

# Allen Bradley's PLC Programming Handbook



**This handbook is a collection of programming overviews, notes, helps, cheat sheets and whatever that can help you (and me) program an Allen Bradley PLC.**

If you have experience with AB then please [contribute](#).

## An Introduction to RSLogix5000 Tags

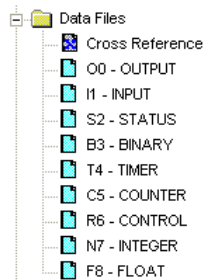
Tags are the method for assigning and referencing memory locations in Allen Bradley Logix5000 controllers. No longer are there any physical addresses such as N7:0 or F8:7 which use symbols to describe them. These have been replaced with tags which are a pure text based addressing scheme. This is a departure from the more conventional ways of programming PLC's, which includes Allen Bradley's 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. So let's dig in and get started.

### The Way We Used To Be

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.

#### Default Data Files



## Data File Descriptions

File #	Type	Description
O0	Output	This file stores the state of output terminals for the controller.
I1	Input	This file stores the state of input terminals for the controller.
S2	Status	This file stores controller operation information useful for troubleshooting controller and program operation.
B3	Bit	This file stores internal relay logic.
T4	Timer	This file stores the timer accumulator and preset values and status bits.
C5	Counter	This file stores the counter accumulator and preset values and status bits.
R6	Control	This file stores the length, pointer position, and status bits for control instructions such as shift registers and sequencers.
N7	Integer	This file is used to store bit information or numeric values with a range of -32767 to 32768.
F8	Floating Point	This file stores a # with a range of $1.1754944e-38$ to $3.40282347e+38$ .

While this method made it easy for using instructions, it provided a challenge for logically grouping different data types together according to function. For instance, in machine control, a motor may have a start, stop, speed and alarm code each with its own data type. Thus, the data was “scattered” throughout the data files.

File #	Name	Data Type
I1	Start	Input
I1	Stop	Input
F8	Speed Setpoint	Floating Point
N7	Alarm Code	Integer

### Comparing the Old and New

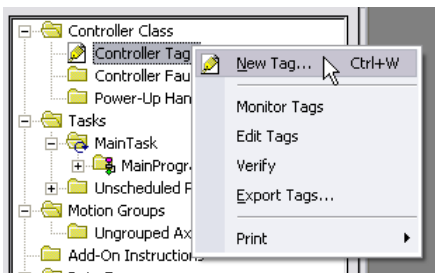
The Logix5000 controllers have done away with data files and in its place is the tag database. The tag database organizes memory locations in one place. Each tag is assigned its own data type. The table below shows the association between the current data types and the older systems with data files.

RSLogix 5 / 500		RSLogix 5000
File #	Type	
O0	Output	Input and output modules, when configured, automatically create their own tags like Local:0:I.Data.0
I1	Input	
S2	Status	Use the GSV and SSV instructions to get status information such as the CPU time, module states and scan times.
B3	Bit	Assign the Boolean (BOOL) data type to the tag.
T4	Timer	Assign the TIMER data type to the tag.
C5	Counter	Assign the COUNTER data type to the tag.

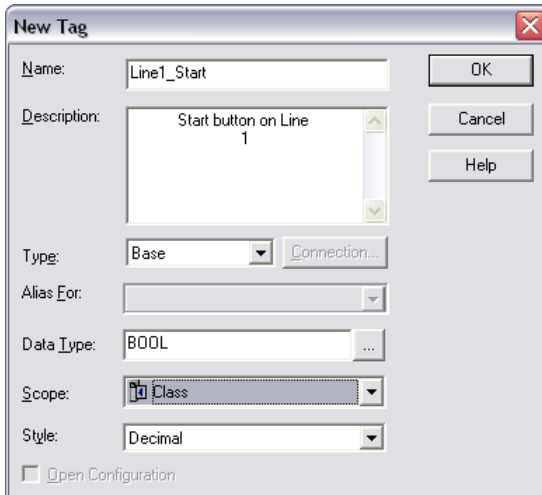
R6	Control	Assign the CONTROL data type to the tag.
<hr/>		
N7	Integer	Assign the double integer (DINT) data type to the tag.
<hr/>		
F8	Floating Point	Assign the REAL data type to the tag.
<hr/>		

## 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

While tags are not case sensitive, it is good practice to mix cases for readability. It is much easier to read Line1\_Start then LINE1START or line1start.

In addition, the tag database list sorts alphabetically. Therefore, it is best to use similar starting characters when you want tags to be together in the monitor list.

Tags Named for Grouping	Tags Not Named for Grouping
Level_High	High_Level
Level_Low	Insert_Nut
Insert_Nut	Knife_Stop
Knife_Stop	Low_Level

Use the **Description** field for a longer description of the tag. It is best to keep names short yet not cryptic. Tag names are downloaded and stored in the controller but the description is not as it is part of the documentation of the project.

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.

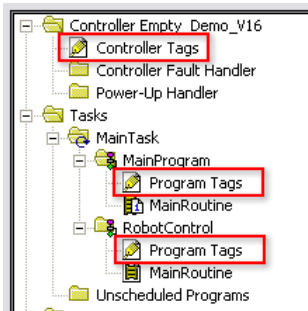
Select a **Data Type** for the tag by typing it in or by clicking on the ellipsis button and selecting it from the list. A data type is a definition of the size and layout of memory allocated for the created tag. Data types define how many bits, bytes, or words of data a tag will use.

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.

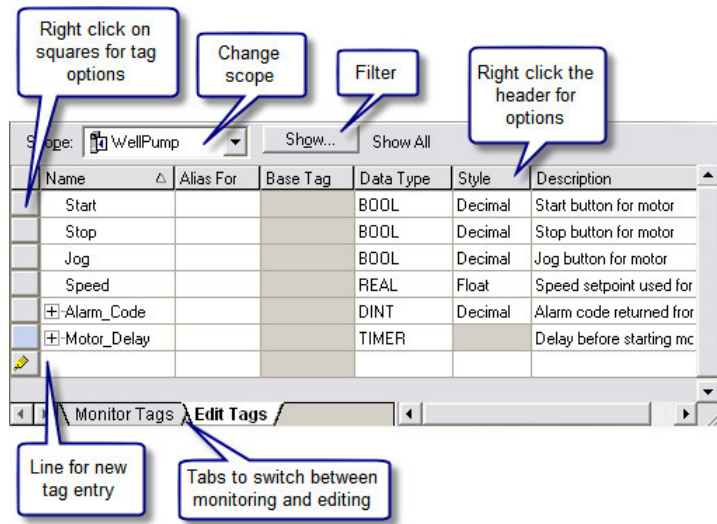
Program Tags are isolated from other programs. Routines cannot access data that is at the program scope of another program. Having program tags make it easy to copy/paste programs and not have to worry about conflicting tag names. Make sure though that no controller tags are named the same as program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

Name	Data Type	Style	Description
Alarm_Code	DINT	Decimal	Alarm code returned from motor
Motor_Delay	TIMER		Delay before starting motor
Motor_Delay.PRE	DINT	Decimal	Delay before starting motor
Motor_Delay.ACC	DINT	Decimal	Delay before starting motor
Motor_Delay.EN	BOOL	Decimal	Delay before starting motor
Motor_Delay.TT	BOOL	Decimal	Delay before starting motor
Motor_Delay.DN	BOOL	Decimal	Delay before starting motor

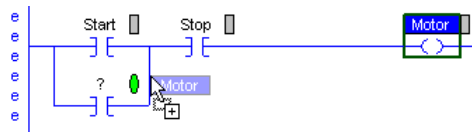
## An Easier Way to Create Tags

The easiest way to create tags is on the fly while programming. When an instruction is first used a "?" will indicate the need for a tag. There are three options at this point:

1. Double click on the "?" and select an existing tag from the drop down box.
2. Right click on the "?" and select new tag.
3. Double click on the "?" and type in the tag name. If it does not all ready exist, then right click on the tag name and select Create "NewTagName". Be careful with this method not to use spaces or special characters.

The nice thing about all these methods is that RSLogix5000 will automatically fill in the correct data type according to the instruction used.

Another quick method is to drag and drop an existing tag to a new instruction. Make sure to click on the tag name rather than the instruction.



## 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's 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

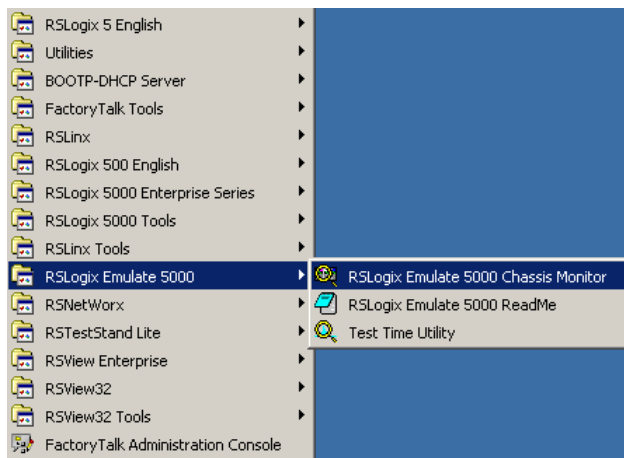
RSLogix Emulator 5000 is a software simulator for the Allen Bradley line of Logix 5000 controllers (ControlLogix®, CompactLogix®, FlexLogix®, SoftLogix5800® and DriveLogix®). The goal is to mimic the function of a [PLC](#) without the actual hardware and thus do advanced debugging. More information can be found in the AB publication LGEM5K-GR015A-EN-P.

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

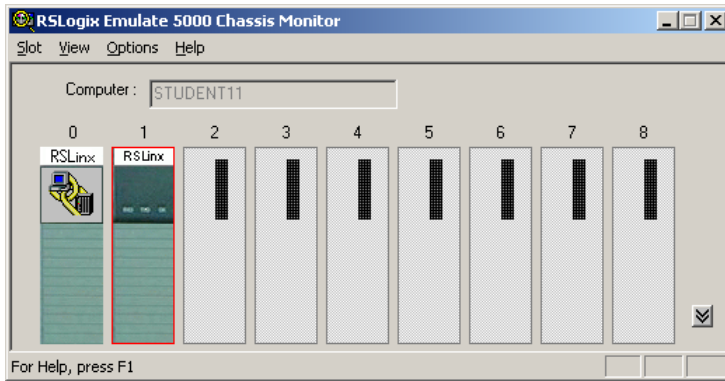
1. Setting up the chassis monitor.
2. Creating a connection in RSLinx.
3. Creating a project with associated emulation hardware.

