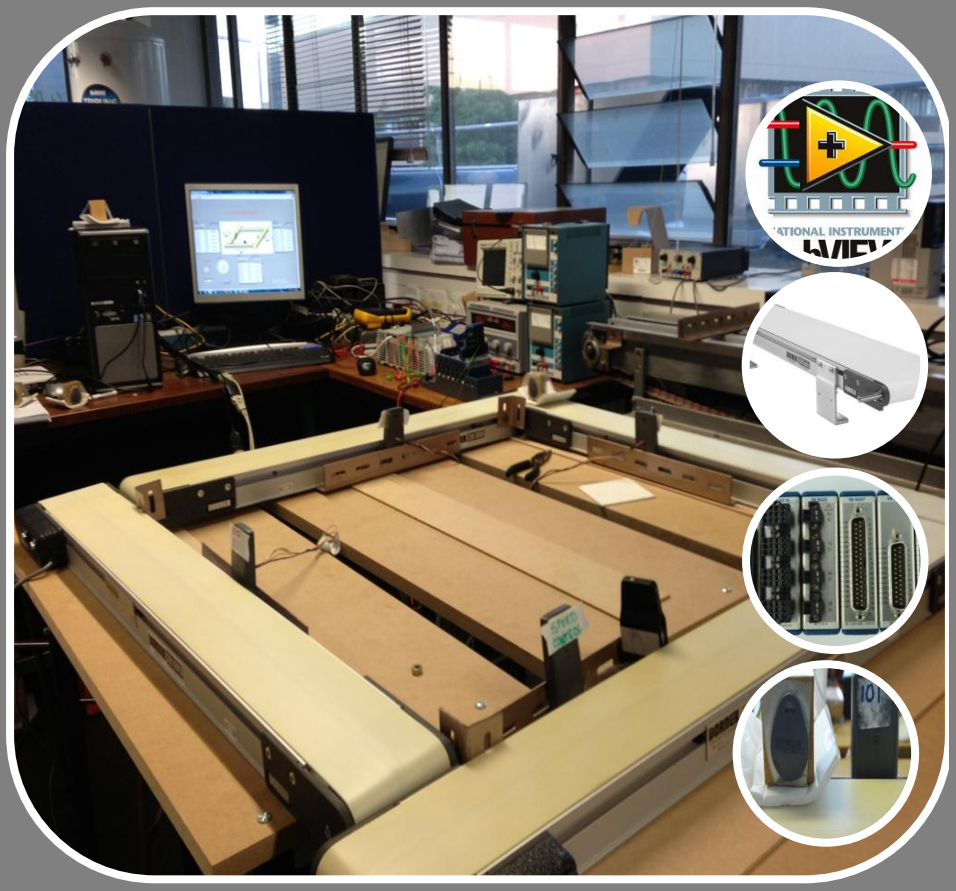
Programmable logic controller



Programmable Logic Controller (PLC)

Alireza Mousavi, Morad Danishvar and Alexandre Spieser

1. Introduction

PLC is a digital computer used for automation of electromechanical processes in plants. The PLC is designed for multiple inputs and outputs arrangements, so we can get the data from the sensors, work with it and command the actuators.

The first Programmable Logic Controller (PLC) was developed by a group of engineers at General Motors in 1968, when the company were looking for an alternative to replace complex relay control systems. The new control system had to meet the following requirements:

- Simple programming
- Program changes without system intervention (no internal rewiring)
- Smaller, cheaper and more reliable than corresponding relay control systems
- Simple, low cost maintenance

2. PLC Components

Fig 1 illustrates the system components of a PLC.

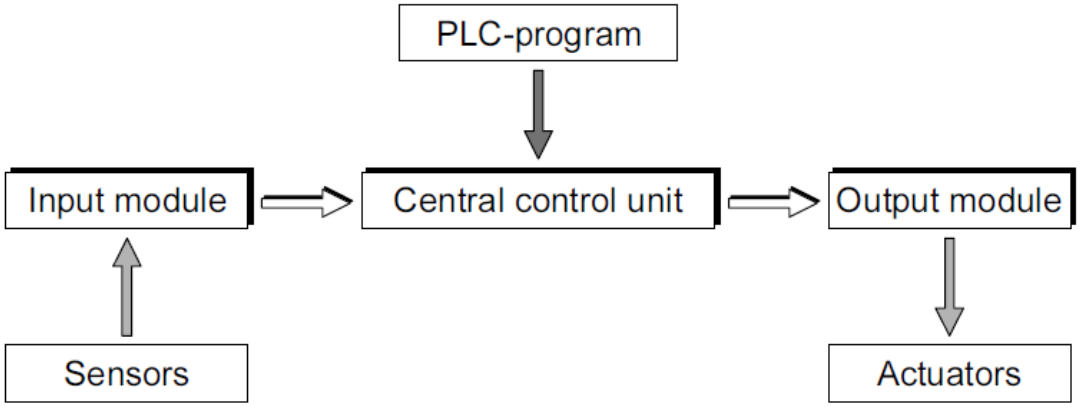


Figure 1. PLC components

The function of an input module is to convert incoming signals into signals that can be processed by the PLC, and to pass those signals to the central control unit. The reverse task is performed by an output module. This converts the PLC signal into signals suitable to operate and invoke the plant actuators. The actual processing of the signals is undertaken in the central control unit and with respect to the program stored in the memory.

The programs and operational routines in a PLC can be created in various ways:

The "Ladder diagram" strongly resembles a schematic diagram of relay logic. The other features are function block diagram (FBD) and statement list (STL). Fig 2 shows an example presented in different ways using controlLogix development tool. Depending on how the central control unit is connected to the input and output modules, various versions of the PLC can be put together. For example, compact PLCs (input module, central control unit and output module in one housing) or modular PLCs.

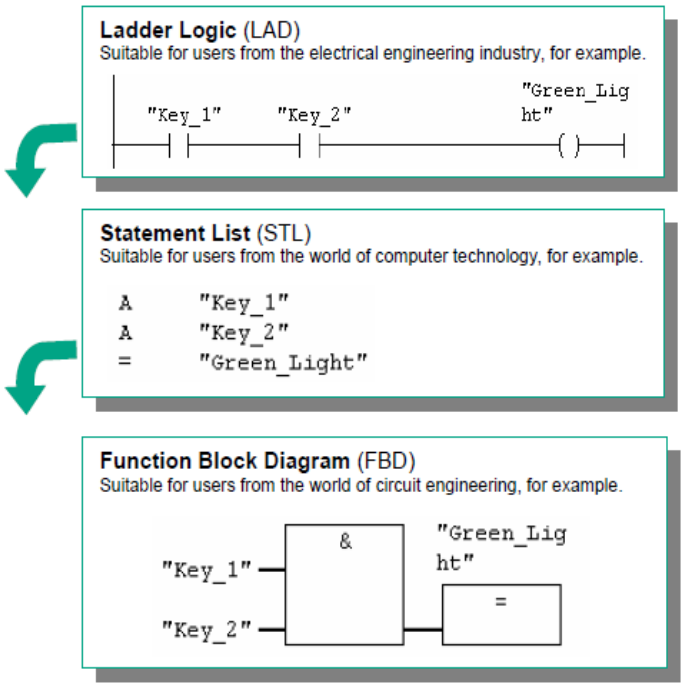


Fig 2. Different PLC programming structures

3. Basic Procedure for PLC programming:

Figure 3 describes the basic procedure for programming PLC.

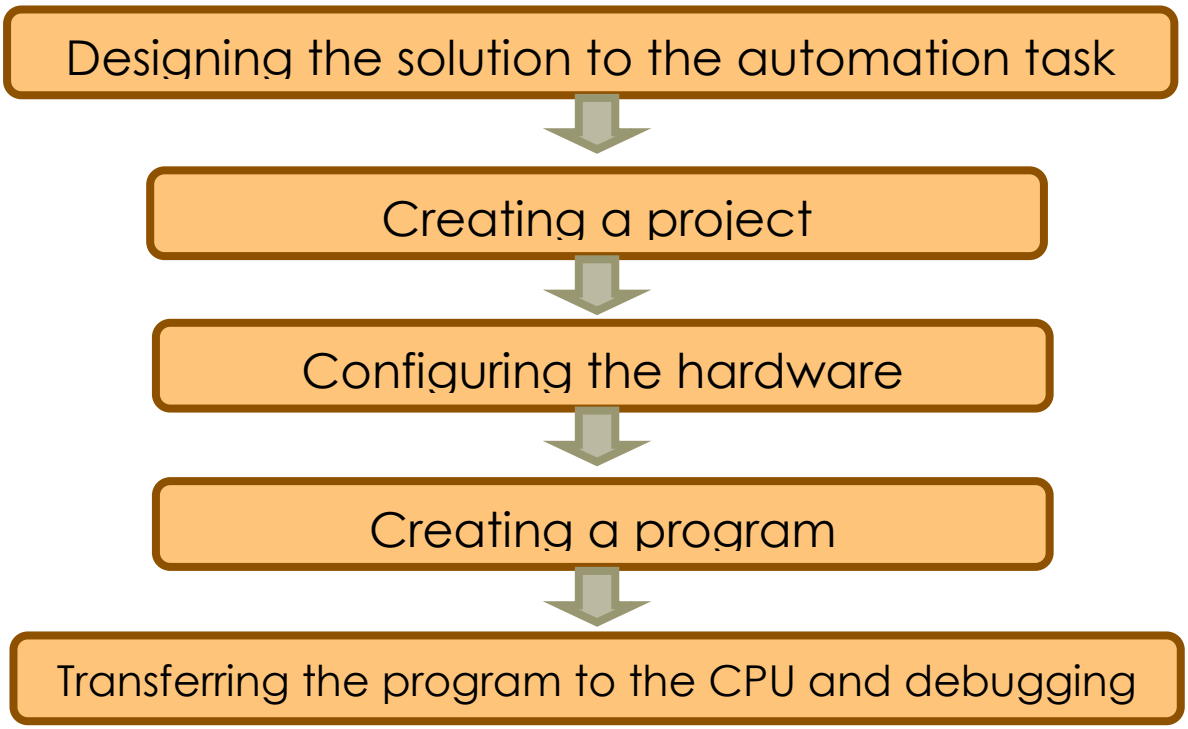


Fig 3. Basic procedure for PLC programming

ALLEN BRADLEY PLC: CompactLogix L32E

Allen-Bradley is the brand-name of a line of Factory Automation Equipment manufactured by Rockwell Automation.

The connection between the controller and the computer is either serial cable or Ethernet cable. We will explain the steps to setting up a project on PLC Allen Bradley using RS logix5000 software.

3.1 Configuring the hardware

Connect the Controller via the Serial Port and Configure the Serial Driver:

Begin with connecting the serial cable to the PC on one side and the controller on the other side.

Next configure a connection. To configure a connection, we use the RSLinx Classic Lite software. For the serial communication we need to configure the RS-232 DF1 Device driver.

1. Choose configure driver.
2. From the Available Driver Types pull-down menu, choose the RS-232 DF1 Device driver.
3. Click Add New to add the driver.
4. The Add New RSLinx Driver dialog box appears.
5. Specify the driver name and click OK.

   The configure dialog box appears:

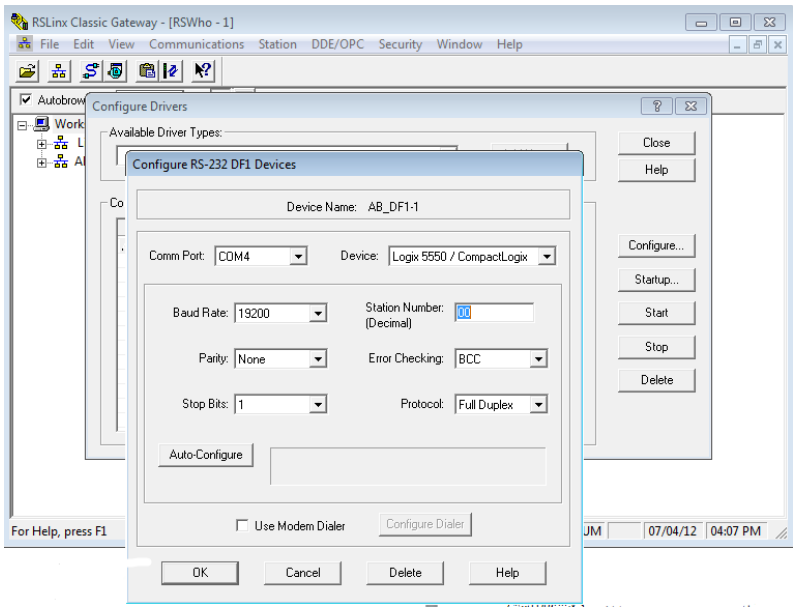


Figure 2.3: Configuring Communication

Specify the serial port settings.

