

# Micro800 Programming Basics



*For Classroom Use Only!*

LISTEN.  
THINK.  
SOLVE.®

 Allen-Bradley • Rockwell Software

**Rockwell  
Automation**

# Important User Information

This documentation, whether, illustrative, printed, “online” or electronic (hereinafter “Documentation”) is intended for use only as a learning aid when using Rockwell Automation approved demonstration hardware, software and firmware. The Documentation should only be used as a learning tool by qualified professionals.

The variety of uses for the hardware, software and firmware (hereinafter “Products”) described in this Documentation, mandates that those responsible for the application and use of those Products must satisfy themselves that all necessary steps have been taken to ensure that each application and actual use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards in addition to any applicable technical documents.

In no event will Rockwell Automation, Inc., or any of its affiliate or subsidiary companies (hereinafter “Rockwell Automation”) be responsible or liable for any indirect or consequential damages resulting from the use or application of the Products described in this Documentation. Rockwell Automation does not assume responsibility or liability for damages of any kind based on the alleged use of, or reliance on, this Documentation.

No patent liability is assumed by Rockwell Automation with respect to use of information, circuits, equipment, or software described in the Documentation.

Except as specifically agreed in writing as part of a maintenance or support contract, equipment users are responsible for:

- properly using, calibrating, operating, monitoring and maintaining all Products consistent with all Rockwell Automation or third-party provided instructions, warnings, recommendations and documentation;
- ensuring that only properly trained personnel use, operate and maintain the Products at all times;
- staying informed of all Product updates and alerts and implementing all updates and fixes; and
- all other factors affecting the Products that are outside of the direct control of Rockwell Automation.

Reproduction of the contents of the Documentation, in whole or in part, without written permission of Rockwell Automation is prohibited.

Throughout this manual we use the following notes to make you aware of safety considerations:

---

**WARNING**

Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

---

---

**IMPORTANT**

Identifies information that is critical for successful application and understanding of the product.

---

---

**ATTENTION**

Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you:

- identify a hazard
  - avoid a hazard
  - recognize the consequence
- 

---

**SHOCK HAZARD**

Labels may be located on or inside the drive to alert people that dangerous voltage may be present.

---

---

**BURN HAZARD**

Labels may be located on or inside the drive to alert people that surfaces may be dangerous temperatures.

---

---

# Micro800 Programming Basics

---

## Contents

Before you begin .....	4
About this lab .....	4
Tools & prerequisites .....	4
Get familiar with the Connected Components Workbench design environment .....	5
Create a CCW project and program a Micro850 controller.....	8
Build and Download your Micro850 Application.....	15
Debug your Micro850 program .....	18
Learn about Variables and Data Types .....	23
Data types.....	24
Learn how to create variables .....	27
Learn how to Implement an Instruction Block .....	25
Learn how to add a plug-in module.....	38
Learn about User Defined Function Blocks .....	42

---

## Before you begin

### About this lab

Connected Components Workbench (CCW) is the integrated design environment software package that is used to program, design, and configure your Rockwell Automation Connected Components devices such as, Micro800 programmable logic controllers, PowerFlex drives, SMC soft-starters, and PanelView Component operator interface terminals.

This lab will demonstrate and help guide you on how to use and program a Micro850 controller using the CCW software.

This lab takes approximately 60 minutes to complete.

### Tools & prerequisites

- Software: Connected Components Workbench v9.00.00
- Hardware: Micro850 Programmable Logic Controller, Catalog 2080-LC50-24QBB

### Please note:

CCW is an all-encompassing software package for component class controllers (or- small / micro controllers). It contains the application programming environment for the Micro800 Programmable Controllers (PLC), Drives (Variable Frequency Drives or VFD's which use AC voltage, converted to DC, generate a Pulse Width Modulated (PWM) signal to control AC induction Motors) Human-Machine Interface (HMI) displays for control, feedback to an operators panel and some Safety PLC's.

With that- all User Manuals are included in CCW as well as a very extensive Help menus.

At any time that you need help or reference to any item, component or object, simply click on the help pulldown

---

## Get familiar with the Connected Components Workbench design environment

This section will help get you familiar with the Connected Components Workbench design environment. As our goal to help simplify your engineering efforts, we've developed CCW using the Microsoft Visual Studio Shell. This common and popular software shell provides you the benefits of a common look, feel, and design environment when transitioning from other similar software packages.

Let's take a couple minutes to get familiar with the CCW design environment.

### 1. Start the Connected Components Workbench software.

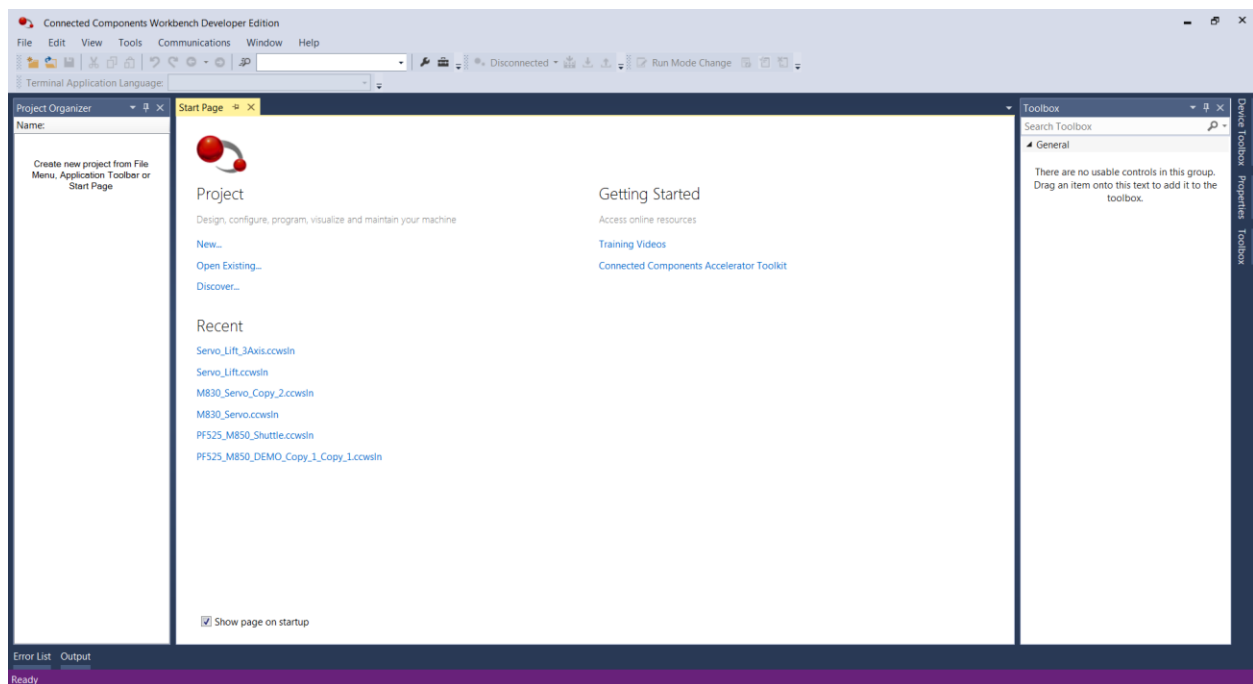
Double-click the Connected Components Workbench shortcut icon on your desktop.



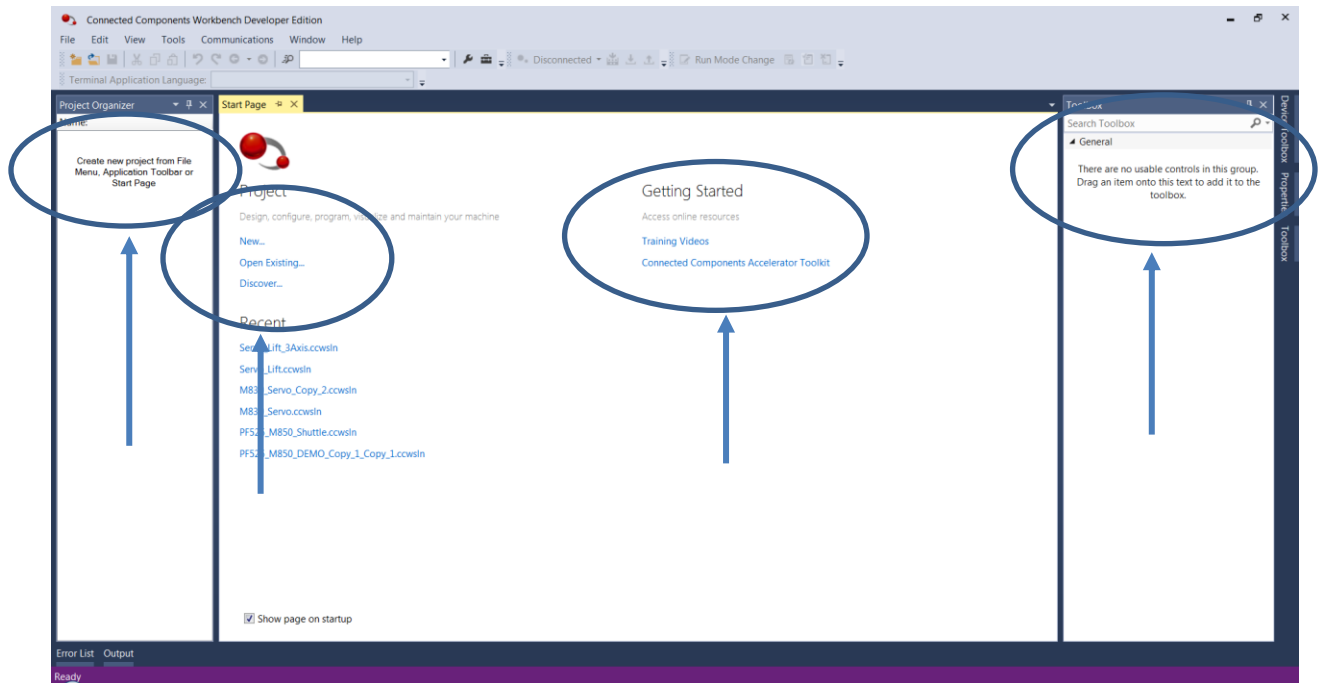
You can also launch the program from your Windows Start Menu by going to: Start > All Programs > Rockwell Automation > CCW > Connected Components Workbench

### 2. Get familiar with the CCW design environment.

This is the default splash screen.



Below are descriptions of each of the panels' contents and the general task the pane is used for.



## Project Organizer

The Project Organizer displays the contents of your project in an organized tree view, providing access to each of the devices and project elements. From the Project Organizer, you can add, move, or delete devices and project elements, as well as double-click them to display their contents.

If your project contains a Micro800 controller, the Project Organizer also displays the logic programs, variables, and user-defined function blocks associated with that Micro800 controller.

More on this later in the Lab.

## Project

Create new or open an existing project, or discover (connect) to a Processor on a network

- Discover - discovers devices that are connected to your computer and recognized by Connected Components Workbench.

## Getting Started

This will launch your web browser to review instructional videos on various aspects of CCW. Take a moment to review the following videos:

- 
- 

Connected Components Accelerator Toolkit

This is for an overall System design which builds an entire system, Bill of Material and sample code. This is for more advanced system generation and will not be covered in this Lab.

## Toolbox

The Toolbox displays icons for items that you can add to programs. From the Toolbox, you can drag and drop individual Toolbox elements onto a design view surface or copy and paste these into a code editor. Each of these actions adds the fundamental code to create an instance of the Toolbox item in the active project file

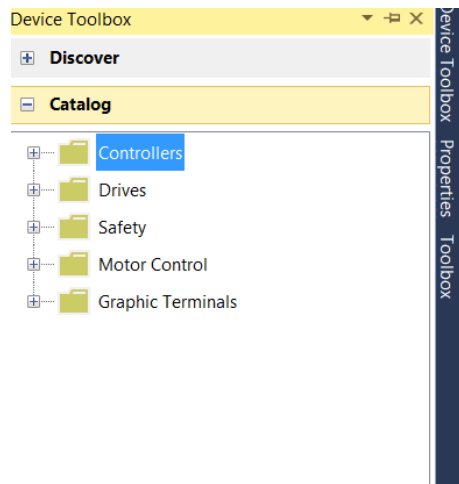
The Toolbox displays all of the devices that you can add to your Connected Components Workbench™ project. On the right side of this box- there are (3) tabs on the side:

Click on Device Toolbox

From the Device Toolbox, you can select devices for your project from the following two tabs:

- Discover - discovers devices that are connected to your computer and recognized by Connected Components Workbench.
  - Catalog - browses a catalog of devices that are included with Connected Components Workbench.
- The Properties tool box will display the properties and attributes of objects selected while programming

Device Toolbox



The Toolbox displays icons for items that you can add to programs. From the Toolbox, you can drag and drop individual Toolbox elements onto a design view surface or copy and paste these into a code editor. Each of these actions adds the fundamental code to create an instance of the Toolbox item in the active project file

---

## Create a CCW project and program a Micro850 controller

In this section, you will create a CCW project and learn how to create a program for a Micro850 programmable logic controller.

You will learn how to:

- Create a CCW project
- Add a Micro850 controller to your project
- Program a simple motor control seal-in circuit

1. If not already open-start the Connected Components Workbench software.

Double-click the Connected Components Workbench shortcut icon on your desktop.

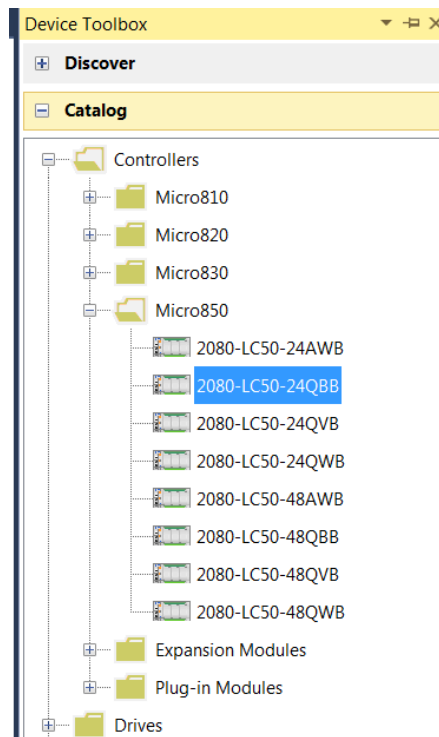


Add a Micro850 to your project. Locate the **Device Toolbox** (upper right-hand corner). Expand **Catalog** and locate the **Controllers** folder. Expand the Controllers folder and locate the Micro850 controller catalog **2080-LC50-24QBB**.