### Setting up the [Chassis Monitor](#)

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

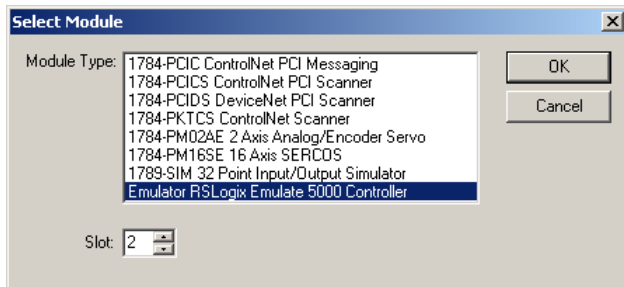


When the emulator opens up you're confronted with what looks like an empty chassis. In slot 0 is an RSLinx module which has to be there for the emulator communications to work. Your slot 1 might have another irremovable RSLinx module depending if you are running RSLogix Enterprise.

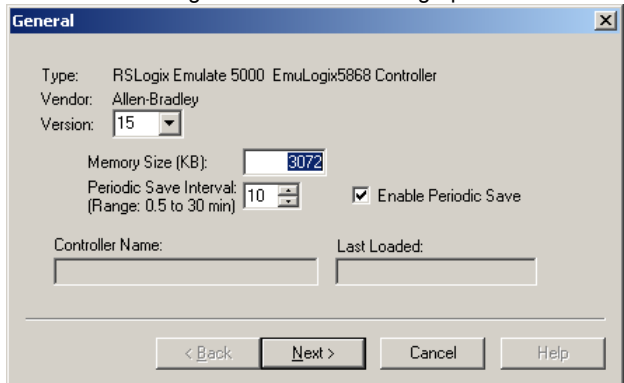


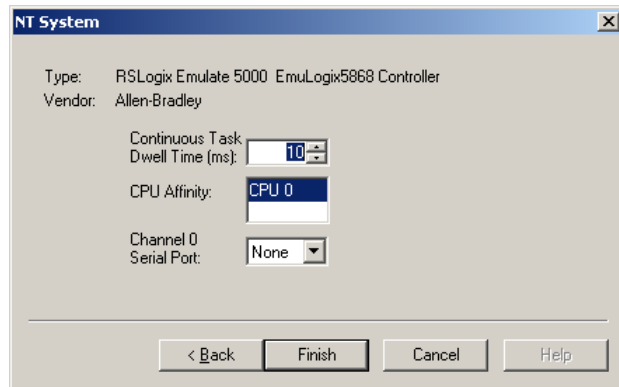
From here we set up our hardware configuration for simulation. Our first step will be to add the [CPU](#). In this case it is a special one called an Emulation Controller.

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.



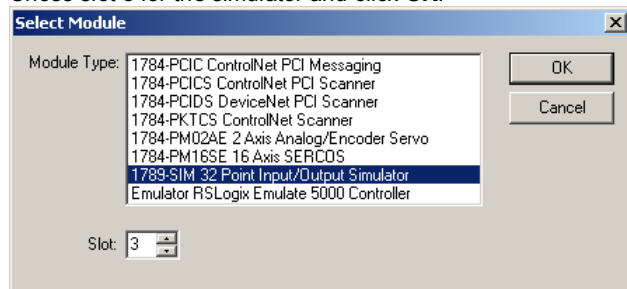
5. At this point you may be accosted with a message about previous configurations. Just select **Reset the Configuration to Default Values** and click **NEXT**.
6. The next two dialog screens are for setting up the controller details. Click **NEXT** and **FINISH** to accept all the defaults.



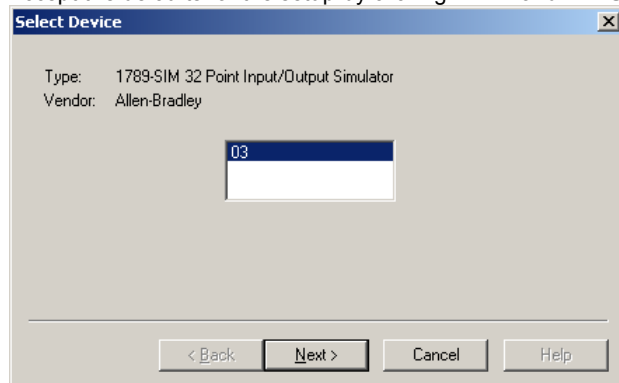


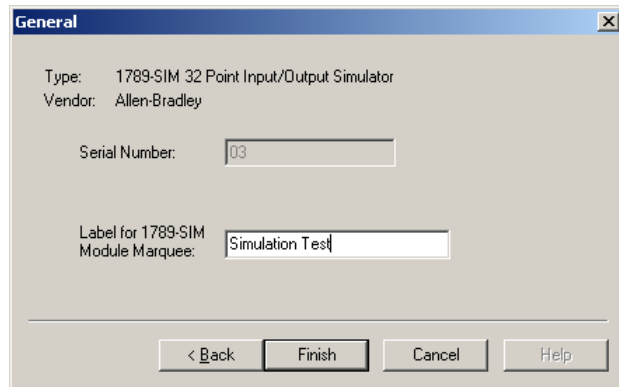
Next we'll add some input/output simulation.

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

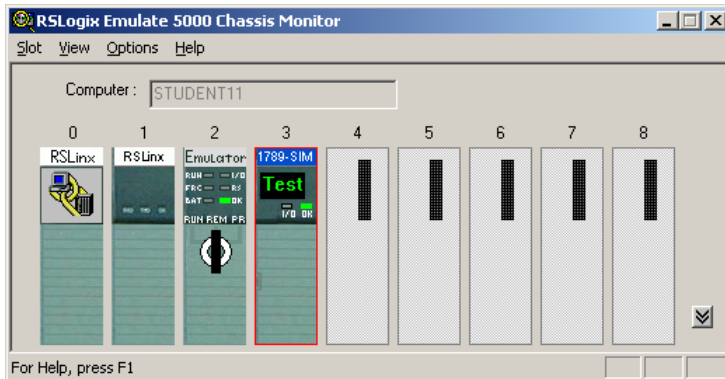


4. Accept the defaults for the setup by clicking **NEXT** and **FINISH**.



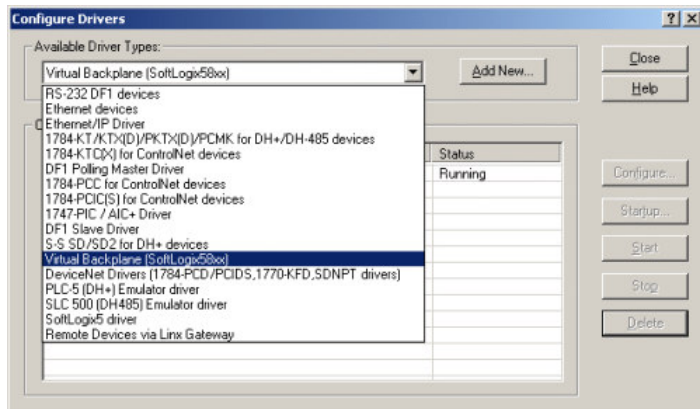


The chassis monitor will now have two emulation modules in it ready to go.



## Creating a connection in RSLinx

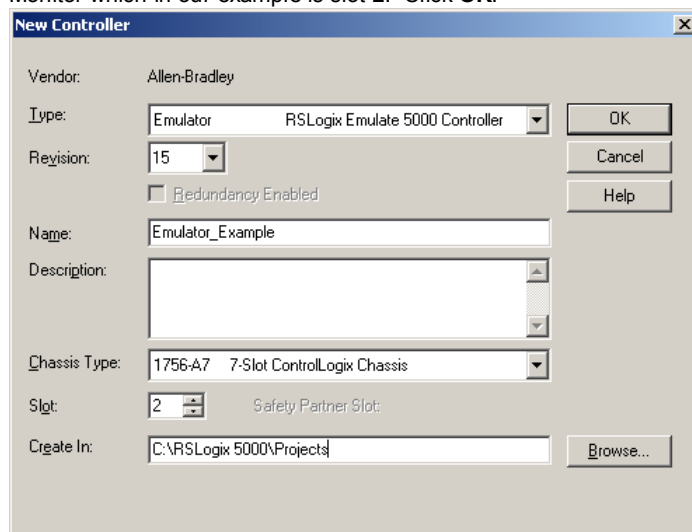
1. Start RSLinx under **Start > Programs > Rockwell Software > RSLinx > RSLinx Classic**
2. Click **Communications > Configure Drivers**.
3. Select the **Virtual Backplane (SoftLogix 58xx)** driver from the **Available Driver Types** list.
4. Click **Add New**. The Add New RSLinx Driver dialog box appears.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



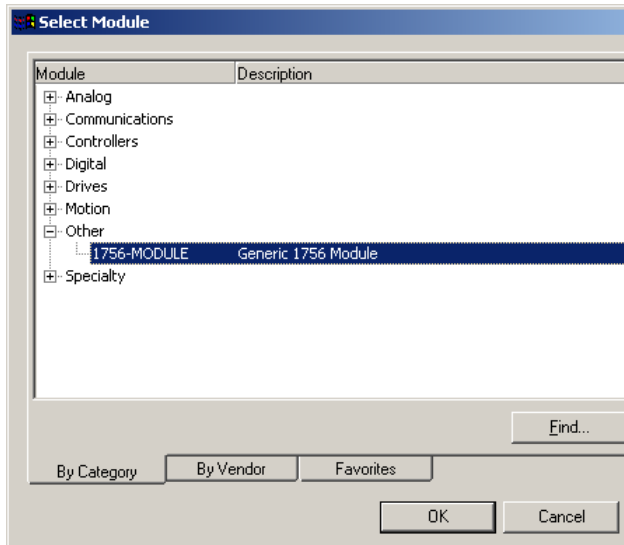
## Using RSLogix Emulator in a Project

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**.