- From the Com Port pull-down menu, choose the serial port on the workstation to that the cable is connected to (e.g. Port 1, Port 2, …).
- From the Device pull-down menu, choose Logix 5550-Serial Port.
- Click Auto-Configure.

Verify that the Auto-Configuration was successful. If it doesn't work, check if you have selected the correct port.

3.2 Configure the I/O modules:

In the next steps we intend to configure the physical modules in the project.

In order to establish a communication between the controller and an I/O module in the system, add the module to the I/O Configuration folder of the controller.

When you add a module, you also define a specific configuration for the module.

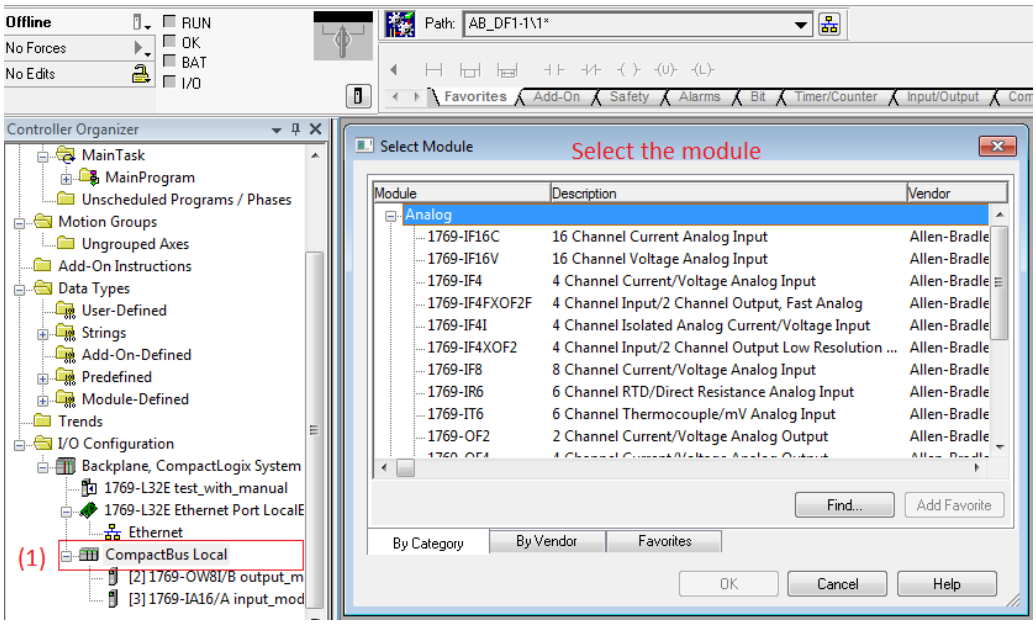


Figure 4. Configuring and adding modules to the project

3.3 Creating a program

a. Ladder Logic programming

In this section we will cover some of the basic and most used instructions in PLC programming. Following the explanations on how to program the controller, we will then have a look at how we can implement and use those instructions in real life applications.

Figure 5 shows a PLC ladder program. A PLC ladder program is a planned set of instructions resembling a hardwired ladder diagram. It consists of a line (L) power rail and a neutral (N) power rail between which one or more rungs are inserted.

Each individual rung contains one or more input instructions on its left-hand (L power rail) side, and a single output instruction or several output instructions placed in parallel on its right-hand (N power rail) side. In Figure 5, for example, the instructions Examine If Closed (XIC) and Examine If Open (XIO) are input instructions analogous to relay contacts. On the other hand, the instruction Output Energize (OTE) is an output instruction analogous to a relay coil.

The PLC ladder program is the main component of the project you download to a PLC. The PLC uses this program to interpret the signals present at its inputs and operate its outputs accordingly.

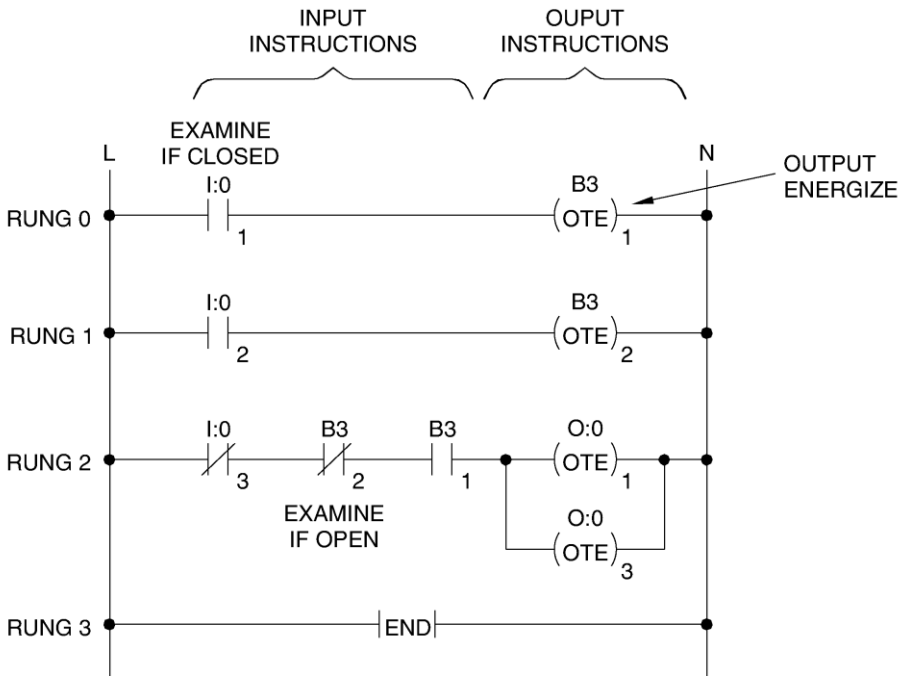


Fig 5. PLC ladder program

b. Logical Continuity.

During PLC operations, and in order to determine whether these PLC inputs are activated or deactivated, the processor reads (scans) the status of the signals applied to the PLC inputs, through the PLC internal input interface, The processor then updates the input data file (data file I1) bits accordingly. The processor then evaluates each rung of the ladder program

individually, updates the timer, binary status, counter, and control data, and then modifies the output data file (data file O0) bits accordingly. The output data file bits are used to energize or de-energize relays in the PLC internal output interface, causing these relays to apply or remove power to/from the devices connected to the PLC output interface terminals.
To evaluate a rung, that is, to determine if the rung is true or false, the processor verifies if a continuous left-to-right path of true input instructions exists between the line (L) and neutral (N) power rails.

c. Basics of PLC Programming

• When a continuous path of true input instructions exists, the rung is evaluated as true and the output instruction on this rung is true.

• When there is no continuous path of true input instructions on the rung, the rung is evaluated as false and the output instruction on this rung is false.

The status of a rung instruction (true or false) depends on the logic state of the data file bit this instruction is shown in Figure 6, for example, indicates the status of the instructions Examine If Closed (XIC) and Examine If Open (XIO), according to the logic state of the corresponding data file bit. From this figure, we can see that:

• The Examine If Closed (XIC) instruction is true when its associated bit is at logic state 1;

• Conversely, the Examine If Open (XIO) instruction is true when its associated bit is at logic state 0.

IF THE CORRESPONDING DATA FILE BIT IS	THE STATUS OF THE INSTRUCTION IS	
	XIC EXAMINE IF CLOSED	XIO EXAMINE IF OPEN
LOGIC 0	FALSE	TRUE
LOGIC 1	TRUE	FALSE

Fig 6. Truth table for the XIC and XIO input instructions

d. Series (AND) and Parallel (OR) Logics

The ladder rung in Figure 2 is an example of series (AND) logic. Series logic means that all the instructions in the rung (XIC I: 0/0 **AND** XIO I: 0/1) must be true in order for output instruction OTE O: 0/1 to be true.

The ladder rung in Figure 7 is an example of parallel (OR) logic. Parallel logic means that one or another path of true instructions must exist on the rung in order for the output instruction to be true. In Figure 7, at least one of the input instructions XIC I:0/1 **OR** XIC I:0/2 must be true in order for instruction OTE O:0/1 to be true. Parallel logic is programmed by branching instructions in a ladder rung.

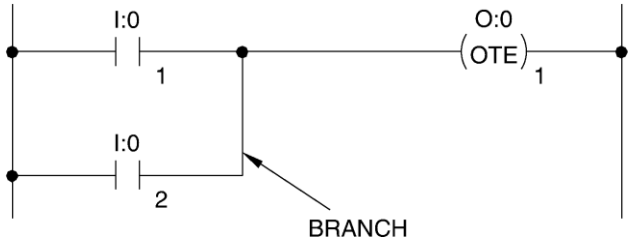


Fig 7. Series and parallel logics.

e. Documenting a Ladder Program

You can document a ladder program by inserting rung comments, instruction descriptions, and address descriptions. This allows you to keep notes on:

• How your ladder program works;

• The purpose of an instruction or a rung;

• The type of input or output device (pilot lamp, pushbutton, limit switch, etc.) associated with each address;

• The conditions required for a rung to be true.

You can insert your comments and descriptions while you enter a ladder program or after you have entered it. The three types of comments and descriptions which can be inserted are described below.

• The **rung comment**: normally used to determine what the rung is meant to do. It is displayed just over the rung in the ladder view window.

• The **instruction description**: used to determine what the instruction is meant to do or the conditions required for the instruction to be true. This description specifies the type and address of the instruction. All instructions of the same type that have a common address will automatically have the same instruction description. The instruction description is displayed over each instruction in the ladder view window.

• The **address description**: used to identify the type of input or output device associated with an address. All instructions having the same address will automatically have the same address

description. Note that address descriptions associated with instructions that are provided with an instruction description are not displayed in the ladder view. However, all address descriptions can be observed by opening the **Cross Reference** data file.

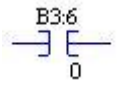
f. Some definitions:

The following is a list of the basic instructions in RSLogix series:

- XIC - Examine if Closed
- XIO - Examine if Open
- OTE - Output Energize
- OTL - Output Latch
- OTU- Output Unlatch
- OSR - One-Shot Rising

XIC Examine if Closed

Symbol



Definition

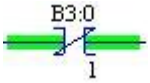
- Examines a bit for an On condition
- Use the XIC instruction in your ladder logic to determine if a bit is ON.
- 0 = False
- 1 = True

Devices

- Start/Stop push buttons
- Selectors
- Limit switch
- Proximity switch
- Light
- Internal bit

XIO Examine if Open

Symbol



Definition

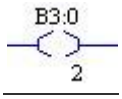
- Examines a bit for an off condition.
- Use an XIO instruction in your ladder logic to determine if a bit if off.
- 1 = True
- 0 = False

Devices

- Start/Stop push buttons
- Selectors
- Limit switch
- Proximity switch
- Light
- Internal bit

OTE Output Energize

Symbol



Definition

- Turns a bit on or off
- Use OTE instruction in your ladder logic to turn on a bit when rung condition is evaluated as true.

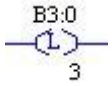
Devices

- Light
- Motor run signal
- Internal bits

OTL Output Latch

Usually we latch a signal with a condition and then unlatch it when a different condition becomes true. Most of time the Latch / Unlatch go together.

Symbol

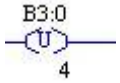


Definition

- Turns a bit on when the rung is executed, and this bit retains its state when the rung is not executed or a power cycle power occurs.
- OTL is a retentive output instruction. OTL can only turn on a bit. This instruction is usually used with OTU with both OTL and OTU addressing the same bit.
- Ladder logic can examine a bit controlled by OTL as often as necessary.
- When you assign an address to the OTL instruction that corresponds to the address of a physical output, the output device wired to the screw terminal is energized when the bit is set. When rung conditions become false, the bit remains set and the corresponding output device remains energized.
- Actuating the latch input turns the function on or causes it to change state. The function then stays on even if the latch input is turned off. To turn the function off, another input must unlatch which turns the function off.

OTU Output Unlatch

Symbol



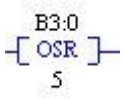
Definition

- Turns a bit off when the rung is executed, and this bit retains its state when the rung is not executed or when power cycle occurs.
- OTU is a retentive output instruction. OTU can only turn off a bit. This instruction is usually used with OTL with both OTL and OTU addressing the same bit.
- Ladder logic can examine a bit controlled by OTU as often as necessary.

- When you assign an address to the OTU instruction that corresponds to the address of a physical output, the output device wired to the screw terminal is de-energized when the bit is cleared.
- The unlatch instruction tells the controller to turn off the addressed bit. Thereafter, the bit remains off, regardless of the rung condition, until it is turned on.