Or

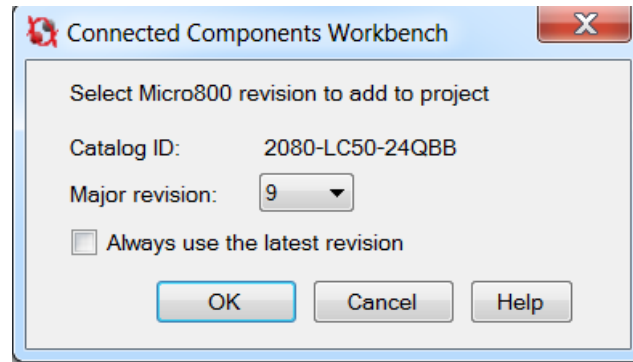
Select New > Pop menu will appear with devices to choose from. Choose Controllers > **2080-LC50-24QBB** > **Add to Project**

Double click this Micro850 controller catalog. This will add a Micro850 controller to your Project.

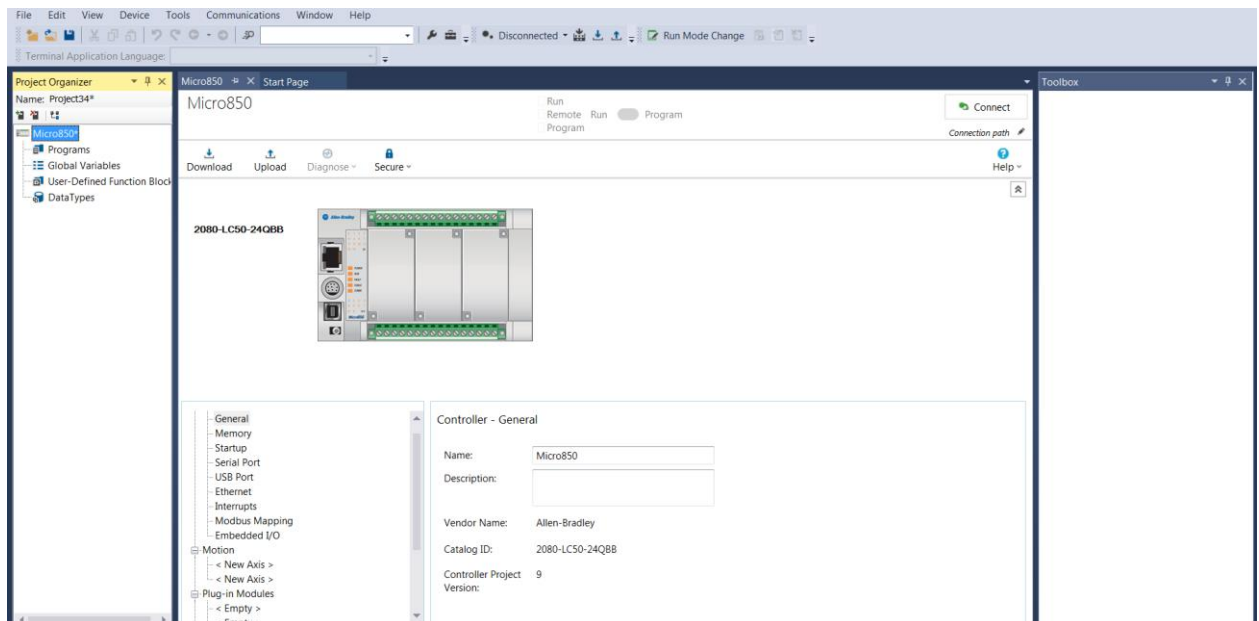




-You will see a pop-up asking what FirmWare (fw) your system will be at.  
Fw levels re-flash the processor to various levels of capability, features or corrections to anomalies.  
Select Major revision 9



2. Notice that the Micro850 shows up in your Project Organizer on the left-hand side.



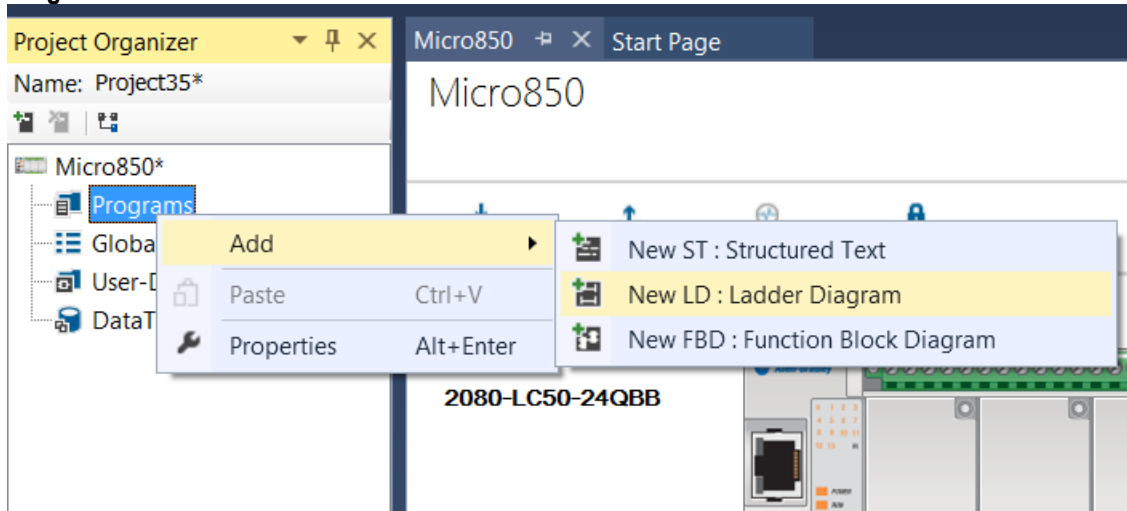
On the left of your screen is the Project Organizer

In the middle are the properties and configuration of the Processor

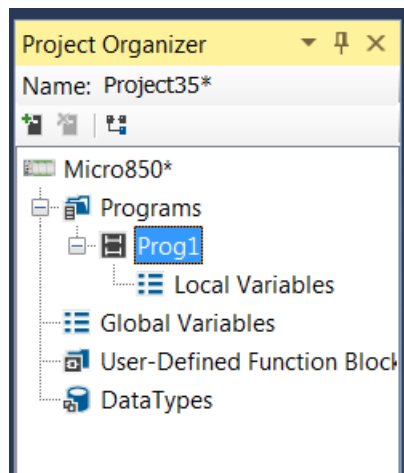
Alternatively, you can drag and drop the controller from the Device Toolbox into the Project Organizer.

3. Add a Ladder Diagram program.

Right-click **Programs** under the Micro850 in your Project Organizer, and select **Add → New LD : Ladder Diagram**.



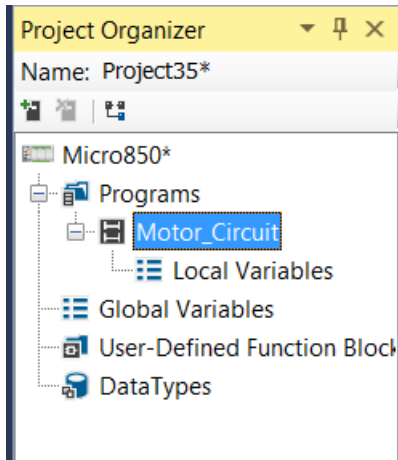
4. Notice a new Ladder Diagram program called **Prog1** will be added under Programs.



Micro850 controllers allow you to create multiple programs as well as use multiple types of programs (such as Structured Text or Function Block Diagram) in the same controller application.

Since we'll be creating a Motor Circuit in this program, let's rename it **Motor\_Circuit**.

5. Right-click the **UntitledLD** program and select **Rename**, and name the Program **Motor\_Circuit**

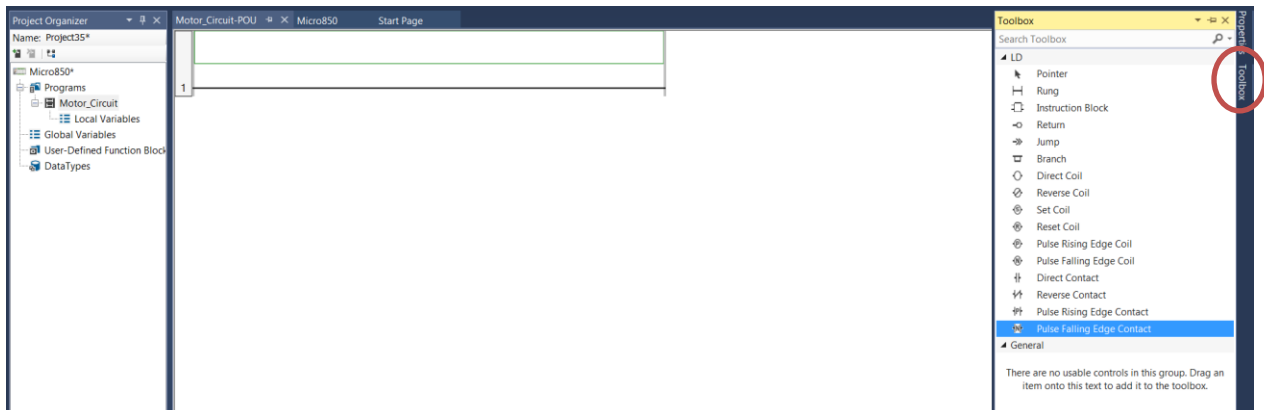


6. Create a motor seal-in circuit in your Motor\_Circuit Ladder Diagram program.

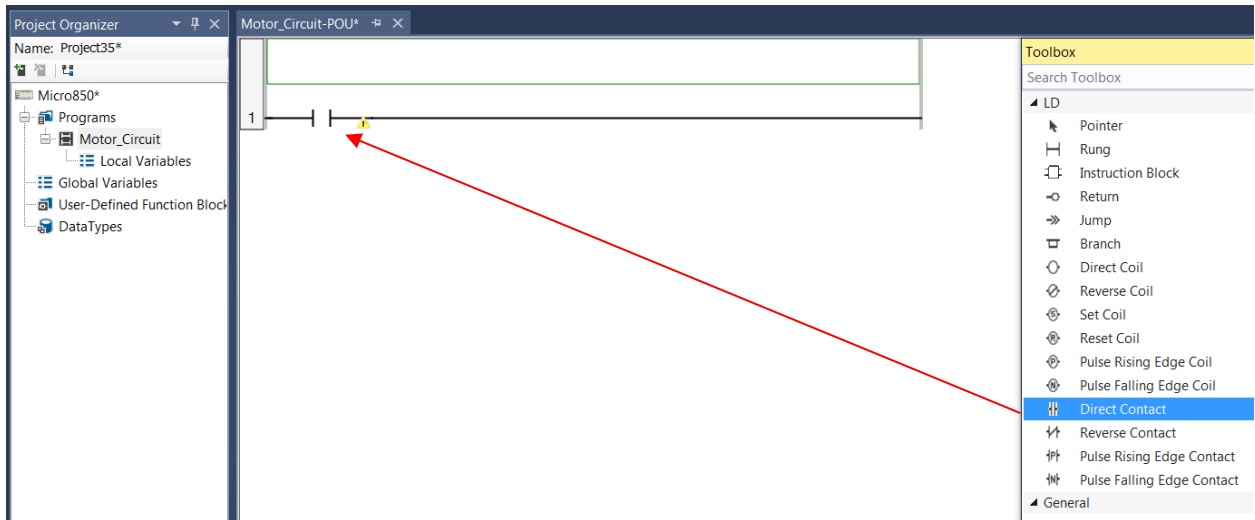
This circuit will use the DI0 switch on the Demo as your Start button, and the DI1 switch as your Stop button. The pilot light wired to DO8 will simulate your motor coil.

In many applications, a 'seal-in circuit not only starts a process or function, but 'seals' it in that ON or OFF state. Many times, per missives are required to turn ON or OFF the output or- to hold it in the required state.

7. Double-click the **Motor\_Circuit** program. A ladder diagram editor will appear in the main project workspace with one empty rung. On the right side is the Toolbox of objects. If your screen does not show these- click on the tab to the right called 'Toolbox'

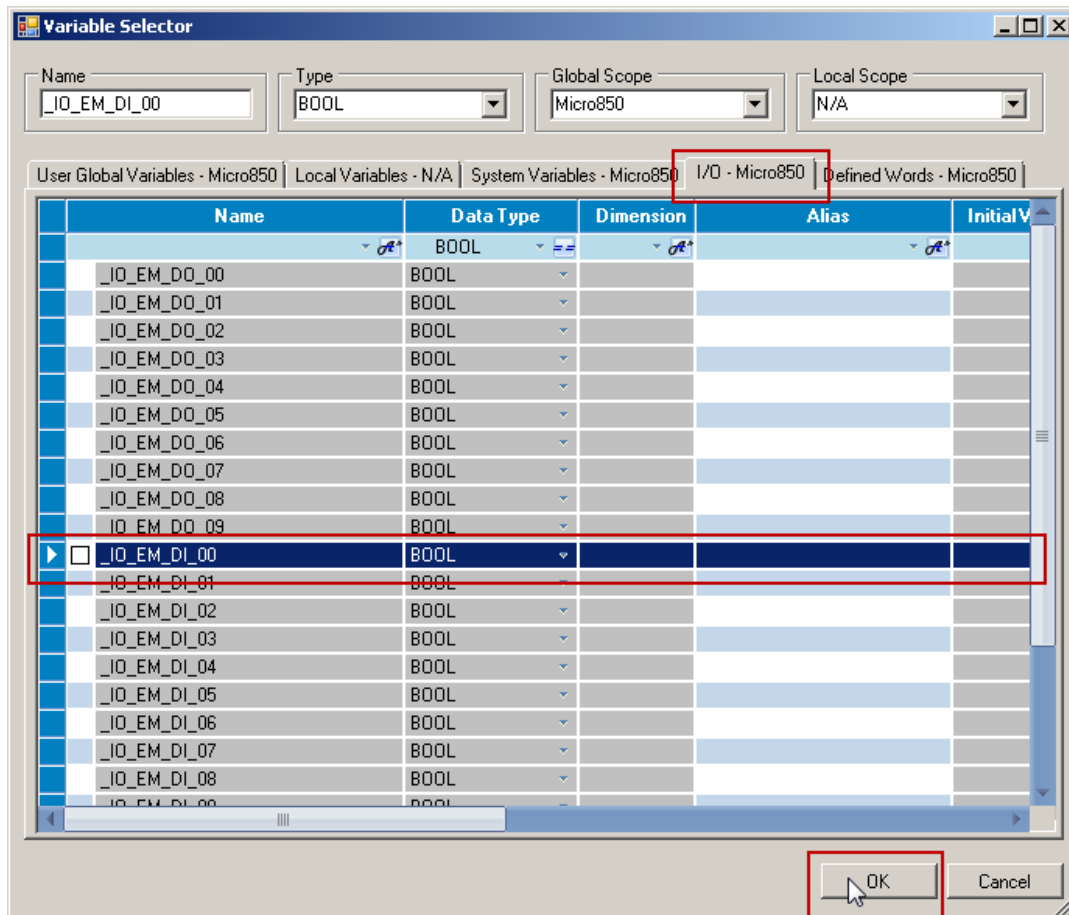


8. Locate the **Direct Contact** instruction in the Toolbox pane (right-hand side), and drag-and-drop it onto the left side of the rung.