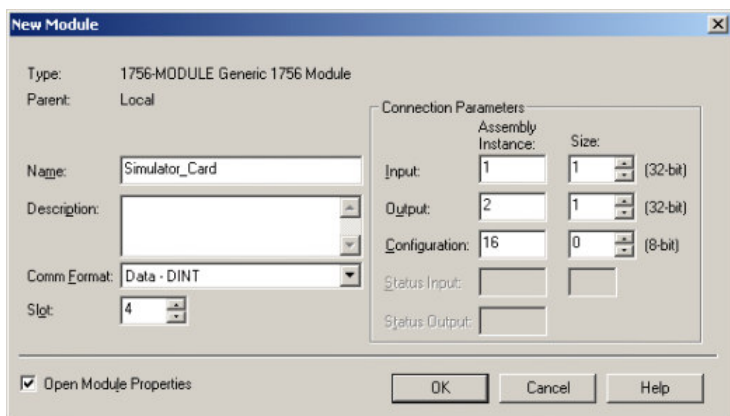


3. In RSLogix 5000's Controller Organizer, right click on the **I/O Configuration** folder, and then click **New Module**. The software displays the **Select Module** window.
4. Open the **Other** folder. Select the **1756-MODULE** from the modules list and then click **OK**.

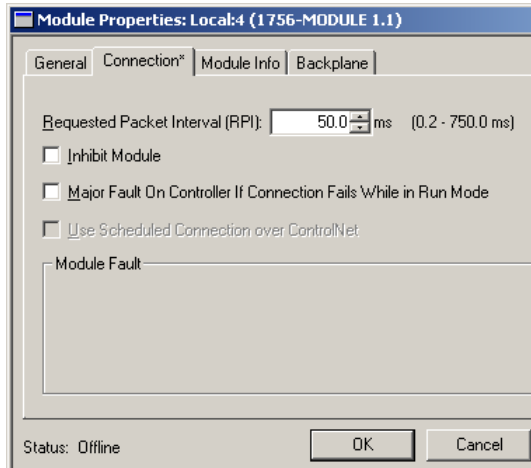


5. The software displays the **New Module** window.
- Add a **Name** for the card.
  - In the **Slot** field put the number that corresponds with the Chassis Monitor.
  - For the **Connection Parameters** put in the following and click **OK**

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

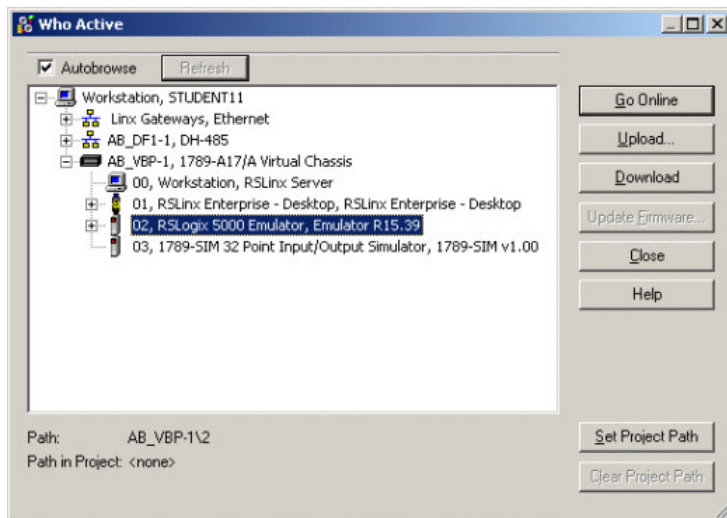


6. On the next **Module Properties** screen make sure to change the **Requested Packet Interval** to 50.0 ms.

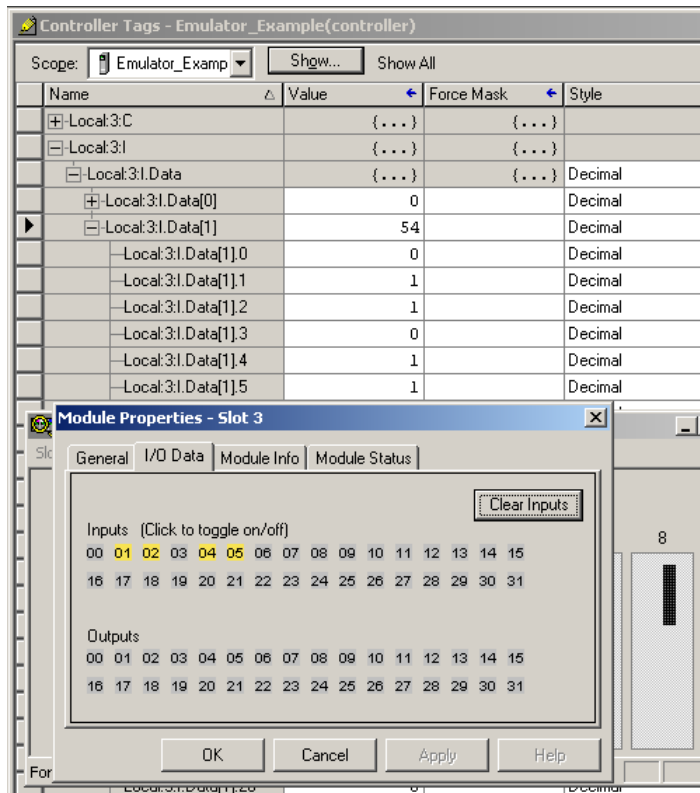


## Ready, Set, Go

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.

**Note:**

RSLogix Emulator is sometimes erroneously called RSEmulator.

## Getting Started with the Logix5000 PIDE Function Block

The PIDE (Enhanced PID) is an Allen Bradley Logix5000 family (ControlLogix, CompactLogix, FlexLogix, SoftLogix) function block that improves on the standard PID found in all their controllers. First impressions of this function block are quite intimidating. If you try to dive into it head first you may just end up banging your head against a wall. Many will be quite happy to stick with the tried and true PID instruction but to compete with the more advanced process control applications the PIDE boasts the following.

- It uses the velocity form of the PID algorithm. This is especially useful for adaptive gains or multiloop selection.
- Control of the instruction can be switched between Program and Operator modes.
- Better support for cascading and ratio control.
- Built in autotuner (requires extra key)
- Support for different timing modes
- More limiting and fault handling selections.

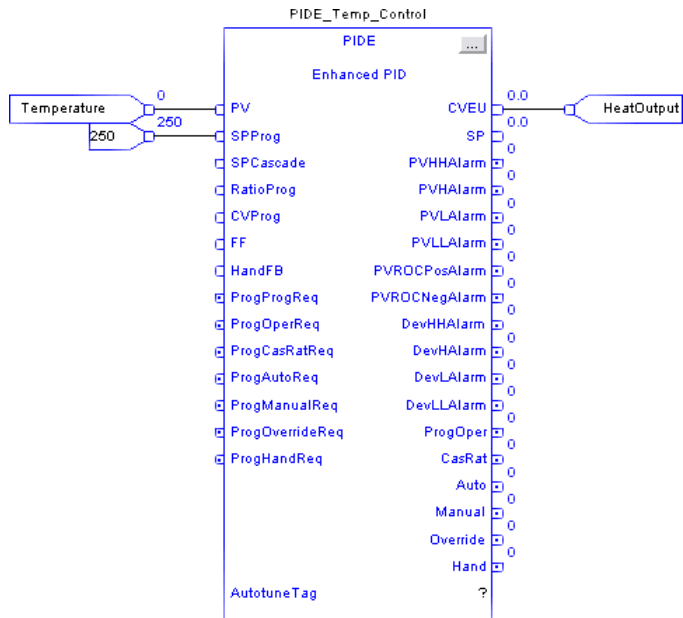
Still interested? What we want to do here is basically get you off the ground with the PIDE, distill all the options to the essentials and get it working.

The PIDE is only available as a function block (sorry, no ladder). Like the PID instruction it is best to set it up in its own periodic task. The period of the task automatically becomes the

sample rate (DeltaT) of the PID loop. Just make sure when adding the new routine to the task to select the *Type* as "Function Block Diagram."

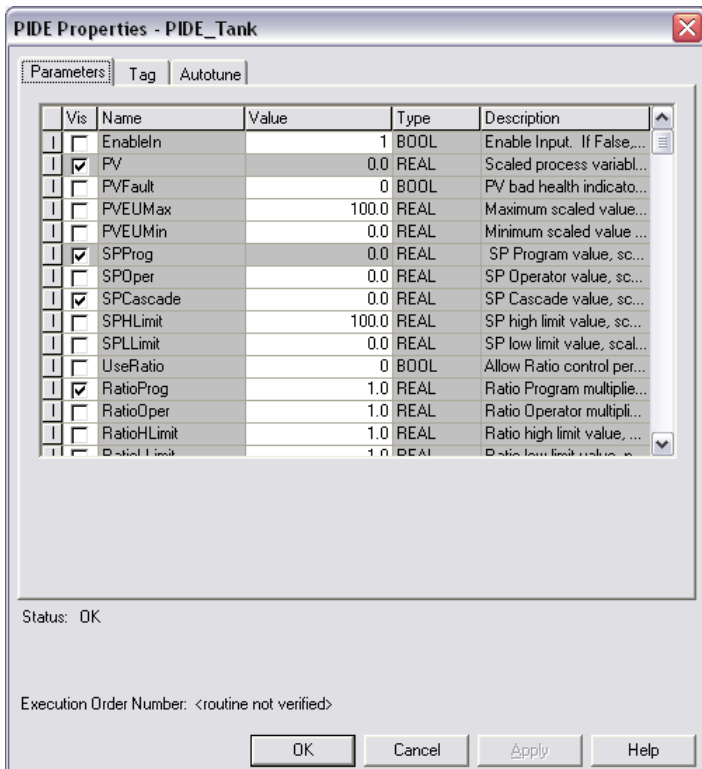
## Adding the PIDE Function Block

The PIDE instruction can be added from the *Instruction Toolbar* under the *Process* tab.



Once you plopp a function block onto a sheet it automatically creates a program tag for the instruction which stores all the settings. The parameters can be set or monitored by wiring input and output references or by clicking on the ellipsis box in the top right corner to reveal the block properties.

Opening the block properties for the PIDE instruction before RSLogix5000 version 15 meant you would be accosted with a long list of parameters.



Version 15 has at least organized some of the more common settings (but not all) under tabs and groups.