OSR One-Shot Rising

Symbol



Definition

- Triggers a one-time event.
- The OSR instruction is a retentive input instruction that triggers an event to occur only one time. Use the OSR instruction when an event must start based on change of state of the rung from false to true.
- When the input instruction goes from false to true, the OSR instruction conditions the rung so that the output goes true for one scan. The output goes false and remains false for successive scans until the input makes another false to true transition.

3. Introduction to Counters

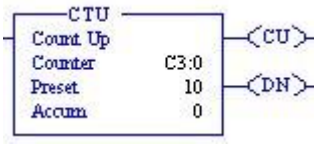
In this section we will cover the count up count down and reset instruction. Counters are very essential in ladder logic programming. Counters are used to index, increment or decrement values.

The following is a list of counter instructions:

- CTU - Count Up
- CTD - Count Down
- RES - Reset

CTU Count UP

Symbol



Definition

- Increments the accumulated value at each false to true transition and retains the accumulated value when the instruction goes false or when power cycle occurs.
- The CTU is an instruction that counts false to true transition. When this transition happens the accumulated value is incremented by one count.
- A CTU accumulation is reset by the RES instruction.
- If the accumulation value is over the maximum range then the overflow (OV) bit will be true.

Each counter address is made of a 3-word element.

Word 1 is the control word

- Bit 0-7: Internal Use
- Bit 10: UA - Update accumulation value.
- Bit 11: UN - Underflow bit.
- Bit 12: OV - Overflow bit.
- Bit 13: DN - Done
- Bit 14: CD - Count down is enabled.
- Bit 15: CU - Count up is enabled.

Word 2 stores the preset value. (PRE)

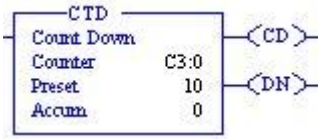
- Specifies the value, which the counter must reach before the controller sets the done bit. When the accumulator value becomes equal to or greater than the preset value, the done status bit is set. You can use this bit to control an output device.
- Preset value is from -32,768 to 32,767
- If a timer-preset value is negative an error will occur.

Word 3 stores the accumulated value. (ACC)

- This is the number of times of false to true transitions that have occurred since the counter was last rest.

CTD Count Down

Symbol



Definition

- Decrements the accumulate value at each false to true transition and retains the accumulated value when the instruction goes false or when power cycle occurs.
- The CTD is an instruction that counts false to true transition. When this transition happens the accumulated value is decrements by one count.
- A CTD accumulation is reset by the RES instruction.
- If the accumulation value is below the minimum range then the underflow (UN) bit will be true.

· Each counter address is made of a 3-word element.

Word 1 is the control word

- Bit 0-7: Internal Use
- Bit 10: UA - Update accumulation value.
- Bit 11: UN - Underflow bit.
- Bit 12: OV - Overflow bit.
- Bit 13: DN - Done
- Bit 14: CD - Count down is enabled.
- Bit 15: CU - Count up is enabled.

Word 2 stores the preset value. (PRE)

- Specifies the value, which the counter must reach before the controller sets the done bit. When the accumulator value becomes equal to or greater than the preset value, the done status bit is set. You can use this bit to control an output device.
- Preset value is from -32,768 to 32,767
- If a timer-preset value is negative an error will occur.

· Word 3 stores the accumulated value. (ACC)

RES Reset

Symbol

Definition

- Resets the accumulated value and status bit of a timer or counter.
- Use a RES instruction to reset timers or counters. When the RES instruction is enabled, it resets the Timer On Delay, Retentive Timer, and Counter Up, Counter Down instruction having the same address as the RES instruction.

4. Introduction to Timers

In this section we will cover timers used in ladder logic programming. Timers are very important in ladder logic programming. Timers give the precision in time. Timer on delay starts timing when instruction is true. Timers are used to track time when instruction is on or off. They could also keep track on a retentive base.

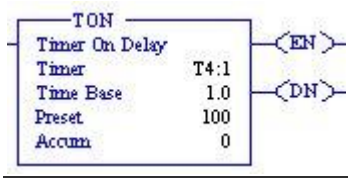
Definition

The following is a list of timer instructions:

- TON - Timer On Delay
- TOF - Timer Off Delay
- RTO - Retentive Timer

TON Timer On Delay

Symbol



Definition

- Count time base intervals when the instruction is true.

- The Timer on Delay instruction begins to count time base intervals when rung conditions become true. As long as rung conditions remain true, the timer adjust its accumulated value (ACC) each evaluation until it reaches the preset value (PRE). The accumulated value is reset when rung conditions go false, regardless of whether the timer has timed out.

Each Timer on Delay is made of a 3-word element.

Word 1 is the control word

- Bit 0-12: Internal Use
- Bit 13: Done (DN) this bit is on when the Accumulation value >= Preset Value
- Bit 14: Timer Timing (TT) this bit is on when the timer is timing
- Bit 15: Enabled (EN), this bit is on when the timer is energized.

Word 2 stores the preset value. (PRE)

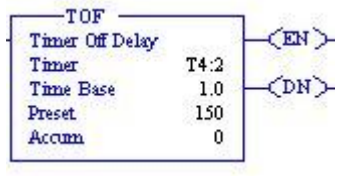
- The programmer specifies this value. When the accumulated time reaches the preset value the controller sets the done bit. When the accumulated value becomes equal to or greater than the preset value, the done bit is set. Usually preset value is from 0 - 32,767
- If a timer-preset value is negative an error will occur.

· Word 3 stores the accumulated value. (ACC)

- This is the time elapsed since the timer was last reset. When enabled the timer updates this continually.
- Time Base: is the timing update interval, this can vary from 0 - 1 second.

TOF Timer Off Delay

Symbol



Definition

- Counts time base intervals when the instruction is false.
- The Timer Off Delay instruction begins to count time base intervals when the rung makes a true to false transition. As long as rung conditions remain false, the timer increments its accumulated value (ACC each scans until it reaches the preset value

(PRE). The accumulated value is reset when rung conditions go true regardless of whether the timer has timed out.

Each timer address is made of a 3-word element.

Word 1 is the control word

- Bit 0-12: Internal Use
- Bit 13: DN- Done
- Bit 14: TT - Timer Timing
- Bit 15: EN - Timer is enabled

Word 2 stores the preset value. (PRE)

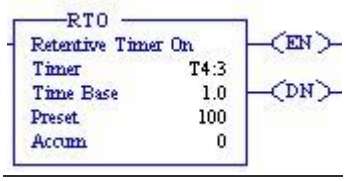
- Specifies the value, which the timer must reach before the controller sets the done bit. When the accumulated value becomes equal to or greater than the preset value, the done bit is se.
- Preset value is from 0 - 32,767
- If a timer-preset value is negative an error will occur.

Word 3 stores the accumulated value. (ACC)

- This is the time elapsed since the timer was last reset. When enabled the timer updates this continually.
- Time Base: is the timing update interval, this can vary from 0 - 1 second.

RTO Retentive Timer

Symbol



Definition

- Counts time base intervals when the instruction is true and retains the accumulated value when the instruction goes false or when power cycle occurs.
- The Retentive Timer instruction is a retentive instruction that begins to count time base intervals when rung conditions become true.
- The Retentive Timer instruction retains its accumulated value when any of the following occurs:
- Rung conditions become false.

- Changing Processor mode from REM run /Test / program mode.
- The processor loses power while battery backup is still maintained and a fault occurs.

Note: To reset the accumulated value in RTO, you must use a reset instruction (RES) with the same address.

Each Retentive Timer is made of a 3-word element.

Word 1 is the control word

- Bit 0-12: Internal Use
- Bit 13: DN- Done
- Bit 14: TT - Timer Timing
- Bit 15: EN - Timer is enabled

Word 2 stores the preset value. (PRE)

- Specifies the value, which the timer must reach before the controller sets the done bit. When the accumulated value becomes equal to or greater than the preset value, the done bit is se.
- Preset value is from 0 - 32,767
- If a timer-preset value is negative an error will occur.

Word 3 stores the accumulated value. (ACC)

- This is the time elapsed since the timer was last reset. When enabled the timer updates this continually.
- Time Base: is the timing update interval, this can vary from 0 - 1 second.

5. An Introduction to RSLogix5000 Tags

Tags are the method for assigning and referencing memory locations in Allen Bradley Logix5000 controllers. No longer are there any physical addresses such as N7:0 or F8:7 which use symbols to describe them. These have been replaced with tags that have a pure text based addressing scheme. This is a departure from the more conventional ways of programming PLCs, which includes Allen Bradley earlier line of PLC5 and SLC 500 controllers.

One of the hardest transitions from the older systems is realizing how the tag database works. The person with experience in Allen Bradley systems will recognize many of the instructions and be at home with the editor in RSLogix 5000. Understanding the tag database is the first major hurdle in becoming comfortable with the ControlLogix and CompactLogix systems.

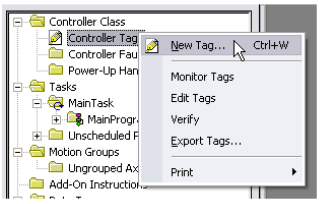
Earlier Allen Bradley PLCs programmed with RSLogix 5 and RSLogix 500 software had data files to store I/O and other internal values. These different data files could only hold one data type. A data type defines the format and the size of the stored value.

The Logix5000 controllers have done away with data files and in its place is the tag database. The tag database organizes memory locations in one place. Each tag is assigned its own data type. The table below shows the association between the current data types and the older systems with data files.

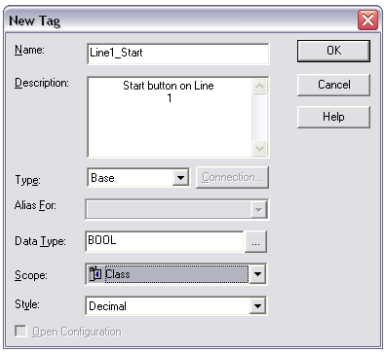
Type	RSLogix 5000
Output Input	Input and output modules, when configured, automatically create their own tags like Local:0:I.Data.0
Status	Use the GSV and SSV instructions to get status information such as the CPU time, module states and scan times.
Bit	Assign the Boolean (BOOL) data type to the tag.
Timer	Assign the TIMER data type to the tag.
Counter	Assign the COUNTER data type to the tag.
Control	Assign the CONTROL data type to the tag.
Integer	Assign the double integer (DINT) data type to the tag.
Floating Point	Assign the REAL data type to the tag.

Creating a Tag

One way to create a new tag is right click on the Controller Tags in the Controller Organizer and select New Tag. Even faster is the Ctrl+W hot key.



The following dialog box pops up.



The Name given to the tag has the following rules:

- only alphabetic characters (A-Z or a-z), numeric characters (0-9), and underscores (_)
- must start with an alphabetic character or an underscore
- no more than 40 characters
- no consecutive or trailing underscore characters (_)
- not case sensitive
-

The tag Type defines how the tag operates in the project

Base	A tag that actually defines the memory where the data is stored
Alias	A tag that represents another tag
Produced	Send data to another controller
Consumed	Receive data from another controller

Alias tags mirror the base tag to which they refer. When the base tag value changes so does the alias tag. Use aliases in the following situations:

- program logic in advance of wiring diagrams
- assign a descriptive name to an I/O device
- provide a more simple name for a complex tag
- use a descriptive name for an element of an array

Produced and consumed tags make it possible to share tags between controllers in the same rack or over a network. This article does not cover this aspect.

Select a Data Type for the tag by typing it in or by clicking on the ellipsis button and selecting it from the list. A data type is a definition of the size and layout of memory allocated for the created tag. Data types define how many bits, bytes, or words of data a tag will use.

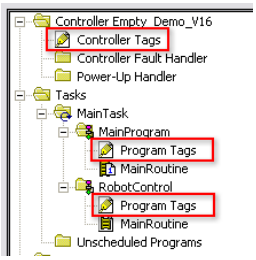
The term Atomic Data Type refers to the most basic data types. They form the building blocks for all other data types.

Data Type	Abbreviation	Memory bits	Range
Boolean	BOOL	1	0-1
Short Integer	SINT	8	-128 to 127
Integer	INT	16	-32,768 to 32,767
Double Integer	DINT	32	-2,147,483,648 to 2,147,483,647
Real Number	REAL	32	+/-3.402823E38 to +/-1.1754944E-38

Logix5000 controllers are true 32-bit controllers, meaning the memory words are 32-bits wide. No matter what, a tag always reserves 32 bits of memory even if it is a Boolean or integer data type. For this reason, it is best to use a DINT when dealing with integers. Furthermore, a Logix5000 controller typically compares or manipulates values as 32-bit values (DINTs or REALs).

A Logix5000 controller lets you divide your application into multiple programs, each with its own data. The Scope of the tag defines if a tag is global (controller tags) and therefore available to all programs or local (program tags) to a select program group. Pay careful

attention to this field as creating it in the wrong area may lead to some confusion later on as to its location.



