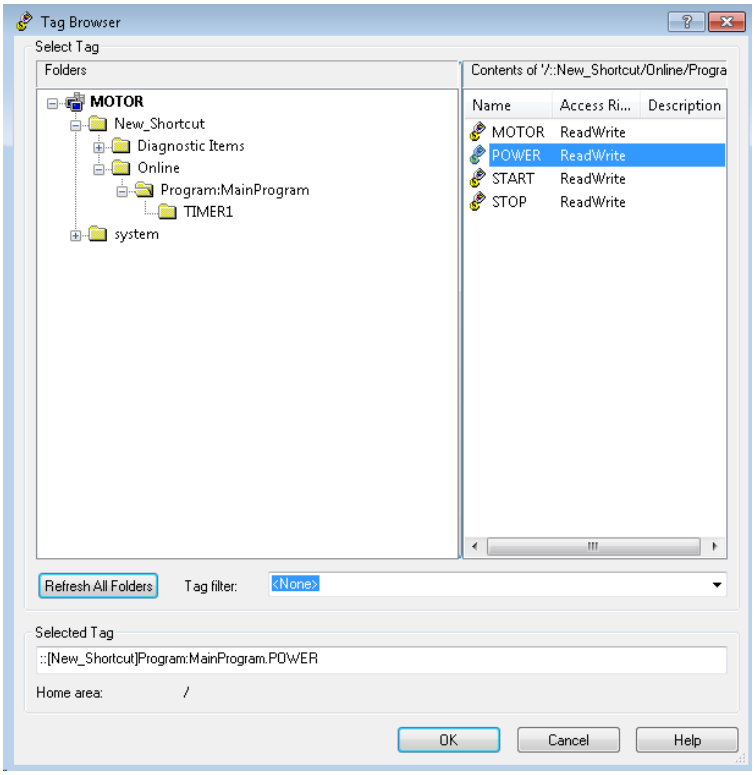
Change the color for State1 to **GREEN**.



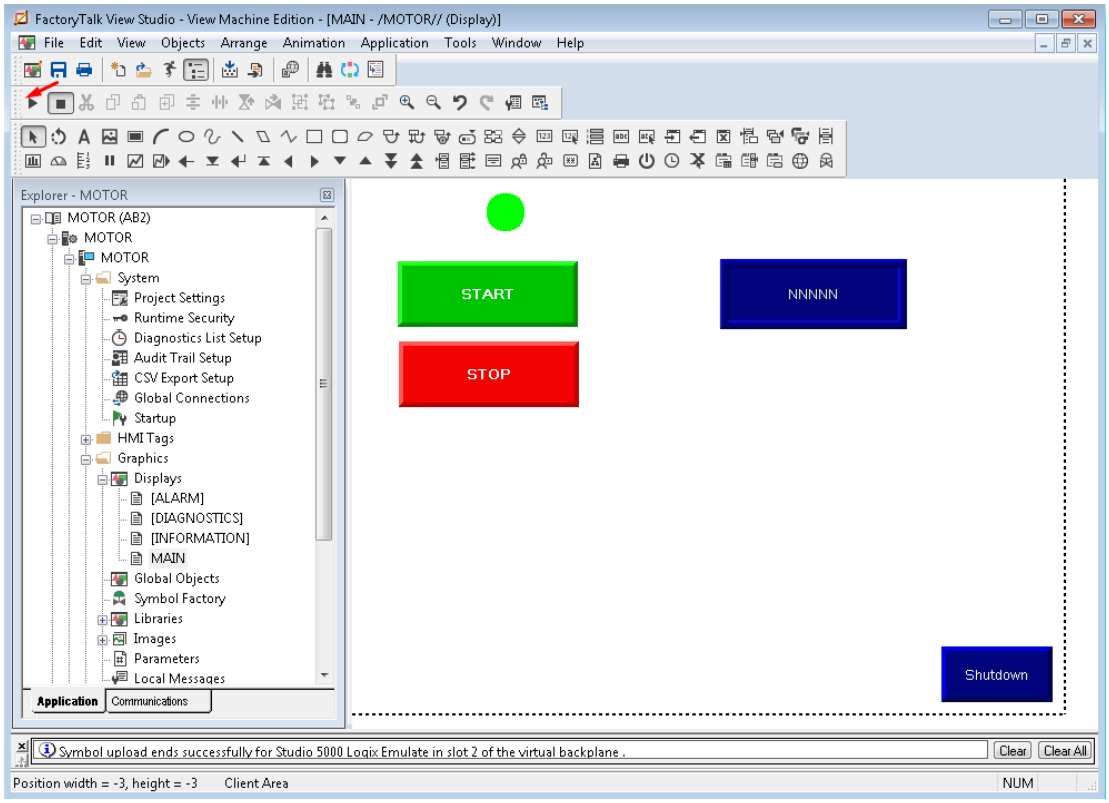
Go to the Connections tab and click on the tag browser button.

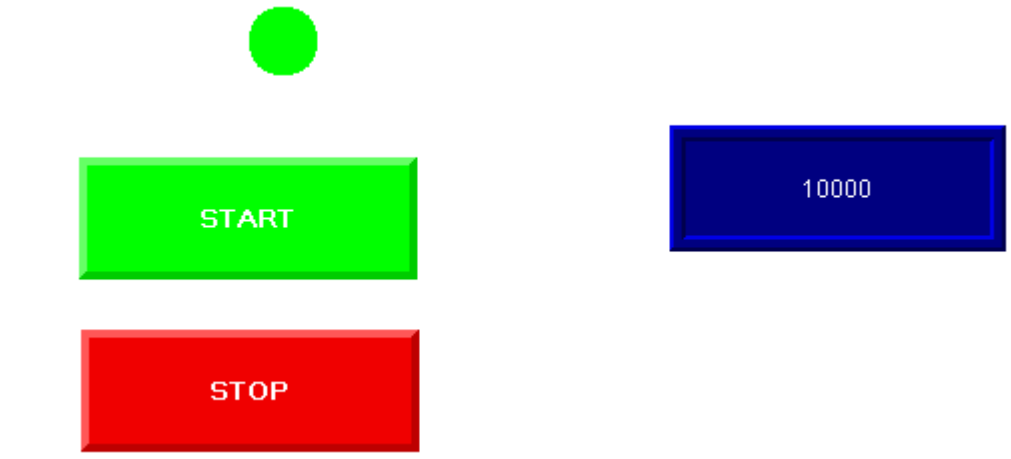


Select the "POWER" tag the click the **OK** button one to apply the change, **OK** button again on the next screen to exit.

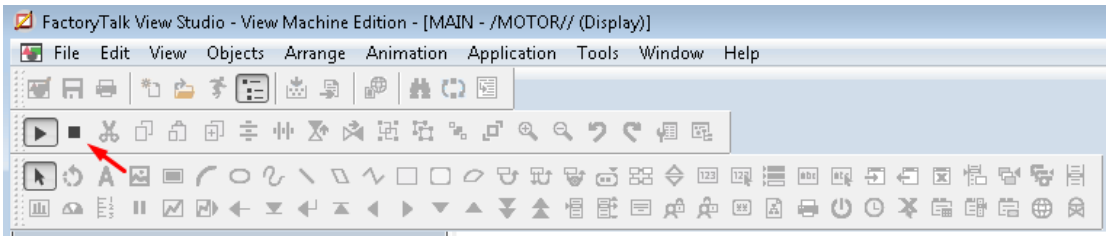


Click this button to test the display. Click the Start button in green color to enable the "POWER".

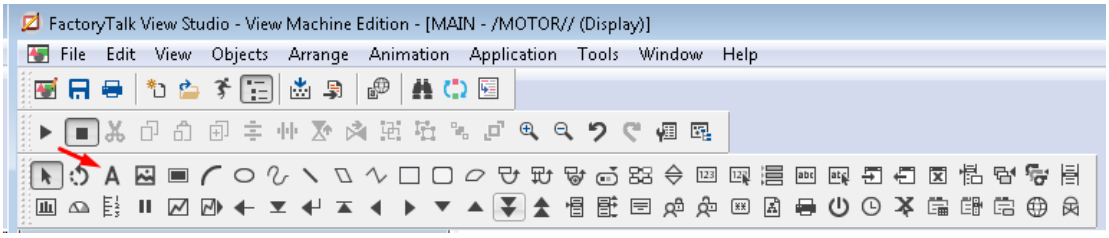




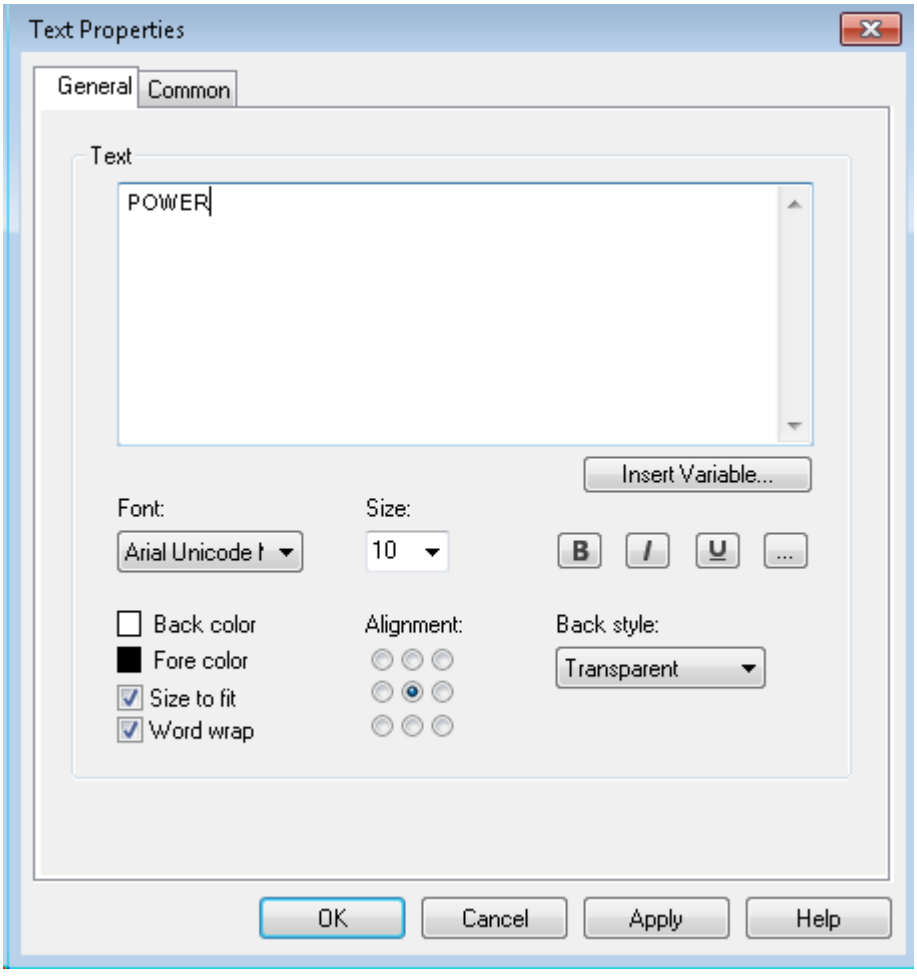
To modify the display, click on the edit display button.

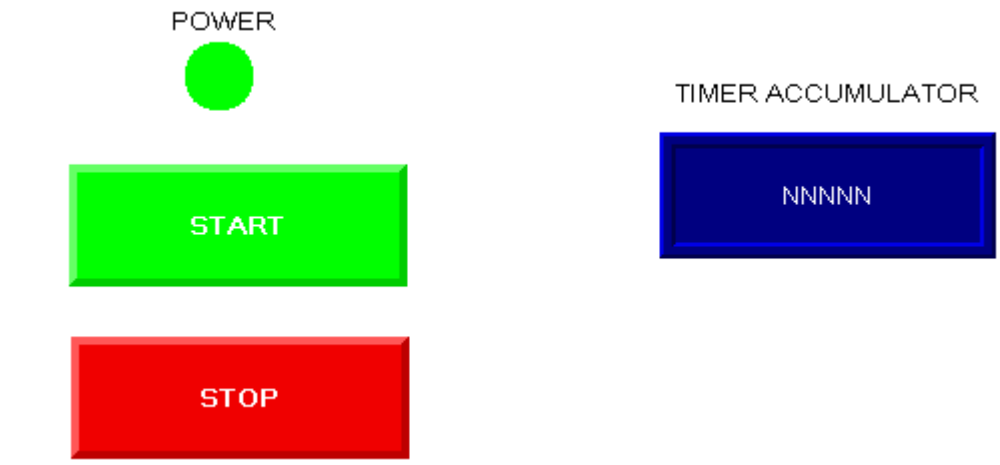


To add text label, click on the A icon and draw a text box on the screen.



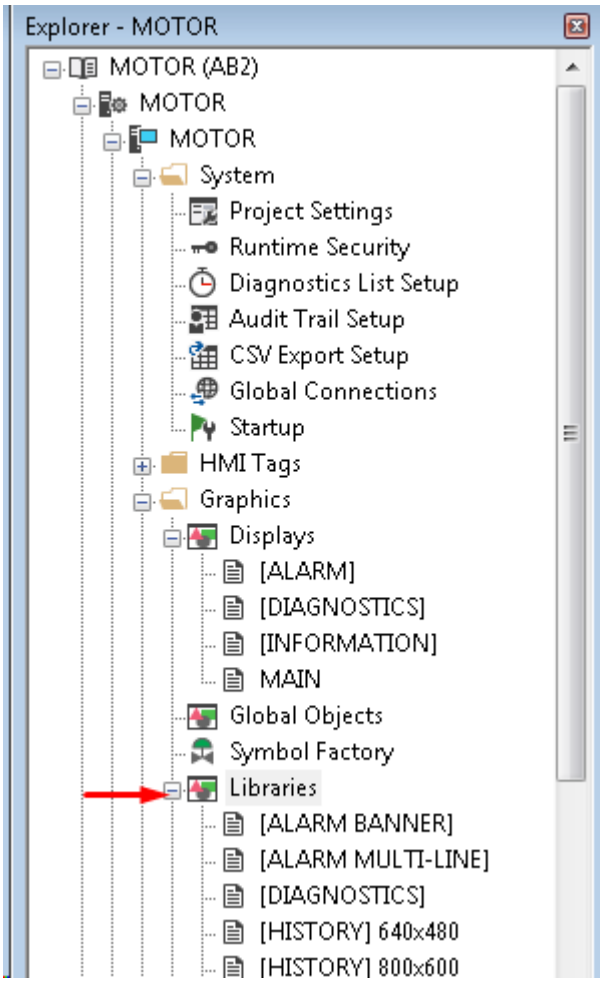
Enter text in the text box, change color, font, size as required.

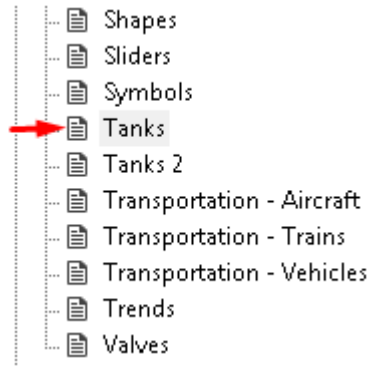



POWER
TIMER ACCUMULATOR
NNNNN
START
STOP

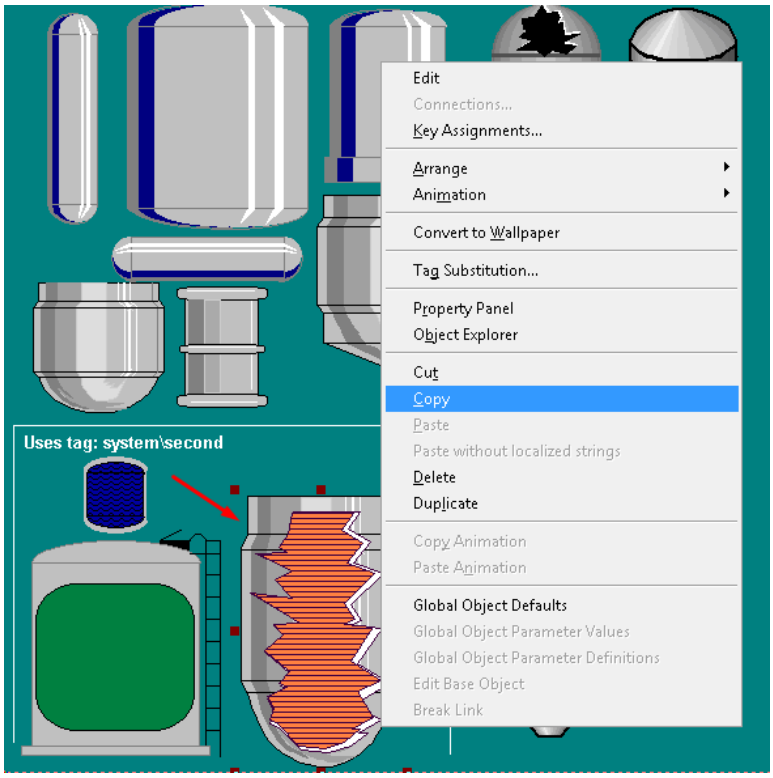
3. EXERCISE 3 - LIBRARIES

Continue from exercise 2, go to the Libraries folder and click on the + sign to view template categories. Scroll down and double click on the Tank sub folder.

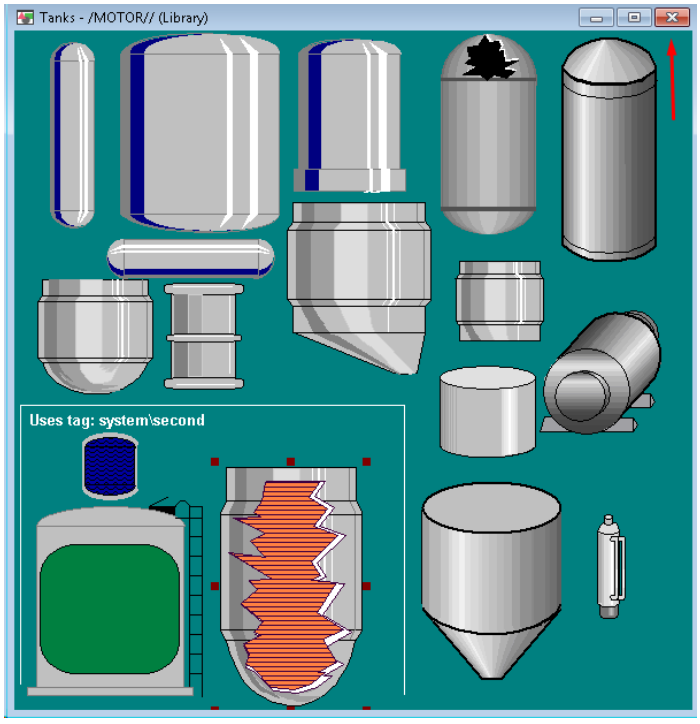




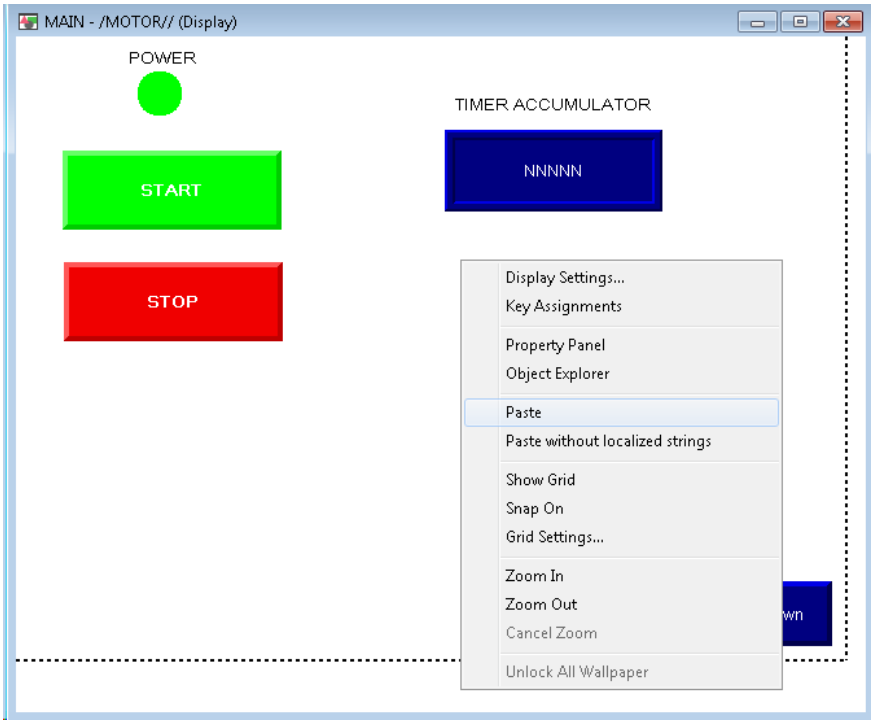
Highlight this template then click the right mouse button, select Copy. Or use keyboard shortcut CTRL + C.