**PID Properties - PIDE\_01**

General Configuration | EUs/Limits | Cascade/Ratio | Alarms | Parameters | Tag | Autotune

Timing  
 Mode:    
 Oversample  $\Delta t$ :  s   
 RTS Period:  ms

Control action:  E = SP - PV   
 E = PV - SP

Calculate Using  
 Proportional term:  E  PV   
 Derivative term:  E  PV

Derivative Smoothing   
 PV Tracking   
 Manual Mode after initialization

Gains  
 Proportional:    
 Integral:  min/repeat   
 Derivative:  min

Equation Type:  Independent   
 Dependent

CV Zero-Crossing Deadband  
 Deadband:    
 Zero-Crossing off

Override value:  %   
 Program Value reset

Status: OK

Execution Order Number: <routine not verified>

OK Cancel Apply Help

The most essential settings are:

Name	V15 Location	Description
<b>.PV</b>	Must be wired in from a tag.	The Process Variable is the reading (temperature, pressure, flow, etc.) that is to be controlled by the PID loop.
<b>.PVEUMax</b> <b>.PVEUMin</b>	EUs/Limit tab in the Engineering Units Scaling group	The Process Variable Engineering Units Maximum and Minimum. The value of PV and SP which corresponds to 100 % span of the process variable.
<b>.SPProg</b> <b>.SPOper</b>	Should be wired in or set in the tag.	The Set Point is the theoretical perfect value of the process variable. SPProg is the value to use when in program mode and SPOper is used when in operator mode.
<b>.SPHLimit</b> <b>.SPLLlimit</b>	EUs/Limit tab in the SP Limits group	The Set Point High Limit and Set Point Low Limit clamp the maximum and minimum values of the set point. If SPHLimit > PVEUMax or SPLLlimit < PVEUMin then a fault will occur.
<b>.PGain</b>	General Configuration tab in the Gains group	Proportional gain. Enter 0 to disable.

Name	V15 Location	Description
.IGain	General Configuration tab in the Gains group	Integral gain. Enter 0 to disable.
.DGain	General Configuration tab in the Gains group	Derivative gain. Enter 0 to disable.

## Program/Operator Control

The first thing to understand when programming a PIDE block is the different controls and modes available.

The Program/Operator control lets you transfer control of the PID loop between the user program and an operator interface such as an HMI. Each control has separate set points and mode controls. It's important to understand that when in Program Control the set point is determined by SPProg while in Operator Control its SPOper. The SP output indicates the set point that the function block is actually using.

Control is determined by the following inputs:

Name	Description
.ProgProgReq	A program request to go to Program control.
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.
.OperOperReq	An operator request to go to Operator control.

The ProgOper output indicates the control of the PIDE instruction. If the output is a 1 then it is in Program control and if the output is a 0 then it is in Operator control. The Program request inputs take precedence over the Operator requests so that the program can lock out any operator overrides. The ProgValueReset input clears all input requests.

## Operating Modes

The PIDE instruction supports the following modes.

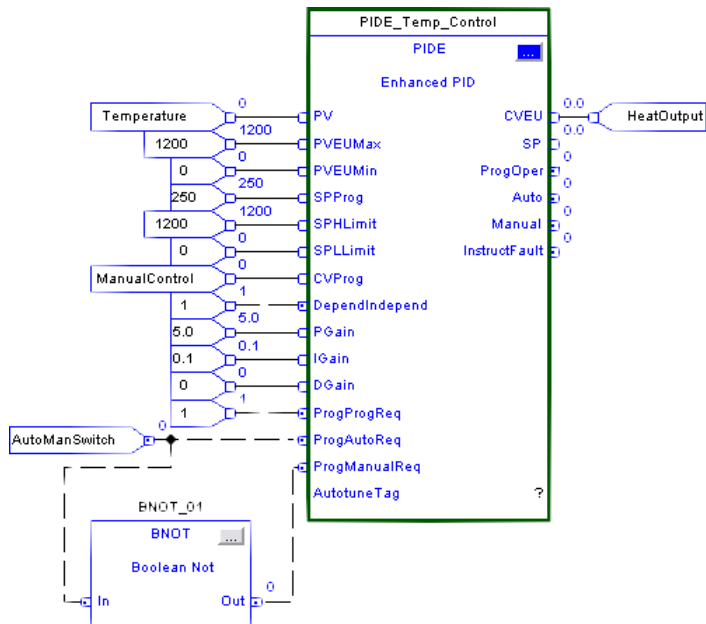
Mode	Description
------	-------------

Mode	Description
<b>Manual</b>	While in Manual mode the instruction does not compute the change in CV. The value of CV is determined by the control. If in Program control, CV = CVProg and if in Operator control, CV = CVOper. Select Manual mode using either OperManualReq or ProgManualReq. The Manual output bit is set when in Manual mode.
<b>Auto</b>	While in Auto mode the instruction regulates CV to maintain PV at the SP value. If in program control, SP = SPProg and if in Operator control, SP = SPOper. Select Auto mode using either OperAutoReq or ProgAutoReq. The Auto output bit is set when in Auto mode.
<b>Cascade/Ratio</b>	While in Cascade/Ratio mode the instruction computes the change in CV. The instruction regulates CV to maintain PV at either the SPCascade value or the SPCascade value multiplied by the Ratio value. SPCascade comes from either the CVEU of a primary PID loop for cascade control or from the "uncontrolled" flow of a ratio-controlled loop. Select Cascade/Ratio mode using either OperCasRatReq or ProgCasRatReq. The CasRat output bit is set when in Cascade/Ratio mode.
<b>Override</b>	While in Override mode the instruction does not compute the change in CV. CV = CVOverride, regardless of the control mode. Override mode is typically used to set a "safe state" for the PID loop. Select Override mode using ProgOverrideReq. The Override output bit is set when in Override mode.
<b>Hand</b>	While in Hand mode the PID algorithm does not compute the change in CV. CV = HandFB, regardless of the control mode. Hand mode is typically used to indicate that control of the final control element was taken over by a field hand/auto station. Select Hand mode using ProgHandReq. The Hand output bit is set when in Hand mode.

If a fault occurs in the PIDE settings then it is forced into Manual mode and sets a corresponding bit in the Status words. The InstructFault output is the indicator of a fault. For more detail open the block properties and look at the Status at the bottom of the dialog box. Refer to the *Logix5000 Controllers Process Control and Drives Instructions* (pub 1756-RM006D-EN-P) for details.

### Basic Example

Here's an example where just the essentials are used. This is a temperature control application if you hadn't guessed all ready. I've changed the look of the function block by going into the block properties, selecting the *Parameters* tab and checking on (or off) the boxes in the *Vis* column besides the inputs and outputs that are of concern.



Here's the run down on each of the inputs.

Input	Description
<b>PV</b>	The process variable coming in from my TC card
<b>PVEUMax</b> <b>PVEUMin</b>	The span of the temperature input that equals 0 to 100%. In this case the temp goes from 0 to 1200 degC.
<b>SPHLimit</b> <b>SPLLimit</b>	We could limit the set point but in this test case just set it equal to the PVEUMax/Min.
<b>SPProg</b>	I've decided to use Program Control so the Set Point needs to come in on this input rather than SPOper.
<b>CVPProg</b>	When in manual mode the CV is controlled by this input.
<b>DependIndependent</b>	I prefer the Dependent form of the PID algorithm.
<b>PGain</b> <b>IGain</b>	The essential PID settings of Proportion, Integral and Derivative.

Input	Description
Dgain	
ProgProgReq	Set the request to use Program Control.
ProgAutoReq ProgManReq	Since we're in Program Control these inputs control the Auto and Manual modes. To run them off one switch the BNOT block is used to invert the bit.

Now for the outputs.

Output	Description
CVEU	The Control Variable output in engineering units. Every PID control needs an output. In this case it goes from 0 to 100%.
SP	The actual set point which in this case equals SPProg.
ProgOper	I want to see a 1 here just to make sure we're in Program Control
Auto Manual	Indicates the operating mode.
InstructFault	If I screw something up then this bit will come on.

### Common Problems

No output	<ul style="list-style-type: none"> <li>The PID loop is in manual mode. Put it into auto mode using ProgAutoReq.</li> <li>Not in program control or SPProg is not set. Use ProgProgReq to go into program control and set SPProg.</li> <li>No values or not enough proportion (PGain) or integral (IGain).</li> </ul>
Output is limited at 100	<ul style="list-style-type: none"> <li>The SP High Limit is still set at the default of 100. Change the value of SPHLimit.</li> </ul>

### Conclusion

Hopefully this basic introduction has gotten you off the ground. Half the battle is just getting it to work. Once that is done you can now really start to tinker with the power of the PIDE function block.

## Further Reference

- *Logix5000 Controllers Process Control and Drives Instructions* (Publication 1756-RM006D-EN-P)
- *Using the PIDE Instruction* (Publication LOGIX-WP008A-EN-P - August 2005)
- *Using a Logix Controller for Barrel Temperature Control on Plastic Injection Molding and Extruding Machines* (Publication RA-AP015A-EN-P €“ February, 2004)

# RSLogix 5000 Tips and Tricks

Everybody enjoys nifty little tips and tricks to get their work done faster. This listing is for Allen Bradley's RSLogix 5000 software. Feel free to add your own tips and tricks using the 'add comment' link.

## General

- To access Release Notes for this version of software, choose Release Notes from the Help menu.
- The Quick View Pane, located below the Controller Organizer, provides "thumbnail" information for the selected component.
- The Watch Pane, located below the language editor window, provides monitoring for all tags referenced in the active routine window.
- The Controller Organizer is dockable. That is, you can drag it to the left or right side of the screen, or float it somewhere in between.
- Hide/show the Controller Organizer via a toolbar button to make more display area for editors.
- RSLogix 5000 supports Cut/Copy/Paste/Drag/Drop of components within the Controller Organizer as well as to other instances of RSLogix 5000.
- Double-clicking on error messages displayed in the Error Window will navigate you to where the error was encountered. F4 and Shift-F4 can be used to move between errors.
- You can reorder the columns in the tag editor by clicking on the title and dragging it to a new position.
- To simultaneously display logic in multiple routines, select Window -> New Window and then arrange the windows manually. Or select Window -> Tile Horizontal.
- To remove a yellow triangle warning symbol on a device, first check the connection status. If the status is "Connection is not scheduled", re-open the RSNetWorx software. Return to RSLogix 5000 software and the yellow triangle should be gone.
- On one computer, you can install and simultaneously launch (run) multiple translated versions of RSLogix 5000 software.
- Once you do a partial import of rungs, add-on instructions, or user-defined data types, you can't undo the import. If the import didn't work as expected, close the project without saving.
- When you select a partial import, make sure to select the correct rung or trend file. Both files have L5X extensions and the software doesn't prevent you from selecting the wrong file. If you try to import a rung where a trend is expected, or vice versa, the software does display an error that the import failed.
- Partial import of rungs works in all ladder routines, including Add-On Instructions.
- In version 15, the Tag Editor added support for New Window.
- To simultaneously display logic in multiple routines, select Window -> New Window and then arrange the windows manually. Or select Window -> Tile Horizontal.