Controller Tags are available to all programs. You cannot go wrong using controller scoped tags unless you easily want to copy and paste programs. A tag must be controller scoped when used in a Message (MSG) instruction, to produce or consume data and to communicate with a PanelView terminal.

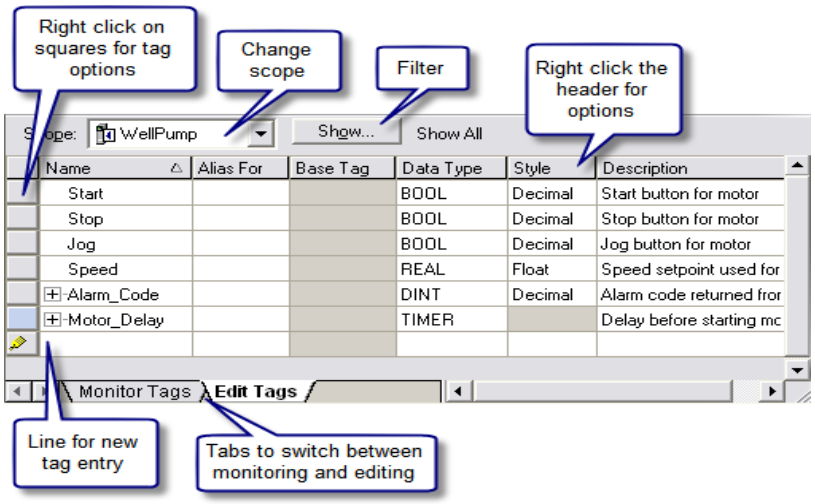
Program Tags are isolated from other programs. Routines cannot access data that is at the program scope of another program. Having program tags make it easy to copy/paste programs and not have to worry about conflicting tag names. Make sure though that no controller tags are named the same as program tags.

Style is the form in which to display the tag by default. The following table provides you with information on the base and notation used for each style.

Style	Base	Notation
Binary	2	2#
Decimal	10	
Hexadecimal	16	16#
Octal	8	8#
Exponential		0.0000000e+000
Float		0.0

Edit and Monitor Tags

To edit existing tags select the *Logic* > *Edit Tags* menu item. A spread sheet like view lets you create and edit tags.



Clicking the + sign next to a tag reveals its structure. For a DINT tag this is the 32 individual bits that make up the tag which will not be of interest if you are using the tag as a number rather than individual bits. If you do wish to use the individual bits then you can address them in this way with the tag name followed by a period and then the bit position (e.g. MyTag.5). Shown below is the expanded structure for a TIMER. Notice it is made of two DINTs and three BOOLs. In this case, the Booleans are packed into one DINT and therefore a timer uses three DINTs of memory.

Conclusion:

These are the basics of tags. The advantages are:

1. Tags, if done right, create a level of documentation that is stored in the PLC.
2. The software does an automatic housekeeping of memory locations. There is no more worrying about physical addressing and memory conflicts.
3. Structures can be more easily put together based on function rather than data type.

Advance subjects include arrays, user defined data types (UDT) and Add-On Instructions. Hopefully, you will continue to learn more about the power of tags. There is no doubt that if you grasp the principles presented here you will be well on your way to using and troubleshooting any Logix5000 controller.

A Quick Tutorial on RSLogix Emulator 5000

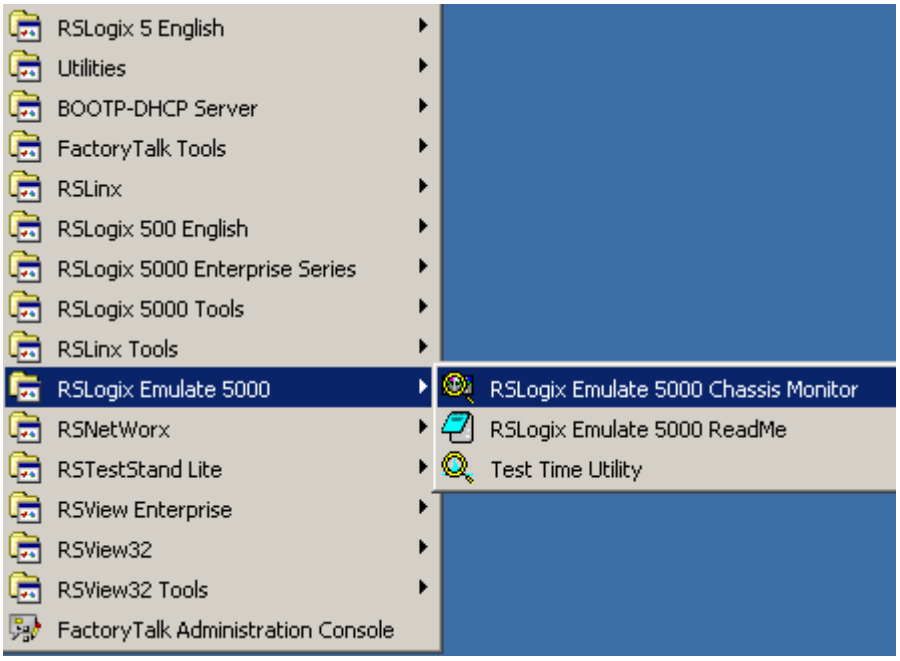
RSLogix Emulator 5000 is a software simulator for the Allen Bradley line of Logix 5000 controllers (ControlLogix®, CompactLogix®, FlexLogix®, SoftLogix5800® and DriveLogix®). The goal is to mimic the function of a PLC without the actual hardware and thus do advanced debugging.

As a quick introduction we'll go through a simple example of setting up a simulation. This involves three major steps.

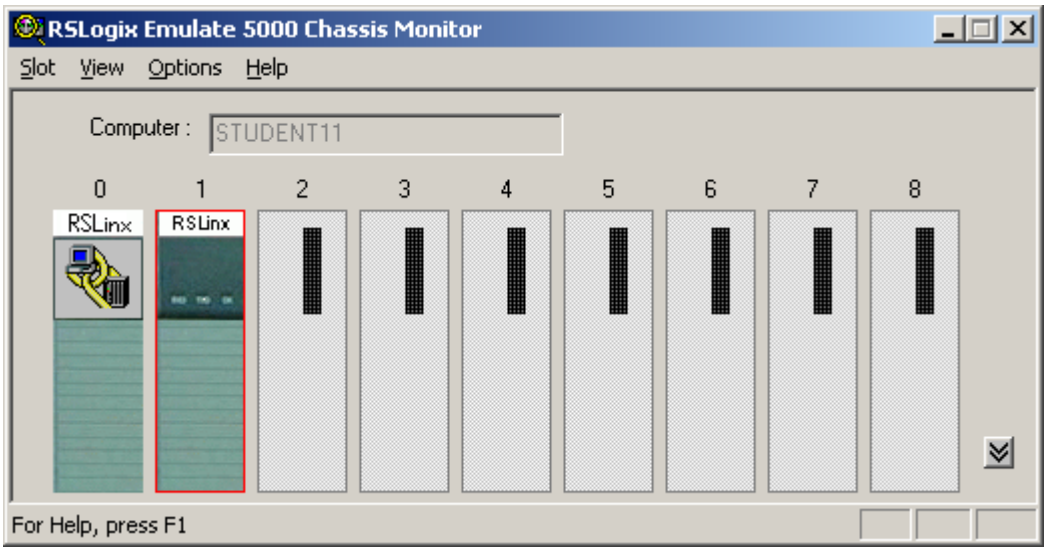
1. Setting up the chassis monitor.
2. Creating a connection in RSLinx.
3. Creating a project with associated emulation hardware.

Setting up the Chassis Monitor

To start the Chassis Monitor, click Start > Programs > Rockwell Software > RSLogixEmulate 5000 > RSLogix Emulate 5000 Chassis Monitor.

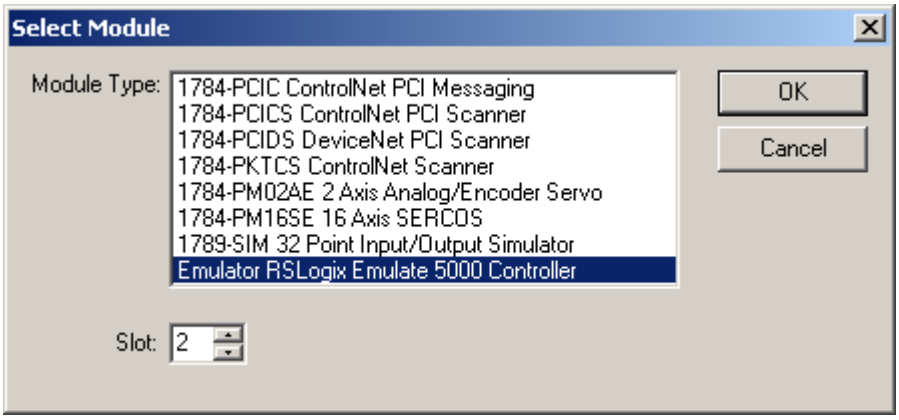


When the emulator opens up you're confronted with what looks like an empty chassis. In slot 0 is an RSLinx module which has to be there for the emulator communications to work. Your slot 1 might have another irremovable RSLinx module depending if you are running RSLogix Enterprise.



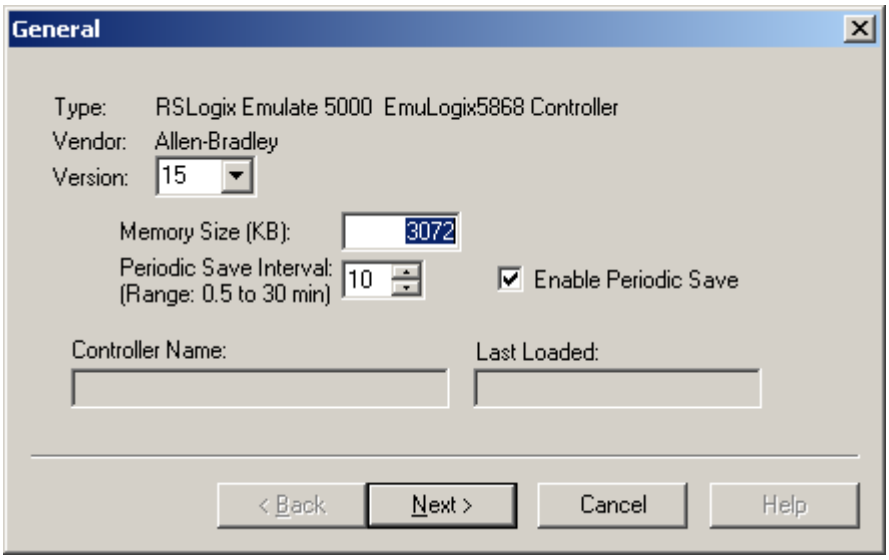
From here we set up our hardware configuration for simulation. Our first step will be to add the CPU. In this case it is a special one called an Emulation Controller.

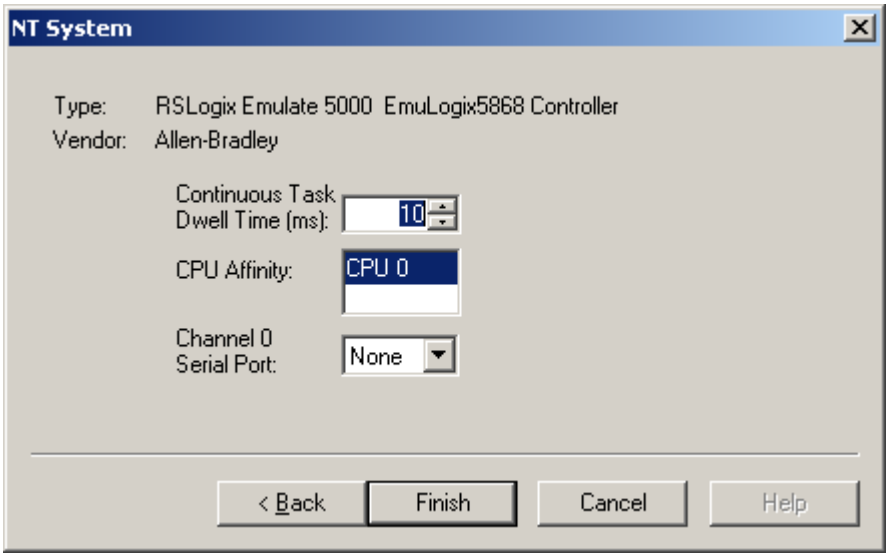
1. Click Slot > Create Module.
2. Choose the Emulator RSLogix Emulate 5000 Controller.
3. Chose slot 2 for the controller
4. Click OK to add it to the chassis monitor.