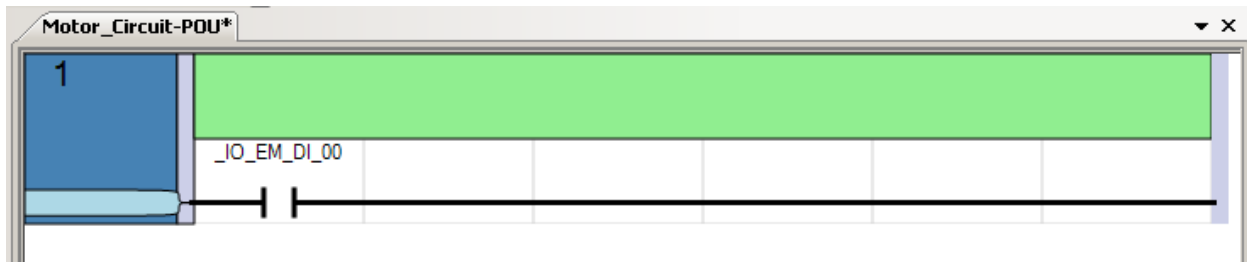


After inserting the Direct Contact instruction, the Variable Selector will automatically pop-up, allowing you to select the variable or I/O point to assign to this instruction.

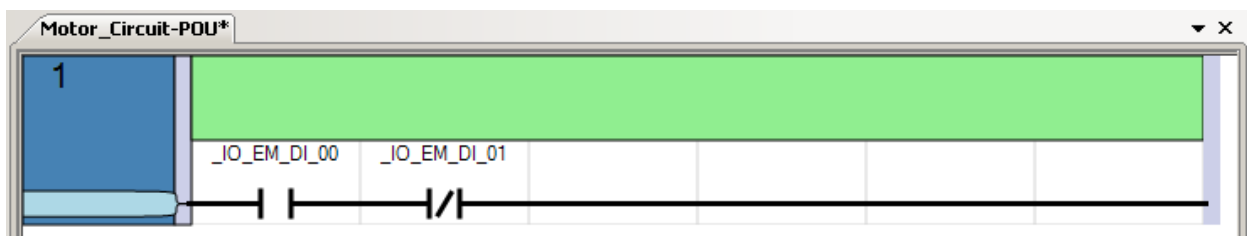
We will be assigning an embedded I/O point to this instruction; therefore we want to select the **I/O – Micro850** tab to show the available embedded I/O points. Then select **\_IO\_EM\_DI\_00**, and click **OK**.



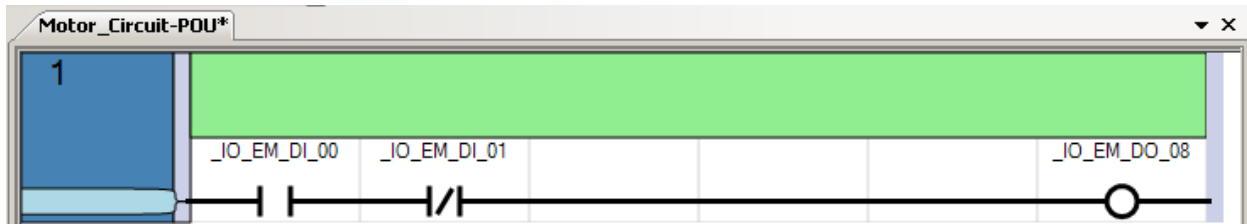
9. Your rung should look like the following.



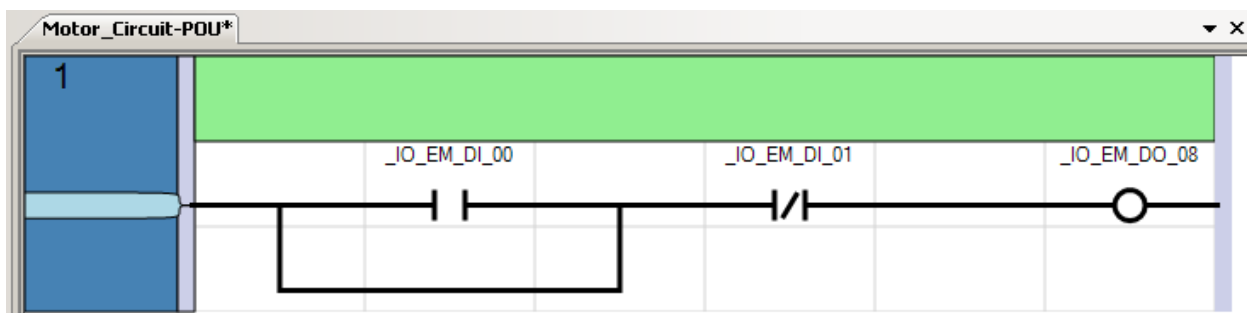
10. Locate the **Reverse Contact** instruction in the Toolbox and drag-and-drop it onto your rung, just to the right of the Direct Contact you inserted in the previous step. Assign it to the embedded I/O point, `_IO_EM_DI_01`. Your rung should look like the following.



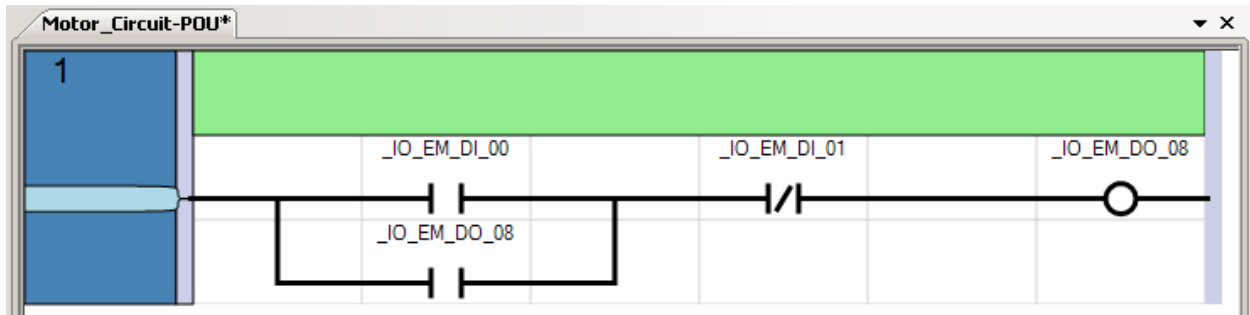
11. Locate the **Direct Coil** instruction in the **Toolbox**, and drag-and-drop it onto the far right side of the rung, and assign it to the embedded I/O point, `_IO_EM_DO_08`. Your rung should look like the following.



12. Locate the **Branch** instruction in the Toolbox and drag-and-drop it on top of the Direct Contact on the far left of the rung. Your rung should look like the following.



13. Drag-and-drop a **Direct Contact** in to the Branch that you just added, and assign it to the embedded I/O point, **\_IO\_EM\_DO\_08**. Your rung should look like the following.



14. You've completed creating your motor seal-in circuit. Save your project and proceed to the next section to build and download your application to the Micro850 controller.

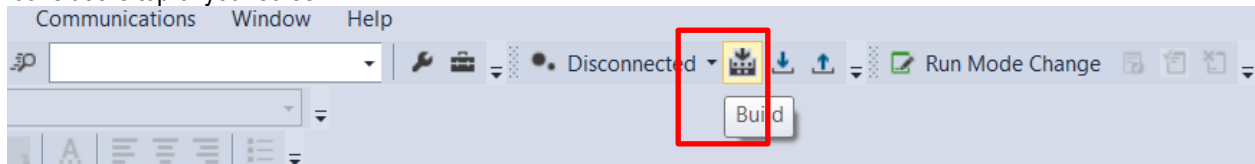
---

## Build and Download your Micro850 Application

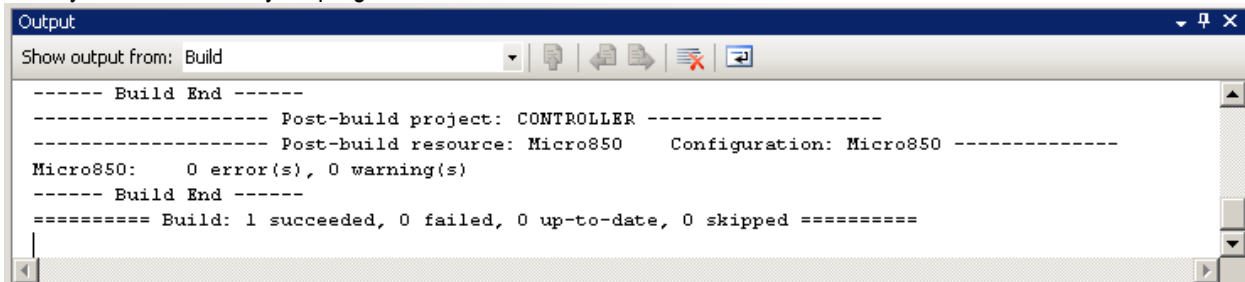
In this section, you will learn how to build and download your Micro850 application to the controller.

Before you can download an application to the controller, you must build it to verify that there are no errors with the programming.

1. Build your application by right-clicking the Micro850 in your Project Organizer, and selecting **Build**, or via the icons at the top of your screen

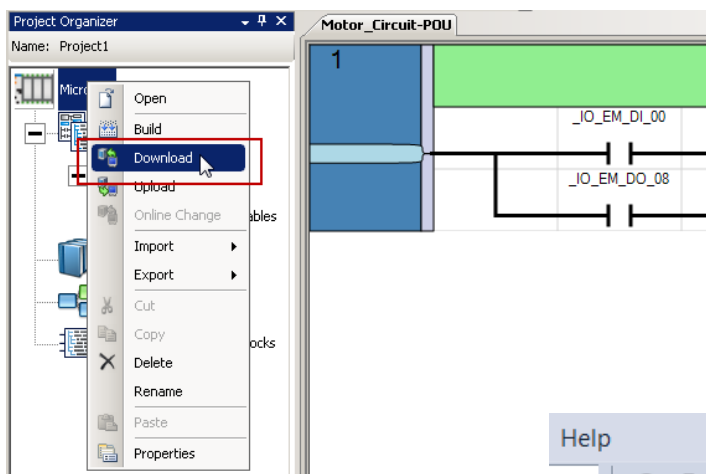


When the build is complete, you will see a message in the **Output** panel that the build has succeeded. If there were errors in your programming, then they would be listed in the **Output** panel as well – and clicking on the error would direct you to the error in your program.

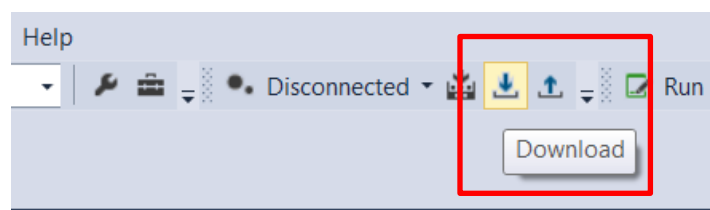


Now that your build has completed, you can download the program to your controller.

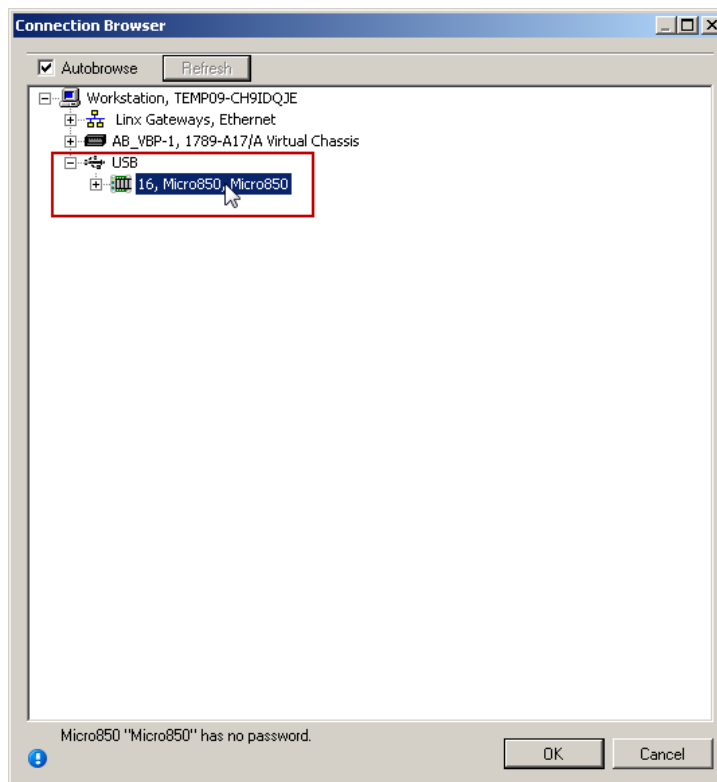
2. Download your program to your Micro850 by right-clicking the Micro850 in your Project Organizer, and selecting **Download**, or click on the download icon



Or

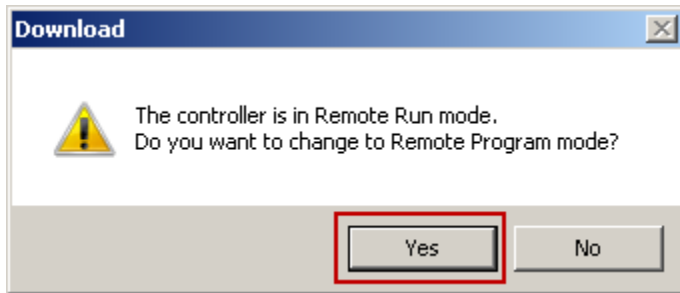


3. The Connection Browser will appear. Browse for your controller by expanding **USB** and selecting **16, Micro850, Micro850**, then clicking **OK**.

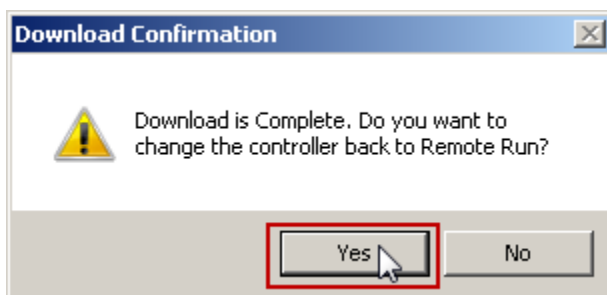




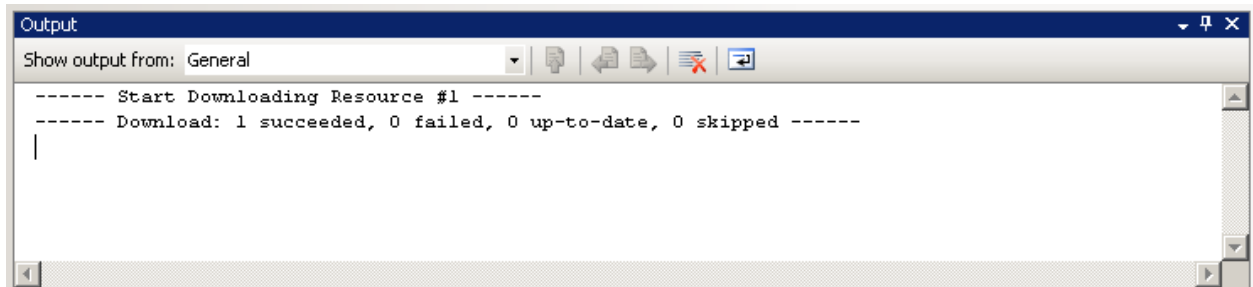
4. If the Controller is in Run Mode, you will be prompted to change it to Remote Program mode. Click **Yes**.