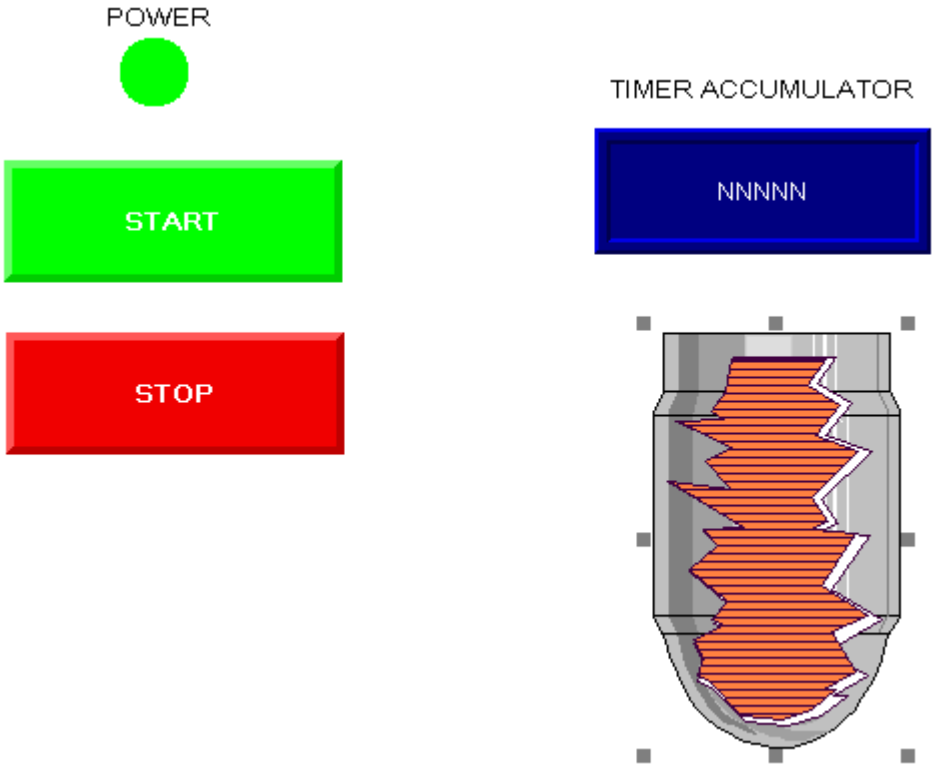
Click on the **X** on the upper right corner to close the Tanks library.



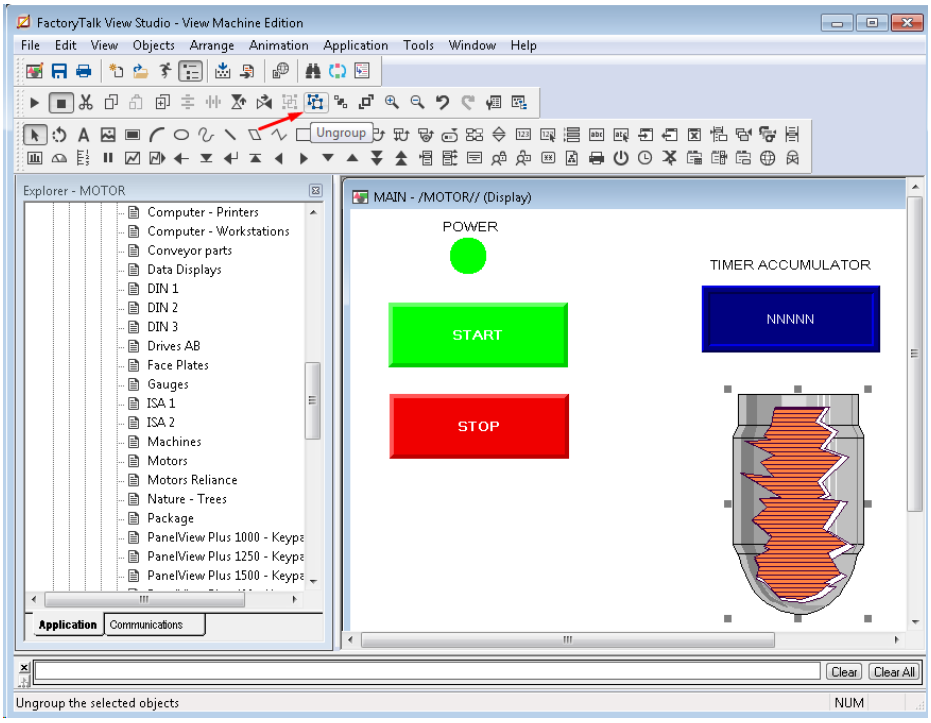
Right click on the Main screen and select Paste. Or keyboard shortcut CTRL + V.



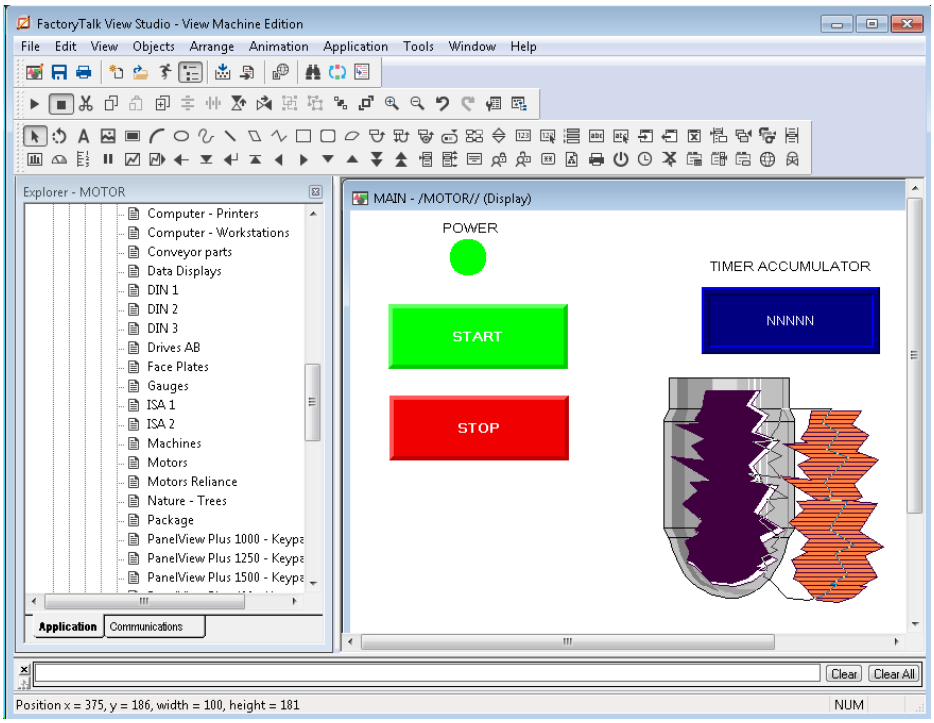
To move the Tank, select the object and hold the left mouse button.



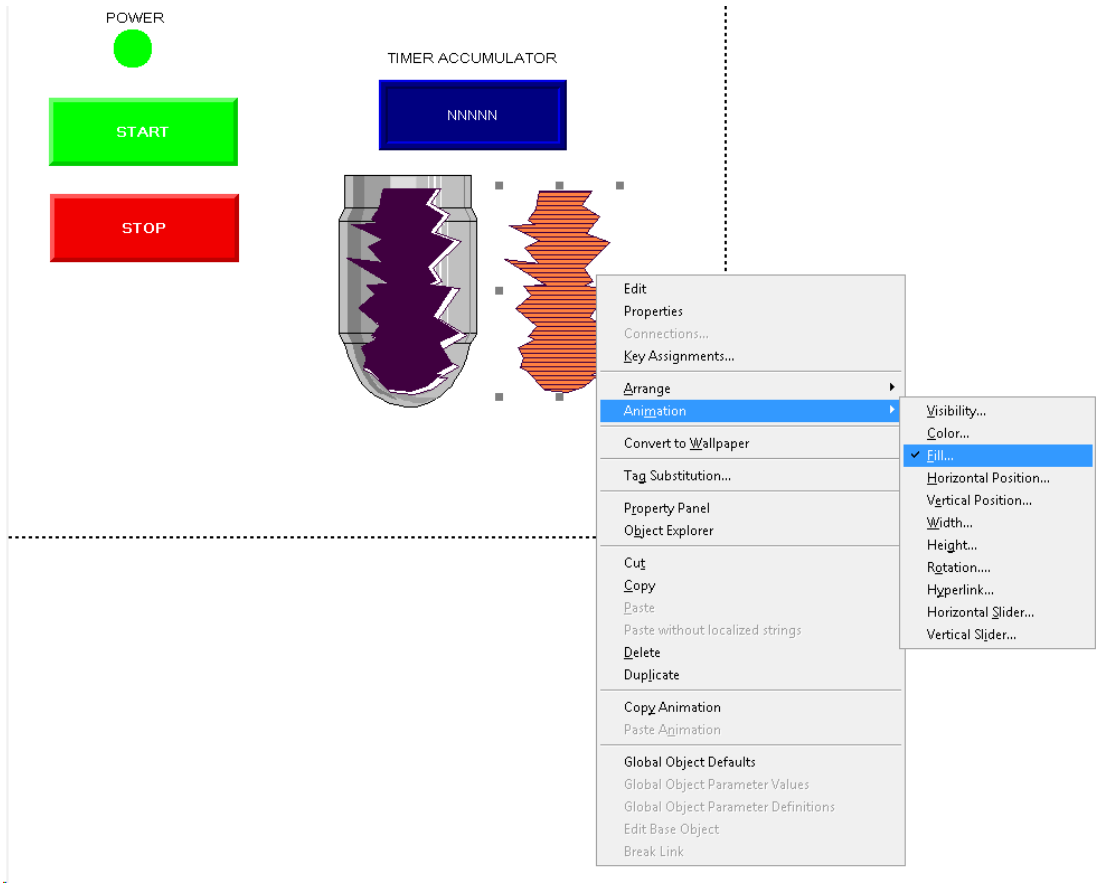
This template is grouped with multiple layers and can not be animated as a whole. Click on the Ungroup icon to separate the layers.



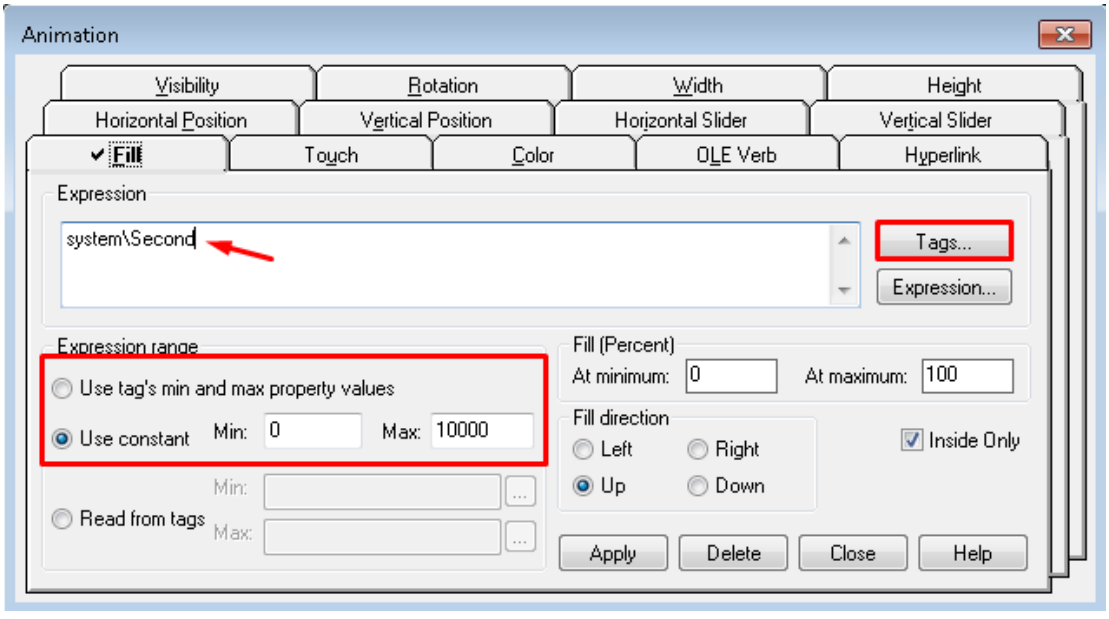
Click on the inner layer (orange colour), and move it to a side.

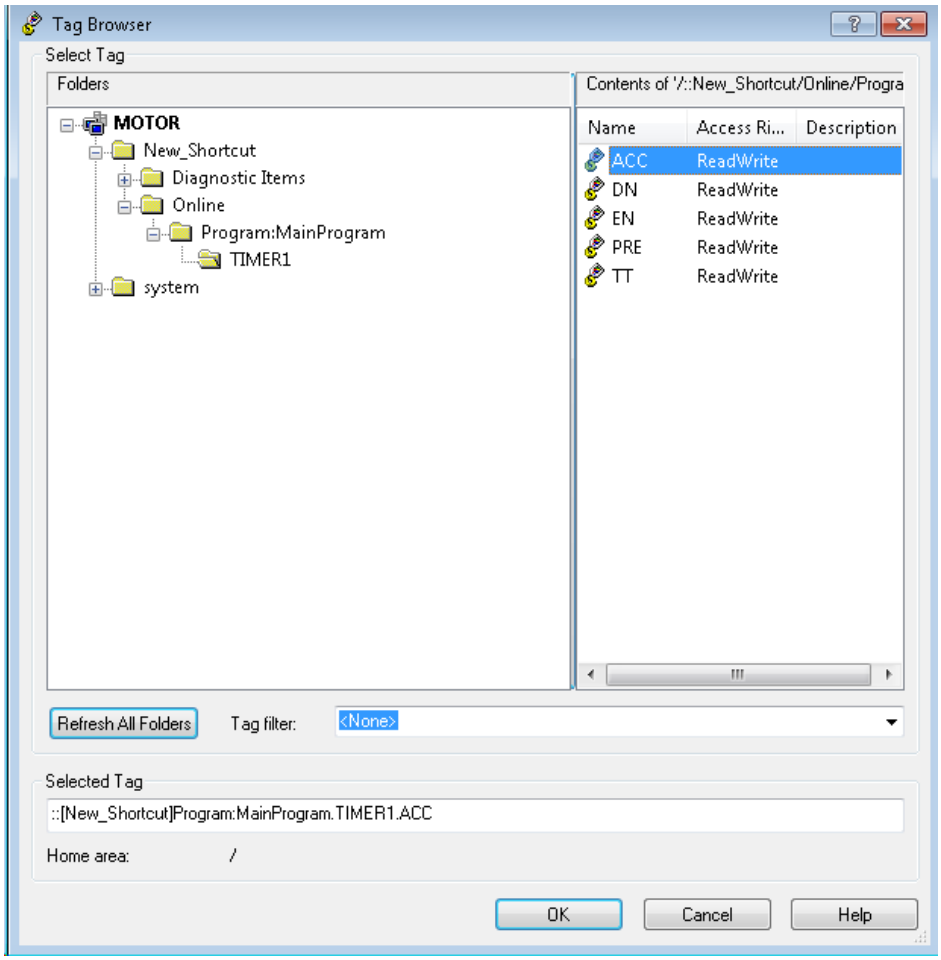


Right click on the orange layer, go to Animation -> Fill...

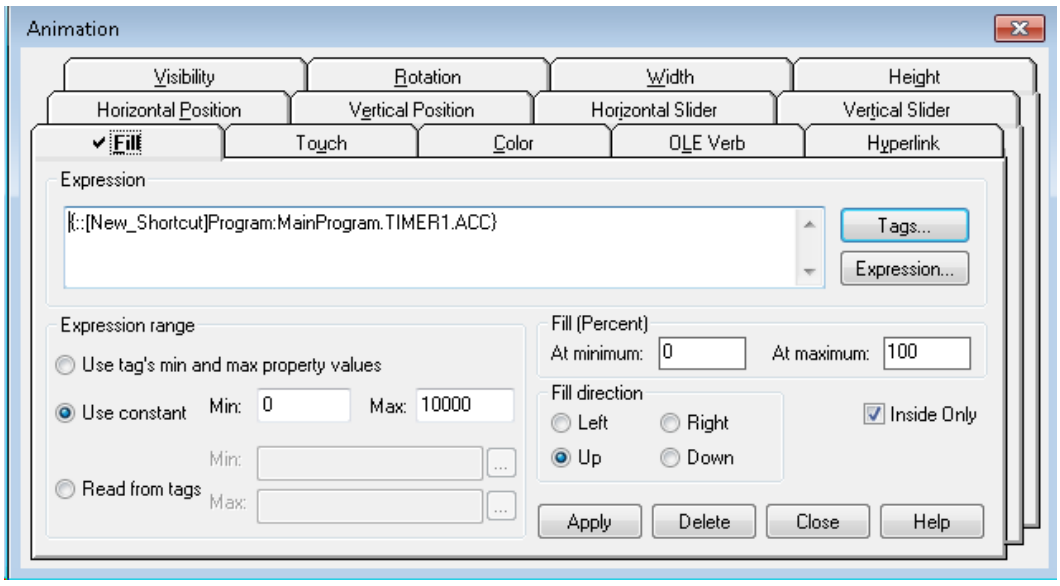


Delete the default value system\Second. Click the Tags... button and select the timer's accumulator.

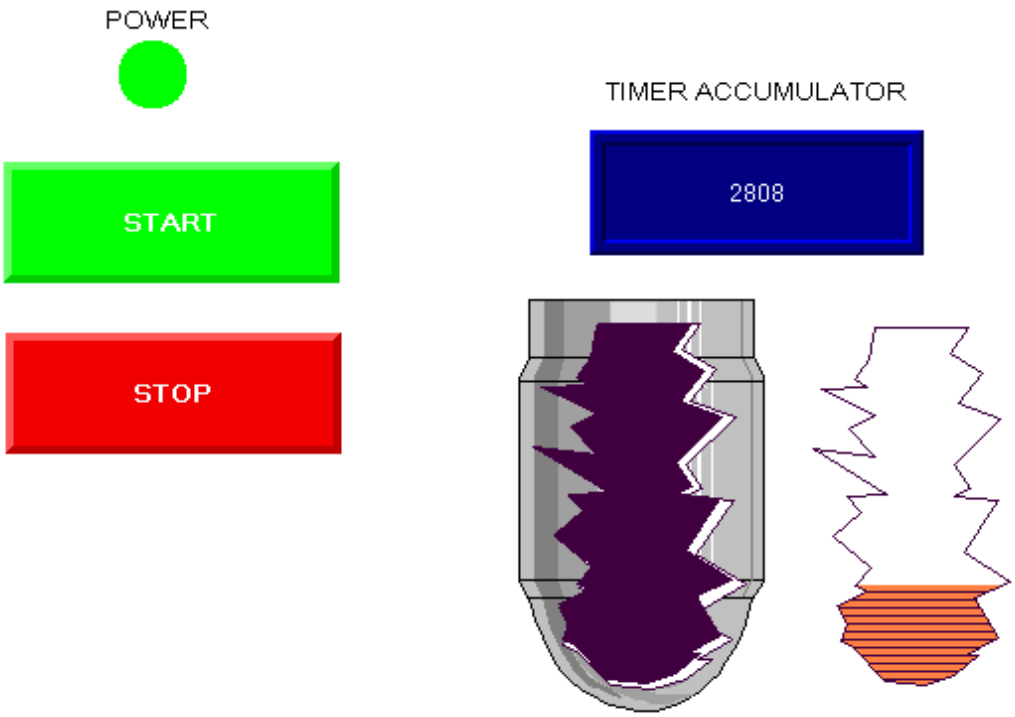




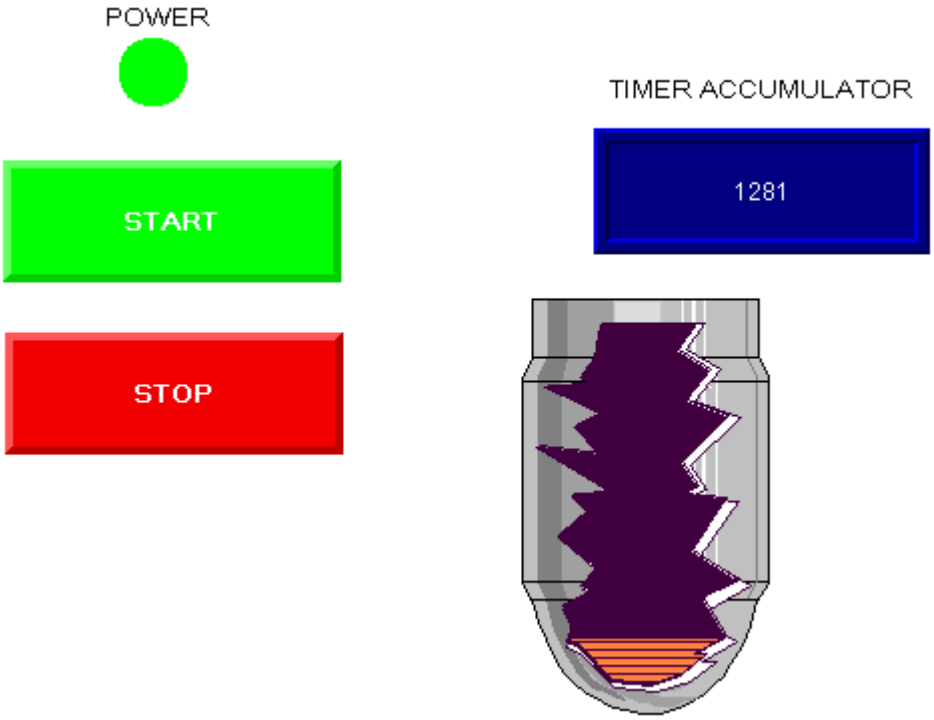
Change the expression range between 0 to 10000. Click **Apply** button to save the change then **Close** button to exit.