5. At this point you may be accosted with a message about previous configurations. Just select Reset the Configuration to Default Values and click NEXT.

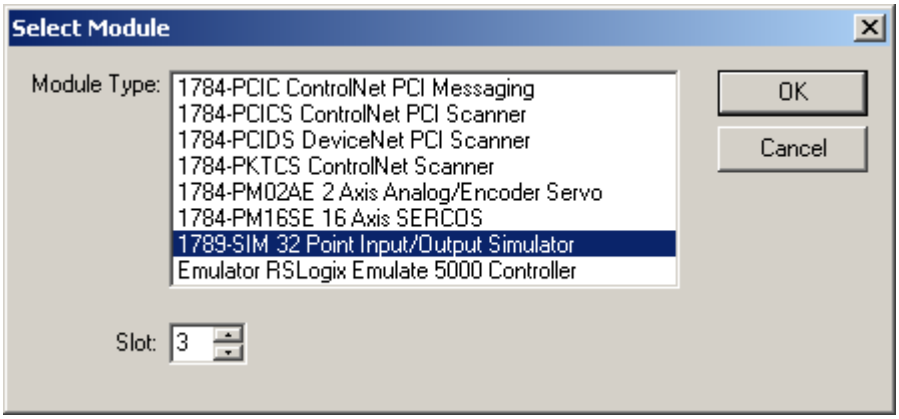
6. The next two dialog screens are for setting up the controller details. Click NEXT and FINISH accepting all the defaults.



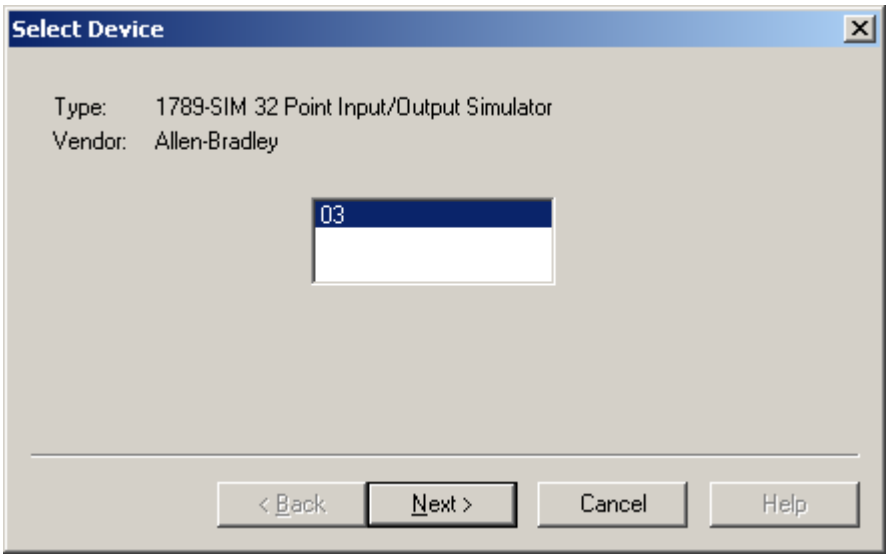


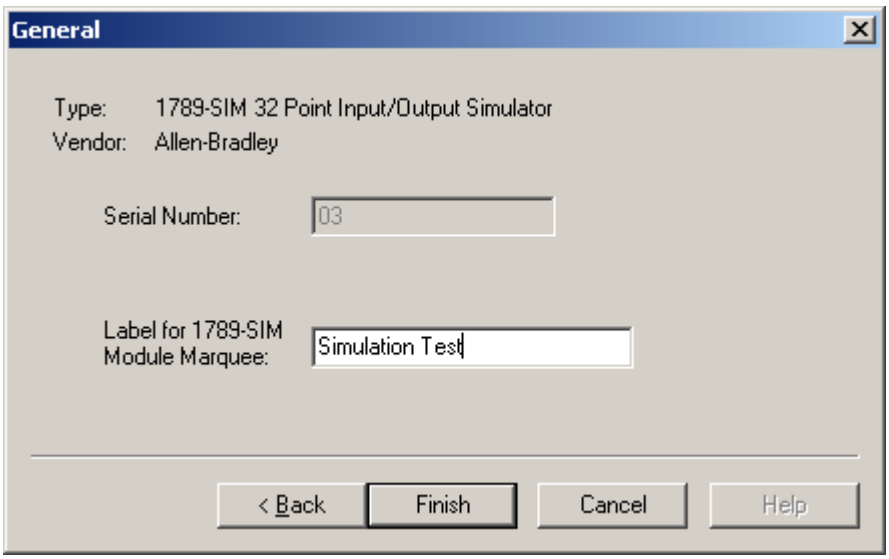
Next we'll add some input/output simulation.

1. Click Slot > Create Module.
2. Choose the 1789-SIM 32 Point Input/output Simulator.
3. Chose slot 3 for the simulator and click OK.

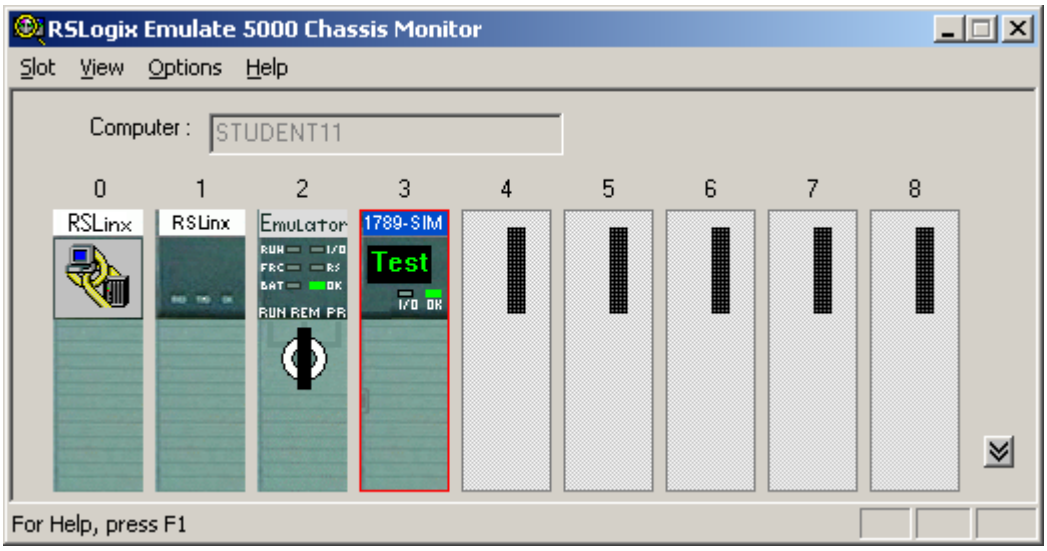


4. Accept the defaults for the setup by clicking NEXT and FINISH.



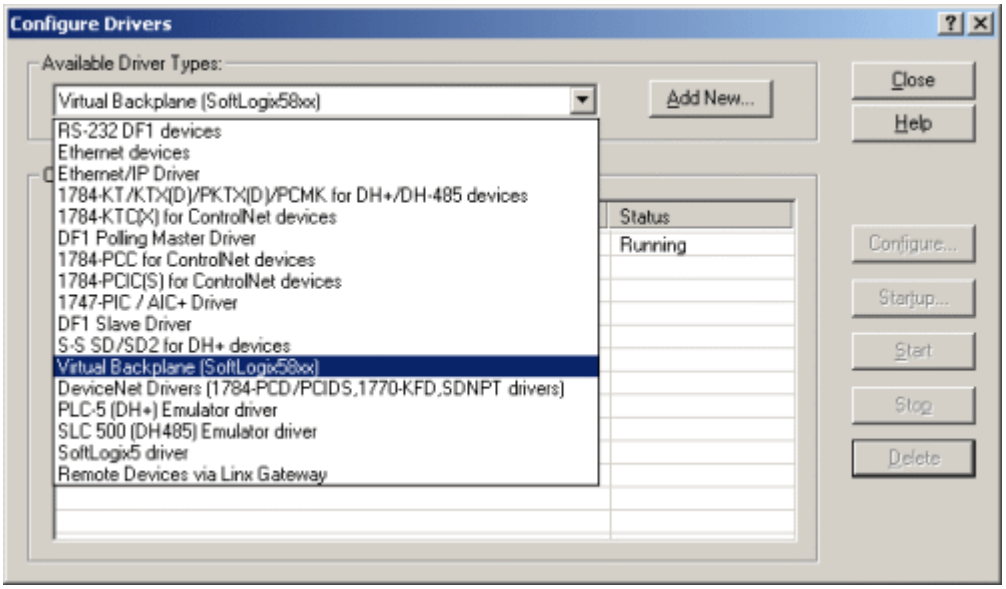


The chassis monitor will now have two emulation modules in it ready to go.



Creating a connection in RSLinx

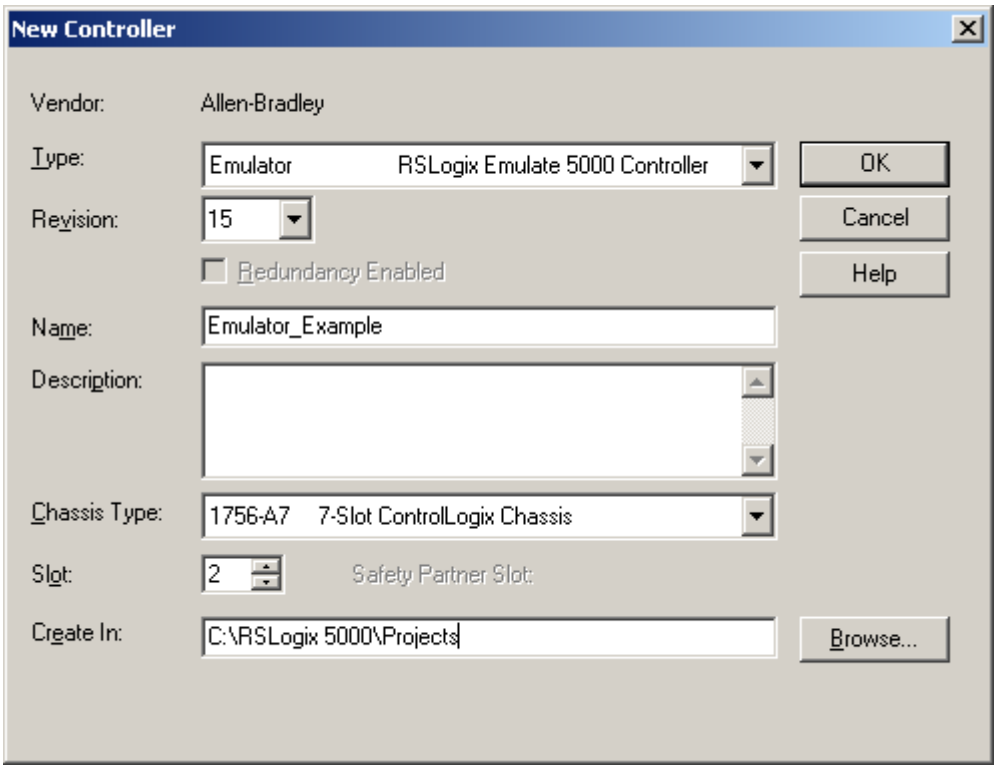
1. Start RSLinx under Start > Programs > Rockwell Software > RSLinx > RSLinx Classic
2. Click Communications > Configure Drivers.
3. Select the Virtual Backplane (SoftLogix 58xx) driver from the Available Driver Types list.
4. Click Add New. The Add New RSLinx Driver dialog box appears. Click OK.
5. The new driver appears in the Configured Drivers list. Click Close.