5. The download will proceed. When the download is complete, you will be prompted to put the controller back in to Run Mode. Click **Yes**.



6. Notice the messages in the Output panel that indicate the Download has completed successfully.



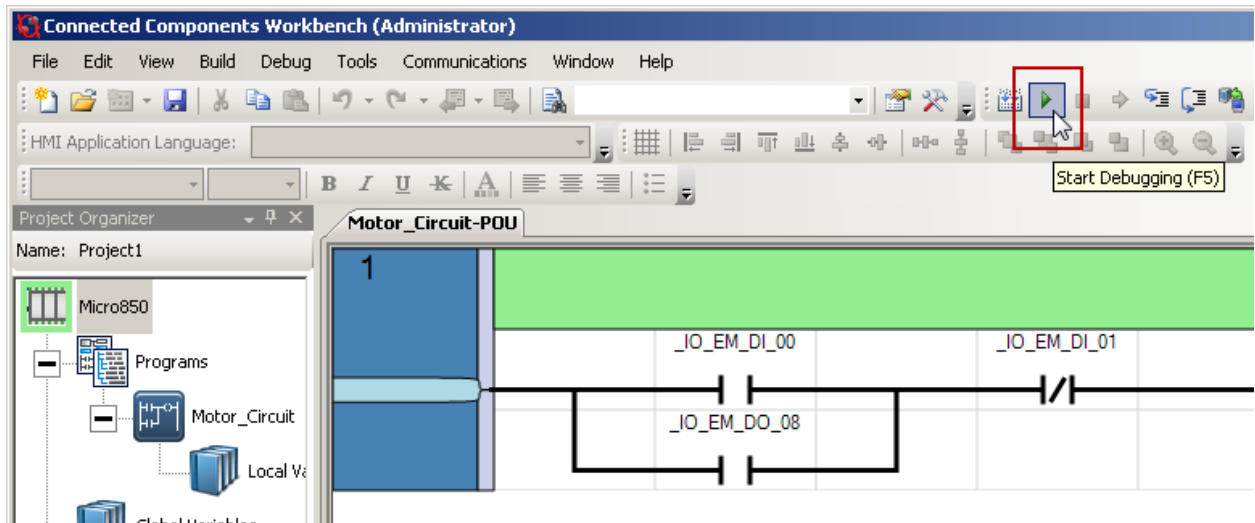
7. You have completed downloading to your Micro850 controller. Proceed to the next section to test and debug your application.

---

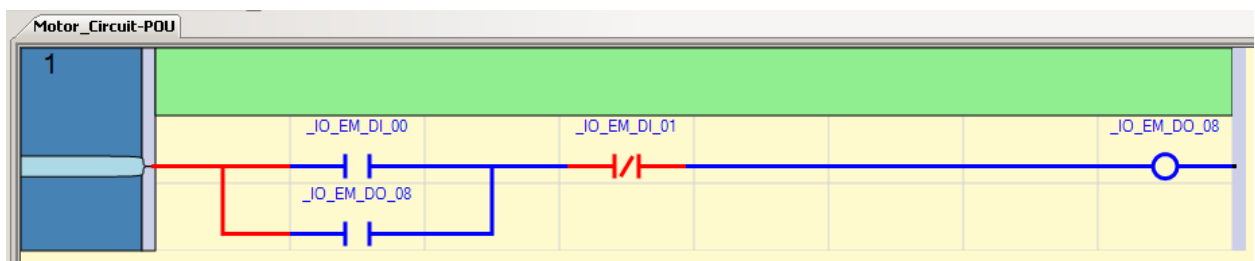
## Debug your Micro850 program

In this section, we will demonstrate how to debug your Micro850 program. By debugging your program, you can view your program visually in real-time and watch values change in the program, as well as visually debug your Ladder Logic or Function Block Diagram.

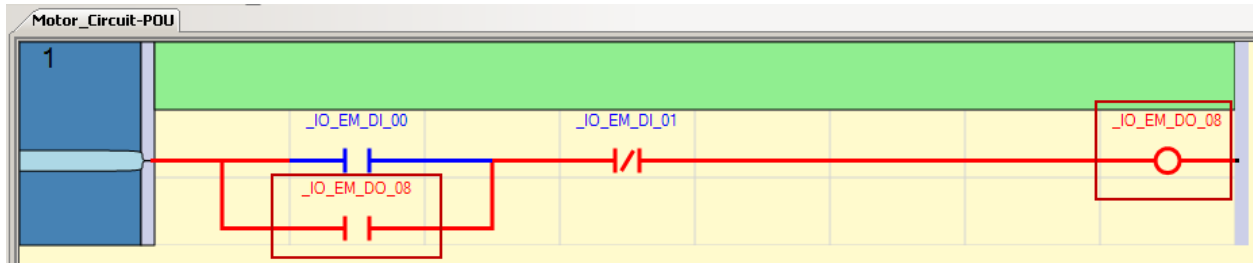
1. Click the “Play” button in the Toolbar at the top of the workspace environment. This will put your program into “Debug” mode.



2. You should see the Ladder Diagram change color. The rung will turn blue, and any Boolean instructions that are active will turn Red.



3. Turn and release the **D10** switch on the Demo hardware. Notice the `_IO_EM_DI_00` Direct Contact instruction turn red as you turn on the switch, and then turn blue as you release it (if you turn and release the switch too fast, you may not see it update in the ladder diagram). Then notice the `_IO_EM_DO_08` Direct Contact and Direct Coil instructions turn red. You should also notice that the DO8 light on the Demo is now on.

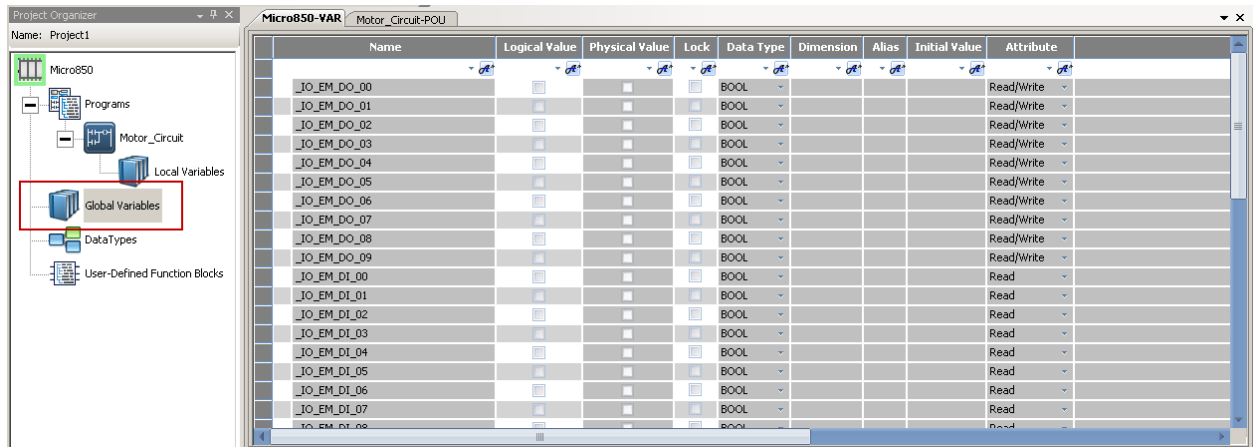


This is a typical motor seal-in circuit (and can also be applied in non-motor circuits as well). The Output Coil is turned on using a Direct Contact and then the active state of the Output Coil seals in the circuit. The circuit is unsealed when a Reverse Contact (normally closed) is opened. DI\_01 would be the permissive to allow the circuit to work, such as a door that must be closed before operation. If the door is closed- the system can start. If it opens- it stops

4. Turn and release the DI1 switch on the Demo to turn off the output. Notice the light on your Demo turn off and the corresponding changes in your Ladder Diagram.
5. So far, you've debugged your program primarily by viewing real-time changes in the Ladder Diagram editor. In some instances, you may just want to view the real-time changes in a list format. You can do this by looking at them in the Variables list.

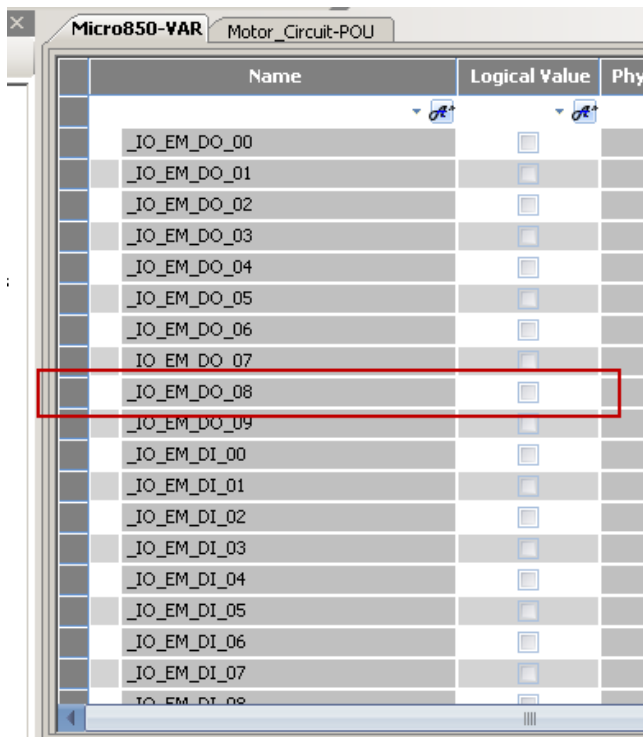
Since the variables we're working with right now are embedded I/O points, we need to open the Global Variables list.

6. Double-click **Global Variables** in your Project Organizer. The Global Variables pane will launch in a new tab in the main project workspace.

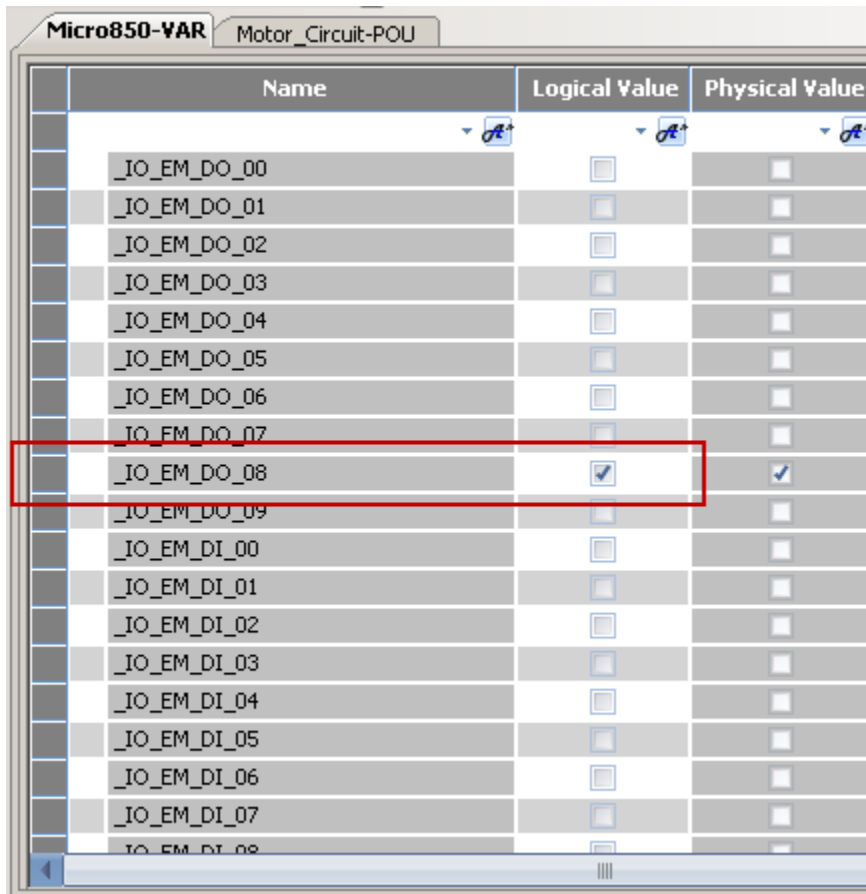


The Global Variables pane is a list of all the Global Variables in your program. You will learn more about Global Variables and other types of variables in the next section.

7. Locate the **\_IO\_EM\_DO\_08** embedded I/O variable in the Global Variable list, and notice that the logical value checkbox is empty.

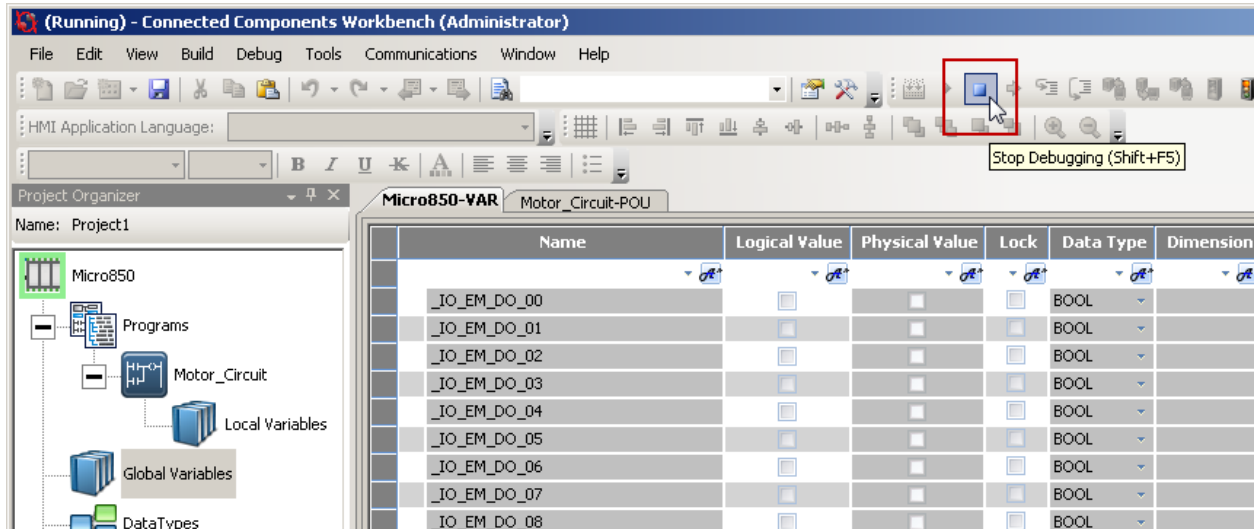


8. Turn and release the DI0 switch on the Demo. Notice there is now a checkmark in the Logical Value for \_IO\_EM\_DO\_08, and the light on the Demo is on. You may have also noticed a checkmark appear in the \_IO\_EM\_DI\_00 logical value as you turned on the switch and then noticed the checkmark go away as you released the switch.