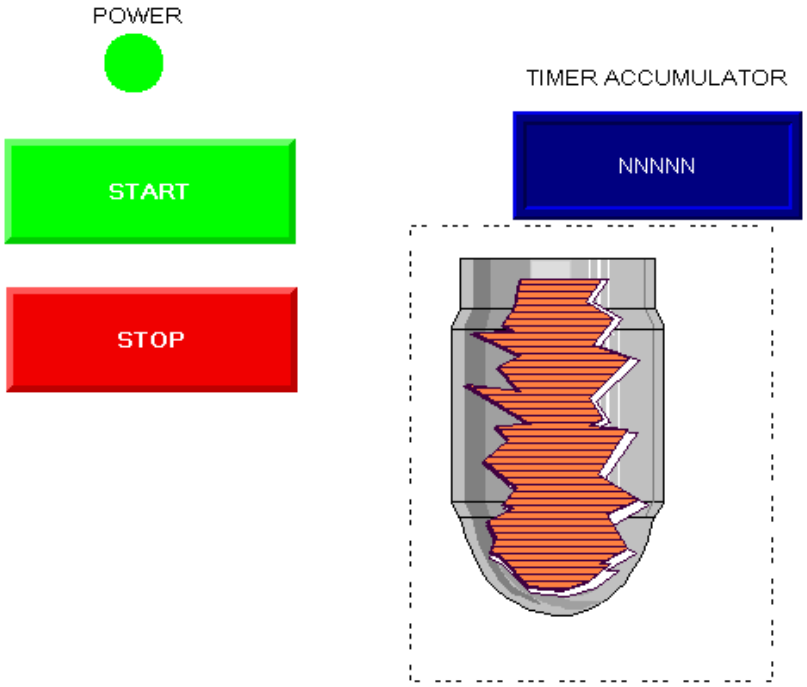
Test the display to make sure the tank is filling correctly.



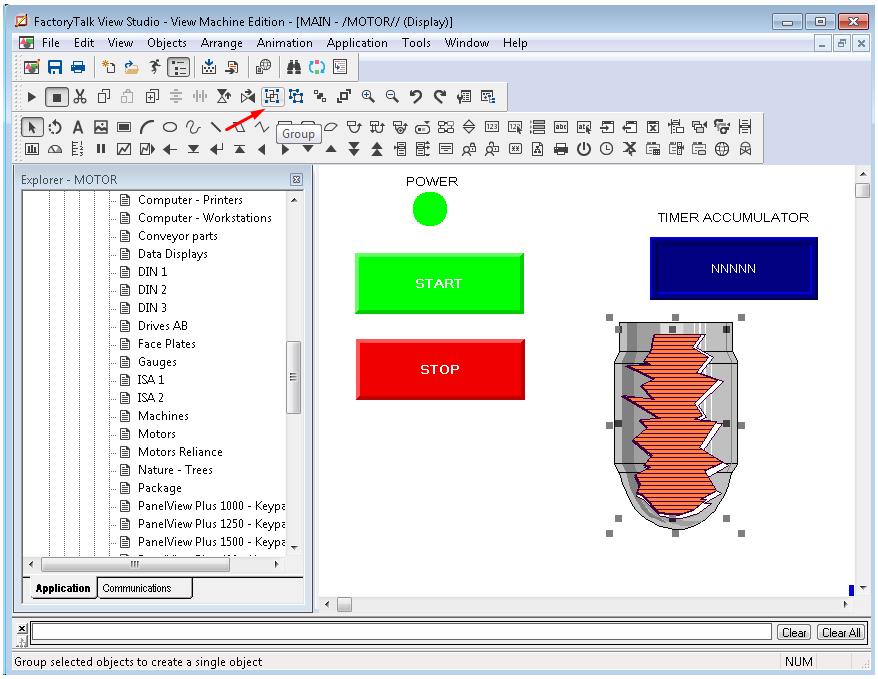
Go back to edit display and move the orange object into the tank.



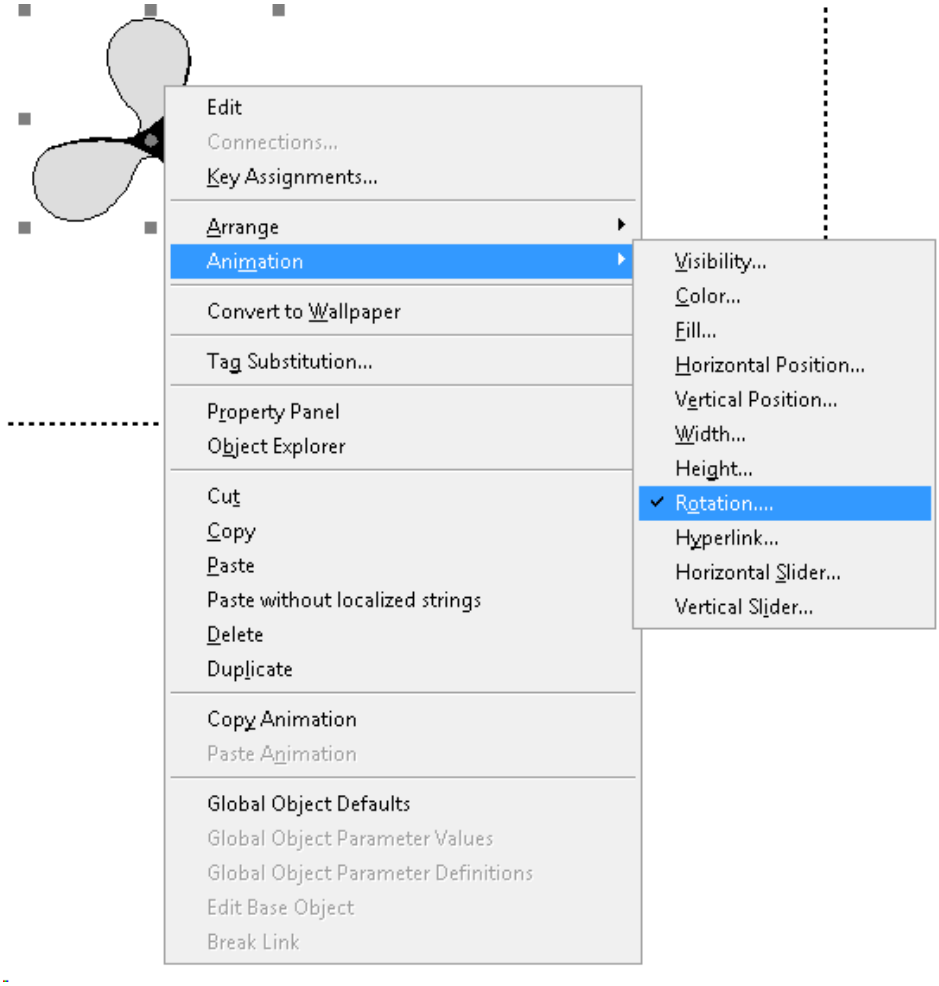
You can regroup the tank by select all the layers and click the Group icon on the menu bar. Regroup allow the tank to re-size as one object.



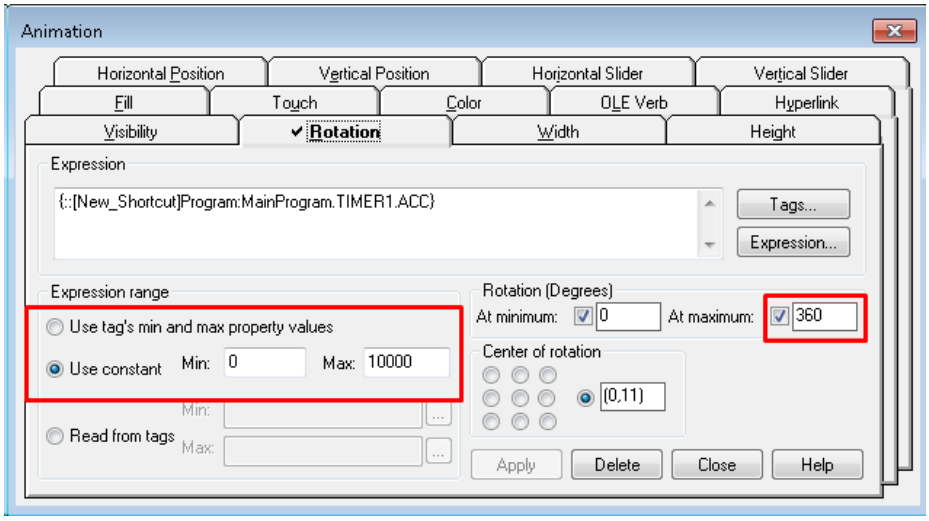
-

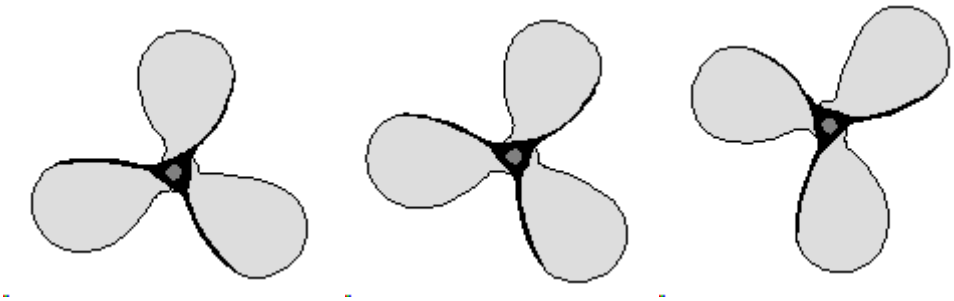


For rotation object like a fan. Go to the motor category, copy and paste a fan to the main screen. Right click on the fan and select Animation -> Rotation



Change the expression range, and Rotation (Degree) to 360. Select the timer's accumulator.

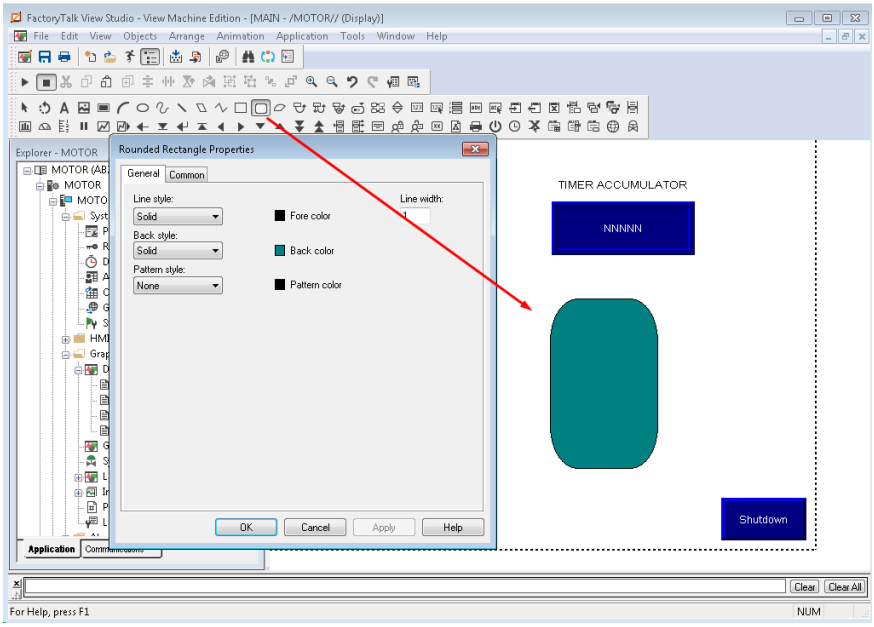




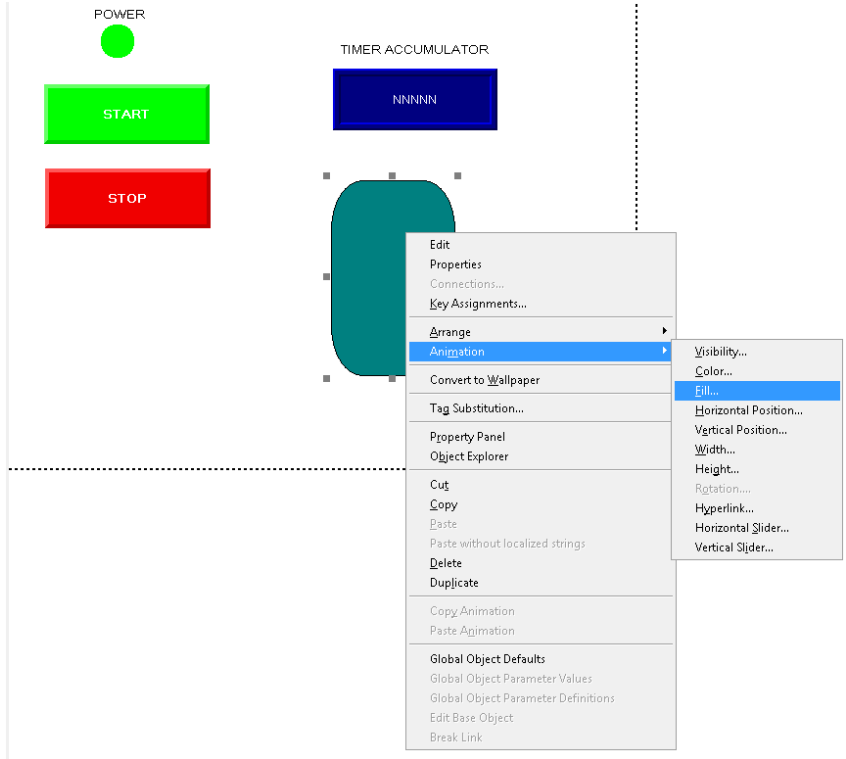
4. EXERCISE 4 - ANIMATIONS

FILL ANIMATION

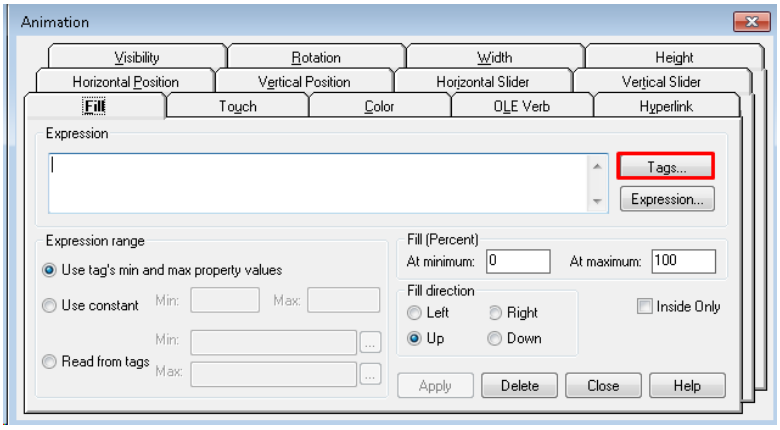
Select a Rounded Rectangle from the menu bar. Click **OK** button to exit.



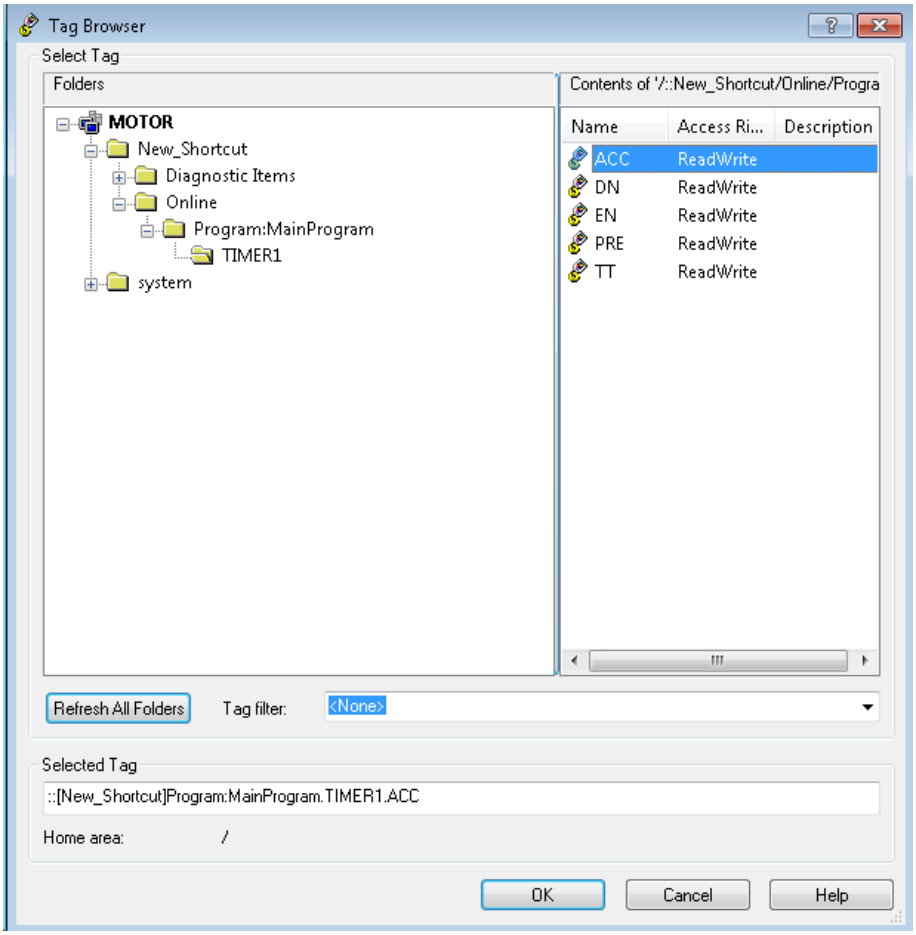
right click on the Rounded Object. Go to Animation -> Fill...



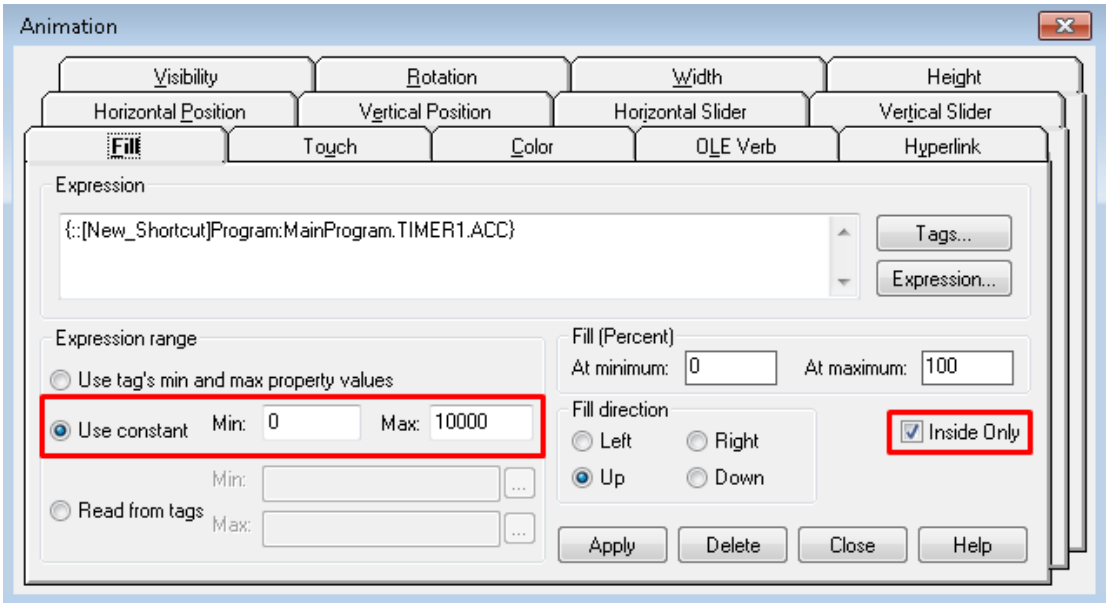
Click on the Tags... button to open the tag browser.