## Keyboard

- Keyboard shortcuts are listed in the Online Help, under the "Navigating the Software" topic.
- You can use Ctrl+Page Down and Ctrl+Page Up to move between tabs in a dialog or routine window.
- You can use Ctrl+Tab and Shift+Ctrl+Tab to move between multiple RSLogix 5000 views.
- You can use Ctrl+G to invoke the Go To dialog. The Go To dialog is convenient for navigating the software.
- You can use Alt+Insert to open the Language Element browser in any of the language editors. You can also invoke this browser by pressing the Insert key in the LD, SFC and FBD editors.
- You can use Ctrl+Space to invoke the Tag browser from within the ST editor.
- You can use the Go To dialog (Ctrl+G) to quickly navigate to routines called by the current routine and to routines that call the current routine.
- In the Sequential Function Chart Editor, you can use the Routine Overview (Ctrl+B) tool to view your entire SFC and help navigate to a specific area of your chart.
- Double-clicking on error messages displayed in the Error Window will navigate you to where the error was encountered. F4 and Shift-F4 can be used to move between errors.
- The Language Element browser is a shortcut to adding logic. In the any of the language editors: use Alt+Insert, type the instruction mnemonic, and press Enter. You can also invoke this browser by pressing the Insert key in the LD, SFC and FBD editors. This short cut can be much quicker than using the instruction toolbar.
- As you use the keyboard to move the cursor around grid cells, press Alt+Down arrow to activate any controls that are active for that cell. This works in all grid-based editors, such as the Tag Editor, Data Monitor, etc. This gives you a way to access cell controls via the keyboard, rather than using the mouse.

## Controller Projects

- Whenever you go online using RSLogix 5000, changes made to controller are simultaneously made to a temporary copy of the project file (.ACD). Save makes these changes permanent. Therefore, an upload is only necessary to obtain the latest copy of the tag data in the controller.
- Both Rockwell Automation and third-party sample projects are installed with RSLogix 5000. You can find them in the RSLogix 5000 Samples folder. These projects demonstrate program techniques and code that you can use to program select modules.
- Avoid pointing one alias tag to another alias tag to ensure the application maintains the appropriate references after an upload.
- Avoid pointing multiple alias tags to the same base tag to ensure the application maintains the appropriate references after an upload.
- All tag names are downloaded and resident in the controller along with your logic.
- On download, if the ControlNet schedule stored in the offline RSLogix 5000 project file is old, RSLogix 5000 will retrieve the latest ControlNet schedule from the associated RSNetWorx project file. To make an association to an RSNetWorx project file, use the RSNetWorx tab in the Module Properties dialog of the ControlNet scanner.
- RSLogix 5000 supports moving your project from one Logix platform/controller to another.
- ACD, L5K, CSV, and L5X files are independent of which translated version of RSLogix 5000 imports or exports the file. The software doesn't create language-specific import/export files.
- Use any translated version of RSLogix 5000 software to go online to a controller without having to re-download.
- In a safety controller, standard tags in a safety mapped relationship follow safety restricted states. For example, a standard tag mapped to a safety tag is read-only in a safety locked state.
- Use Add-On Instructions to initialize tag values to specific values at the beginning of each routine or program scan. Then source protect the AOI to assure that values are correctly initialized and not overwritten manually.
- The order of members within a User-Defined Data Type affect the memory size of the data type. Within the UDT, keep members of the same data type together.

## I/O Configuration

- Module icons in the I/O Configuration folder change to indicate the module has faulted or the connection to the module has been interrupted.
- To remove a yellow triangle warning symbol, first check the connection status. If the status is "Connection is not scheduled", re-open the RSNetWorx software. Return to RSLogix 5000 software and the yellow triangle should be gone.
- To easily find a module in the Select Module Type dialog, simply start typing any part of the module name or description. When you start typing, the Find Module dialog is launched automatically.
- Use rack optimized communication formats for digital I/O modules to minimize amount of controller memory and communications overhead associated with these modules.
- RSLogix 5000 automatically creates controller tags when you create an input or output module. You can reference these tags directly in your logic.
- Use alias tags to assign names to specific input/output data and/or to provide a short alternative to lengthy structure member names.
- When you configure an analog I/O module, hold the shift key as you move the slider to increment HH, H, L, and LL values in whole numbers.
- Copy I/O data to a User-Defined Type (UDT) so you can synchronize I/O data with program scan. The UDT also enables easier mapping of physical I/O.

## Tasks, Programs and Equipment Phases

- An event task in Logix is similar to the processor input interrupt (PII) in the PLC-5. Multiple event tasks can exist in the controller, each configured to execute at the initiation of independent triggers.
- A periodic task in Logix is similar to the selectable timed interrupt (STI) in the PLC-5. Multiple periodic tasks can exist in the controller, each configured to execute at independent rates.
- Double-click on a state in an Equipment Phase to navigate to the logic for that state.
- Use RSBizWare Batch software to create Equipment Phases. Use the Equipment Editor to create the phases, define parameters, and synchronize the phases with an RSLogix 5000 project.
- Use any programming language (Ladder, Structured Text, FBD, or SFC) to program state routines in Equipment Phases.
- The fault routine for an Equipment Phase is the same as the fault routine for a program. Use a fault routine to allow logic to run before the controller faults due to a programming error.
- The Prestate routine runs all the time, even when the Equipment Phase is not active.
- The Prestate routine for an Equipment Phase is optional. Use the Prestate routine to execute the error detection logic for your phases.
- You don't have to implement all the available states in an Equipment Phase. On the Equipment Phases properties, check the "Complete State if not implemented" option.
- In the Phase Monitor, the states you can write code have action names and have a command word leading into the state, such as Start leads to the Running state. You add routines to these states. Waiting states don't require routines. The phase waits for a command to move to the next state. For example, Idle and Hold.

## Tags, Data Types and Other Data

- As you organize, add, or delete members of a User-Defined Data Type, the software adjusts the associated tag members and values accordingly so that remaining members retain their values.
- In the tag browser, click the >> button to display the tag filter. Use the tag filter to display unused tags or tags of a particular data type.
- The tag browser filters tags in some situations. If you don't see a tag you expect, change the tag filter.
- In version 15, the Tag Editor added support for New Window.
- You can use arrays to do indirect addressing. RSLogix 5000 supports arrays of one, two, and three dimensions.
- You can create a recipe by creating a new data type and then creating a tag which uses that data type. Your new data type can contain descriptive field names.
- RSLinx uses memory in a Logix controller to read data values. Use the following equation to estimate the memory needed:  $(1.5\text{Kbyte} + (\text{Number of individual tags} * 45 \text{ bytes}) + (\text{Number of array or structure tags} * 7))$
- The Watch Pane, located below the language editor window, provides monitoring for all tags referenced in the active routine window.
- You can trend a tag by right-clicking the tag and choosing "Trend Tag".
- You can find all occurrences of a tag by right-clicking the tag in logic and choosing "Find All".
- Logix controllers are optimized for the DINT and REAL data types. Use these data types to avoid conversion overhead and optimize performance.
- You can optimize the communication performance of acquiring data from Logix controllers by consolidating multiple data values into a User-Defined Data Type (UDT) or array.
- Indexed references to array elements add additional scan time overhead to the application. Use single dimension arrays whenever possible.
- When building a User-Defined Type (UDT), locate all bits or BOOLS adjacent to each other to minimize the amount of controller memory required to store the data.
- RSLogix 5000 automatically creates controller tags when you create an input or output module. You can reference these tags directly in your logic.
- Use alias tags to assign names to specific input/output data and/or to provide a short alternative to lengthy structure member names.
- Avoid pointing one alias tag to another alias tag to ensure the application maintains the appropriate references after an upload.
- Avoid pointing multiple alias tags to the same base tag to ensure the application maintains the appropriate references after an upload.
- Controller tags apply to the entire controller and can be referenced by any program. Program tags apply only to individual programs. This means program tags can have the same names in more than one program, allowing programs to be copied and reused.
- You can reorder the columns in the tag editor by clicking on the title and dragging it to a new position.
- All tag names are downloaded and resident in the controller along with your logic.
- You can export (and import) tag definitions to a comma separated value (CSV) file and manipulate them using external tool, e.g. spreadsheet, text editor.
- For tables of bits (BOOL), use a DINT array to ensure full access via the file and diagnostic instructions COP, DDT, FBC, etc.
- In a safety controller, standard tags in a safety mapped relationship follow safety restricted states. For example, a standard tag mapped to a safety tag is read-only in a safety locked state.
- The order of members within a User-Defined Type affect the memory size of the data type. Within the UDT, keep members of the same data type together.

## Routines

- Logix supports four controller programming languages: Ladder, Function Block Diagram, Structured Text, and Sequential Function Chart.
- To simultaneously display logic in multiple routines, select Window -> New Window and then arrange the windows manually. Or select Window -> Tile Horizontal.
- Multiply the number of words in a PLC/SLC program times 18 to estimate the amount memory (in bytes) needed in a Logix controller.
- To display context-specific instruction help, select an instruction or element and press F1.
- The Language Element browser is a shortcut to adding logic. In the any of the language editors: use Alt+Insert, type the instruction mnemonic, and press Enter. You can also invoke this browser by pressing the Insert key in the LD, SFC and FBD editors. This short cut can be much quicker than using the instruction toolbar.
- You can find all occurrences of a tag by right-clicking the tag in logic and choosing "Find All".
- You can use the Go To dialog (Ctrl+G) to quickly navigate to routines called by the current routine and to routines that call the current routine.
- You can drag and drop from the instruction toolbar in any of the language editors. In SFC editor, the elements auto-connect.
- Use the CPS instruction to provide buffering of communications and I/O data to minimize impact of asynchronous data arrival.
- For tables of bits (BOOL), use a DINT array to ensure full access via the file and diagnostic instructions COP, DDT, FBC, etc.
- Controller tags apply to the entire controller and can be referenced by any program. Program tags apply only to individual programs. This means program tags can have the same names in more than one program, allowing programs to be copied and reused.
- In the Sequential Function Chart Editor, you can use the Routine Overview (Ctrl+B) tool to view your entire SFC and help navigate to a specific area of your chart.
- Logix controllers perform a prescan of logic on startup to perform initialization. A tag used as an index can cause a startup fault if its value is larger than the array length. Use a Fault routine to detect and reset this condition.
- In the Sequential Function Chart Editor, you can use the indicator tag field in an action to specify a tag value to monitor during execution.
- In the Sequential Function Chart Editor, you can select multiple SFC elements and use the Layout SFC Elements feature to automatically rearrange the selected elements as needed.

to provide adequate spacing, avoid page boundaries, and left or center justify branches.