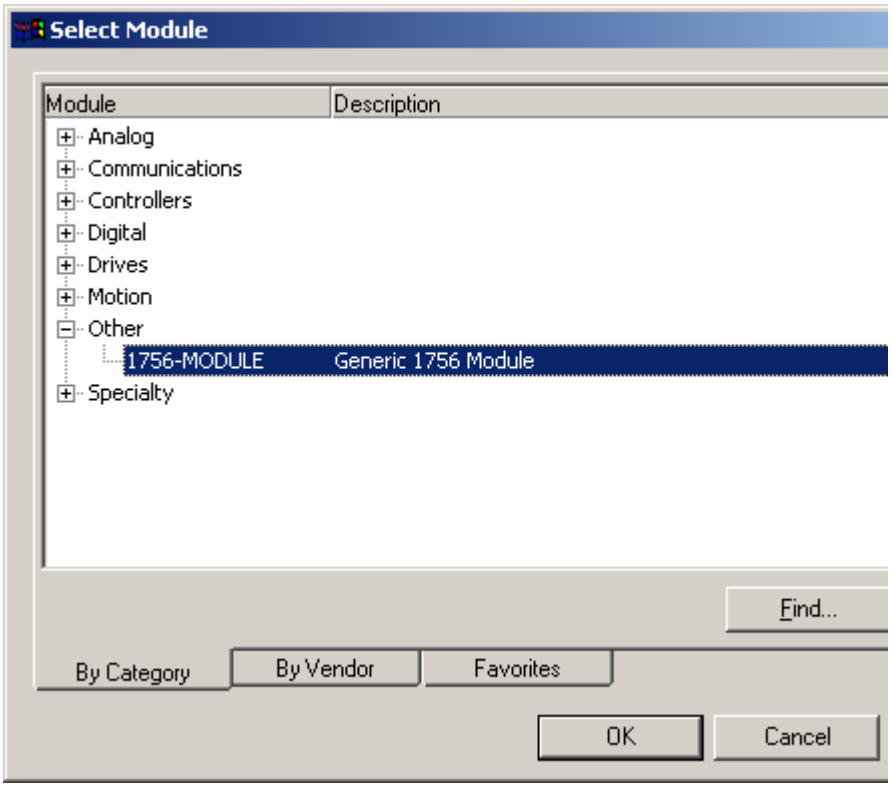
Using RSLogix Emulator in a Project

To use the emulator in a project you must setup the hardware correctly.

1. Start the RSLogix 5000 software and create a new project.
2. Under the New Controller window type select an Emulator – RSLogix Emulator 5000 Controller. Give it a name and assign it to the same slot as the one you put in the Chassis Monitor which in our example is slot 2. Click OK.

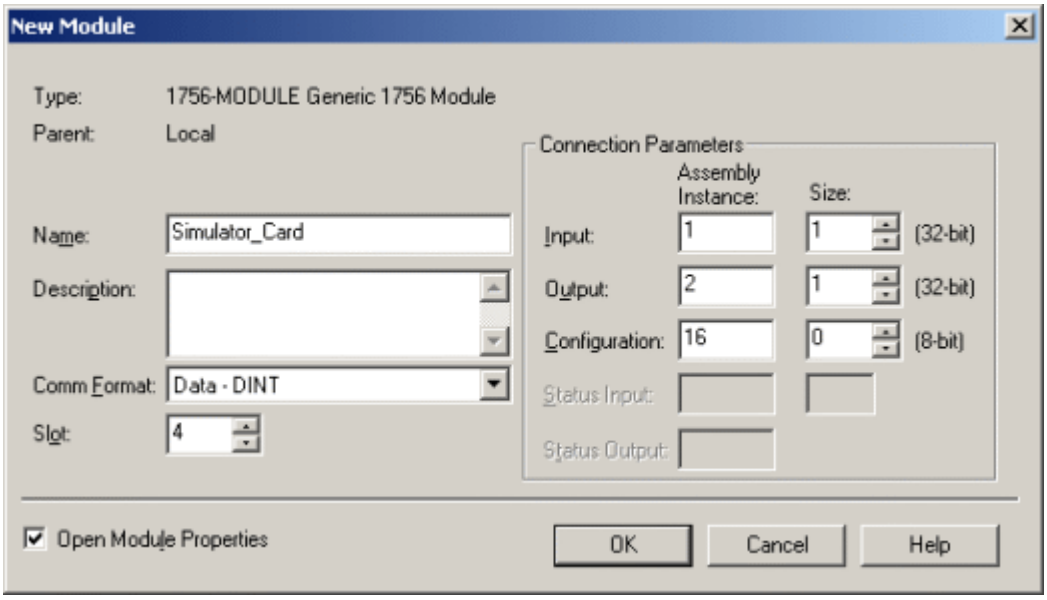


3. In RSLogix 5000's Controller Organizer, right click on the I/O Configuration folder, and then click New Module. The software displays the Select Module window.
4. Open the Other folder. Select the 1756-MODULE from the modules list and then click OK.

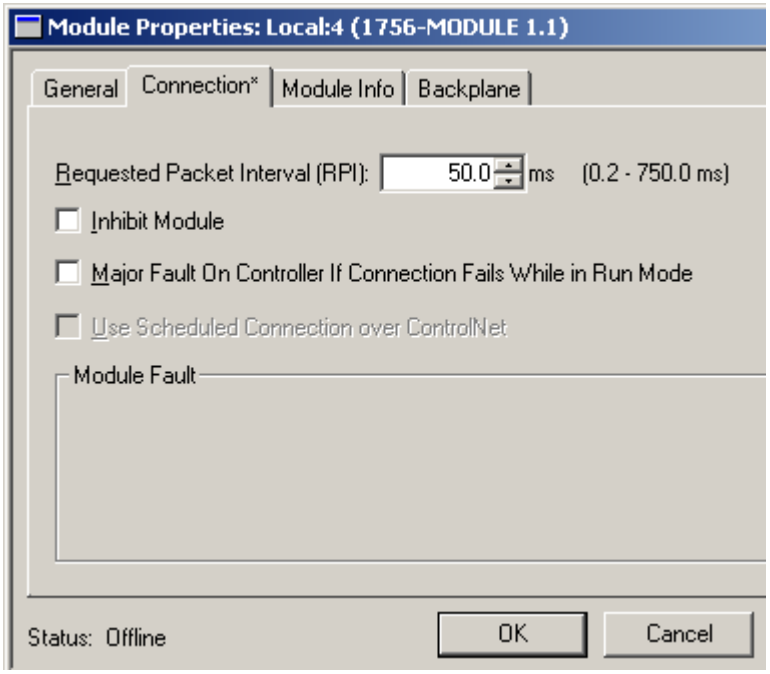


5. The software displays the New Module window.
   a. Add a Name for the card.
   b. In the Slot field put the number that corresponds with the Chassis Monitor.
   c. For the Connection Parameters put in the following and click OK

	Assembly Instance	Size
Input	1	2
Output	2	1
Configuration	16	0

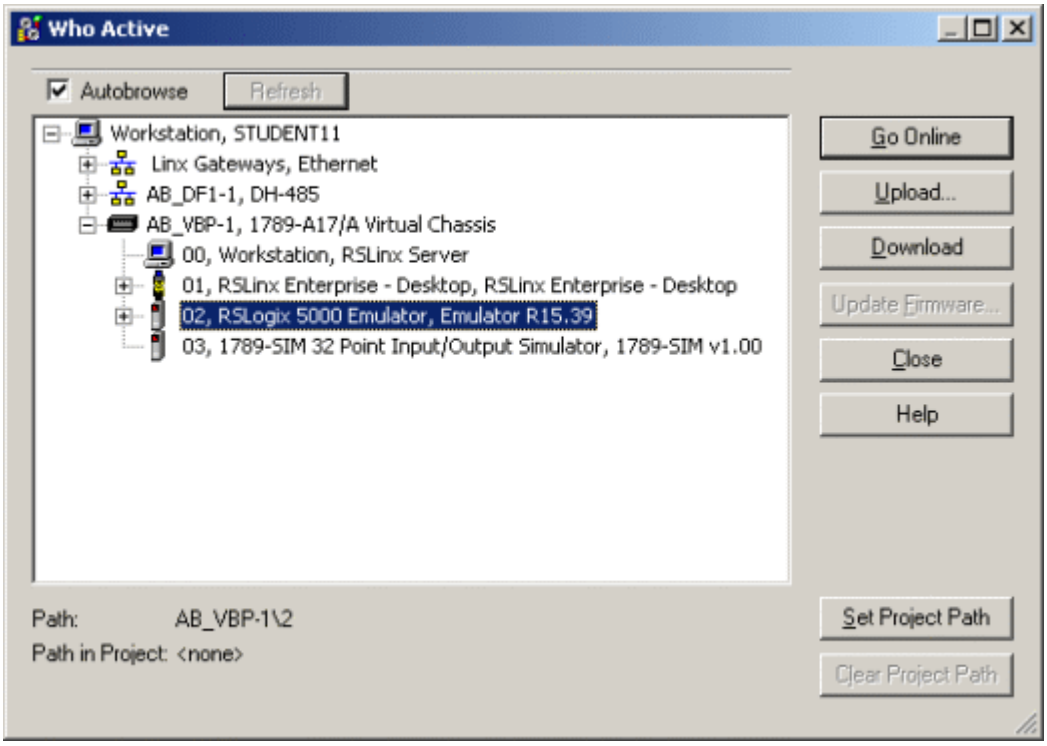


6. On the next Module Properties screen make sure to change the Requested Packet Interval to 50.0 ms.

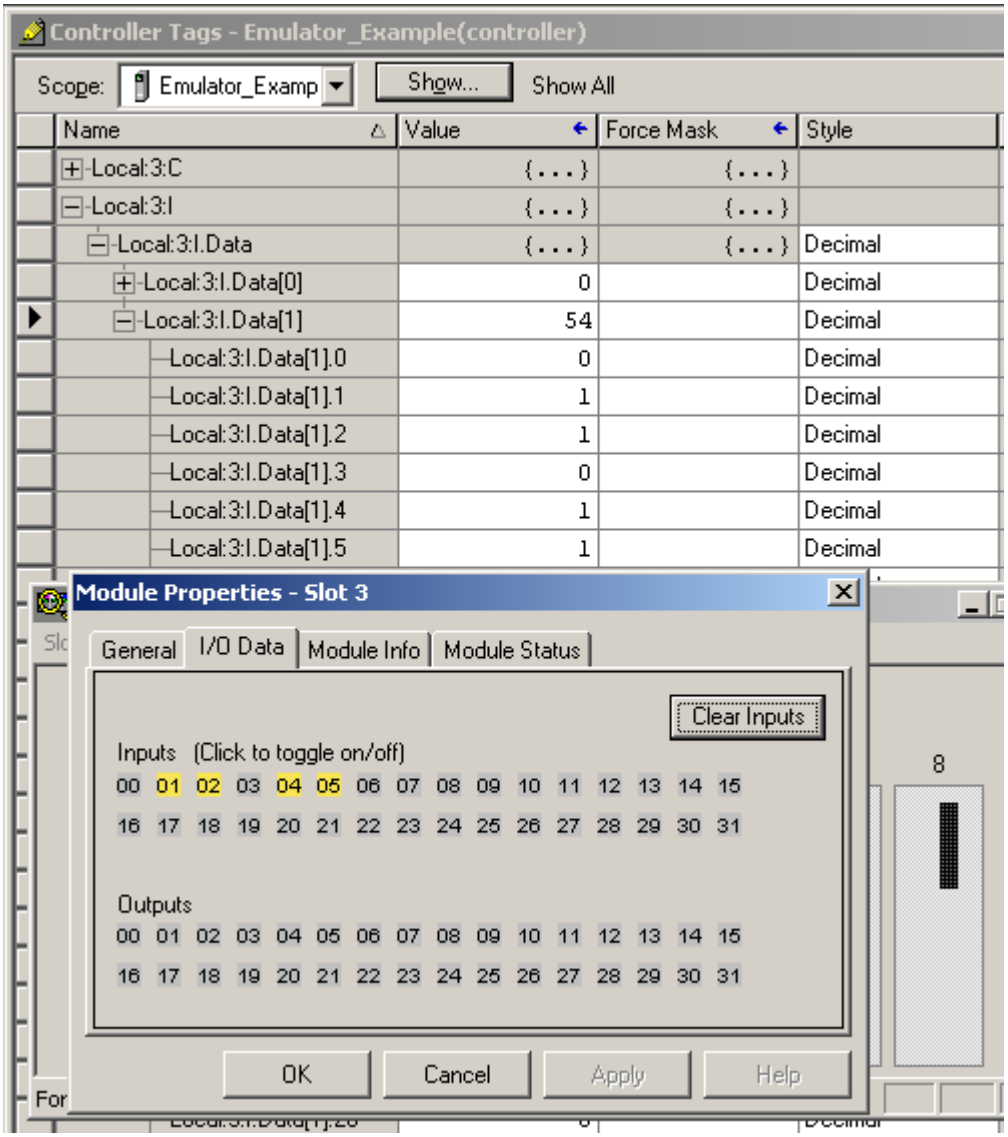


Ready, Set, Go

You are now ready to use the emulator just like you would any other PLC. Open Who Active and set the path to the RSLogix 5000 Emulator.



The inputs can be simulated in the emulator's Chassis Monitor by right clicking on the module and selecting Properties. Under the I/O Data tab is the ability to toggle each of the inputs on or off.