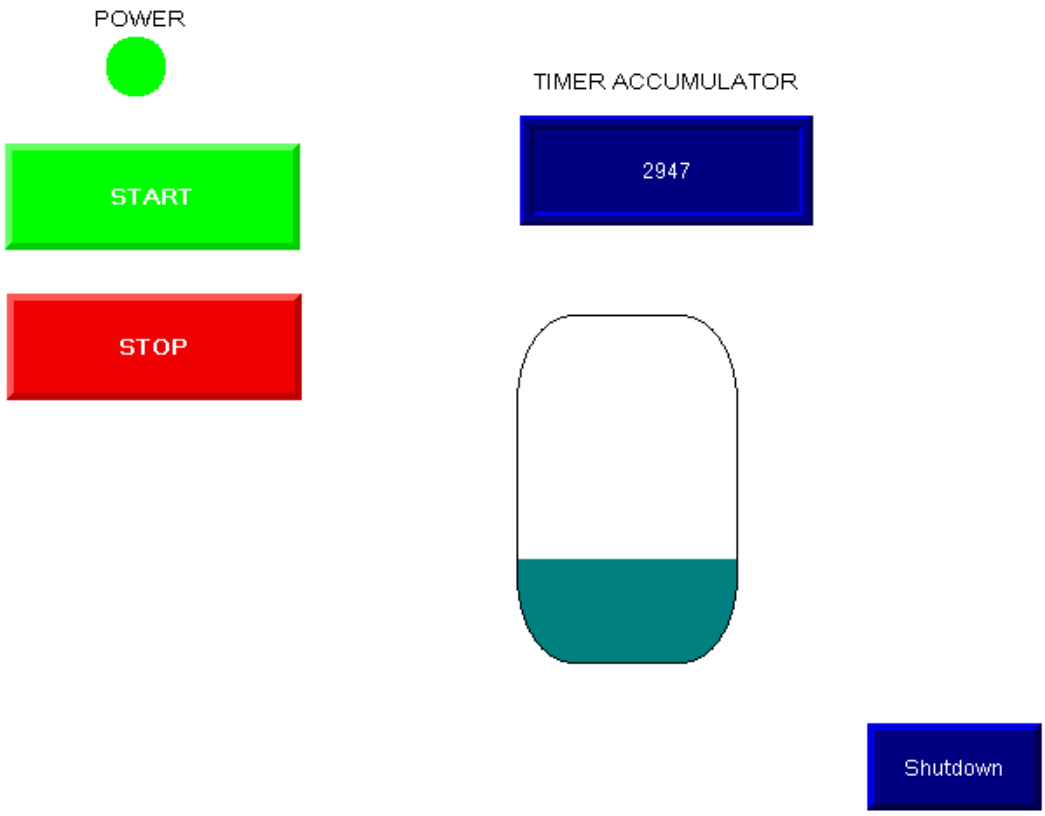
Select Timer's accumulator, click **OK** button to continue.



Change the expression range to user constant. Set the minimum value 0 and maximum to 10000 (timer's preset value). Select the check box Inside Only. Click **Apply** then **Close** button to exit.



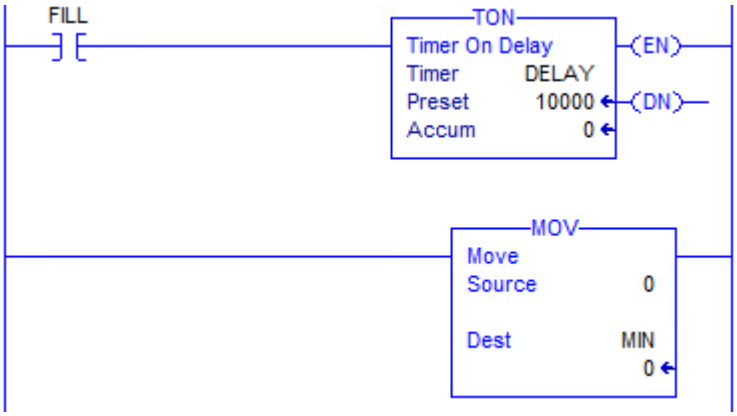
Test the fill animation.



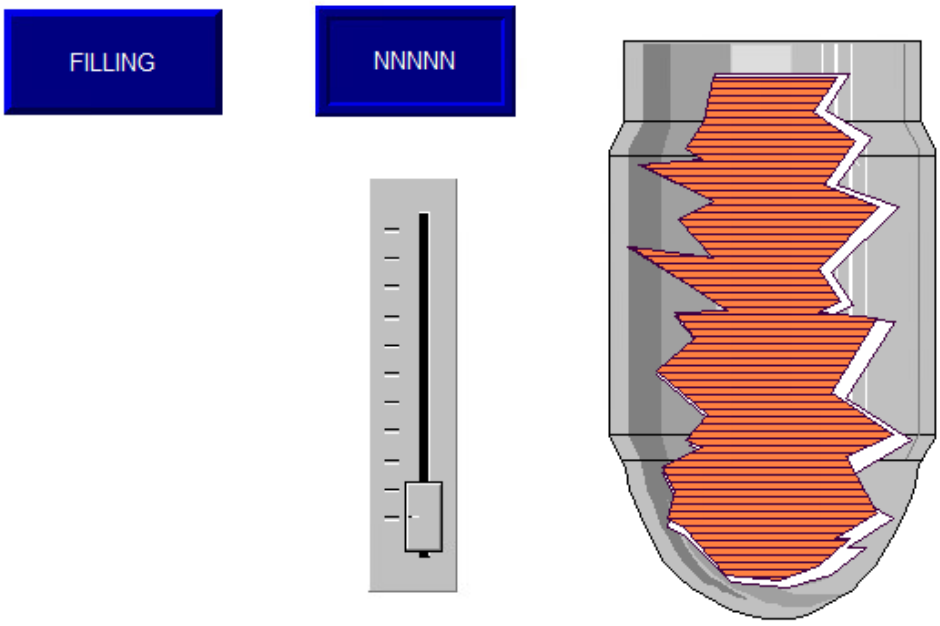
5. EXERCISE 5 - ANIMATION WITH MAX AND MIN RANGE

There are two methods of assign minimum and maximum values for an animated object in FactoryTalk View Studio. This lab will cover both methods.

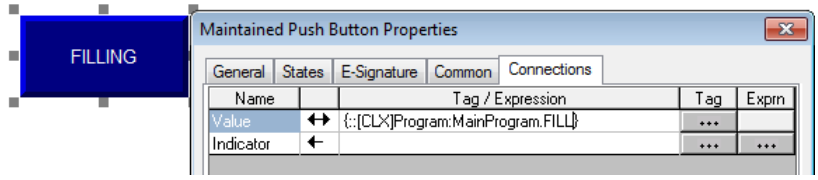
1. Create a simple logic with “START” input and a timer “DELAY”. Use a MOV instruction, assign value 0 for the source, create a new double integer tag name “MIN” as the destination. Download the program then switch to Run Mode.

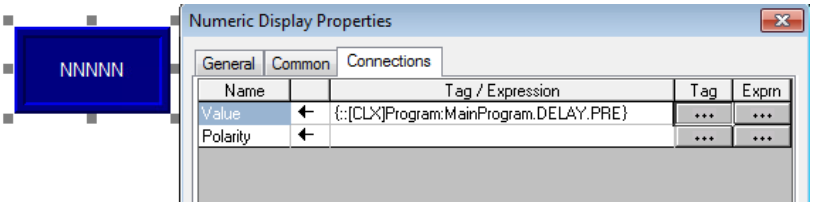


2. Open FactoryTalk View Studio Machine Edition. Setup RSLinx Enterprise communication by create a shortcut to the controller. Create a maintained button, a numeric display, a slider, and a tank. Sider and Tank animation object can be found in the Libraries sub-folder.

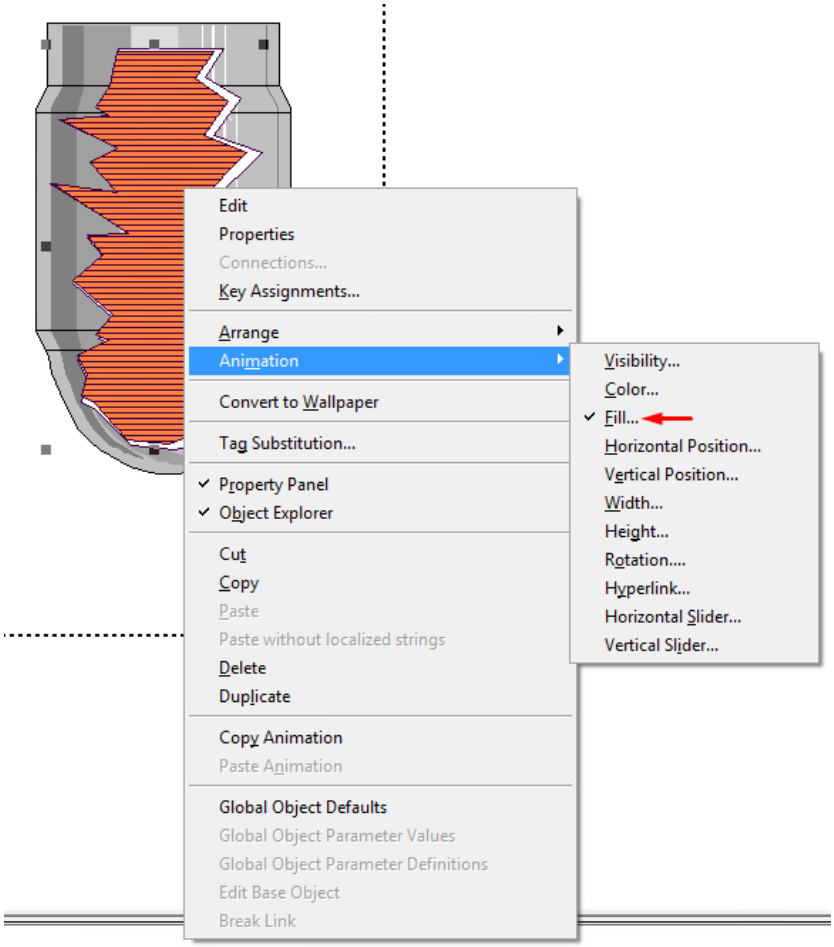


3. Link the Fill button to the “FILL” tag, link the numeric display to the timer “DELAY” preset.

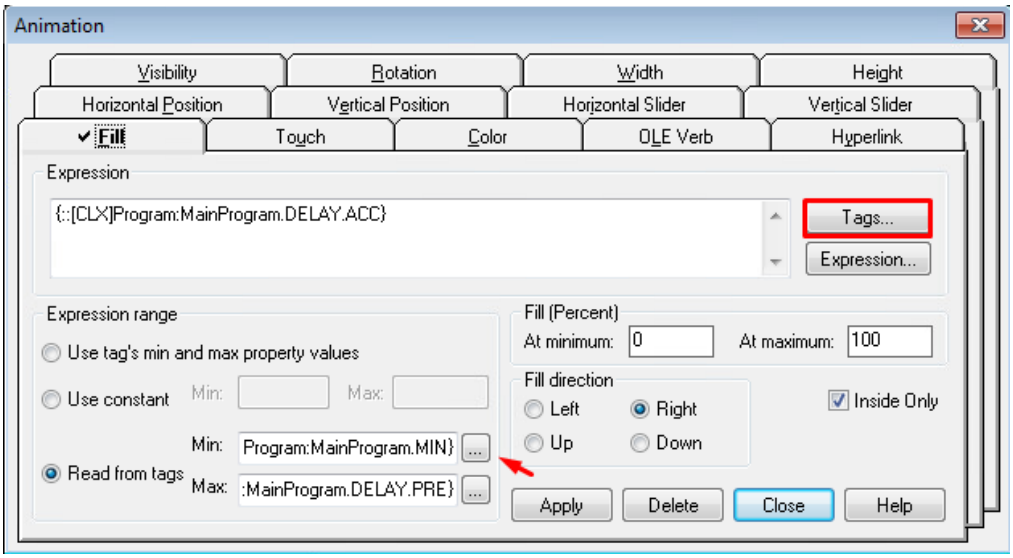




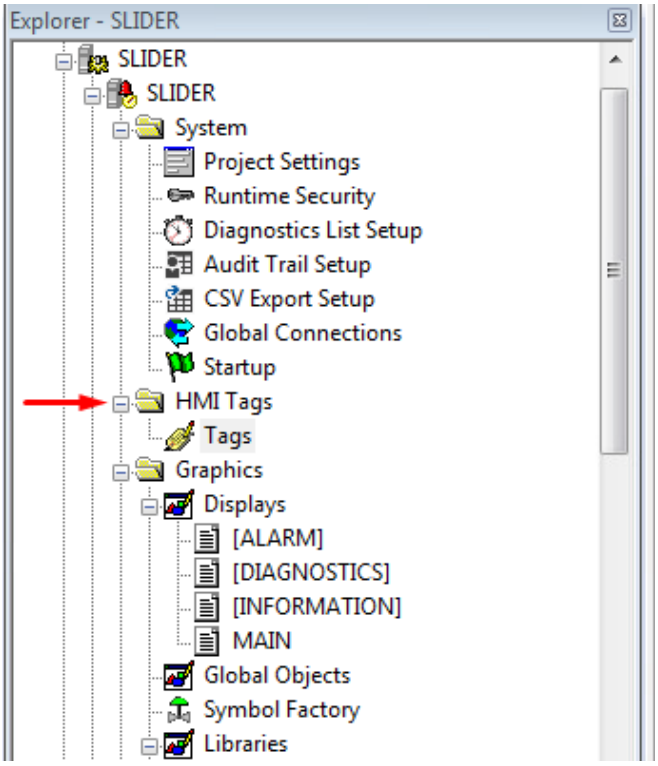
4. Right click on the orange color of the object, go to Animation -> Fill.

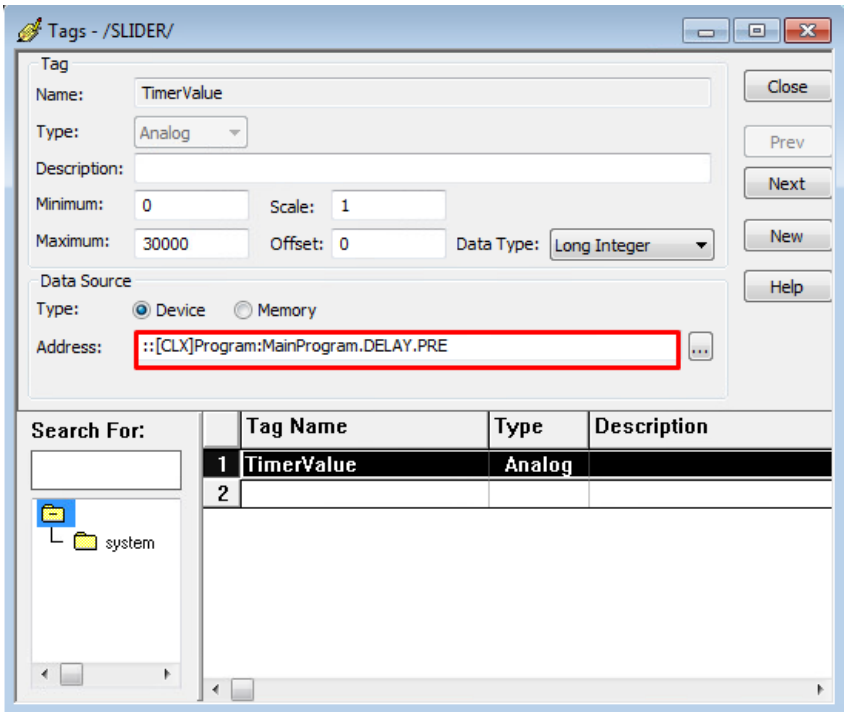


5. Assign timer “DELAY” accumulator for the expression. Select Read from tags, click on the tag browser button and select “MIN” tag for the Min value. Use timer “DELAY” preset for the Max value.

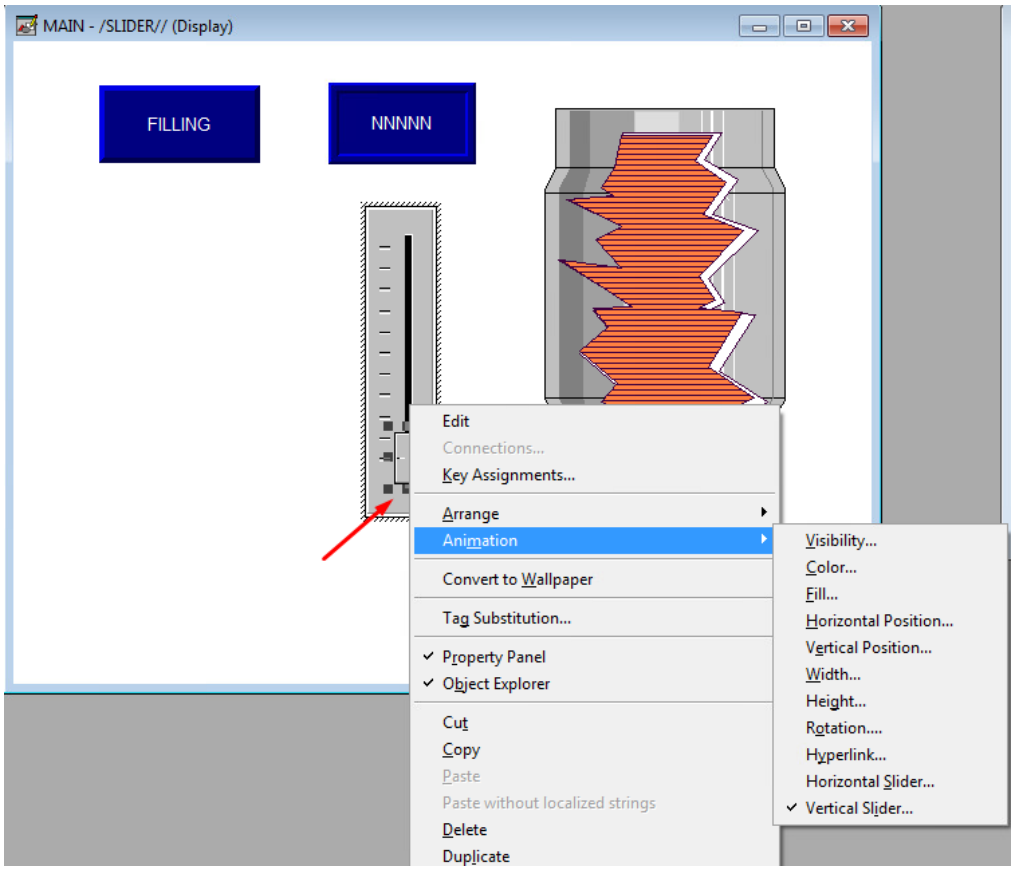


6. Go to HMI Tags -> Tags, double click on the Tags to open the Tags window. Create an HMI tag name "TimeValue", change type to **Analog**, enter **0** for Minimum and **30000** for Maximum, select timer "DELAY" preset in the address window. Close the window and return to the display Main.

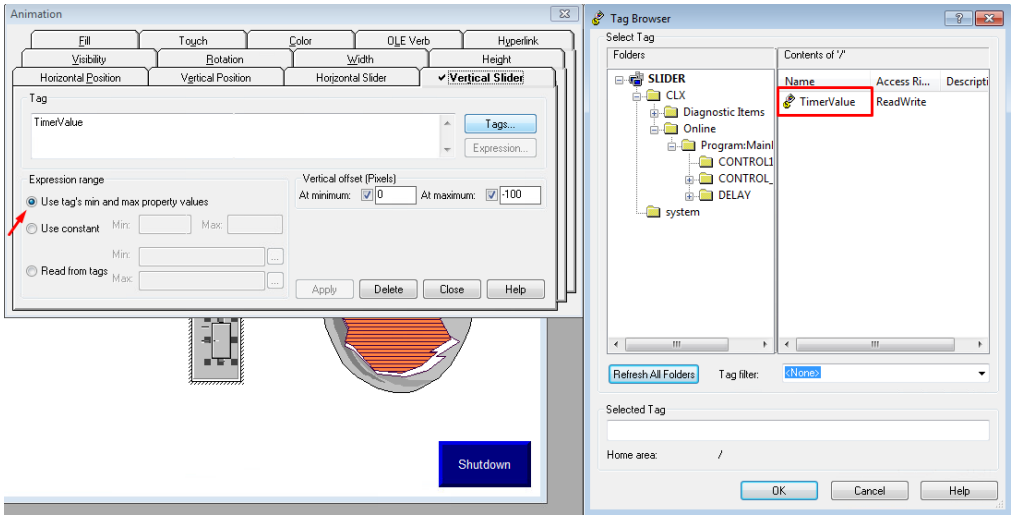




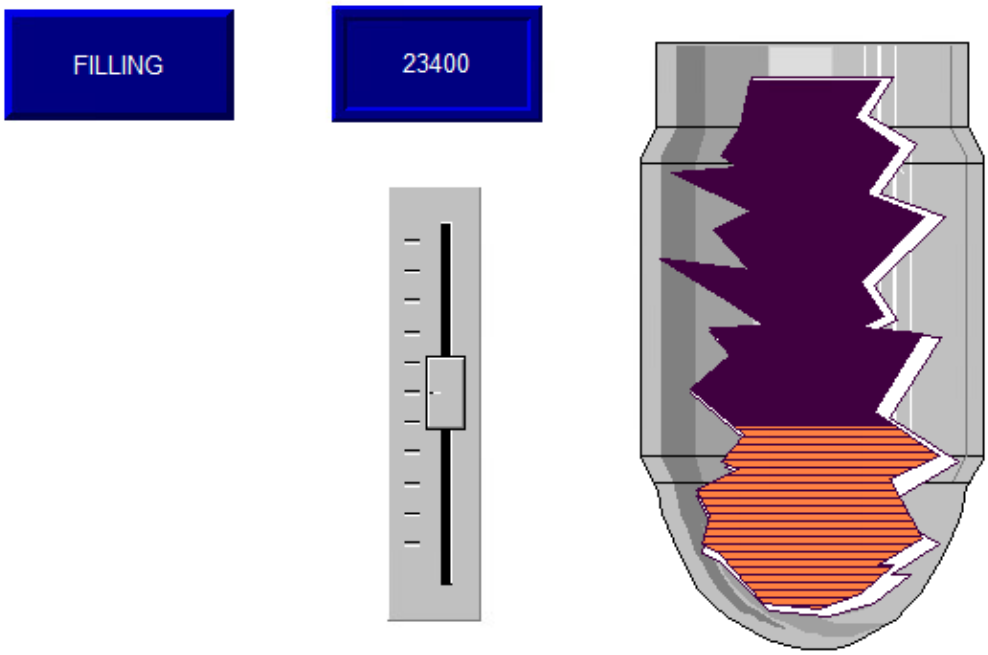
7. For the slider, highlight the square object in the slider. Go to Animation -> Vertical Slider.