Name	Logical Value	Physical Value
_IO_EM_DO_00	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_01	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_02	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_03	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_04	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_05	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_06	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DO_07	<input type="checkbox"/>	<input type="checkbox"/>
<b>_IO_EM_DO_08</b>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
_IO_EM_DO_09	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_00	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_01	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_02	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_03	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_04	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_05	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_06	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_07	<input type="checkbox"/>	<input type="checkbox"/>
_IO_EM_DI_08	<input type="checkbox"/>	<input type="checkbox"/>

9. Turn and release the DI1 switch on the Demo. Notice the checkmark go away in the logical value for \_IO\_EM\_DO\_08, and the light turn off on the Demo.
10. You have completed debugging your program. To exit Debug mode, click the Stop button in the toolbar.



Being able to debug a program in real-time is a very valuable tool in the programming and trouble-shooting process, and Connected Components Workbench makes this design step very simple and easy.

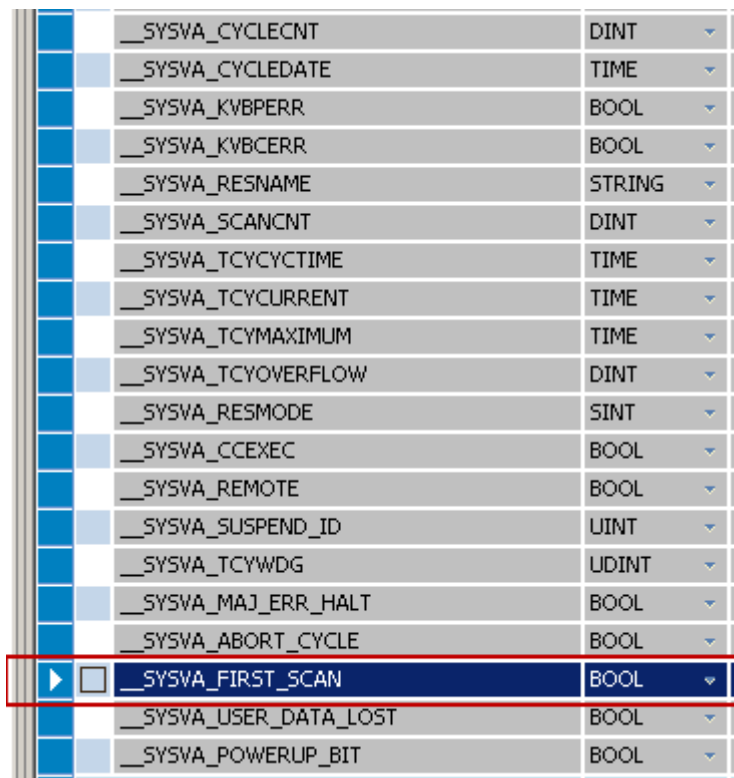
---

## Learn about Variables and Data Types

In this section, we will discuss what a Variable is, and the different Data Types available.

A variable is a unique identifier of data. A basic example of a variable is what you've already been referencing in the lab for Embedded I/O points. The Embedded I/O variables are Boolean data types that are direct references to the embedded input and outputs on the controller. They are identified by variables that start with the prefix `_IO_EM`, and are globally scoped. We will discuss variable scope a little later.

Micro800 controllers also have System Variables of varying data types that reference internal system values of the controller that a user may want to use in their programming, or for troubleshooting purposes. System Variables start with the prefix `__SYSVA`. An example of a system variable that is commonly used is the `__SYSVA_FIRST_SCAN` variable. This is a Boolean variable that is TRUE when the Micro800 controller is going through its first scan of the program – typically used for programming startup routines.



The image shows a screenshot of a software interface displaying a list of system variables. The variables are listed in a table with three columns: a checkbox, the variable name, and the data type. The variable `__SYSVA_FIRST_SCAN` is highlighted with a red rectangle. The data types for the variables are: DINT, TIME, BOOL, BOOL, STRING, DINT, TIME, TIME, TIME, DINT, SINT, BOOL, BOOL, UINT, UDINT, BOOL, BOOL, BOOL, BOOL, and BOOL.

<input type="checkbox"/>	<code>__SYSVA_CYCLECNT</code>	DINT
<input type="checkbox"/>	<code>__SYSVA_CYCLEDATE</code>	TIME
<input type="checkbox"/>	<code>__SYSVA_KVBPERR</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_KVBCERR</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_RESNAME</code>	STRING
<input type="checkbox"/>	<code>__SYSVA_SCANCNT</code>	DINT
<input type="checkbox"/>	<code>__SYSVA_TCYCYCTIME</code>	TIME
<input type="checkbox"/>	<code>__SYSVA_TCYCURRENT</code>	TIME
<input type="checkbox"/>	<code>__SYSVA_TCYMAXIMUM</code>	TIME
<input type="checkbox"/>	<code>__SYSVA_TCYOVERFLOW</code>	DINT
<input type="checkbox"/>	<code>__SYSVA_RESMODE</code>	SINT
<input type="checkbox"/>	<code>__SYSVA_CCEXEC</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_REMOTE</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_SUSPEND_ID</code>	UINT
<input type="checkbox"/>	<code>__SYSVA_TCYWDG</code>	UDINT
<input type="checkbox"/>	<code>__SYSVA_MAJ_ERR_HALT</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_ABORT_CYCLE</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_FIRST_SCAN</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_USER_DATA_LOST</code>	BOOL
<input type="checkbox"/>	<code>__SYSVA_POWERUP_BIT</code>	BOOL

Variables can be created dynamically as you need them, and they can be named anything you want (as long as it's not a reserved name). You can also create variables for local program use only, or you can create them for Global use (for all programs to use) – this is what we refer to as variable scope. Global Variables are created in the Global Variables list, and Local Variables are created in the Local Variables list of the specific program.

Being able to create variables dynamically and use custom names provides you, as a programmer, great flexibility and customization that will help you create code and troubleshoot faster.

## Data types

When you create a variable, you have to specify its data type. A data type defines the type of data that the variable represents, such as an integer, real (floating point), Boolean, time, double integer, etc. Data types can also be data structures of an Instruction Block.

CCW supports the 19 elementary IEC 61131-3 data types below.

- BOOL
- SINT
- USINT
- BYTE
- INT
- UINT
- WORD
- DINT
- UDINT
- DWORD
- LINT
- ULINT
- LWORD
- REAL
- LREAL
- TIME
- DATE
- STRING

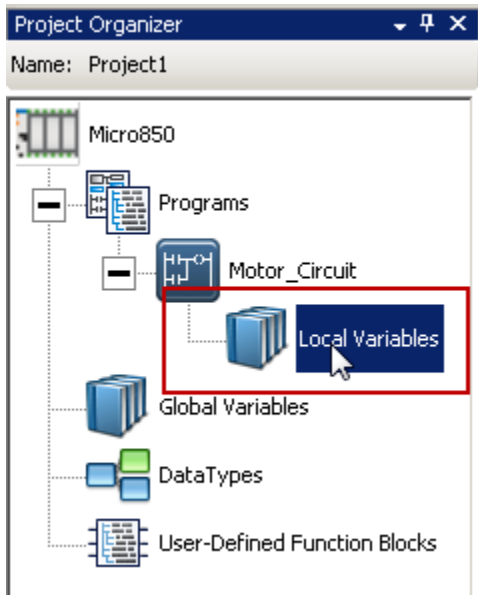


---

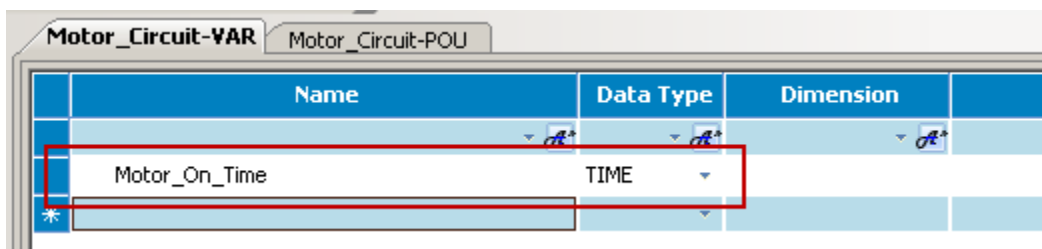
## Learn how to create variables

In this section of the lab, you will learn how to create variables for use in your program. The variables you create in this section of the lab will be used in the next section of the lab.

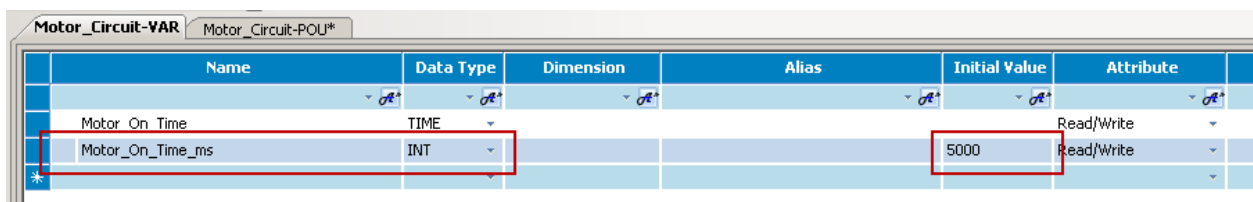
1. Double-click **Local Variables** in your **Motor\_Circuit** program to launch the Variables panel.



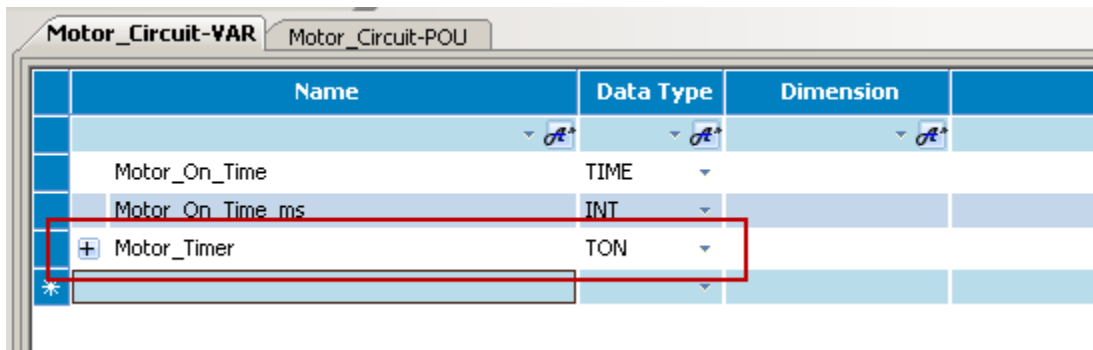
2. Create a variable called **Motor\_On\_Time** of Data Type **TIME**.



3. Create a variable called **Motor\_On\_Time\_ms** of Data Type **INT** and with an Initial Value of **5000**.



4. Create a variable called **Motor\_Timer** of Data Type **TON**.



Name	Data Type	Dimension
Motor_On_Time	TIME	
Motor_On_Time_ms	INT	
Motor_Timer	TON	

A TON data type is actually the data structure of a Timer-on-Delay Instruction Block. We will discuss Instruction Blocks in the next section.

5. You have completed creating variables to be used in the next section of the lab.

---

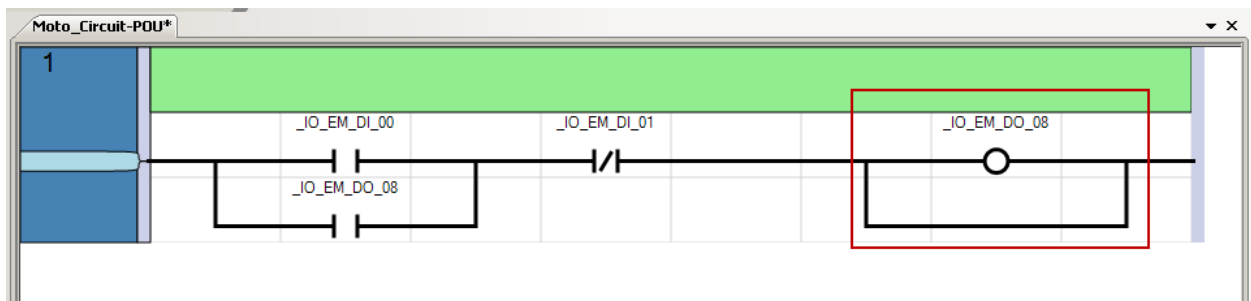
## Learn how to Implement an Instruction Block

An Instruction Block is essentially a function block that has been predefined to perform a specific task or function. Instruction Blocks include functions such as Timer-on-delay, Timer-off-delay, Math instructions, Data-type conversions, Motion instructions, and so forth.

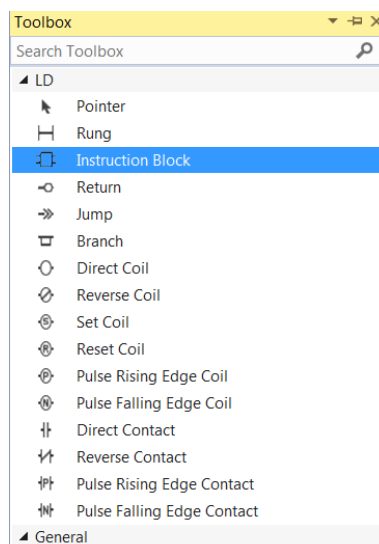
In this section of the lab, you will learn how to implement a Timer-On-Delay Instruction Block (TON). This instruction block will be inserted into your motor circuit and will turn on the motor coil, and then automatically turn off the motor coil after 5 seconds.

You will also learn how to implement an ANY\_TO\_TIME Data Conversion Instruction Block to convert an Integer to a Time value.