Controller Tags - Emulator_Example(controller)
Scope: Emulator_Examp
Show...
Show All
Name
Value
Force Mask
Style
Local:3:C
Local:3:I
Local:3:I.Data
Local:3:I.Data[0]
Local:3:I.Data[1]
Local:3:I.Data[1].0
Local:3:I.Data[1].1
Local:3:I.Data[1].2
Local:3:I.Data[1].3
Local:3:I.Data[1].4
Local:3:I.Data[1].5
Decimal
Module Properties - Slot 3
General
I/O Data
Module Info
Module Status
Clear Inputs
Inputs (Click to toggle on/off)
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Outputs
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
OK
Cancel
Apply
Help
8

Ladder Logic programming with RSLogix 5000 and RSEmulator 5000

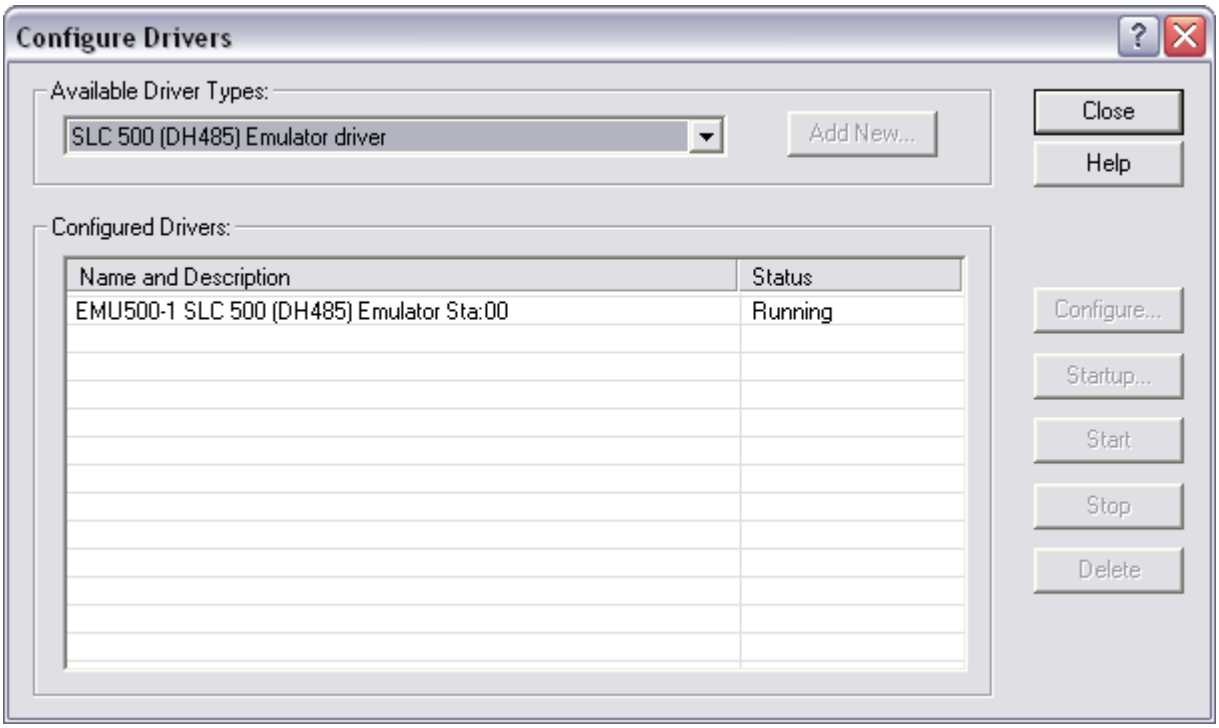
First Step with RSLinx

RSLinx is the software RSLogix will use to communicate with your PLC or in our case to the emulator.

Let's start by running the RSLinx software under the *START > All Programs > Rockwell Software > RSLinx > RSLinx Classic*. A Follow these steps to set it up:

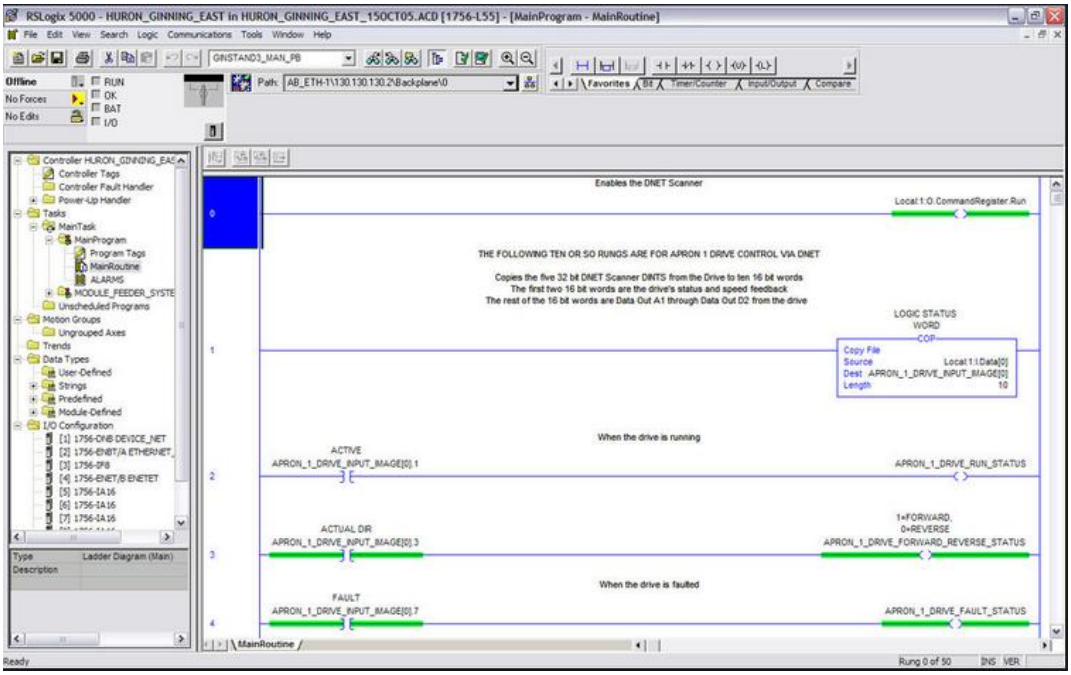
1. Under the Communications menu select Configure Drivers.
2. Under the Available Drivers Types select the PLC Emulator driver and click the Add button.
3. You can give the driver a name but I just leave it at the default.
4. Leave the configuration options as Station Number 00 and click OK.

Your driver should now be running and look likes the picture below.

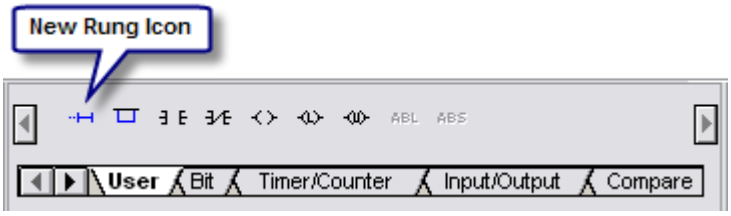


Second step running RSLogix 5000

START > All Programs > Rockwell Software >RSLogix 5000.



Let's make a simple rung to test. Make sure the cursor is on the rung with the END on it and then click the New Rung icon in the instruction toolbar.

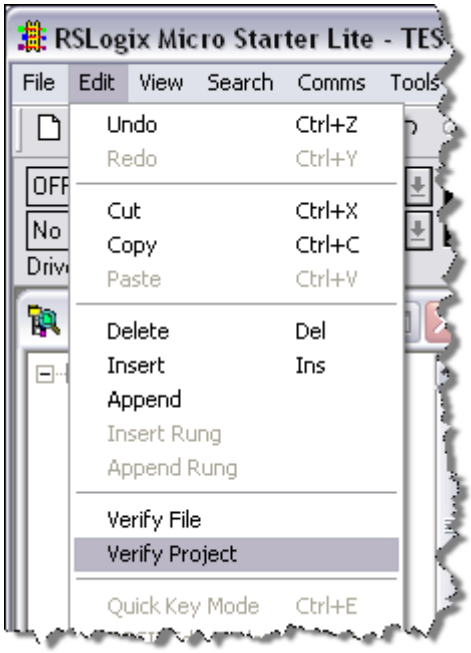


Now click on the Examine if Closed • instruction to add it to the rung. Â Double click on the question mark above it and enter I:0/0 as its input address. Just leave the description pop up box empty by clicking OK.

Next, click on the Output Energize instruction to add it to the right side of the rung. Double click on the question mark above it and enter O:0/0 as its output address. Just leave the description pop up box empty by clicking OK.

You should now have something like below.

The next very important step is to verify the project with the *Edit > Verify Project* menu item. This will compile the project and get it ready for the emulator.



Save the project as something like Test.

RSEmulator 5000

The emulator allows us test our work by running a virtual PLC and help us to download our program to it and run it in a very similar fashion to a real PLC. Start the emulator with the

START > All Programs > Rockwell Software > RSLogix Emulate 500 > RSLogix Emulate 5000

Set the Emulator up as it has been explain above in Emulator part.

Connecting Excel to ControlLogix

By John Schop at http://www.plcdev.com/connecting_controllogix_excel

Have you ever lost data in a CLX processor, because you downloaded new code? Unfortunately, when you download a program to a ControlLogix processor, you also download the values of the tags (variables).

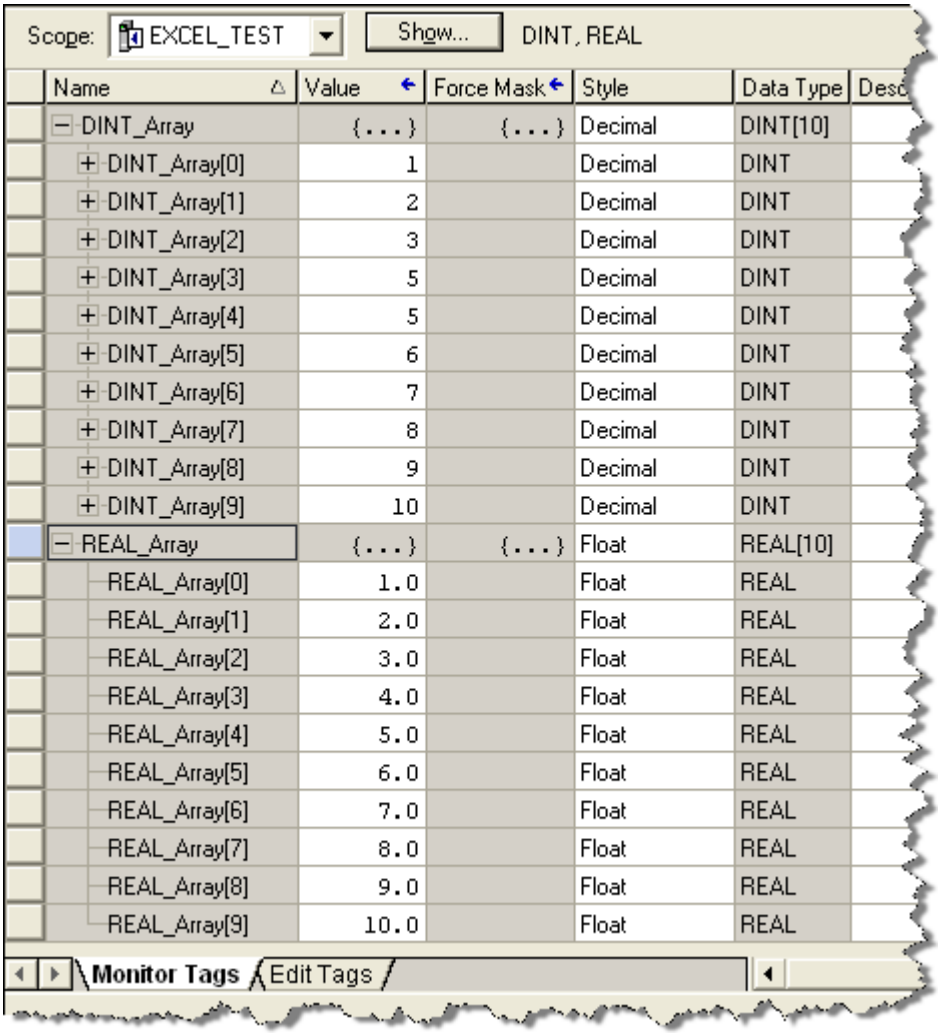
A solution to this problem that could be useful is an Excel sheet that reads and writes values to the ControlLogix processor using the DDE/OPC capabilities of RSLinx.

In this article, I will show you how to create one of these sheets for your projects.