8. Check Use tag's min and max property values. Go to tag browser and select "TimeValue" under the project tree. Click **OK** and **Close** button to exit.

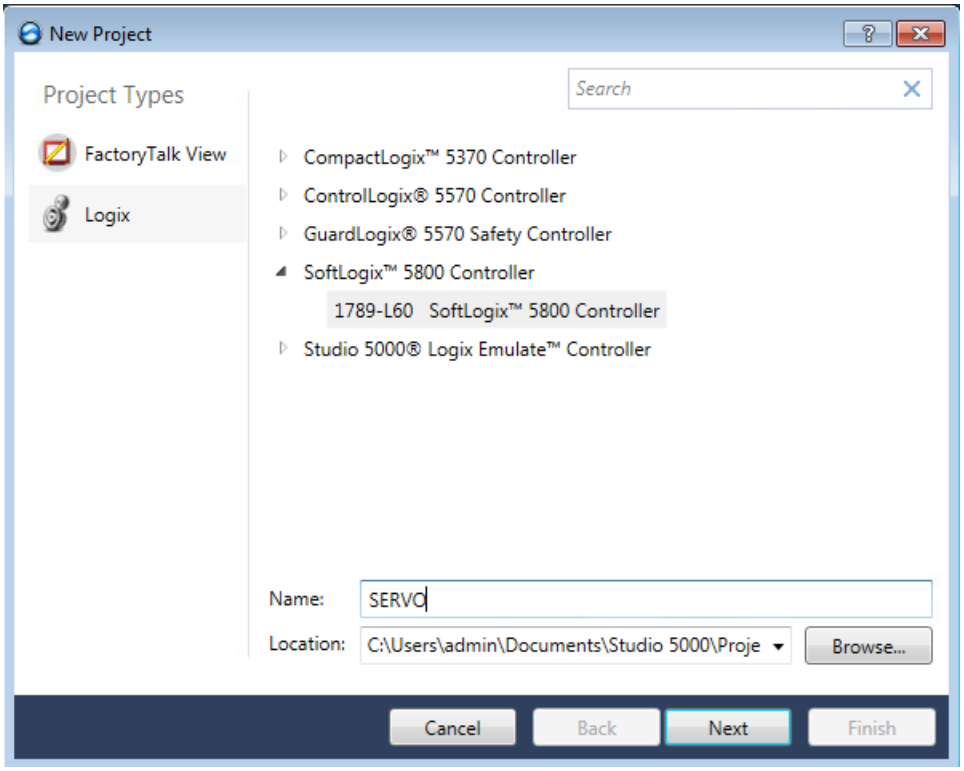


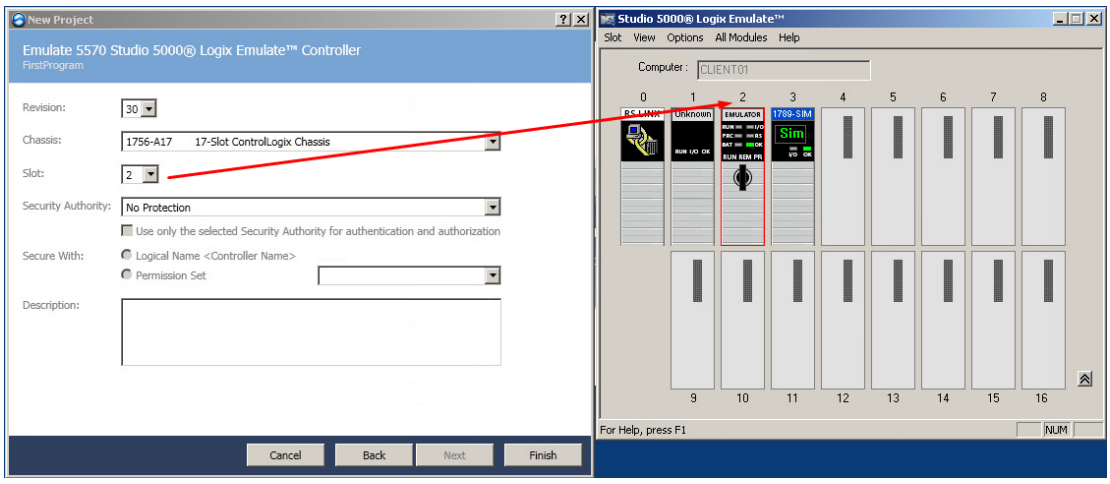
9. Use the slider to change the timer's preset value range between **0** to **30000**. Press the **FILL** button to start filling the tank.



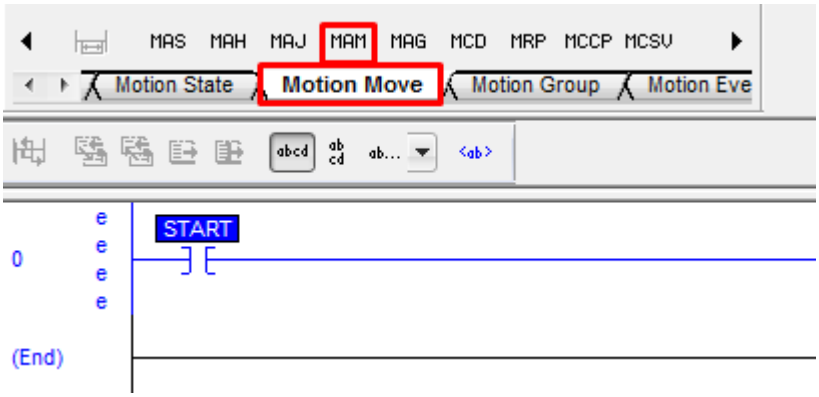
Part IX Introduction To Servo And Virtual Axis

Start a new project name **SERVO**.

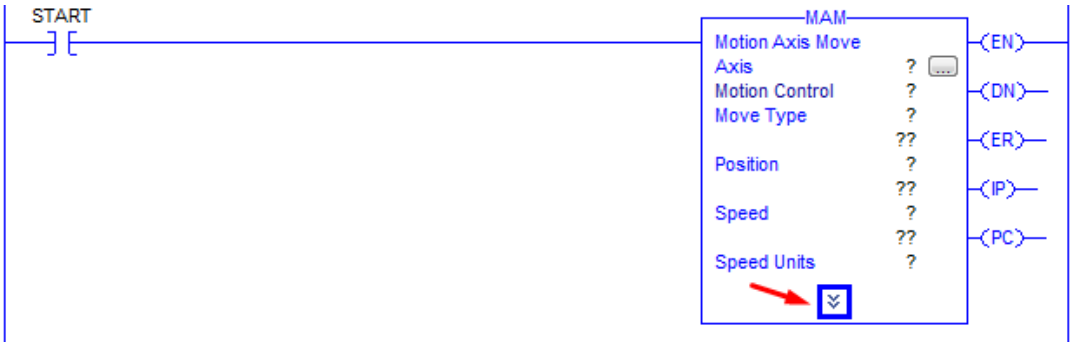




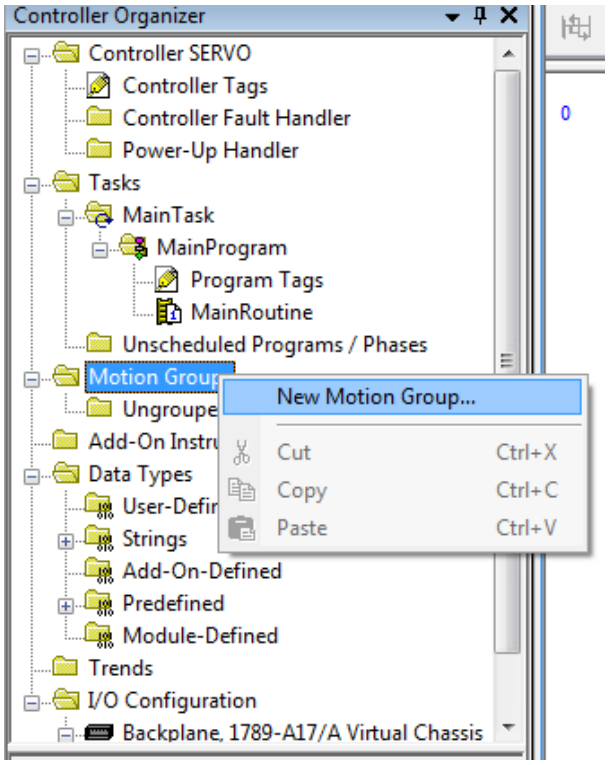
Create a "START" button, navigate to the Motion Move tab in the instruction menu. Drag and drop the MAM instruction to the right side of the first rung.



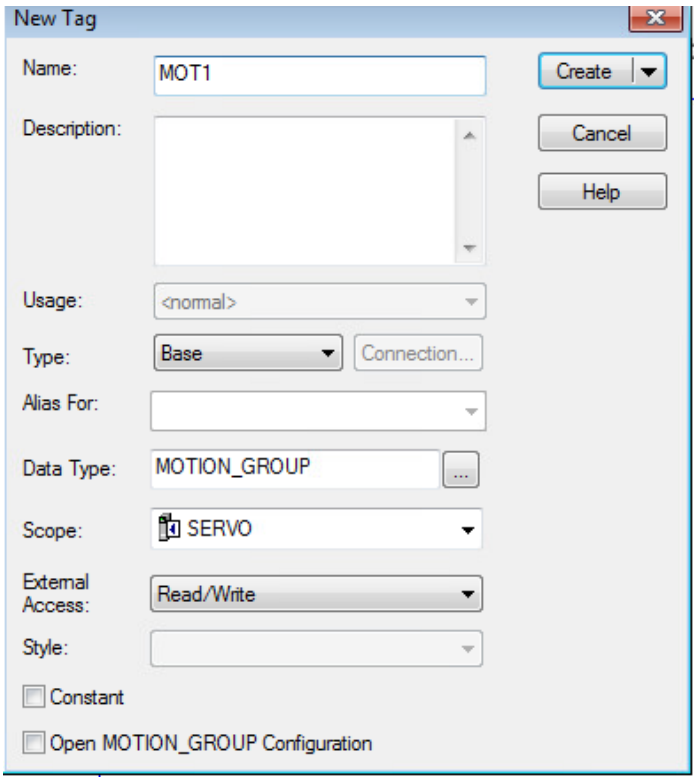
Click on the two arrows pointing down to expand the hidden parameters.



Right click on the Controller Organizer -> Motion Group and select New Motion Group...

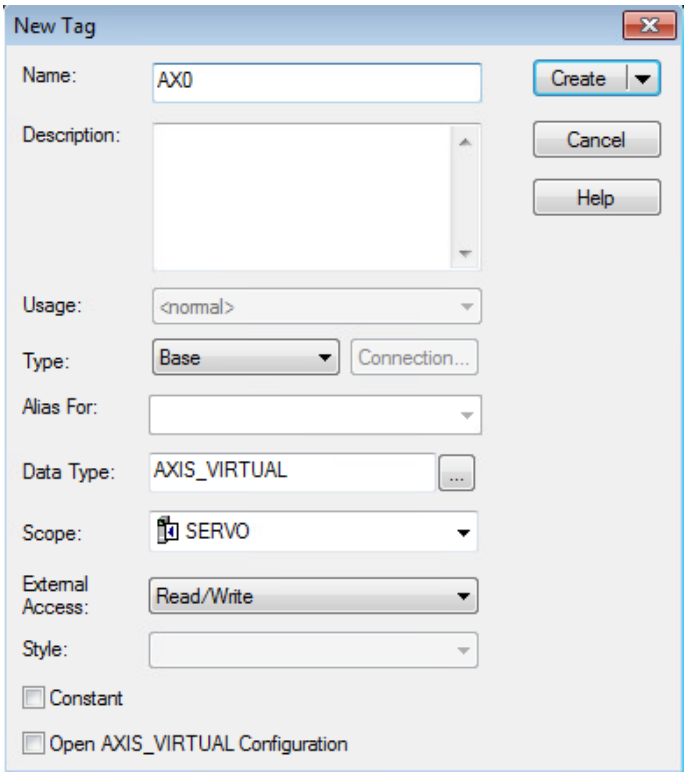


Name the motion group as MOT1, click **Create** button to continue.

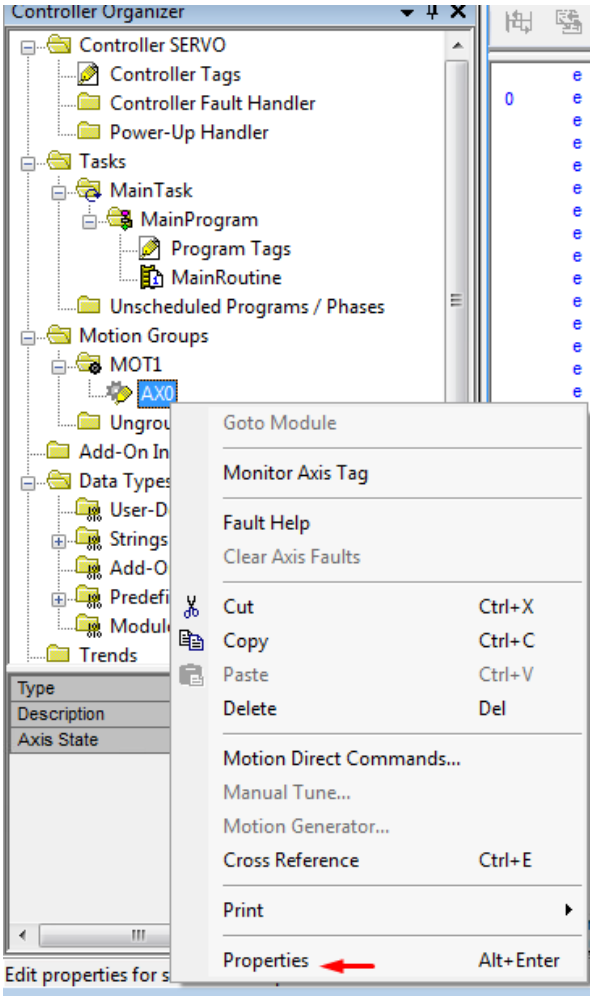


Right click on MOT1 and select **New Axis** -> **Axis Virtual...**

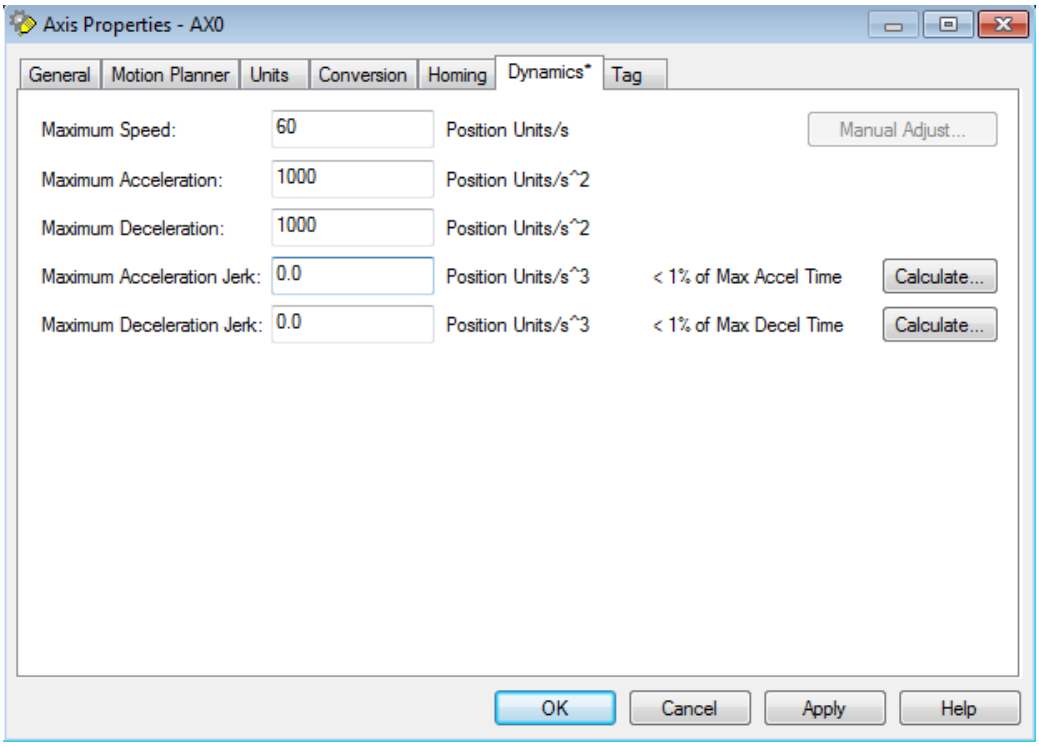
Enter AX0 in the name box then click **Create** button to continue.



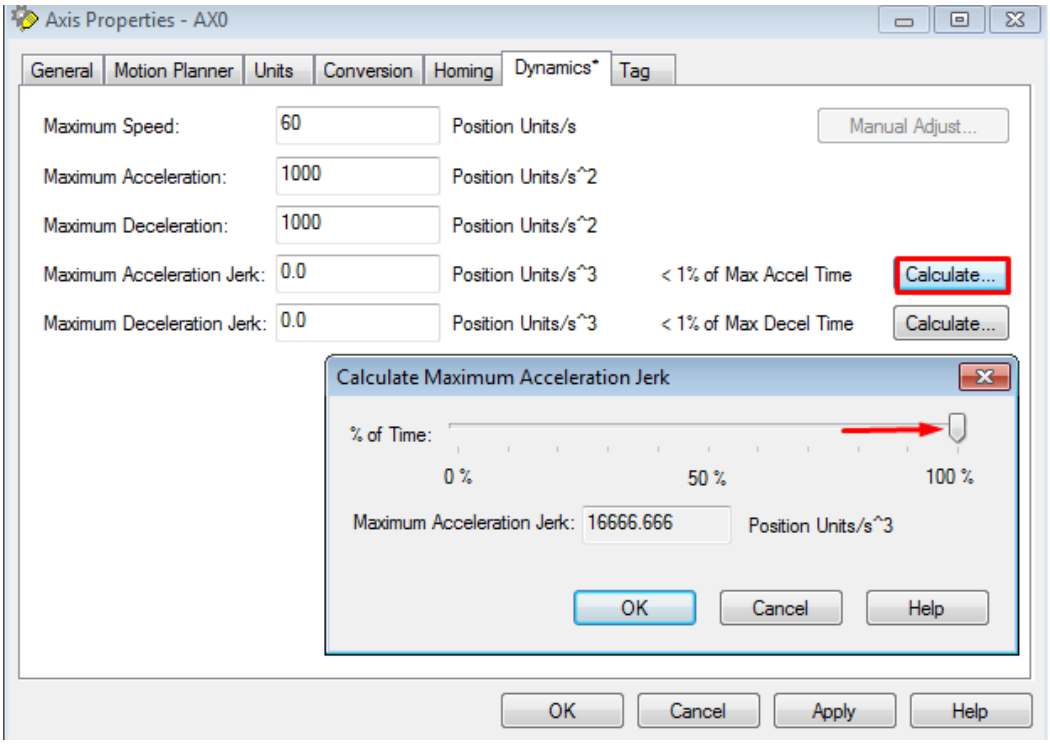
Right click on the AX0 and go to properties.



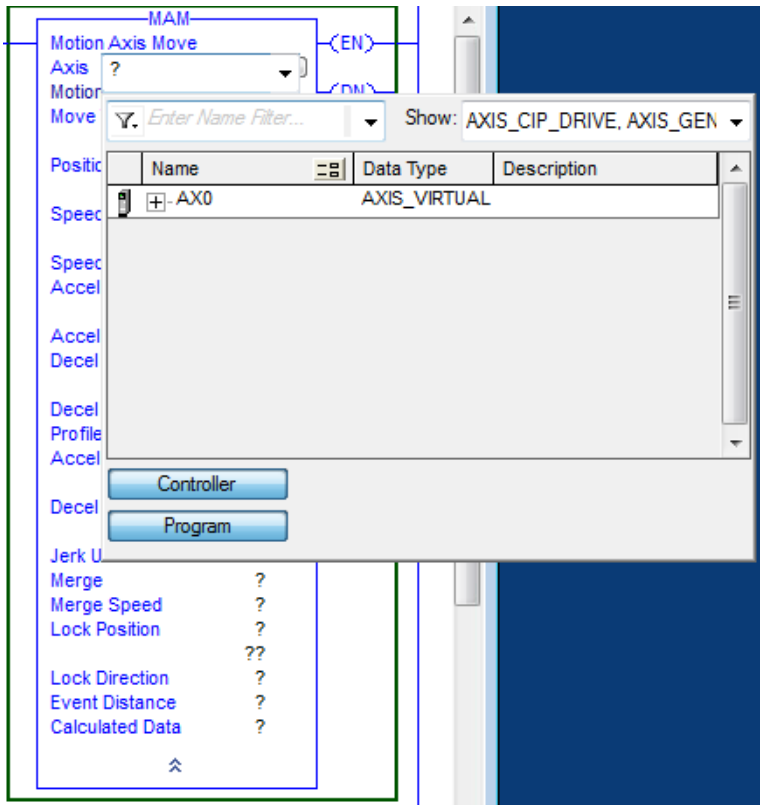
Navigate to the Dynamics tab in properties. Enter values for maximum speed, acceleration, and deceleration.