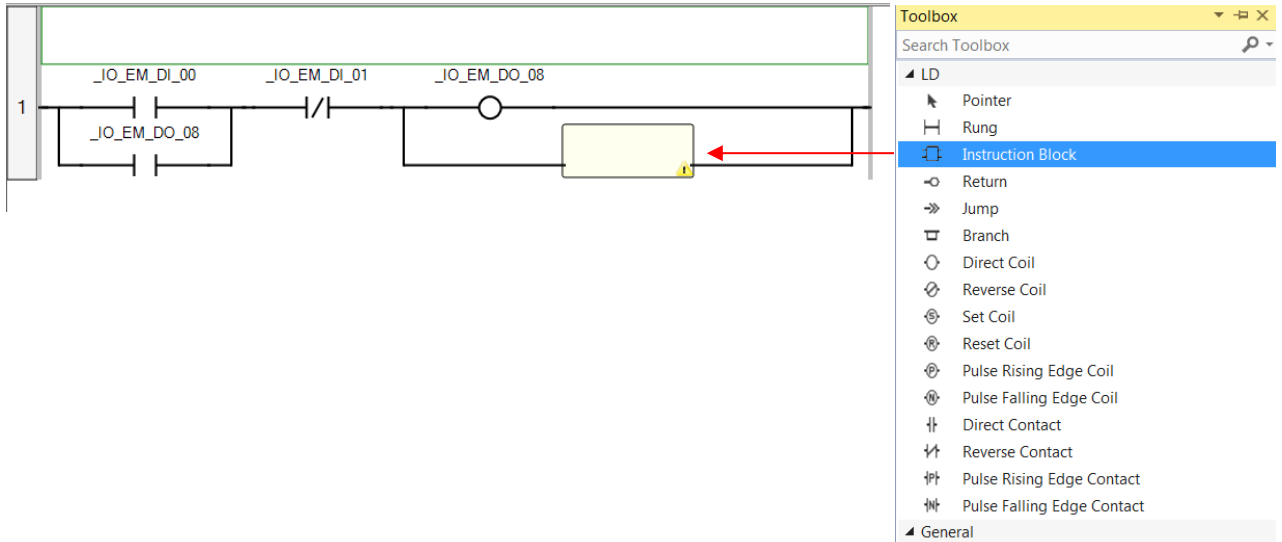
1. Drag-and-drop a **Branch** instruction to right side of the rung, wrapping around the coil instruction.



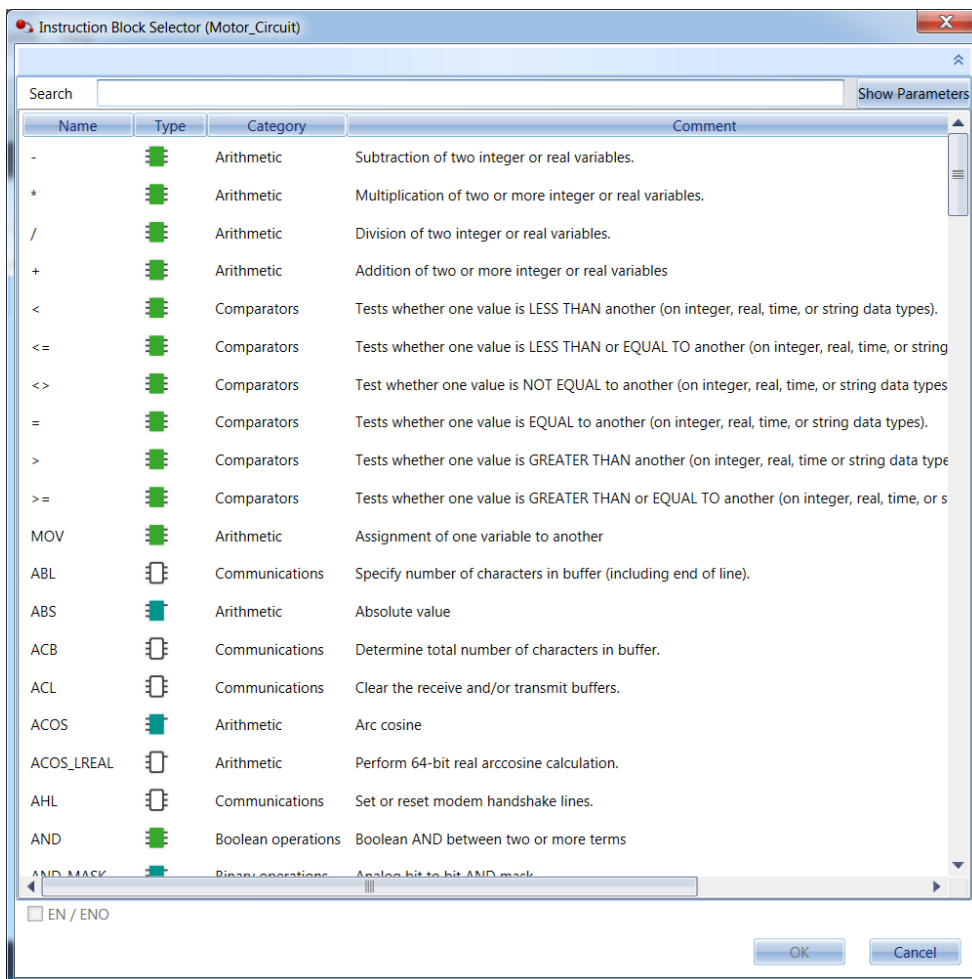
2. Locate the **Block** instruction in the Toolbox.



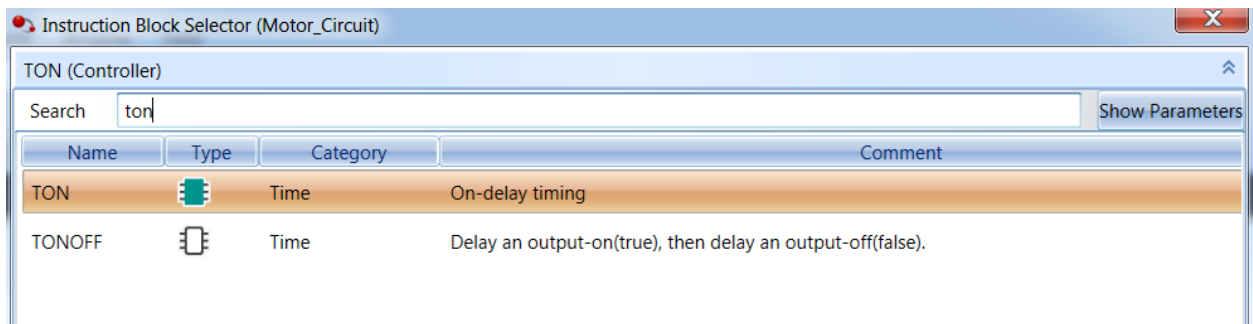
3. Drag-and-drop this Block instruction into the branch that you just added.



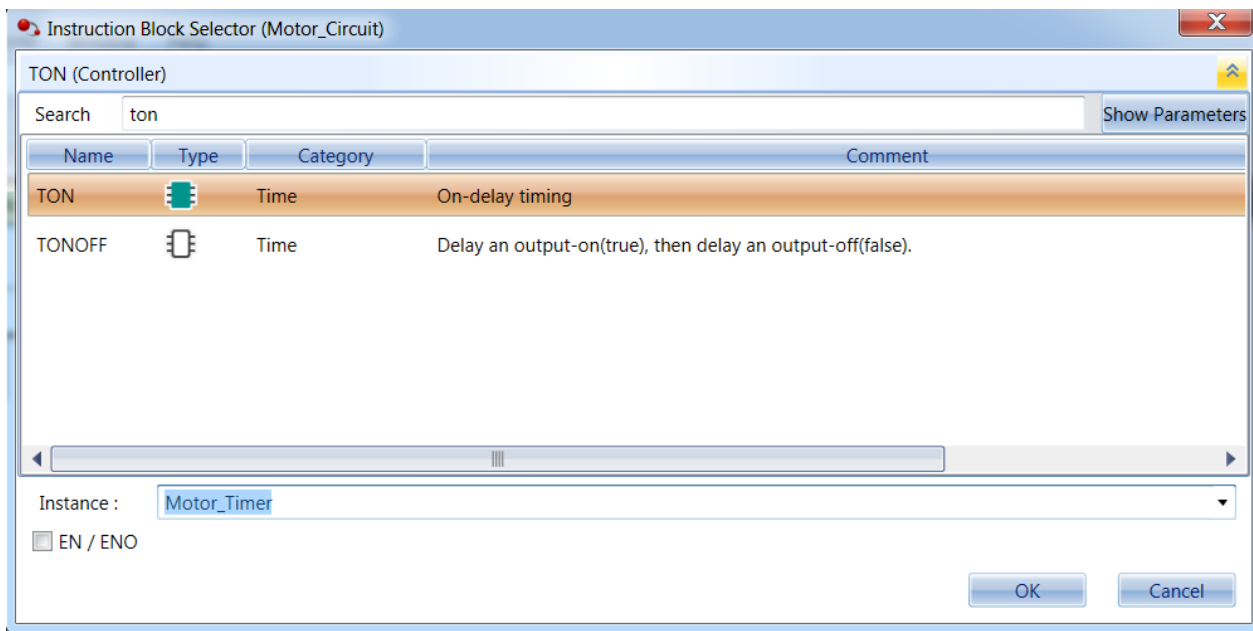
4. The Instruction Block Selector will appear. This is where you can select the type of Instruction Block you would like to use. As you can see, there is a long list of different types of instruction blocks that you can choose from. Feel free to take a minute to scroll through this list to see what types of instruction blocks are available.



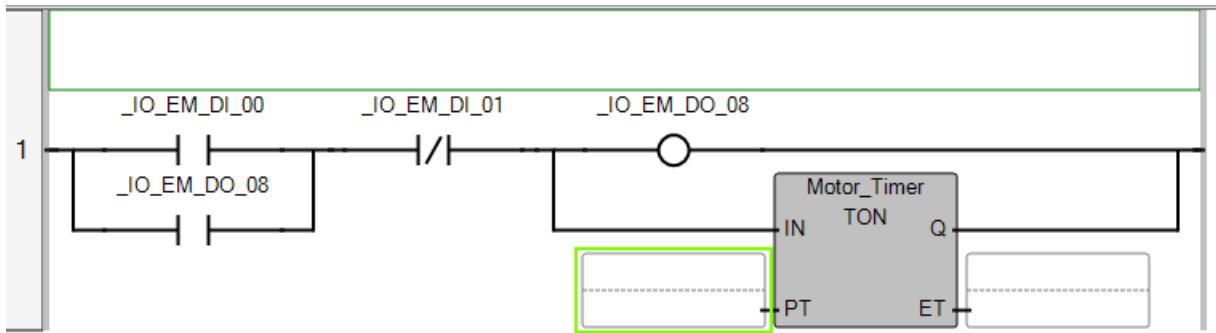
5. You can filter the instruction blocks by Name, Category, or Type. Since we want to use a Timer-On-Delay instruction block, type **TON** in the Name filter box at the top of the Name column. This will filter the choices to only Instruction Blocks that start with TON.



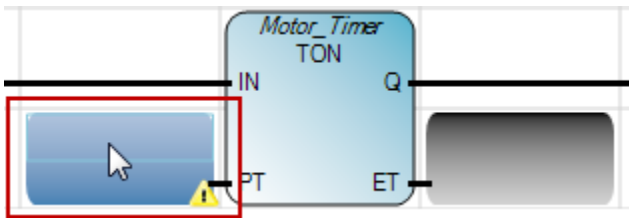
6. Highlight the **TON** Instruction Block – this is the Timer-on-Delay. Then at the bottom select the **Instance** combo box pull-down, and select your previously created **Motor\_Timer**.click **OK**



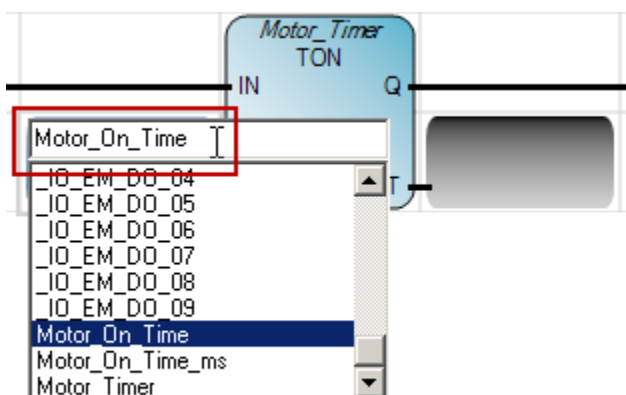
7. Your ladder program should look like the following.



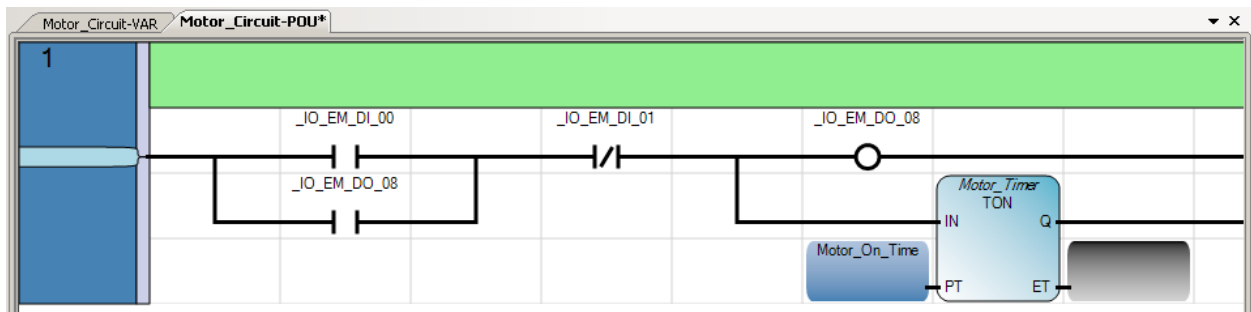
8. Next, hover the mouse cursor over the PT parameter of the Motor\_Timer TON instruction. You will notice a light blue highlighting the box..



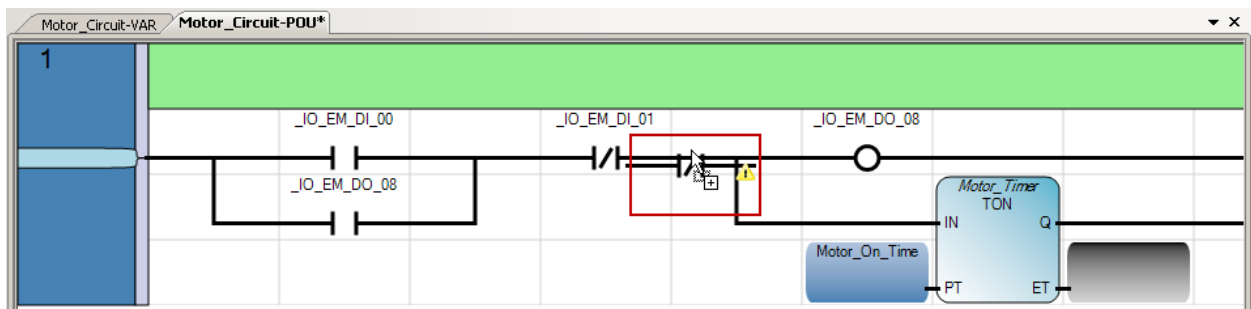
9. Click this box, and a pull down combo box will appear. Find and select the variable **Motor\_On\_Time** and then press the Enter key.