Here's what you'll need:

- Microsoft Excel, with some basic knowledge about programming macro's in Visual Basic
- RSLinx (not the 'Lite' version, because that does not have DDE/OPC capabilities)
- A ControlLogix processor of course

Let's fire up RSLogix first, and create a bunch of tags with values. In this example, I created 2 arrays, of the types DINT and REAL, each with a length of [10] tags. These arrays I filled with some values:



Scope: EXCEL_TEST | Show... | DINT, REAL

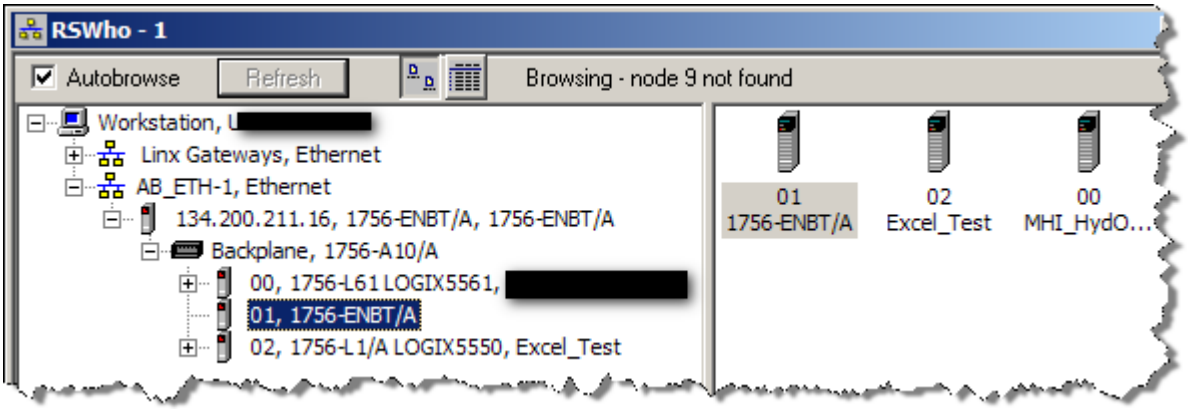
Name	Value	Force Mask	Style	Data Type	Desc
DINT_Array	{...}	{...}	Decimal	DINT[10]	
DINT_Array[0]	1		Decimal	DINT	
DINT_Array[1]	2		Decimal	DINT	
DINT_Array[2]	3		Decimal	DINT	
DINT_Array[3]	5		Decimal	DINT	
DINT_Array[4]	5		Decimal	DINT	
DINT_Array[5]	6		Decimal	DINT	
DINT_Array[6]	7		Decimal	DINT	
DINT_Array[7]	8		Decimal	DINT	
DINT_Array[8]	9		Decimal	DINT	
DINT_Array[9]	10		Decimal	DINT	
REAL_Array	{...}	{...}	Float	REAL[10]	
REAL_Array[0]	1.0		Float	REAL	
REAL_Array[1]	2.0		Float	REAL	
REAL_Array[2]	3.0		Float	REAL	
REAL_Array[3]	4.0		Float	REAL	
REAL_Array[4]	5.0		Float	REAL	
REAL_Array[5]	6.0		Float	REAL	
REAL_Array[6]	7.0		Float	REAL	
REAL_Array[7]	8.0		Float	REAL	
REAL_Array[8]	9.0		Float	REAL	
REAL_Array[9]	10.0		Float	REAL	

Monitor Tags | Edit Tags

I'm not going to do anything with the PLC program; I just need some data in a number of tags.

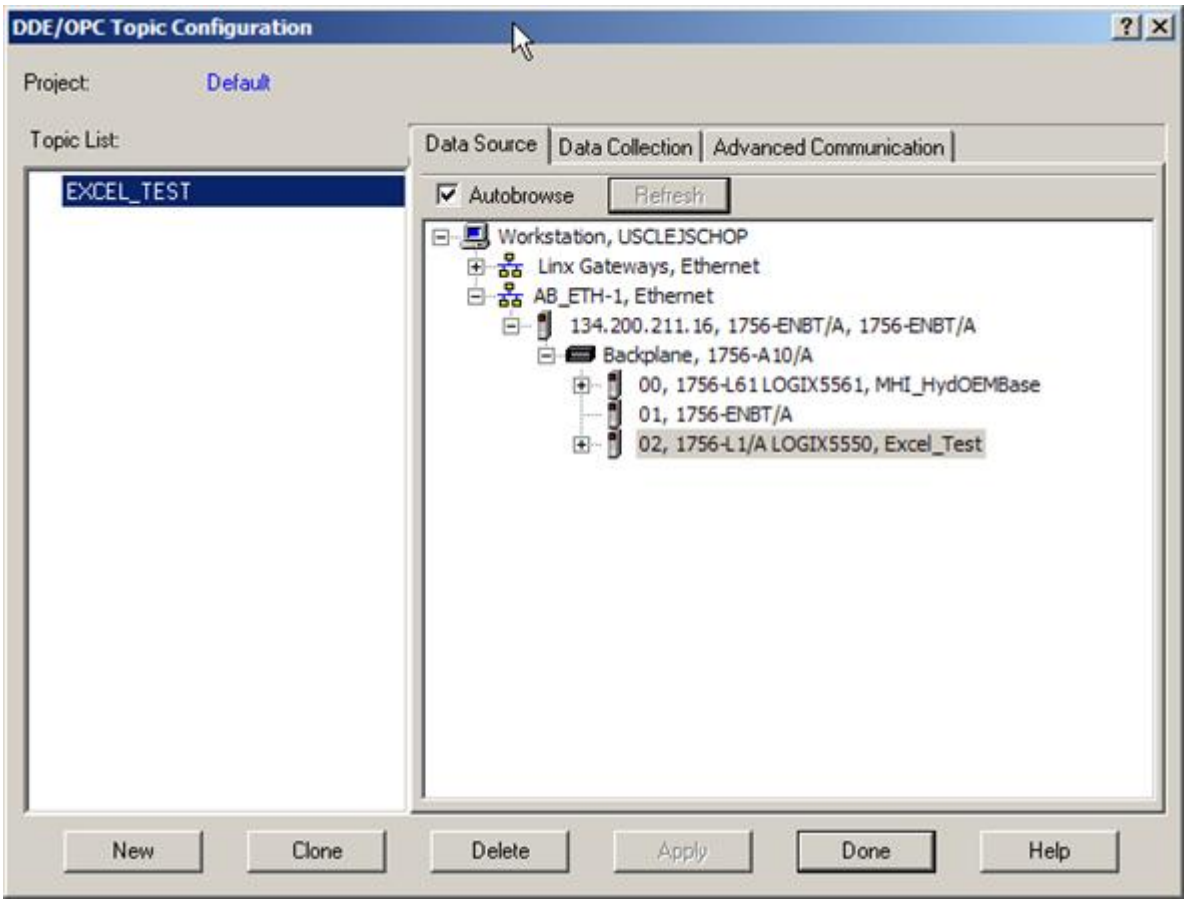
Next, we're going to set up a DDE/OPC Topic in RSLinx. Depending on the version of RSLinx you use, it might look slightly different, but you should be able to follow this with the screenshots.

Assuming that you know how to setup RSLinx initially to get online with your controller, I've skipped some steps. The setup I use looks like this in RSLinx:



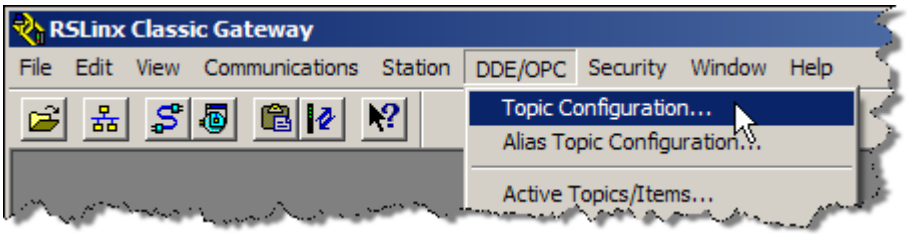
As you can see, I have a 10 slot CLX rack, with a 1756-ENBT card in slot 1 (address 134.200.211.16), and two processors, one in slot 0, and one in slot 2. The one in slot 2 is the processor we are going to use for this exercise.

Now, open up the DDE/OPC topic configuration by clicking 'DDE/OPC' and then 'Topic Configuration' in the top menu of RSLinx.



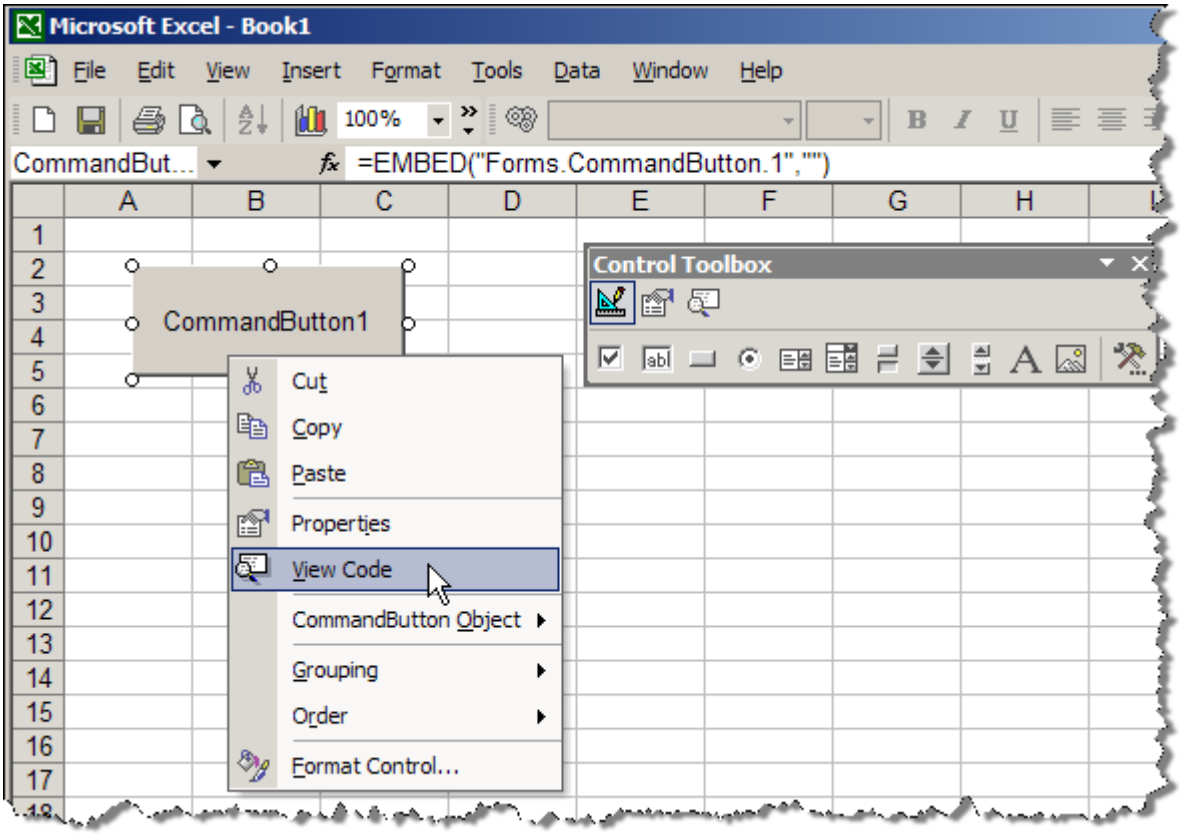
I'm going to create a new DDE/OPC topic called 'EXCEL_TEST', and use the Logix5550 processor in slot 2 as the data source. In order to do this, you have to click the 'New' button,

give the topic the desired name, and make sure the processor in slot 2 is selected as the source before you click 'Done'



To test if your setup is working, at this point you can use the OPC test client provided with RSLinx. I'm not going into detail about that, but I did make sure this worked before continuing with the next step, creating the Excel sheet.

Let's start up good old Excel, and create a new workbook. On this workbook, place a new command button. You can find the Command Button control in the 'Control Toolbox' toolbar in Excel. When you have the button, right click on it and choose 'View Code'. This will take you to the Visual Basic Editor:



First, create a function that will open the DDE topic to Excel:

```
Private Function OpenRSLinx()
    On Error Resume Next

    'Open the connection to RSLinx
    OpenRSLinx = DDEInitiate("RSLINX", "EXCEL_TEST")

    'Check if the connection was made
    If Err.Number <> 0 Then
        MsgBox "Error Connecting to topic", vbExclamation, "Error"
        OpenRSLinx = 0 'Return false if there was an error
    End If

End Function
```

Now, if I call this function from the CommandButton1_Click event, it will open the link to RSLinx:

```
Private Sub CommandButton1_Click()

    rslinx = OpenRSLinx()

End Sub
```

References:

http://literature.rockwellautomation.com/idc/groups/literature/documents/um/1769-um011_-en-p.pdf

http://literature.rockwellautomation.com/idc/groups/literature/documents/qs/1756-qs001_-en-p.pdf

http://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm001_-en-e.pdf

http://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1756-rm003_-en-p.pdf

http://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1756-rm094_-en-p.pdf