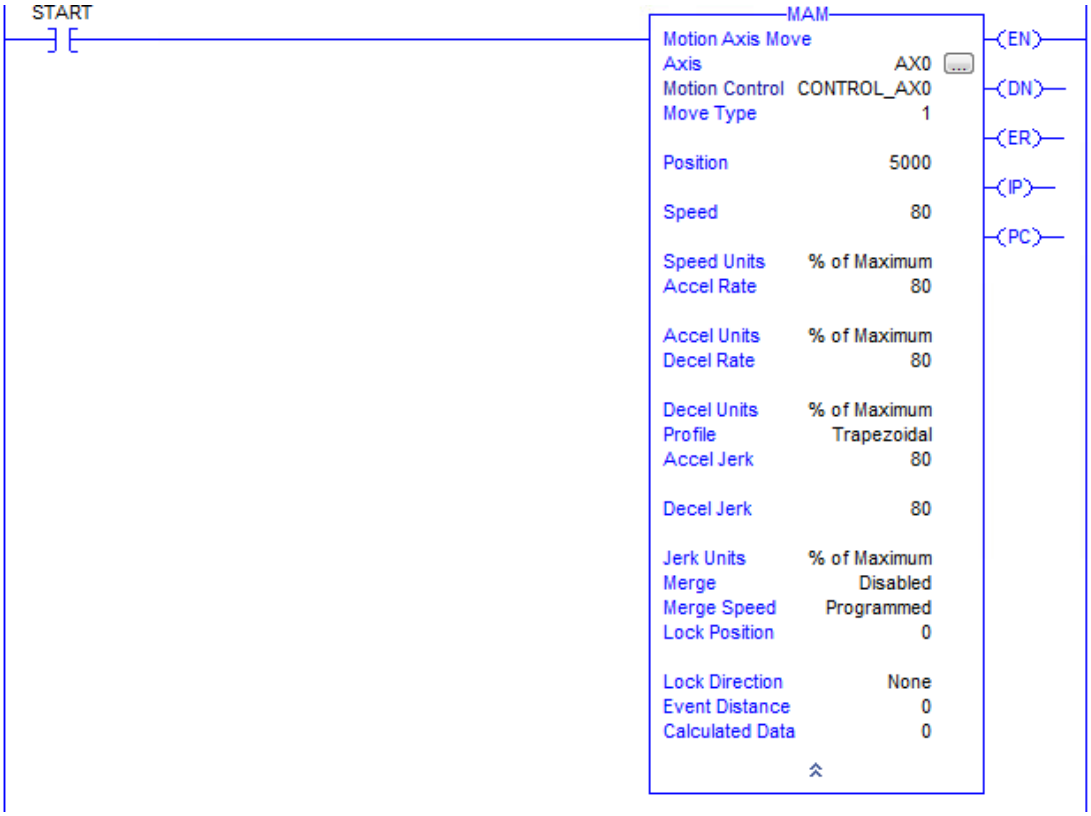
For maximum acceleration and deceleration jerk. Click the **Calculate** button then slide the value from 0 to 100%



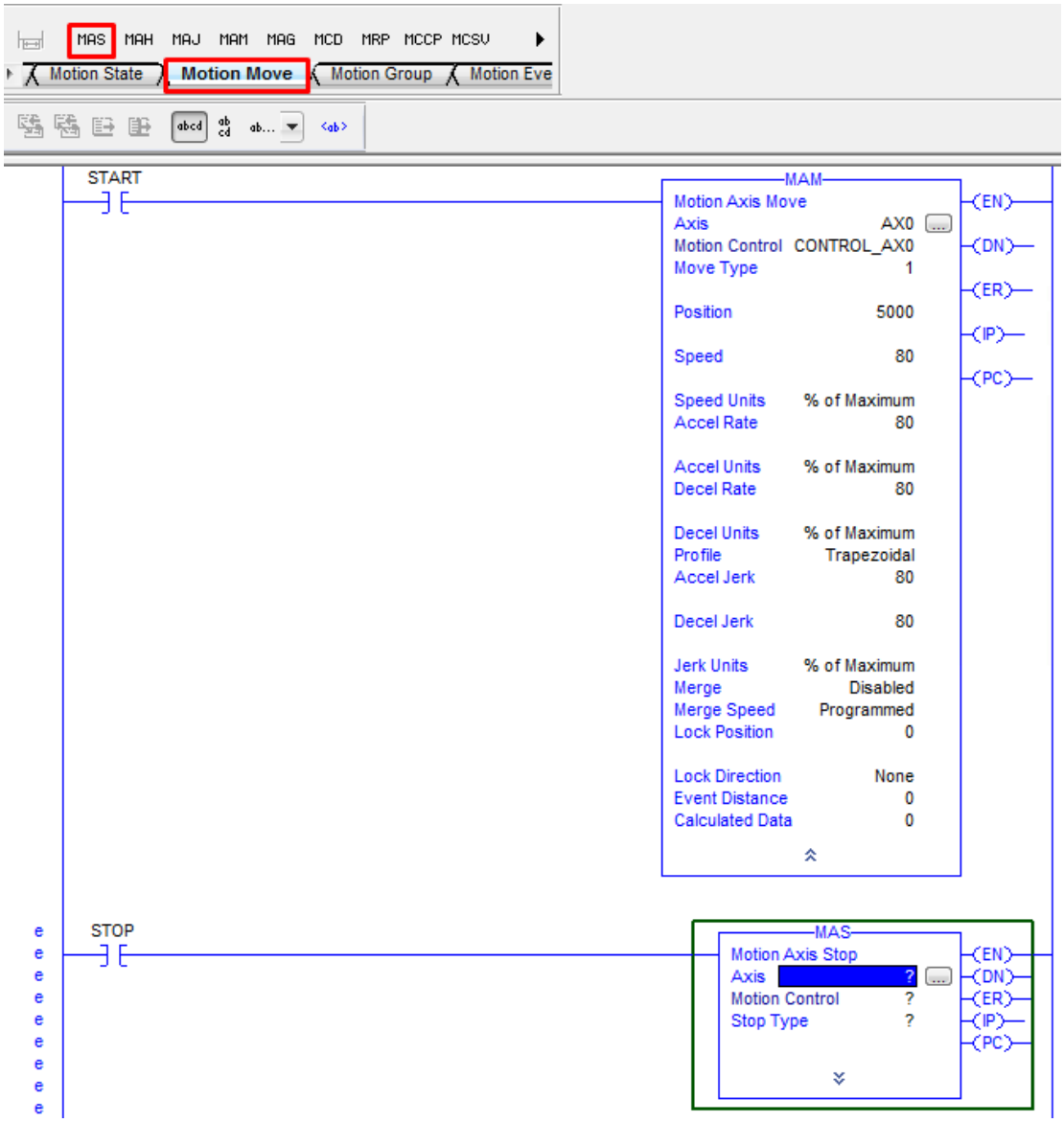
Enter the parameter for the MAM instruction. Click the **?** mark then select AX0 for the Axis.



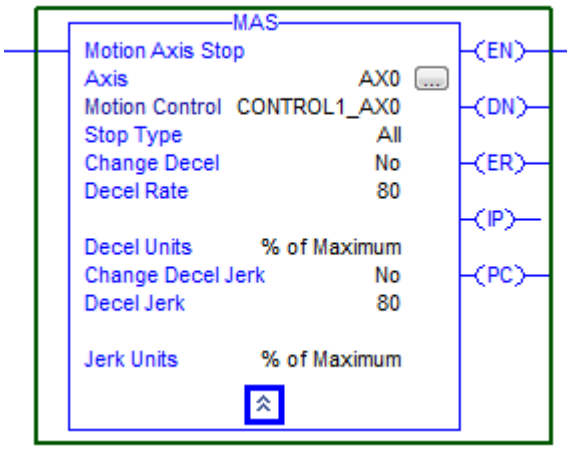
Create a new tag call "CONTROL_AX0" in Motion Control. Make sure to define the new tag "CONTROL_AX0".



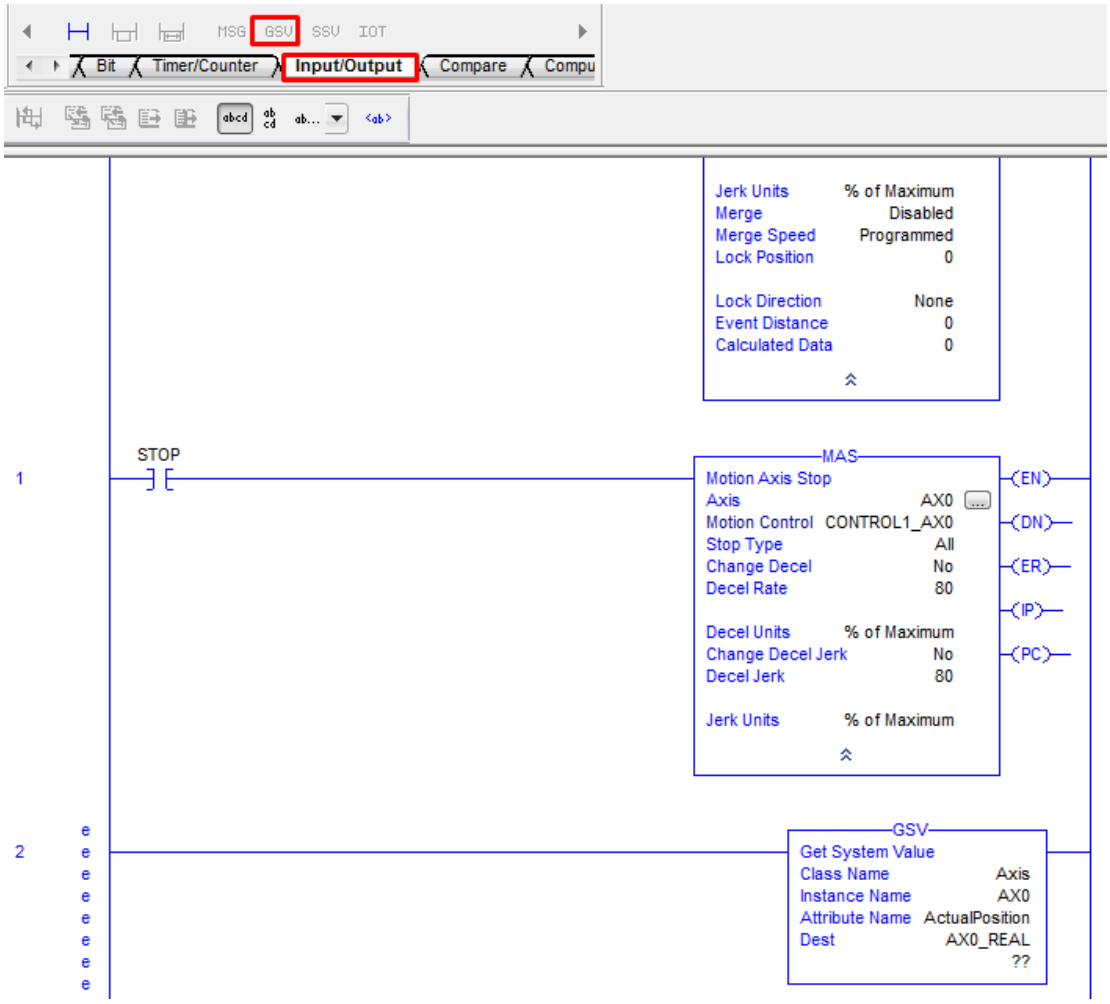
Insert a new rung, put a "STOP" button and MAS instruction which can be found in the Motion Group located in the instruction menu.



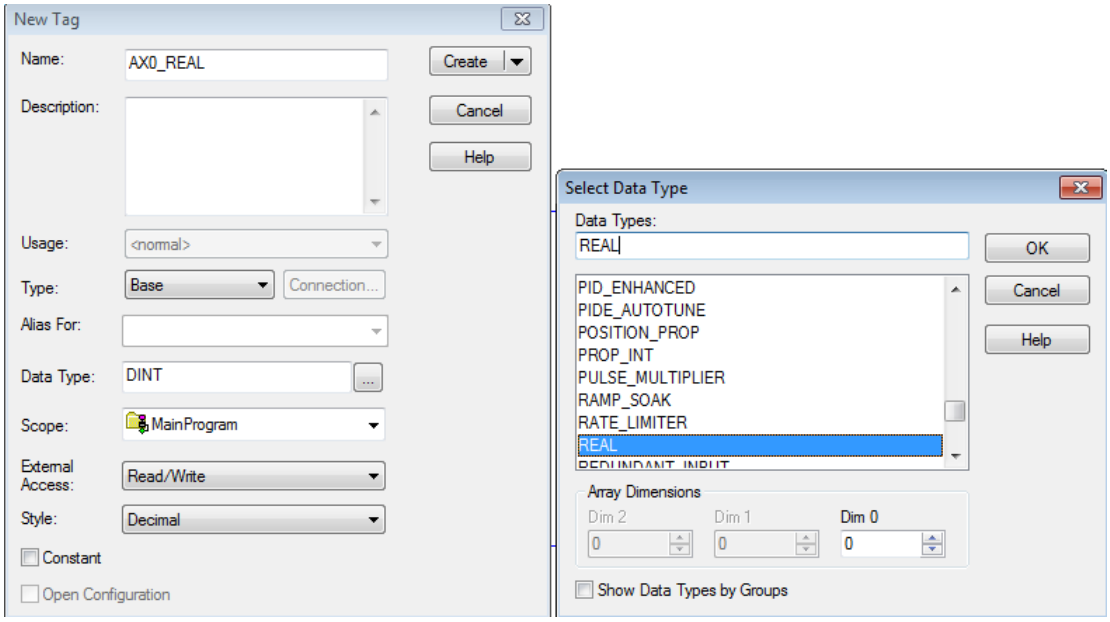
Enter parameters for the MAS instruction. Create a new motion control call CONTROL1_AX0 and define the new tag.

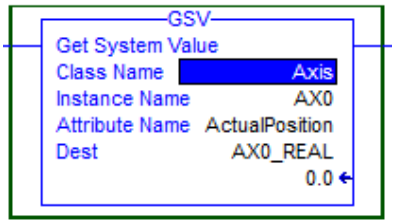


Go to Input/Output tab then select the GSV instruction. Insert the GSV instruction on a new rung and enter the parameters.

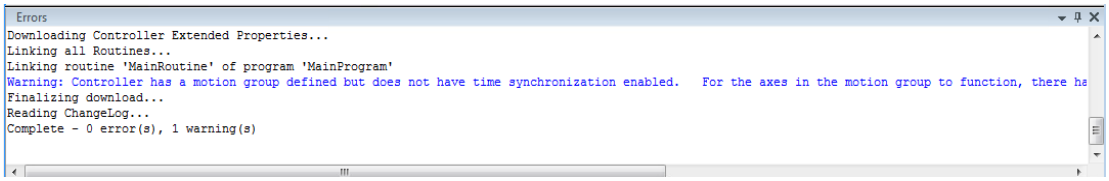


Change data type for AX0_REAL from DINT to REAL.

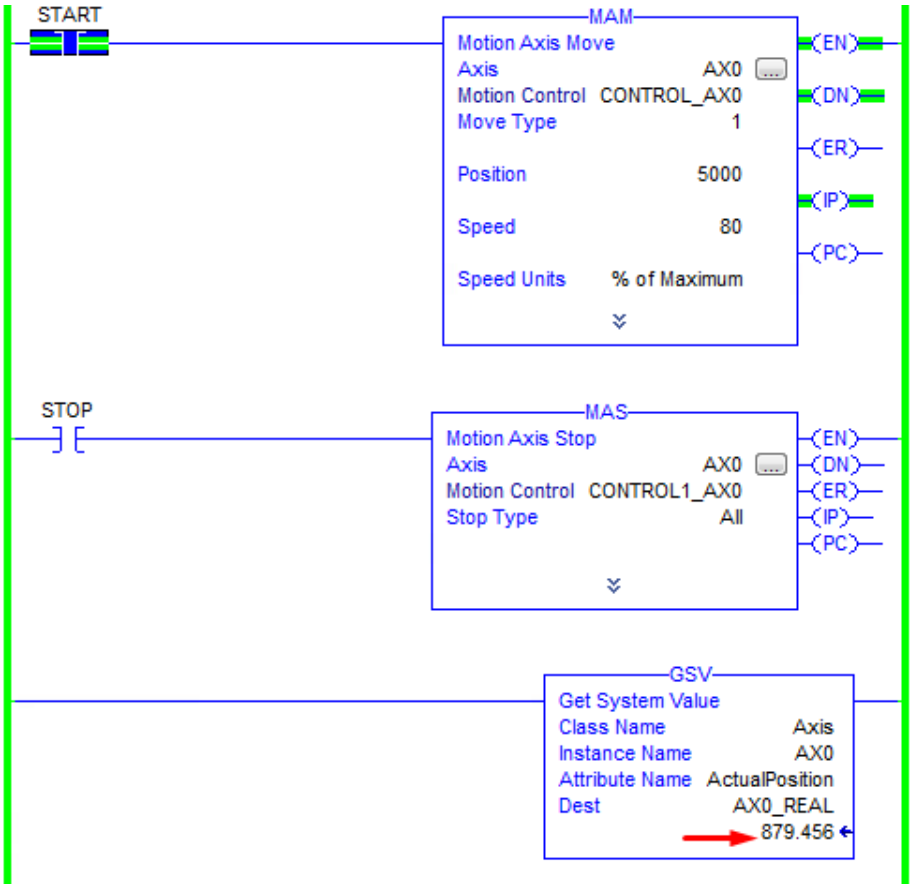




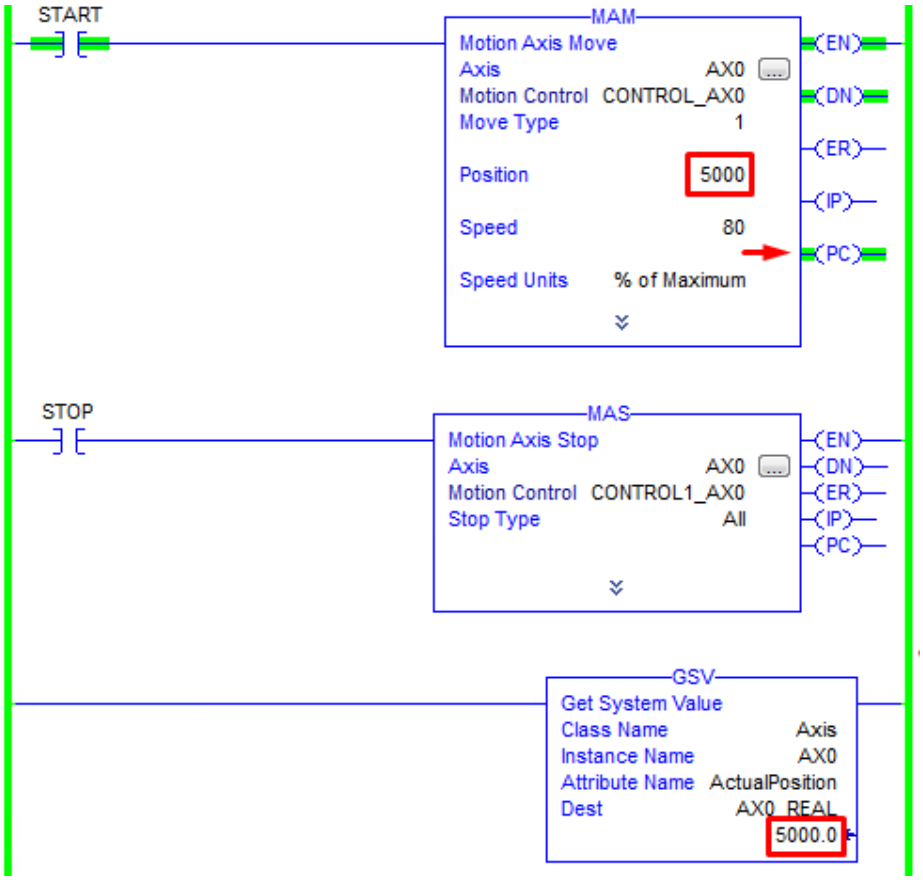
Ignore the warning when you try to download the program.



You should see the axis moving toward position 5000 after toggled the "START" button.



When the position reaches 5000, the PC bit is ON to indicate the process is completed. You can stop the AX0 with the "STOP" button.