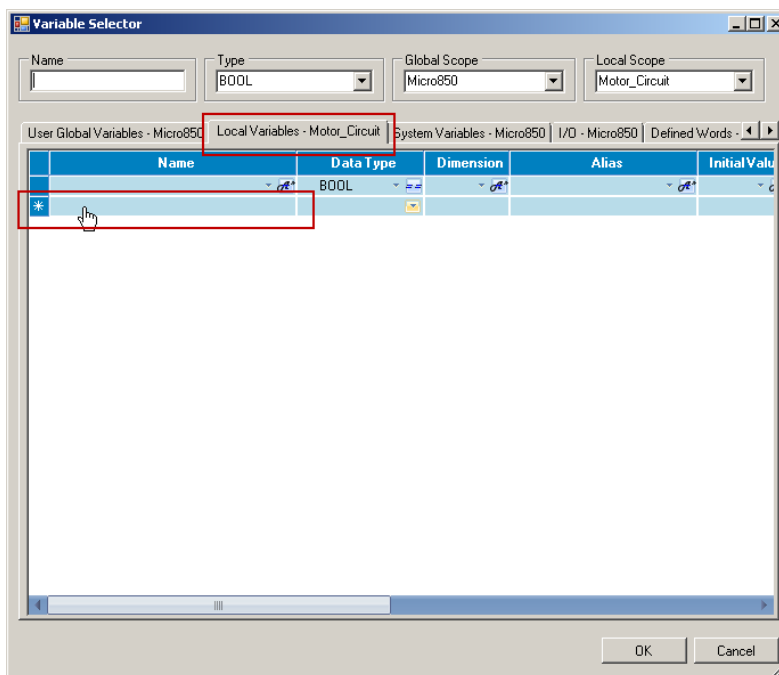
10. Your ladder program should look like the following.



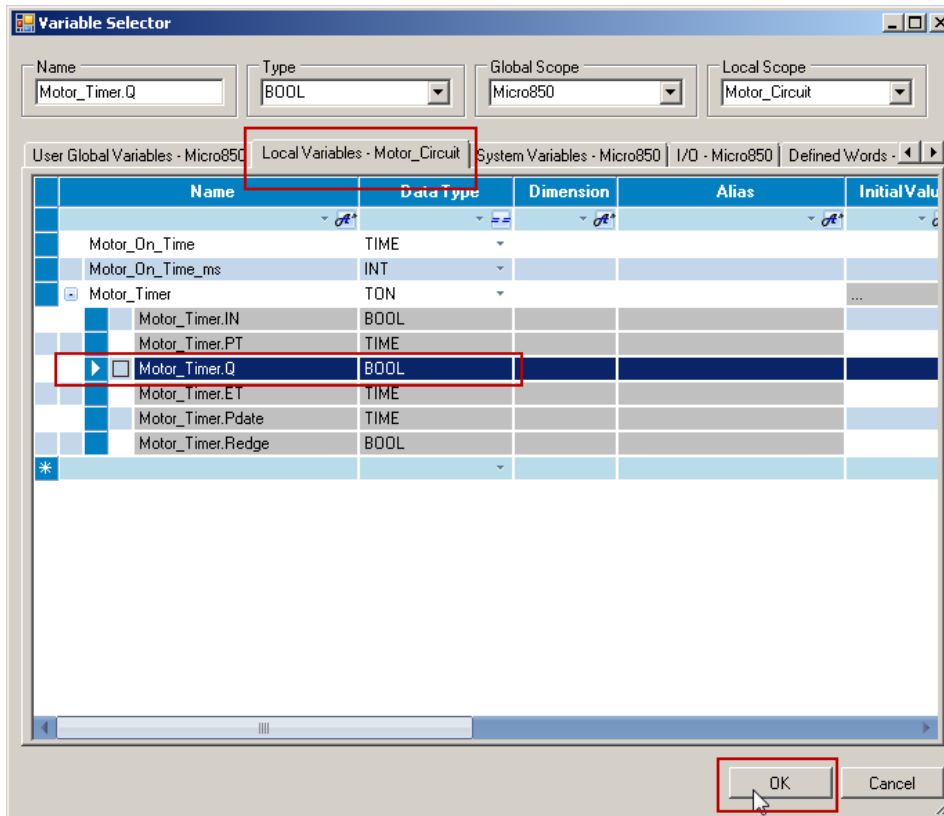
11. Insert a Reverse Contactor after the \_IO\_EM\_DI\_01 Reverse Contactor, as shown below.



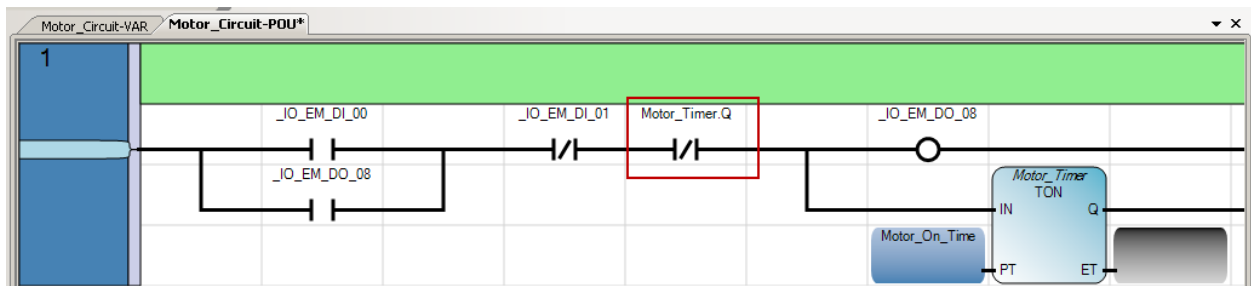
12. The Variable Selector will display. Select the **Local Variables – Motor\_Circuit** tab, and then click the empty cell shown below.



13. Expand the variable, **Motor\_Timer** and select, **Motor\_Timer.Q**. Then click **OK**.

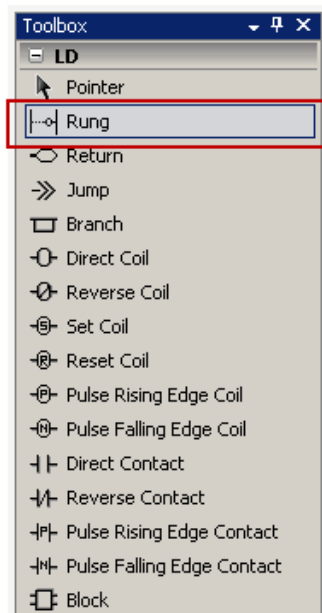


14. Your ladder program should look like the following.

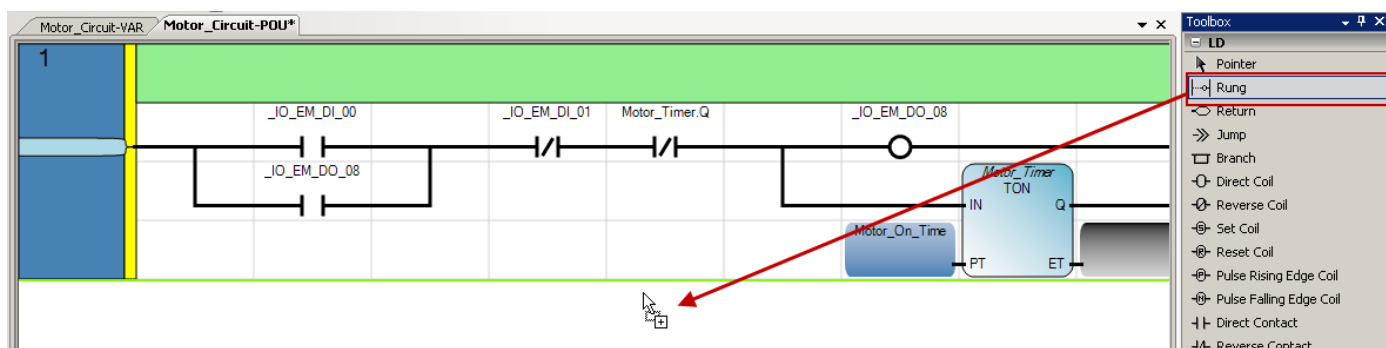




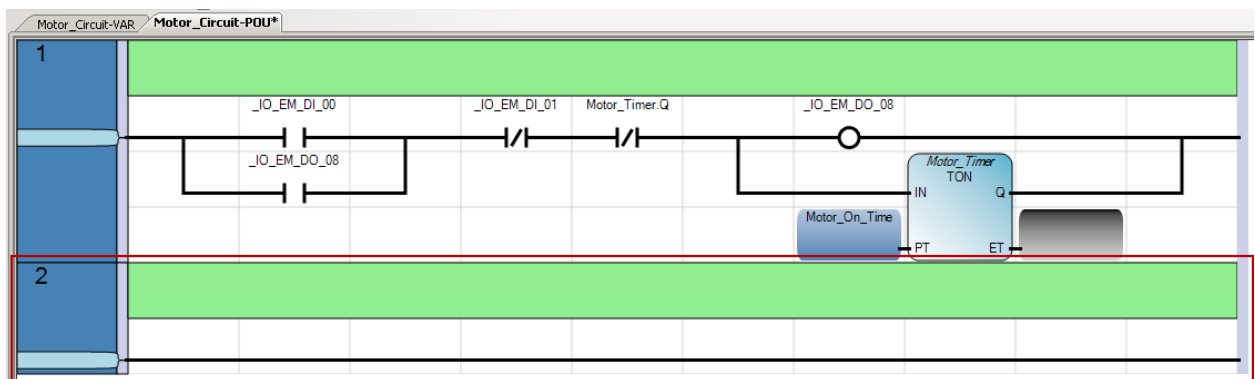
15. Locate the rung instruction in the Toolbox.



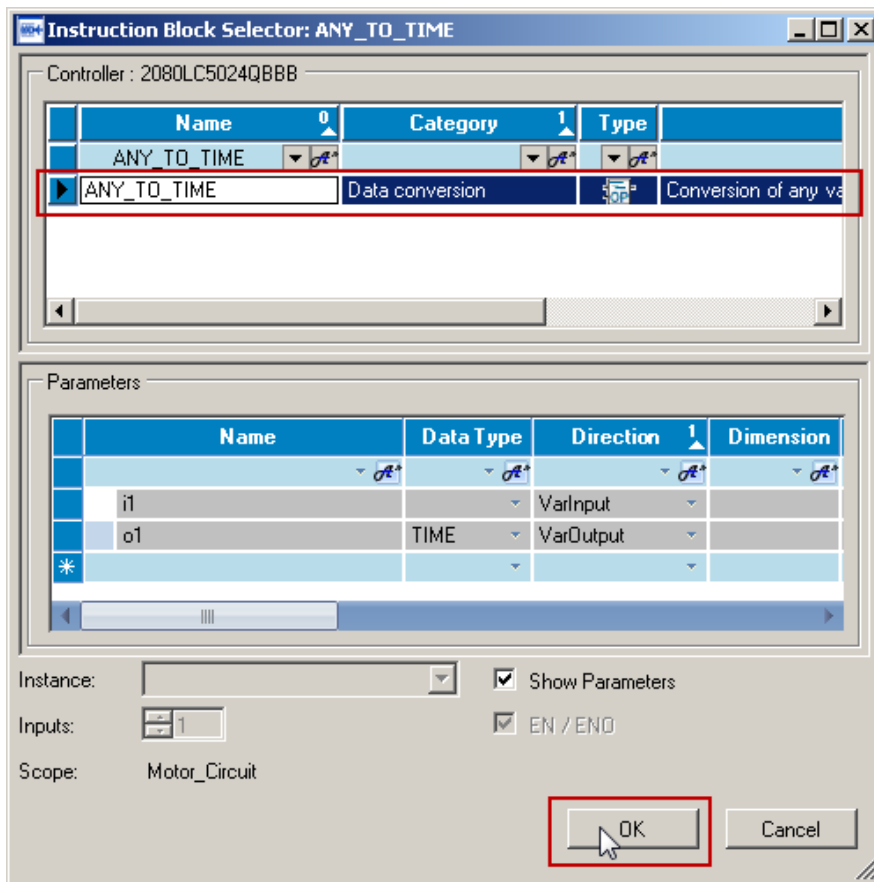
16. Drag and drop the **Rung** instruction below Rung 1.



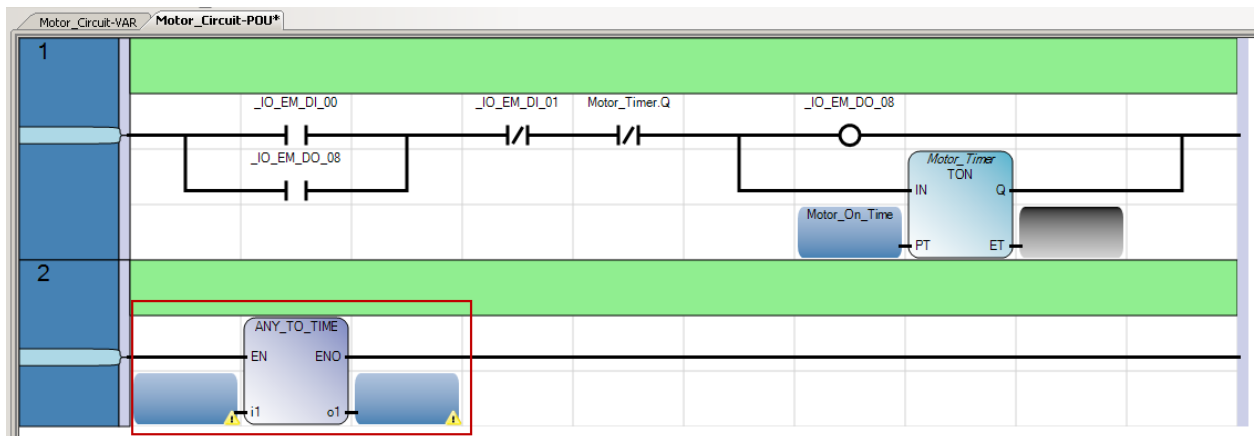
17. Your program should look like the following.



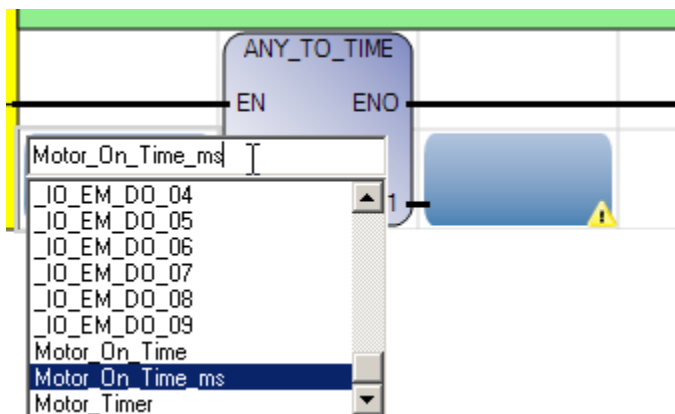
18. Insert a **Block** instruction into the rung you just created, and select the **ANY\_TO\_TIME** Instruction Block. Then click **OK**.