- In the Sequential Function Chart Editor, you can change the order in which selection branch legs are evaluated from the Set Sequence Priorities dialog.
- Refer to the Online Help for the Action Properties dialog - General Tab Overview for a useful timing diagram that explains how the various action qualifiers affect the execution of an action.
- You can customize the auto-naming of Sequential Function Chart Steps, Actions, Transitions, and Stop elements from the Workstation Options and Routine Properties dialogs.
- You can attach text boxes to language elements in FBD and SFC logic to maintain their relative positions if you move logic.
- You can use Ctrl+Space to invoke the Tag browser from within the ST editor.
- Comments in Structured Text are downloaded to the controller. This includes comments in Structured Text routines and embedded Structured Text in SFC routines.
- In the Structured Text Editor, you see the words colored to indicate keywords, tag names, and other recognized words. You can change the colors used by the editor by choosing Options from the Options menu.
- In the Structured Text Editor, you can configure the instruction's parameters by right-clicking an instruction name and choosing "Instruction Properties".
- You can use instructions available in Ladder and Function Block Diagram routines also in Structured Text routines.
- You can configure the sheet size for your Function Block Diagram or Sequential Function Chart routines from the Routine Properties dialog.
- In the Function Block Editor, you can configure the block's parameters by clicking the Browse (...) button on the upper right side of the block.
- RSLogix 5000 supports pending edits on multiple rungs when online editing ladder logic.
- When editing ladder routines, you can create logic using ASCII (for example: "XIC MYTAG") by either typing when a rung is selected, pressing the Enter key when a rung is selected, or double clicking to the left of a rung.
- In the Ladder Diagram Editor, you can insert a branch level above the current level by right clicking the left side of the branch and select Add branch. To insert a branch level below the current level, right click the right side of the branch and select Append New Level.
- Partial import of rungs works in all ladder routines, including Add-On Instructions.
- When performing a partial import of rungs, change tag names to create new tags in the imported logic.
- Once you do a partial import of rungs, add-on instructions, or user-defined data types, you can't undo the import. If the import didn't work as expected, close the project without saving.
- On a partial import of rungs, the exported data values are also imported. This includes configured message instructions. Partial imports/exports can save time versus copying and pasting since copying and pasting does not copy data values.
- Copy pieces of logic into other applications like Microsoft Word in a bitmap or metafile format.
- If you want a subroutine to execute every scan, copy the first instruction and paste it right next to the original instruction. Use the same tags on the duplicate instruction as on the first instruction. Then insert an AFI instruction before the duplicate.
- To copy a group of rungs to paste into another routine later, select the rungs and drag them to the desktop. This copies the rungs into a file that you can later drag into another routine.
- You can drag components from the Controller Organizer into the Ladder Editor instruction.
- Double click or press Enter at the end of a ladder rung to create and start a textual edit of that rung.
- To drag a language element from one routine to another, drag the element over the routine tabs at the bottom of the editor to switch the routine.

## Add-On Instructions

- To display the logic of an Add-On Instruction, select the instruction and use the context menu (right click) to open the logic.
- You can drag an Add-On Instruction from the Controller Organizer into any language editor.
- Copy an Add-On Instruction Definition from one project and paste into another to move that AOI and referenced AOIs in to the project.
- Use Add-On Instructions to initialize tag values to specific values at the beginning of each routine or program scan. Then source protect the AOI to assure that values are correctly initialized and not overwritten manually.
- Use source protection on an Add-On Instruction to protect local tags, data and logic.

## Communications

- Reserve 20% or more of the controller's memory to accommodate communications and changes in future Logix controller firmware releases.
- Use rack optimized communication formats for digital I/O modules to minimize amount of controller memory and communications overhead associated with these modules.
- Use the CPS instruction to provide buffering of communications and I/O data to minimize impact of asynchronous data arrival.
- On download, if the ControlNet schedule stored in the offline RSLogix 5000 project file is old, RSLogix 5000 will retrieve the latest ControlNet schedule from the associated RSNetWorx project file. To make an association to an RSNetWorx project file, use the RSNetWorx tab in the Module Properties dialog of the ControlNet scanner.
- When working with multiple controller projects in different chassis, use RSLinx shortcuts to identify those chassis with meaningful names.

## Drives and Motion

- RSLogix 5000 integrated motion supports camming, gearing, single-axis, and multi-axis instructions in Ladder Diagram, Structured Text, and Structured Text embedded in Sequential Function Charts.
- Execute motion direct commands directly from the context menu for any configured motion axis. The motion direct commands let you control motion instruction execution without creating or adding logic. This can be useful when first commissioning an axis or drive.
- To tune motor and drive parameters, such as gains for velocity and acceleration loops, as well as load dynamics, use the Tune tab or the MRAT and MAAT instructions. You can use the Tune tab in either Remote Program or Remote Run.
- The software automatically populates some SERCOS drive parameters when you configure an Axis\_Servo\_Drive. Display the axis properties to view or edit these parameters.
- In a SERCOS drive's configuration, you can change the number of counts returned per revolution to make the counts per inch or degree an rational number.
- In a motion system, you can copy over all motion hardware from an existing project to a new project without losing any axis settings or tuning. First drag the motion control module over to the new project. Then, drag any drives, the Motion Group, and then the axes.

## Optimizing Performance

- Logix controllers are optimized for the DINT and REAL data types. Use these data types to avoid conversion overhead and optimize performance.
- You can optimize the communication performance of acquiring data from Logix controllers by consolidating multiple data values into a User-Defined Type (UDT) or array.
- Indexed references to array elements add additional scan time overhead to the application. Use single dimension arrays whenever possible.
- Reserve 20% or more of the controller's memory to accommodate communications and changes in future Logix controller firmware releases.
- Use rack optimized communication formats for digital I/O modules to minimize amount of controller memory and communications overhead associated with these modules.
- Use the CPS instruction to provide buffering of communications and I/O data to minimize impact of asynchronous data arrival.
- If the memory estimation button is disabled, it means that your estimation is up to date. This happens after an estimate, but it also happens when you go offline with the controller because the offline memory numbers reflect actual use.
- The order of members within a User-Defined Type (UDT) affect the memory size of the data type. Within the UDT, keep members of the same data type together.

## Project Documentation

- Comments in Structured Text are downloaded to the controller. This includes comments in Structured Text routines and embedded Structured Text in SFC routines.
- You can print RSLogix 5000 views by clicking on the view and then pressing Ctrl+P or choosing Print from the File menu.
- When you print FBD logic, the editor automatically makes the logic fit the page. A 2:1 ratio is generally readable. For example, set the FBD sheet size to 11 x 17 (B Size) and print on 8 1/2 x 11 size paper.
- Copy pieces of logic into other applications like Microsoft Word in a bitmap or metafile format.

## Security

- If you are have trouble downloading a project even though you have privileges, make sure that you have the project and that you are online with the controller.
- If you can't access routine source protection when security is enabled, ask your administrator to grant you "Routine: Modify Properties" to obtain access.
- If your system uses FactoryTalk Security with RSLogix5000 software, version 16, software users can log into and log off of RSLogix 5000 software.
- If security functions are enabled, you must have appropriate access to import rungs or to copy/paste tags and data.

## The Logix5000 Essential Manuals

The Allen Bradley Logix5000 family (ControlLogix, CompactLogix, FlexLogix, SoftLogix) has some very good manuals. If you are just starting out or need a refresher here are the key manuals and the order I would read them. If you have RSLogix 50000 installed then you will find some of these in the *Help > Online Books* menu. Revision 16 also has some great videos in the Learning Center.

### The Basics

For starters there is the Quick Start manual.

 [Logix5000 Controllers Quick Start](#)

If you don't deal with the PLCs or RSLogix 5000 too much and just need a quick reminder about the hardware or programming then the System Reference is perfect.

 [Logix5000 Controllers System Reference](#)

## The Essentials

If you are getting into programming and designing a system then you'll want to start off with the Common Procedure Manual. It has a lot of helpful examples dealing with all aspects of the system.

 [Logix5000 Controllers Common Procedures Programming Manual](#)

Next comes the nitty gritty of each instruction. It's a good idea to at least peruse all the instructions so you have an idea of what is available.

 [Logix5000 Controllers General Instructions Reference Manual](#)

 [Logix5000 Process Control and Drives Instructions Reference Manual](#)

 [Logix5000 Controllers Motion Instructions](#)

 [GuardLogix Safety Application Instruction Set Reference Manual](#)

An often overlooked manual but filled with great information for getting the most out your designs is the Design Considerations Reference Manual. Certainly a must read if you are knee deep in the development and programming of Allen Bradley PLCs.

 [Logix5000 Controllers Design Considerations Reference Manual](#)

## Hardware Specifics

Specifics for the hardware can be found in the User Manuals and Installation Instructions for the PLC.

### ControlLogix

 [ControlLogix System User Manual](#)

 [ControlLogix Installation Instructions](#)

 [ControlLogix Controller and Memory Board Installation Instructions](#)

### CompactLogix

 [CompactLogix System User Manual](#)

 [CompactLogix 1769-L20, 1769-L30 Installation Instructions](#)

 [CompactLogix 1769-L32E, 1769-L35E Installation Instructions](#)

 [CompactLogix 1769-L32C, 1769-L35CR Installation Instructions](#)

### FlexLogix

 [FlexLogix System User Manual](#)

 [FlexLogix Controllers Installation Instructions](#)

### SoftLogix

 [SoftLogix System User Manual](#)

 [SoftLogix Controllers Installation Instructions](#)

### GuardLogix

 [GuardLogix Controllers User Manual](#)

 [GuardLogix Controllers Installation Instructions](#)

## Networking

If you are deciding on which network to use then see the [Design Considerations](#) manual and the section "Determine the Appropriate Network". Otherwise, for existing networks the following are helpful.