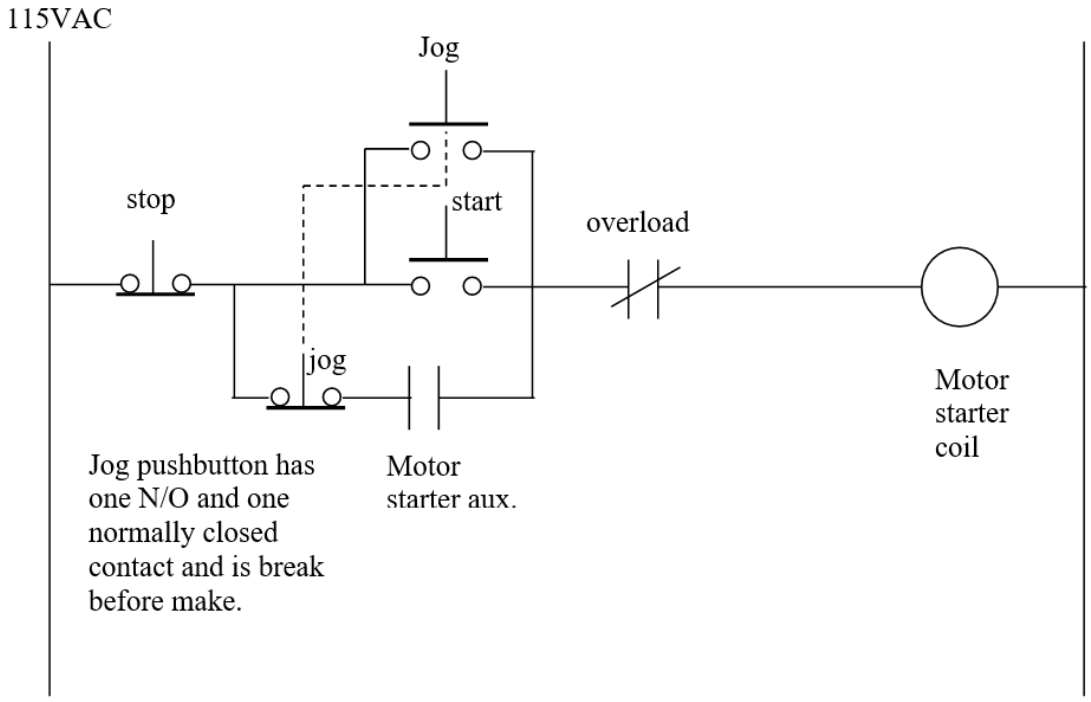
Part X Assignments

Complete the assignment and submit to your instructor for evaluation.

1. BREAK BEFORE MAKE (15%)

Start – Stop – Jog logic

Create a ladder logic program that mimic a typical hardwired stop/start/jog control.



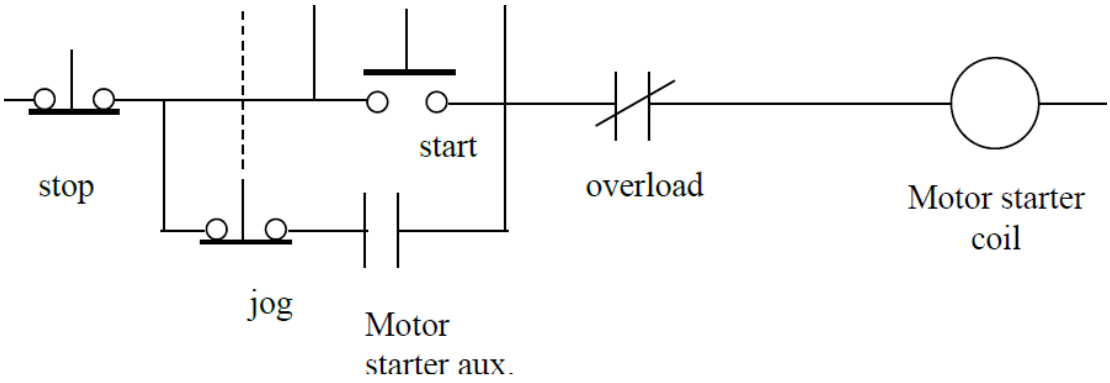
Note: Jog push-button has one normally open and one normally closed contact and is break before make.

Tags Required:

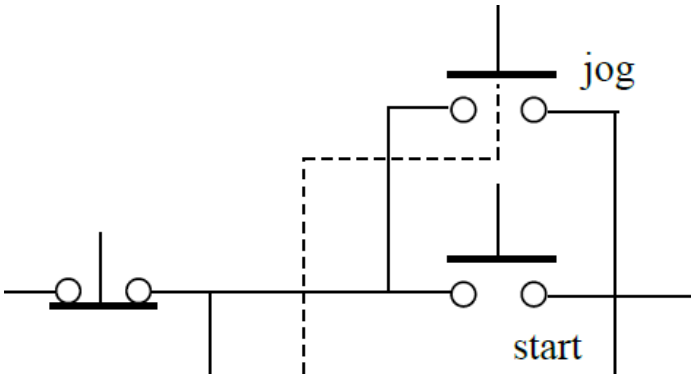
"Stop", "Start", "Jog", "Overload", "Motor", "Coil"

Purpose of the JOG buttons:

The normally **closed** "JOG" button can be used as a "STOP" button because "JOG" and "MOTOR" starter aux are connected in series. When Press and Release the "JOG" button, "MOTOR" turns OFF.



The normally **opened** "JOG" button is connected in parallel with "START" button. Therefore "JOG" button can be used to turn on the "MOTOR" as long as the "JOG' button is pressed.



Hint: Two "JOG" button on the same rung will have same condition. But "JOG" button on two different rungs will create a small timing difference.

2. TRAFFIC LIGHT (15%)

Make a traffic light simulation for a 4 way intersection with the use of timers.

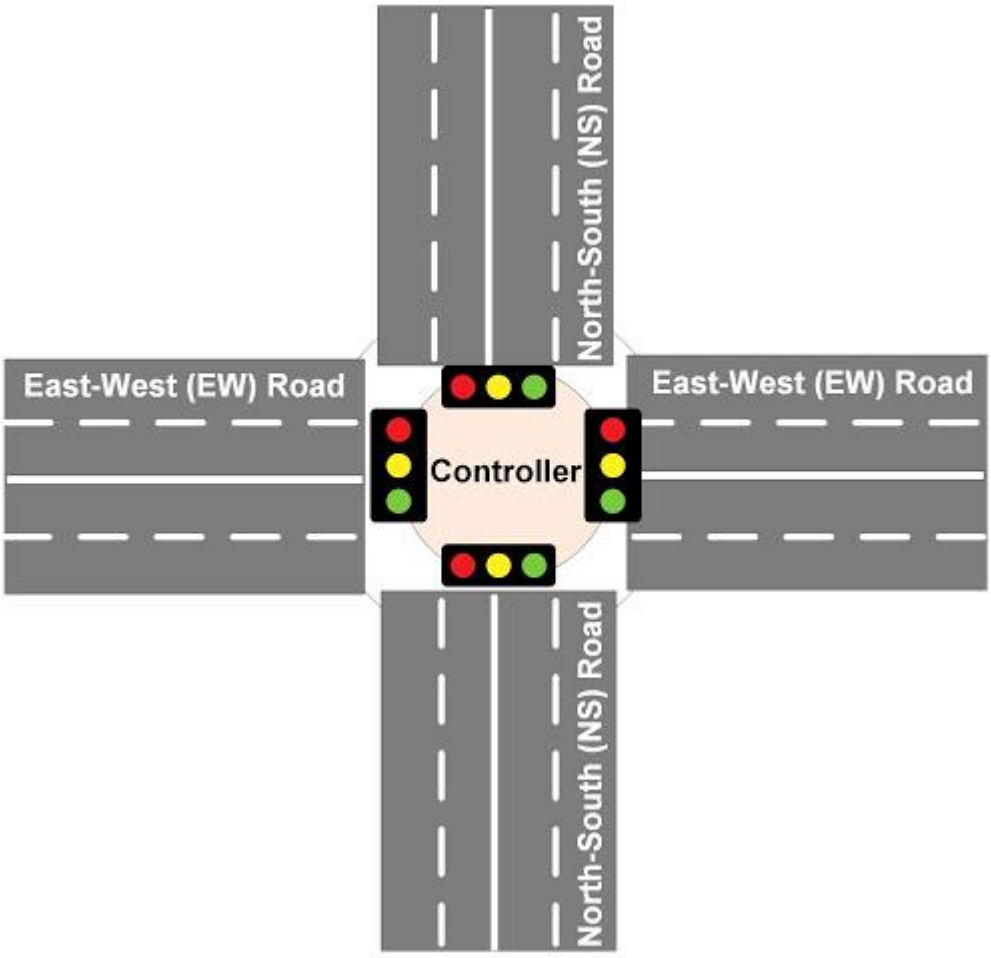
Try to make it as real as possible , this means : Both directions are red for a few seconds before one direction gets a green light. The other sequence of green – yellow – red speaks for itself.

Note:

Refer to the flashing light example to implement the traffic light with 6 timers. Ignore the outputs for now, focus on the timers logic.

Make sure when "Timer1" done then Timer2 starts. Timer2 is done then Timer3 starts etc...

Use the done bit from the last timer to reset the sequence.



TIMER SEQUENCE TABLE

	EAST-WEST			NORTH-SOUTH			
TIMER(S)	RED	YEL-LOW	GREEN	RED	YEL-LOW	GREEN	TIMING (ms)
1	1	0	0	1	0	0	2000
2	1	0	0	0	0	1	10000
3	1	0	0	0	1	0	2000
4	1	0	0	1	0	0	2000
5	0	0	1	1	0	0	10000
6	0	1	0	1	0	0	2000

Translate the timer sequence table into logic

Example:

East_West_RED is TRUE when Timer1, Timer2, Timer3, or Timer4 is timing.

East_West_YELLOW is TRUE with Timer6 is timing

East_West_GREEN is TRUE with Timer5 is timing

	EAST-WEST		
TIMER(S)	**RED**	**YELLOW**	**GREEN**
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	0	0	1
6	0	1	0


TIMER1.TT
TIMER2.TT
TIMER3.TT
TIMER4.TT
EAST_WEST_RED

3. COMPRESSOR (15%)

COMPRESSOR

Design a system needed to start a compressor with separate oil pump and cooling fan motors.(Using the least amount of ladder logic possible). Use a simple start/stop circuit to start & stop the system in the following order:

PART I

1. The oil pump will start and stop immediately with push buttons.

2. The compressor will start 2 minutes later if the oil pressure switch has confirmed oil pressure. It will stop immediately when the stop button is pressed or on an oil pressure fault.

3. The cooling fan will start with the compressor, and stop 5 minutes after the compressor is stopped.

4. If you can, make a lockout in case of an pressure fault, no pressure is detected at start-up or drops, compressor is running but cooling fan stops. When the machine is in lockout mode, use a "Reset" button to re-enable the machine. Turn ON a light when "LOCKOUT" is true.

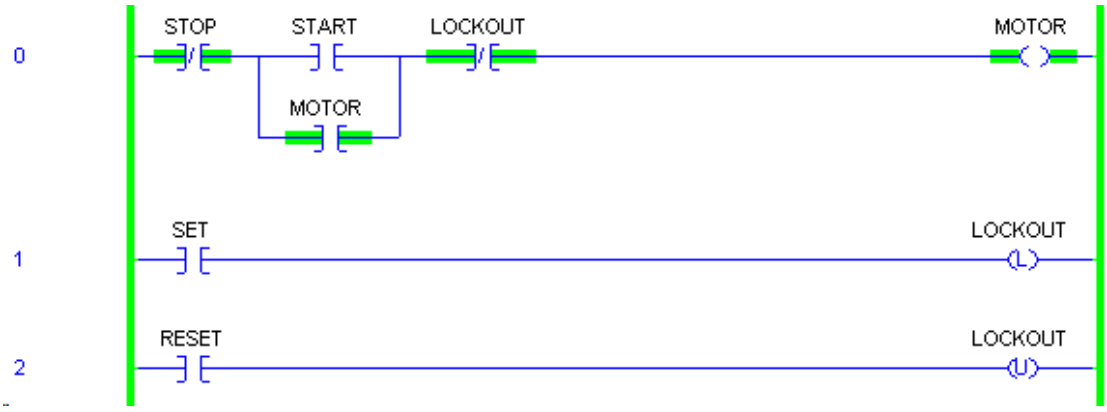
** When debug the logic, reduce the waiting time from 2 minutes to 20 seconds and cooling fan to 10 seconds. Try to use TOF timer for the cooling fan.*

Tags:

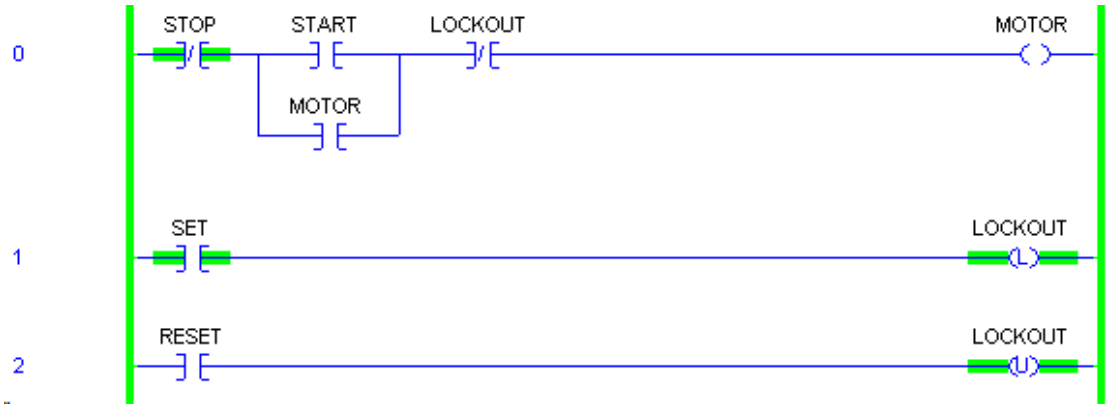
"Start", "Stop", "Oil_Pressure_Switch", "Oil_Pump_Coil", "Compressor_Coil", "Colling_Fan_Coil", "RESET", "LOCKOUT"

Example:

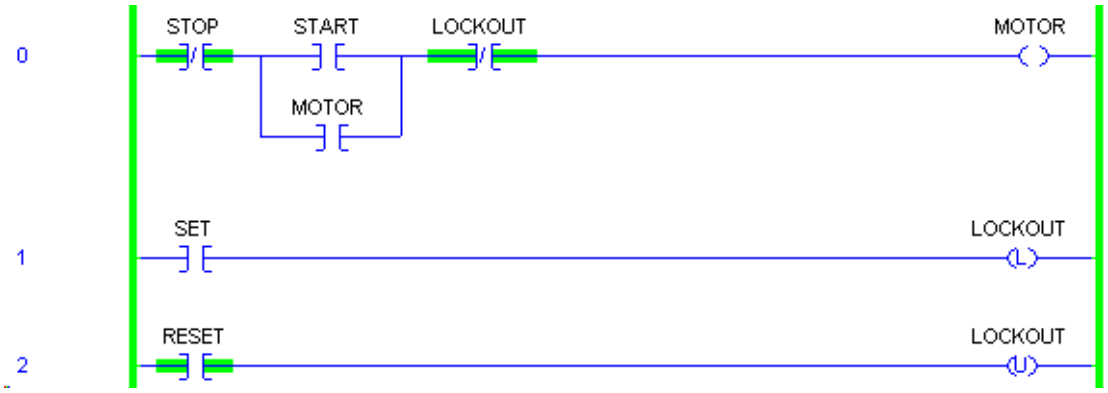
Use latch and unlatch instruction for the lockout.



Toggle "SET" will latch the "LOCKOUT" bit.



Toggle "RESET" will re-enable the "LOCKOUT" bit.



4. TEMPERATURE CONVERSION (15%)

Write a program that converts the temperature from Fahrenheit to Celsius or Celsius to Fahrenheit.

Create an HMI in FactoryTalk View Studio (Machine Edition):

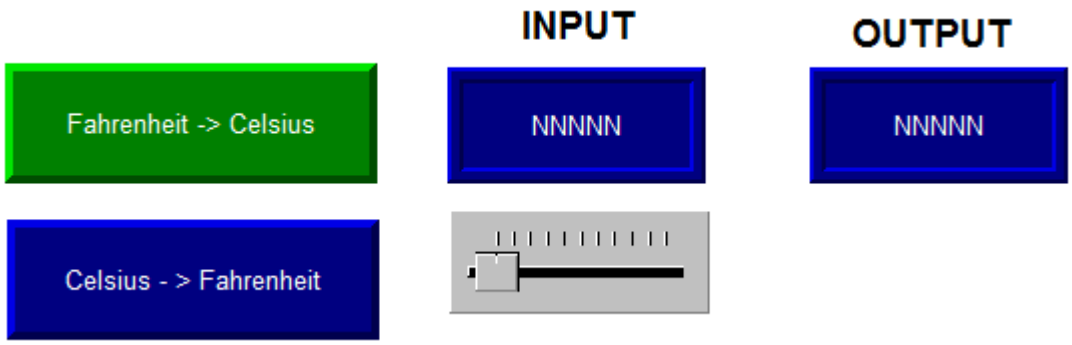
1. Use slider from the library for adjusting the temperature.

2. Use numerical display to show the input value and converted value.

3. A selector is required to switch between F to C or C to F.

4. A light to indicate when the temperature is in negative (flashing light is preferred)

5. Create a HMI in FactoryTalk View Studio. Use Slider from the library for adjust temperature.



Formulas:

C = (F –32) * 5 / 9

F = 9 * C / 5 + 32

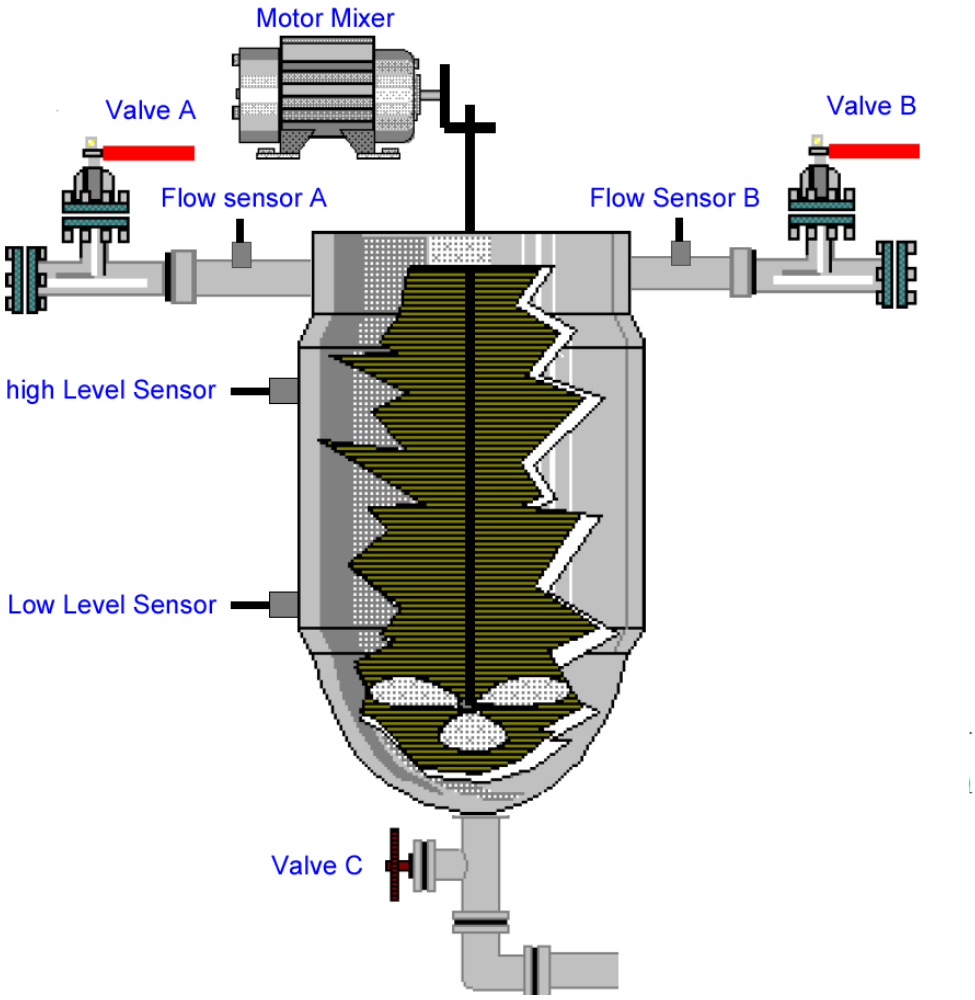
5. CHEMICAL BATCH MIXER APPLICATION

The sequence of the batch process is as follows:

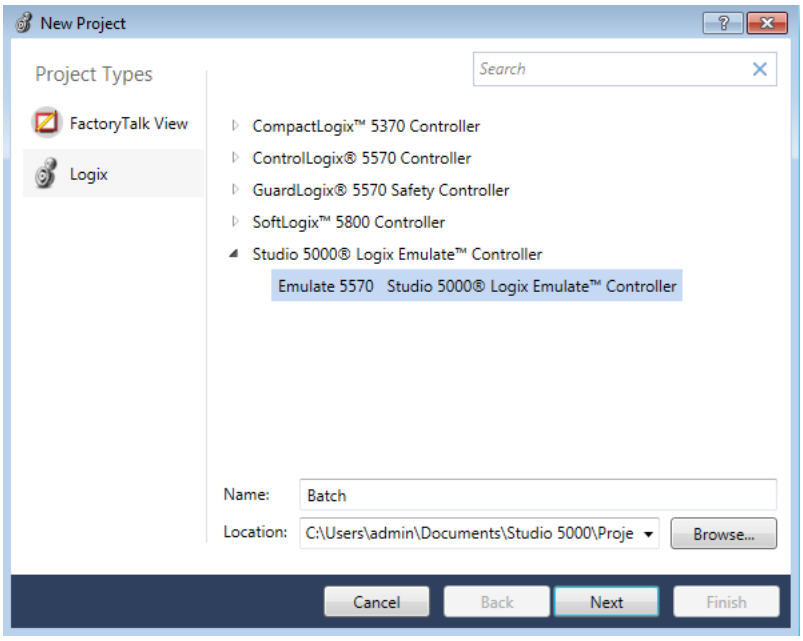
1. provide a start/stop station to start the process.
2. When the high level sensor turns on motor starts for 10 seconds. Use an On Delay Timer to control the motor.
3. When the mixer stops, valve C will open to release the product until the low level sensor detects no more product.
4. Valve C closes and the process repeats again.

5. Implement flow sensors A & B. Limit the flow sensor A to 5 liters, and flow sensor B to 8 liters with CTU instruction. Valve A and B should be disabled when the limits are reached.

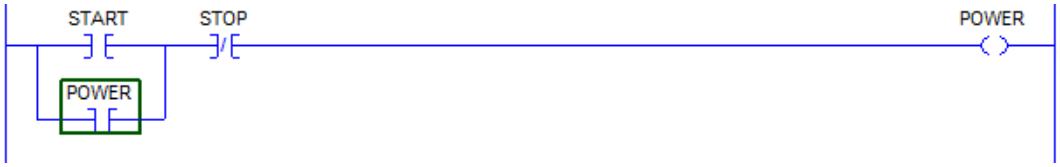
Note: try to simplify the logic by combine rungs with common instructions.



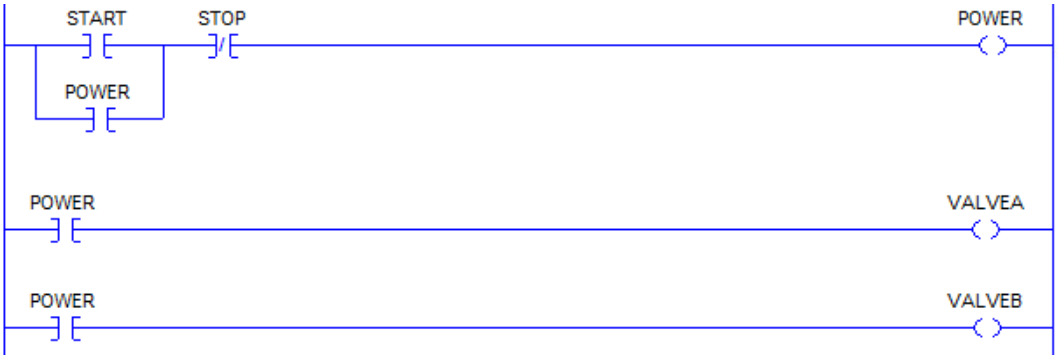
Let's create a new application in Studio 5000 name **Batch**.



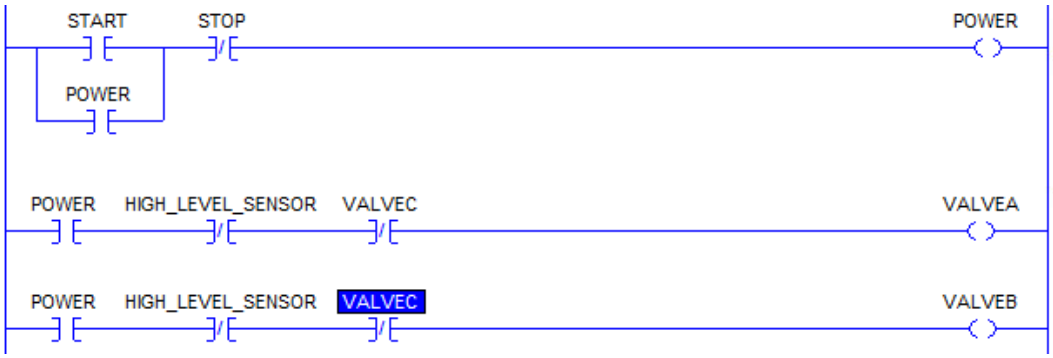
You need a "START" and "STOP" logic to control the output "POWER" ON or OFF.



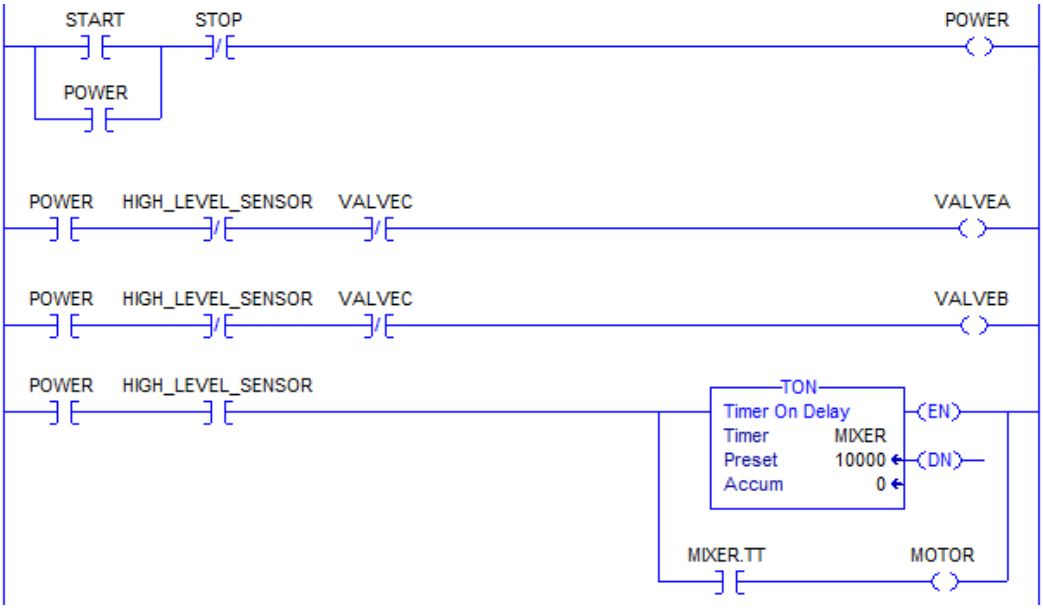
The output "POWER" enables valve A and B.



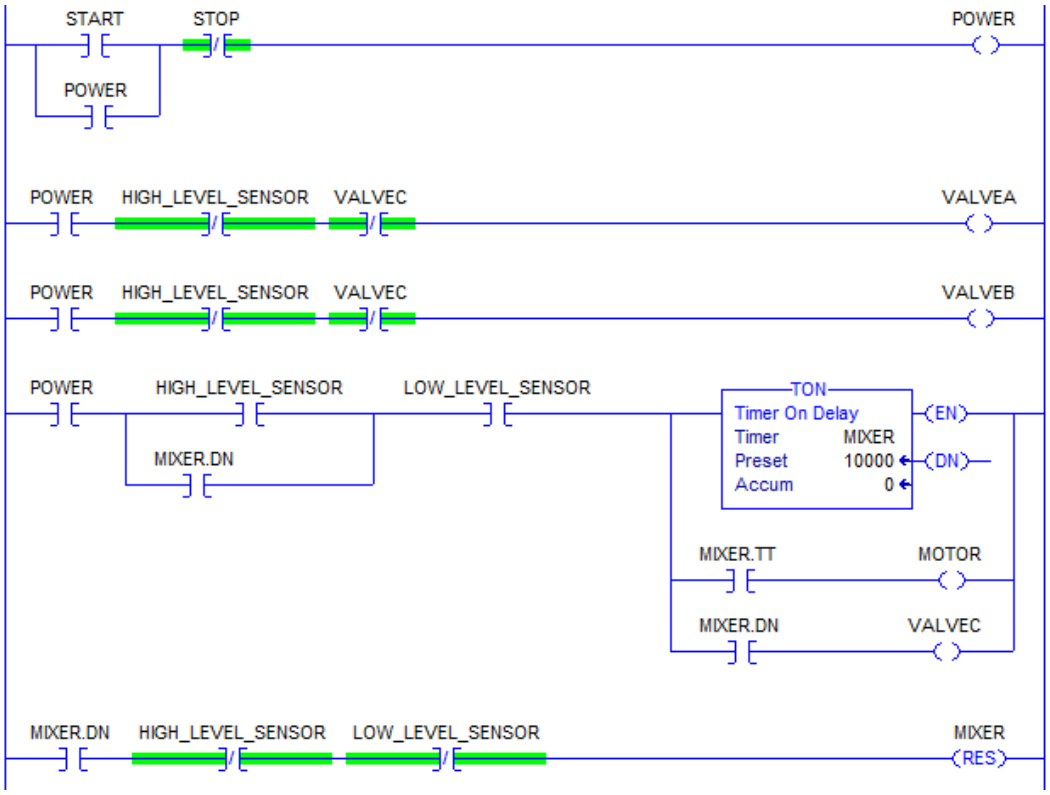
We need to implement some safeties into the logic to prevent the tank from over flow or valve c is not close. If high level sensor is ON or Valve C is open will interrupt the filling process.



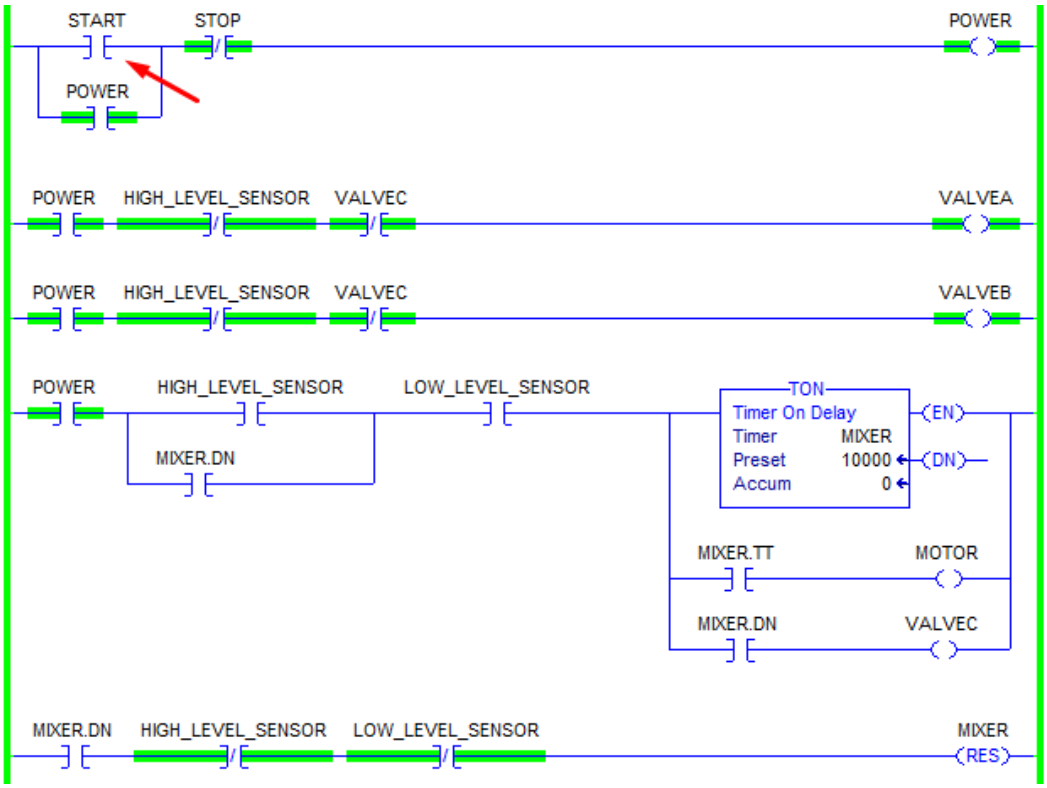
Create a timer name "MIXER" and then use timer timing bit to turn on the "MOTOR" for 10 seconds.



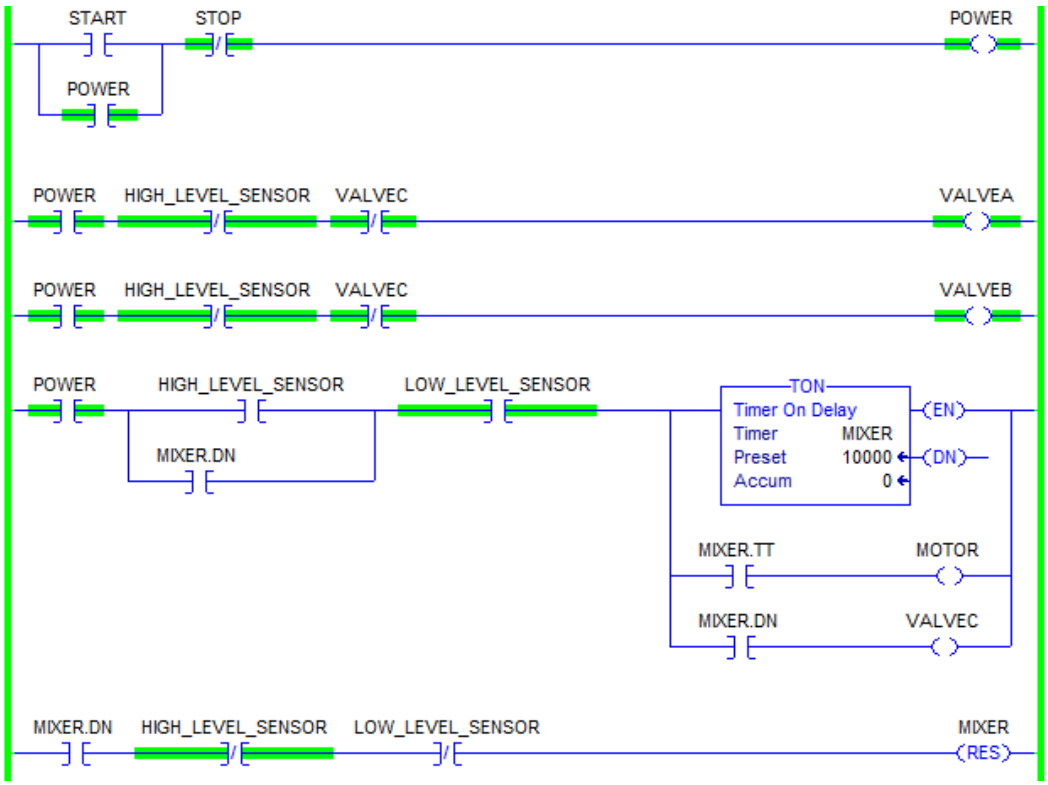
When the "MOTOR" stops then "VALVEC" opens to drain the chemical. After certain time, "HIGH_LEVEL_SENSOR" will switch OFF follow by "LOW_LEVEL_SENOR". The low level sensor indicate the tank is empty and resets the process for the next batch.



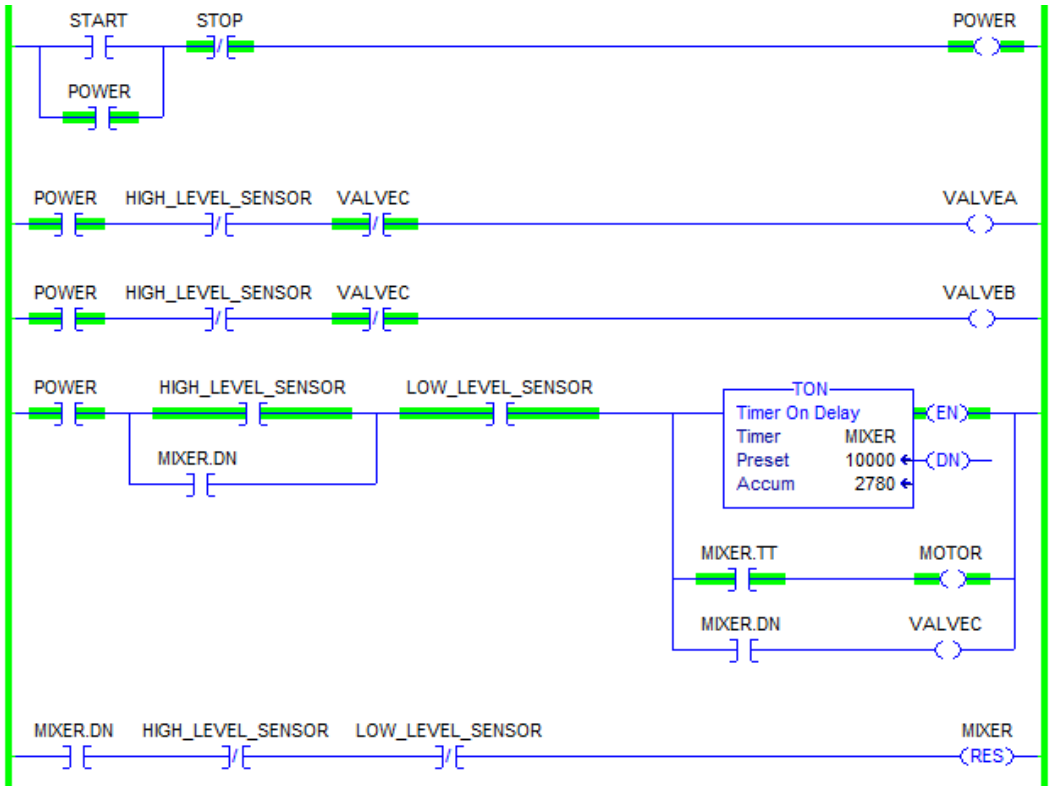
Simulate the process by download the program, and toggle the "START" to turn on the "POWER".



When the power is on and the valves start filling. Low sensor sensor should be ON to indicate the tank is no longer empty. Toggle "LOW_LEVEL_SENSOR" to ON position.

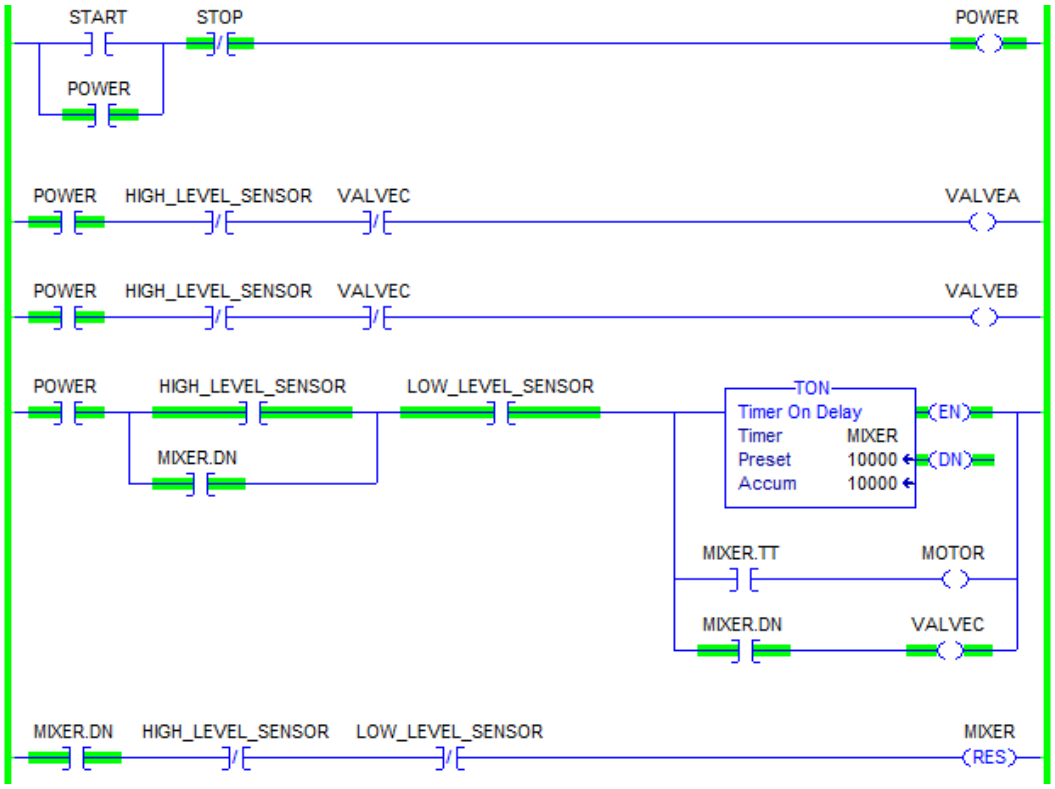


Toggle "HIGH_LEVEL_SENSOR" to on position. Assume the tank is fulled and the high level sensor is ON.



After the mixer stops, the timer done bit becomes TRUE. Valve C should be ON to release the chemical. High level level will turn off first follow by low level sensor.

Toggle the high level sensor off after the valve C opens, follow by low level sensor. The purpose of low level sensor is to reset the timer for the next batch.


START
STOP
POWER
POWER
POWER
HIGH_LEVEL_SENSOR
VALVEC
VALVEA
POWER
HIGH_LEVEL_SENSOR
VALVEC
VALVEB
POWER
HIGH_LEVEL_SENSOR
LOW_LEVEL_SENSOR
MIXER.DN
TON
Timer On Delay
Timer MIXER
Preset 10000
Accum 10000
EN
DN
MIXER.TT
MOTOR
MIXER.DN
VALVEC
MIXER.DN
HIGH_LEVEL_SENSOR
LOW_LEVEL_SENSOR
MIXER
RES

INDEX

No index entries found.

This is the last page of this template, the "back cover", so to speak. Put anything here that you want to have on the back cover or simply delete this last page.