### Ethernet/IP

- [Ethernet Design Considerations for Control System Networks](#)
- [EtherNet/IP Modules in Logix5000 Control Systems User Manual](#)
- [Guard I/O EtherNet/IP Safety Modules User Manual](#)

### ControlNet

- [ControlNet Modules in Logix5000 Control Systems](#)

### DeviceNet

- [DeviceNet Modules in Logix5000 Control Systems User Manual](#)
- [Guard I/O DeviceNet Safety Modules](#)

The links are from the AB site so they are the latest and greatest manuals. Let [me know](#) if any of them are broken or if I forgot one you think is essential.

## User Defined Data Types (UDTs) and OOP

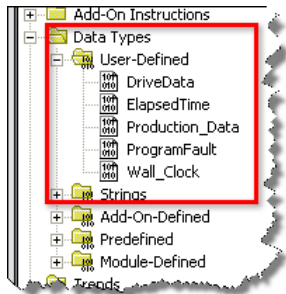
by John Schop

For years now, Object Oriented Programming paradigm (or OOP) has been a commonly used programming practice, and has of course found its way into industrial automation as well.

In the mean time, most [PLC](#) manufacturers have found ways to make the programmers life easier by introducing User Defined Types or UDTs. The name says it all; it is a 'type' that you, the programmer, can define all by yourself. This means that your programming environment will not only have the regular integers (INT) and Booleans (BOOL), but could also have a 'VALVE' type or a 'MOTOR' type.

I can't speak for other brands of PLC's, but the Allen Bradley ControlLogix series of PLC's, together with RSLogix 5000 programming software, makes it very easy to work with these UDTs, and since the introduction of RSLogix version 17 earlier this year, it is now even possible to edit your UDTs while online with a running system.

The Controller Organizer has a folder called Data Types > User-Defined with all the UDTs in the project.



I am of the opinion that every PLC program should rely heavily on UDTs to improve readability, and if you are an OOP adept, it can be a great help to organize your classes.

Let's go over the fundamentals of OOP for a little bit:

- **Classes:** Classes define the abstract characteristics and behavior of an object. For example, a simple 'VALVE' class would have the characteristics (or *attributes*) that it can be open or closed (the **things it can be**), and as far as behavior goes, it could have the *methods* 'to open' and 'to close' (the **things it can do**)
- **Objects:** An object is an instance (occurrence) of a class. In our example, there could be a Valve\_001 and a Valve\_002, which are both instances of the class 'VALVE', with the same attributes and methods.

Of course the definition of OOP goes a lot further than this. There is a very understandable explanation here: [http://en.wikipedia.org/wiki/Object-oriented\\_programming#Fundamental\\_concepts](http://en.wikipedia.org/wiki/Object-oriented_programming#Fundamental_concepts) for those who would like to read more. For now, let's leave it at this, and see how we can apply this to an industrial environment.

If you look at a valve as an object in a typical industrial automation environment, you should note the following:

- It has inputs and outputs that are specific for the object (proximity switches and solenoids).

- It can be either 'open' or 'closed'
- You can tell it to go 'open' or 'close'.
- It could have an alarm timer, that would tell us if the valve did not open or close in a given time period after a command.
- It might have interlocks, which allow the valve to open or close under certain conditions.

A UDT for this class, would fit all these properties and methods in one simple type. But, as always, we can expect further complications of the class 'VALVE' during the realization of a project. To be as flexible as possible, I highly recommend the practice of nesting UDT's, which will become clear along the way.

Let's start with defining our class, and keep in mind that it will have to be easily accessible for maintenance people or other programmers.

If we start at the I/O end, the best method is to create sub-classes called VALVE\_IN and VALVE\_OUT, which will contain our I/O.

The following example uses RSLogix5000 V16. First, create the sub-classes. From the File menu select New Component > [Tag](#). The following dialog box appears to create and edit the members of the UDT.

**Data Type: VALVE\_IN**

Name: VALVE\_IN

Description: Inputs for a valve

Members: Data Type Size: 4 byte(s)

Name	Data Type	Style	Description
Open	BOOL	Decimal	Open Prox Switch
Closed	BOOL	Decimal	Closed Prox Switch

Move Up Move Down OK Cancel Apply Help

**Data Type: VALVE\_OUT**

Name: VALVE\_OUT

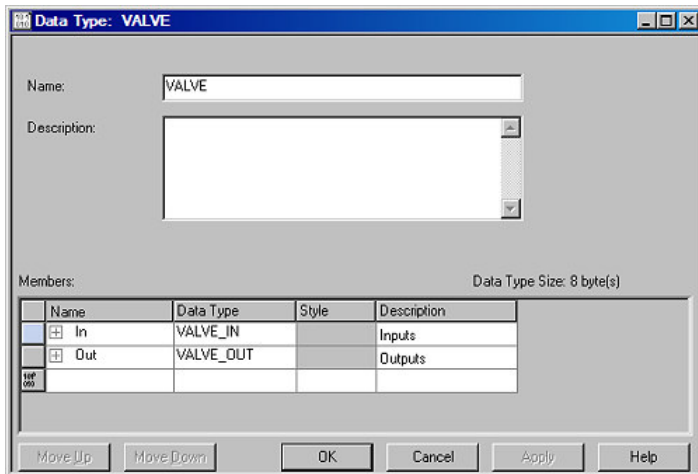
Description:

Members: Data Type Size: 4 byte(s)

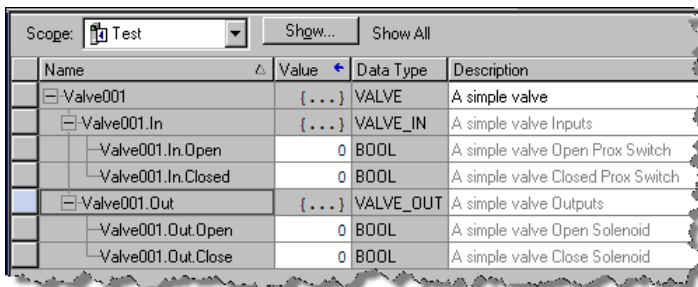
Name	Data Type	Style	Description
Open	BOOL	Decimal	Open Solenoid
Close	BOOL	Decimal	Close Solenoid

Move Up Move Down OK Cancel Apply Help

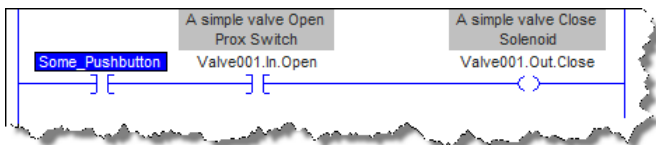
Now, make a UDT called VALVE, and 'nest' these sub-UDTs in it:



As you see, I am allowed to take the types I just created as the data type in this UDT. The real advantage of this feature will become clear if you create a object called Valve001 of the type VALVE, and look at the object in the 'monitor tags' window:



Wow! Just by creating a new tag of the type VALVE, it gets all these I/O points right away, and referenced in the program:



Of course, going further with this concept, everything for a valve can be included in one object. Allow me to skip some steps, and show you a possible final result:

The 'VALVE' class is now contained in a UDT called VALVE, which looks like this:

Name:

Description:

Members: Data Type Size: 48 byte(s)

Name	Data Type	Style	Description
[-] In	VALVE_IN		Inputs
[-] Open	BOOL	Decimal	Open Prox Switch
[-] Close	BOOL	Decimal	Close Prox Switch
[-] Out	VALVE_OUT		Outputs
[-] Open	BOOL	Decimal	Open Solenoid
[-] Close	BOOL	Decimal	Close Solenoid
[-] Timer	VALVE_TIMER		
[-] OpenTimer	TIMER		
[-] CloseTimer	TIMER		
[-] Status	VALVE_STATUS		
[-] Open	BOOL	Decimal	
[-] Close	BOOL	Decimal	
[-] Int	VALVE_INTERLOCKS		
[-] Open_Interlock	BOOL	Decimal	
[-] Close_Interlock	BOOL	Decimal	
[-] Cmd	VALVE_COMMANDS		
[-] Open	BOOL	Decimal	
[-] Close	BOOL	Decimal	
[-] Alarm	VALVE_ALARMS		
[-] Open_Timeout	BOOL	Decimal	
[-] Close_Timeout	BOOL	Decimal	
[-]			

As you see, the class VALVE now consists of the sub-classes VALVE\_IN, VALVE\_OUT, VALVE\_TIMER, VALVE\_STATUS, etc.

And an instance of this class, the object Valve001, would look like this:

Scope:  Show... Show All

Name	Value	Style	Data Type
[-] Valve001	{...}		VALVE
[-] Valve001.In	{...}		VALVE_IN
[-] Valve001.In.Open	0	Decimal	BOOL
[-] Valve001.In.Closed	0	Decimal	BOOL
[-] Valve001.Out	{...}		VALVE_OUT
[-] Valve001.Out.Open	0	Decimal	BOOL
[-] Valve001.Out.Close	0	Decimal	BOOL
[-] Valve001.Timer	{...}		VALVE_TIMER
[-] Valve001.Timer.OpenTimer	{...}		TIMER
[-] Valve001.Timer.CloseTimer	{...}		TIMER
[-] Valve001.Status	{...}		VALVE_STATUS
[-] Valve001.Status.Open	0	Decimal	BOOL
[-] Valve001.Status.Close	0	Decimal	BOOL
[-] Valve001.Int	{...}		VALVE_INTERLOCKS
[-] Valve001.Int.Open_Interlock	0	Decimal	BOOL
[-] Valve001.Int.Close_Interlock	0	Decimal	BOOL
[-] Valve001.Cmd	{...}		VALVE_COMMANDS
[-] Valve001.Cmd.Open	0	Decimal	BOOL
[-] Valve001.Cmd.Close	0	Decimal	BOOL
[-] Valve001.Alarm	{...}		VALVE_ALARMS
[-] Valve001.Alarm.Open_Timeout	0	Decimal	BOOL
[-] Valve001.Alarm.Close_Timeout	0	Decimal	BOOL

While adding stuff to my class, I did not have to re-create the object Valve001. RSLogix updated it for me, so all the properties and methods are available in my program.

Now, let's say you're working on this project with a couple hundred valves, and the customer decides to go with a different type of valve, that also has an analog input, that tells us the exact position of the valve. All we have to do is modify our VALVE\_IN sub-class to add this to every instance of the type VALVE:

Name:

Description:

Members: Data Type Size: 4 byte(s)

Name	Data Type	Style	Description
Open	BOOL	Decimal	Open Prox Switch
Closed	BOOL	Decimal	Closed Prox Switch
Position	INT	Decimal	Position Input (0-100%)

Of course, you would still have to write code to tell your program what to do with that information, but that is also the reason why PLC programmers still have a job.

For somebody that is not familiar with your program, it might be confusing to look at all your UDT's. We just made eight UDT's for one simple valve class! But remember, you only have to do this during the design phase. Once you have a solid design for all your classes (and made sure their names are self-explanatory), you will never have to look at your UDT folder again, and creating a new instance will be a breeze.

# Connecting Excel to ControlLogix

by John Schop

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

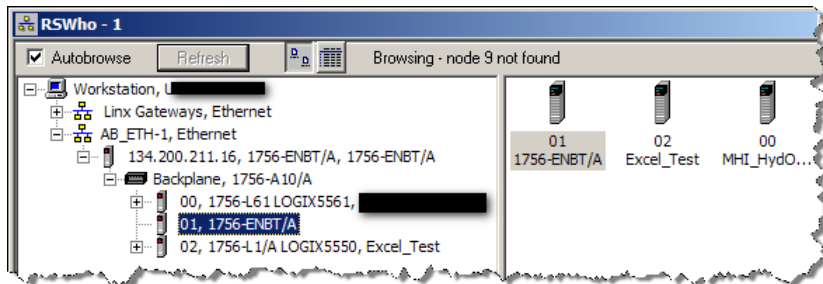
Let's fire up RSLogix first, and create a bunch of tags with values. In this example, I created 2 arrays, of the types [DINT](#) and REAL, each with a length of [10] tags. These arrays I filled with some values:

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	

I'm not going to do anything with the [PLC](#) program, I just need some data in a number of tags.

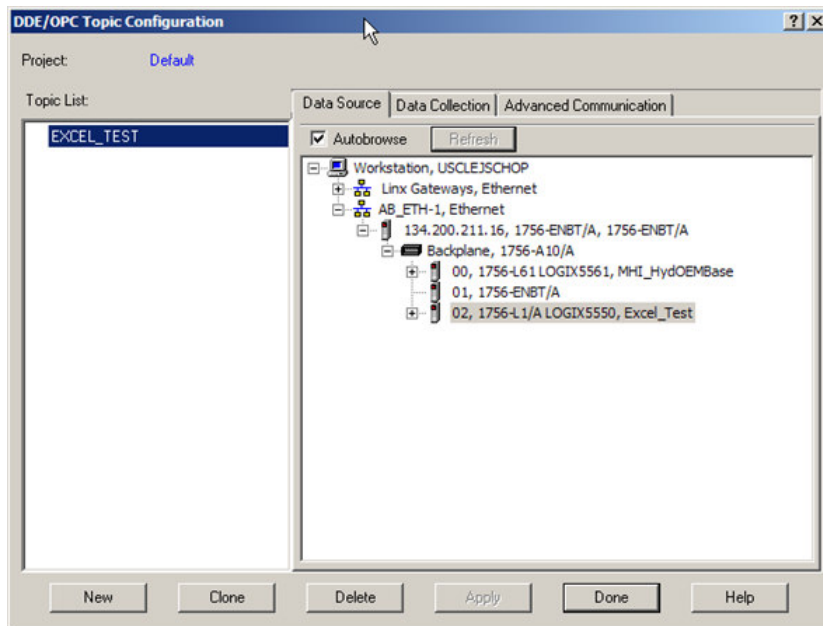
Next, we're going to set up a DDE/OPC Topic in RSLinx. Depending on the version of RSLinx you use, it might look slightly different, but you should be able to follow this with the screenshots.

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:

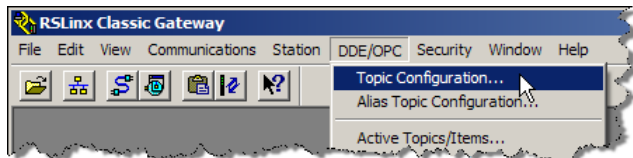


As you can see, I have a 10 slot CLX rack, with a 1756-ENBT card in slot 1 (address 134.200.211.16), and two processors, one in slot 0, and one in slot 2. The one in slot 2 is the processor we are going to use for this exercise.

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



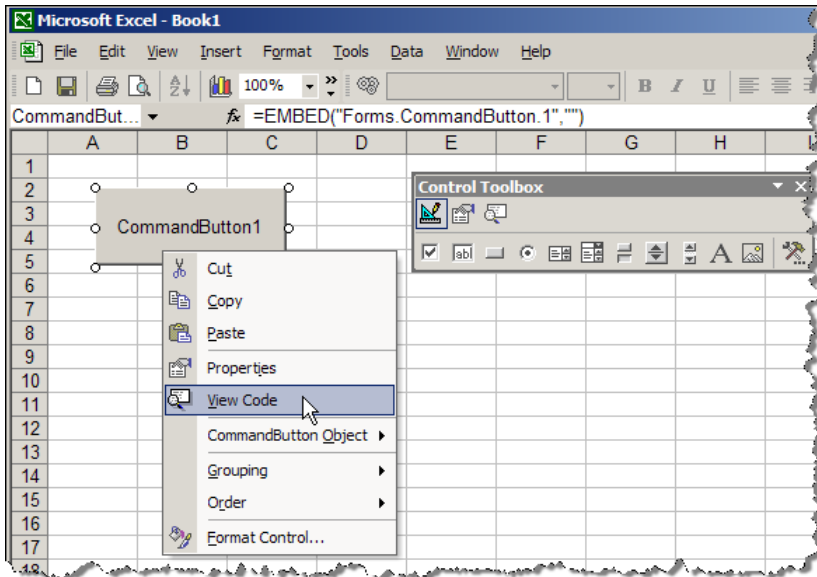
I'm going to create a new DDE/OPC topic called 'EXCEL\_TEST', and use the Logix5550 processor in slot 2 as the data source. In order to do this, you have to click the 'New' button, give the topic the desired name, and make sure the processor in slot 2 is selected as the source before you click 'Done'



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
```

The variable 'rslinx' will hold the number of the open channel. All subsequent DDE functions use this number to specify the channel.

To save you all the steps to program the rest of the code, here is the final code to get the array of REALs out of the controller, and put them in cells D2 – D11, and the array of DINTs in cells E2-E11.

```
Private Sub CommandButton1_Click()

    rslinx = OpenRSLinx() 'Open connection to RSLinx

    'Loop through reading the CLX array tags and
    'put the values into cells
    For i = 0 To 9
        'First the array of REALs
        'Get the value form the DDE link
        realdata = DDERequest(rslinx, "REAL_Array[" & i & "],L1,C1")

        'If there is an error, display a message box
        If TypeName(data) = "Error" Then
            If MsgBox("Error reading tag REAL_Array[" & i & "]. " & _
                "Continue with Read?", vbYesNo + vbExclamation, _
                "Error") = vbNo Then Exit For
        Else
            'No error, place data in cell
            Cells(2 + i, 4) = realdata
        End If

        'Now the array of DINTs
        'Get the value from the DDE link
        dintdata = DDERequest(rslinx, "DINT_Array[" & i & "],L1,C1")

        'If there is an error, display a message box
        If TypeName(data) = "Error" Then
            If MsgBox("Error reading tag DINT_Array[" & i & "]. " & _
                "Continue with Read?", vbYesNo + vbExclamation, _
                "Error") = vbNo Then Exit For
        Else
            'No error, place data in cell
            Cells(2 + i, 5) = dintdata
        End If
    Next i

    'Terminate the DDE connection
    DDETerminate rslinx

End Sub
```

Now we know how to read, it would of course be a lot of fun if we could write values as well. I would like to be able to change the values in the cells, and then hit a 'Write Data' button.

First, make another button on the sheet (mine looks like below now)

The screenshot shows a Microsoft Excel spreadsheet with columns A through E and rows 1 through 12. The data in the spreadsheet is as follows:

	A	B	C	D	E
1					
2				20	0
3		Read Data		20.1	1
4				20.2	2
5				20.3	3
6				20.4	4
7		Write Data		34.78	5
8				20.6	6
9				20.7	233
10				20.8	8
11				20.9	9
12					

And then write some code for the button:

```
Private Sub CommandButton2_Click()

    rslnx = OpenRSlnx() 'Open connection to RSlnx

    'Loop through the cells and write values to the CLX array tags
    For i = 0 To 9

        'First the array of REALs
        'Get the value from the DDE link
        realdata = DDERequest(rslnx, "REAL_Array[" & i & "],L1,C1")
        'If there is an error, display a message box
        If TypeName(data) = "Error" Then
            If MsgBox("Error reading tag REAL_Array[" & i & "]. " & _
                "Continue with write?", vbYesNo + vbExclamation, _
                "Error") = vbNo Then Exit For
        Else
            'No error, place data in CLX
            DDEPoke rslnx, "REAL_Array[" & i & "]", Cells(2 + i, 4)
        End If

        'Now the array of DINTs
        'Get the value from the DDE link
        dintdata = DDERequest(rslnx, "DINT_Array[" & i & "],L1,C1")
        'If there is an error, display a message box
        If TypeName(data) = "Error" Then
            If MsgBox("Error reading tag DINT_Array[" & i & "]. " & _
                "Continue with write?", vbYesNo + vbExclamation, _
                "Error") = vbNo Then Exit For
        Else
            'No error, place data in CLX
            DDEPoke rslnx, "DINT_Array[" & i & "]", Cells(2 + i, 5)
        End If
    Next i

    'Terminate the DDE connection
    DDETerminate rslnx
End Sub
```

The way this is implemented is of course very rudimentary, but once you get the concept, the sky is the limit.

To make this easier on everybody, I've included the [Excel file with the code already in it](#). The only thing you have to do to make this Excel sheet work, is make sure there is an DDE/OPC topic in your RSLinx setup called 'EXCEL\_TEST', and the arrays REAL\_Array and DINT\_Array in your controller (of at least length 10).