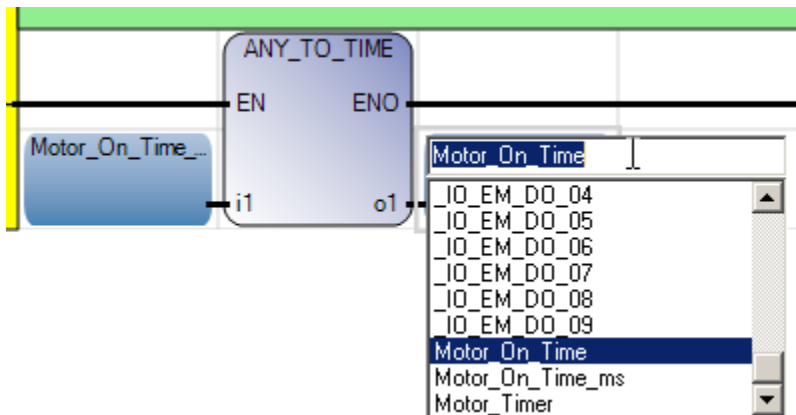
19. Your program should look like the following.



20. Select the variable **Motor\_On\_Time\_ms** for the i1 parameter.

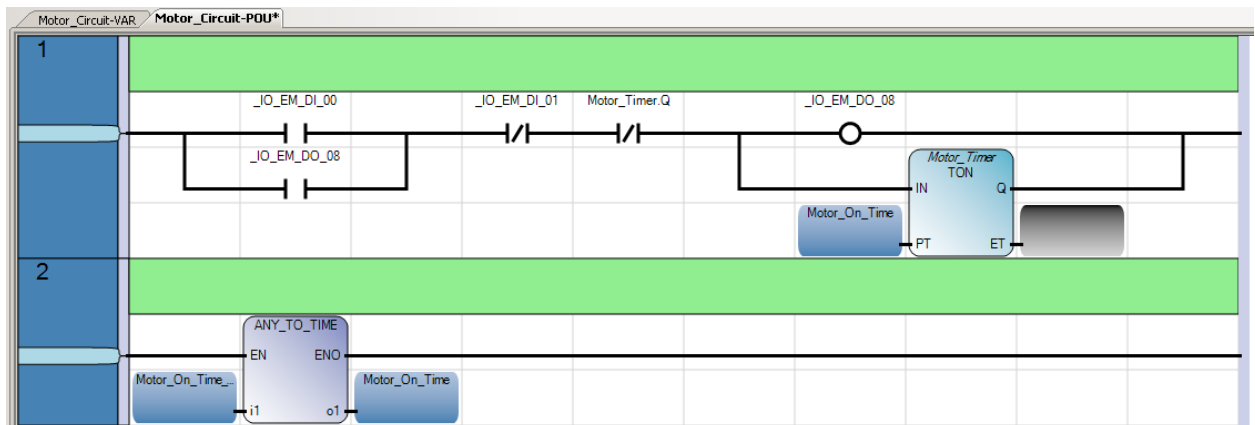


21. Select the variable **Motor\_On\_Time** for the o1 parameter.



22. The ANY\_TO\_TIME instruction block is being used to convert an integer value into a time value that is used as the preset time for the Motor\_Timer. The integer value represents time in milliseconds.

23. Your program should look like the following.

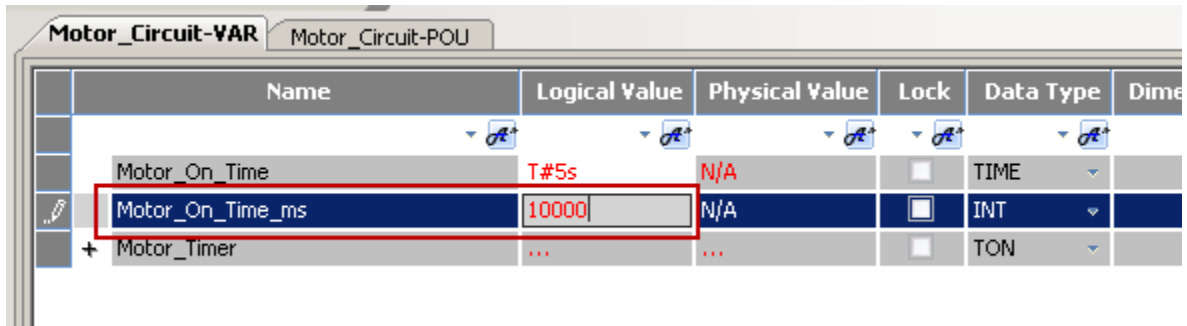


24. Build your program, and download it to the Micro850 (if you forgot how to do this, go back and reference the section **Build and Download your Micro850 Application**).

25. After completing the download, put the program into Debug mode by clicking the play button (or pressing the F5 key).

26. Now test your program. Turn the DI0 switch on, and watch the DO8 light turn on. After 5 seconds, the light should turn off.

27. Change the value of the variable, **Motor\_On\_Time\_ms**, to change the amount of time the light stays on to 10 seconds (remember we enter the value in milliseconds). Make sure to press enter after changing the value.



Name	Logical Value	Physical Value	Lock	Data Type	Dimension
Motor_On_Time	T#5s	N/A	<input type="checkbox"/>	TIME	
Motor_On_Time_ms	10000	N/A	<input type="checkbox"/>	INT	
Motor_Timer	...	...	<input type="checkbox"/>	TON	

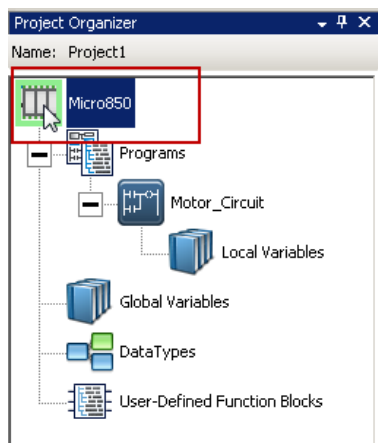
28. Now test your program again. Turn the DI0 switch on. The DO8 light should now stay on for 10 seconds, before turning off.
29. Click the Stop button to exit Debug Mode (or press Shift+F5).
30. You have completed this section of the lab.

---

## Learn how to add a plug-in module

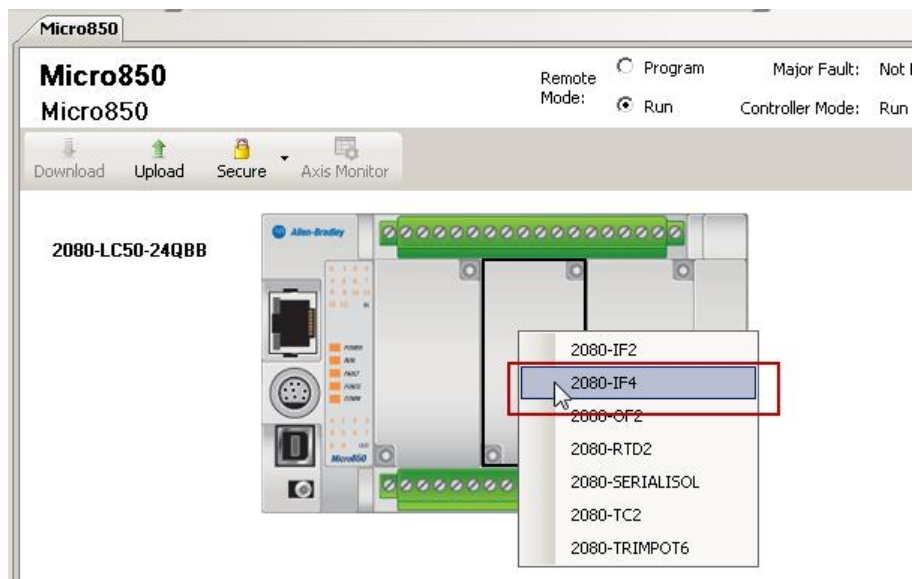
In this section of the lab, you will learn how to add an Analog Input plug-in module. A plug-in module is a module that you can plug into the Micro800 chassis to allow you to add additional I/O or Communications Options to your controller.

1. Double-click your Micro850 controller in the Project Organizer.

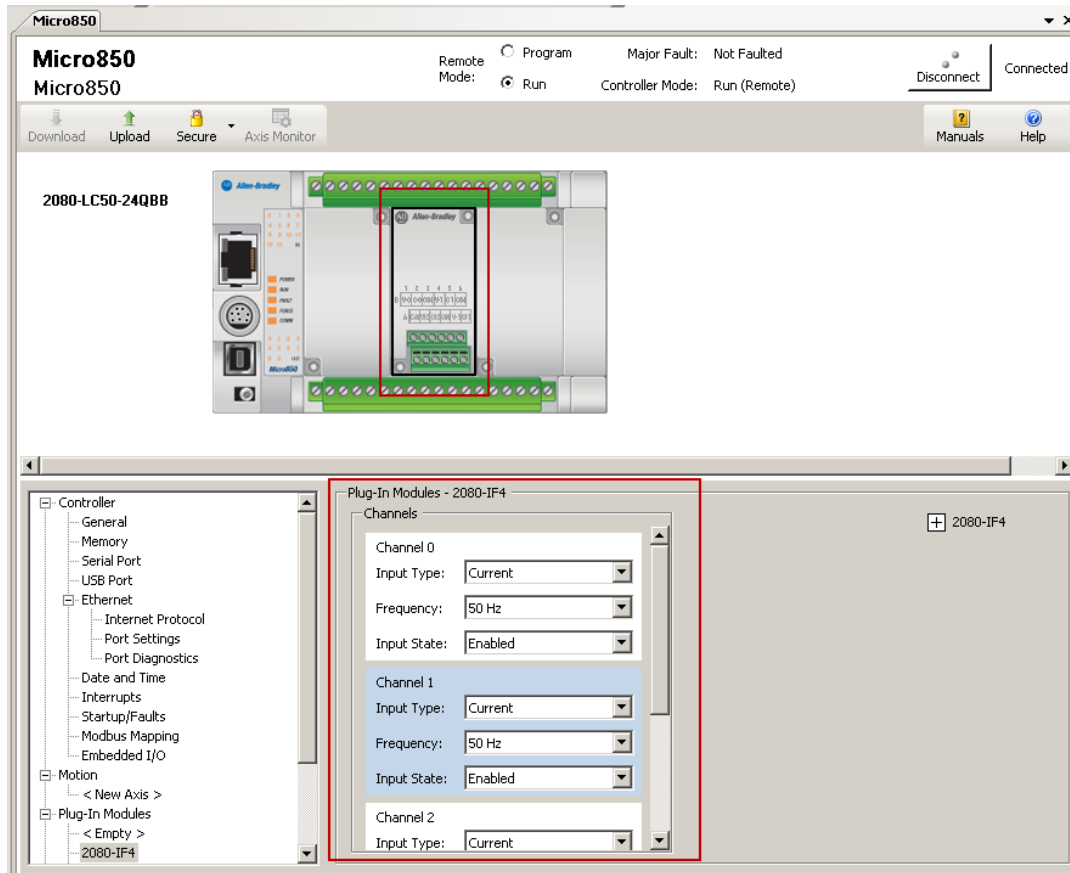


This will bring up the General Controller Properties in the main project window.

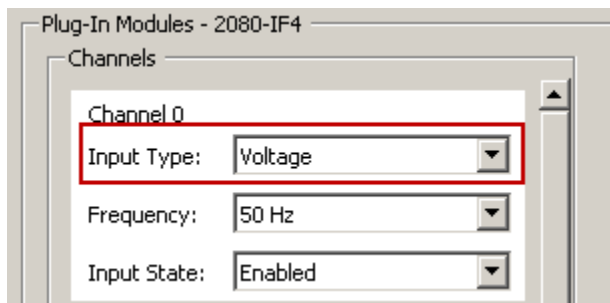
2. Right-click the second plug-in module slot, and select 2080-IF4.



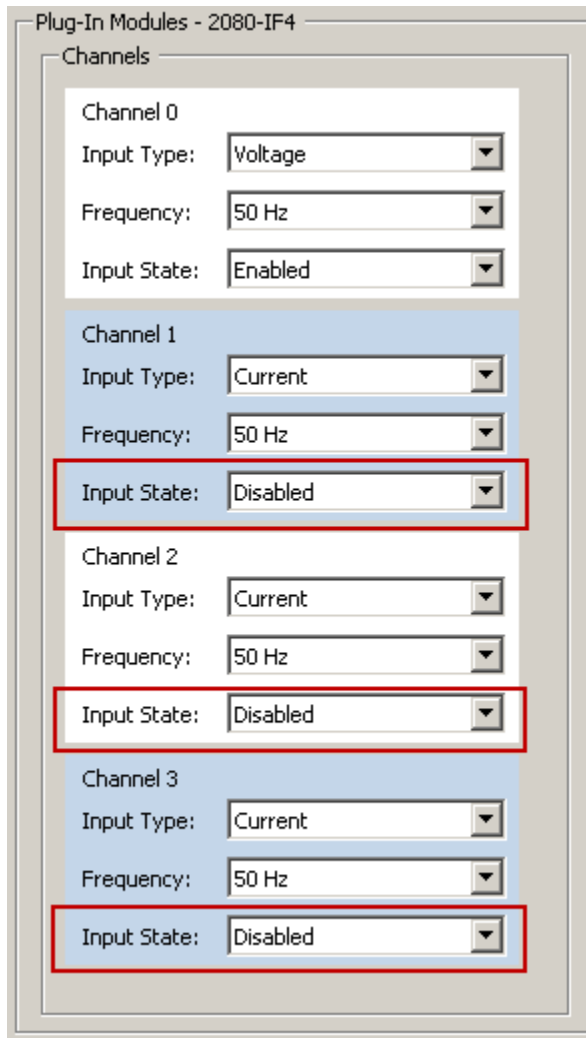
3. Notice that the 2080-IF4 module is now added to your chassis. The configuration properties should also show up in the pane below it.



4. Configure the Input Type for Channel 0 to Voltage.



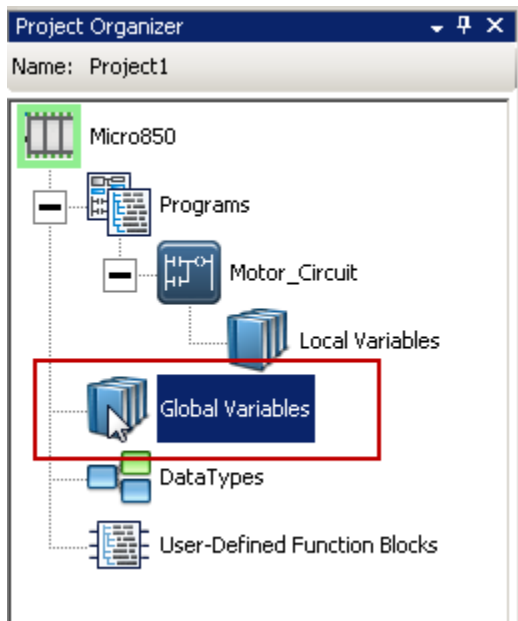
5. Configure the input State for Channel 1, 2, and 3 to **Disabled**.



6. Build and download your program to the Micro850.
7. Start debugging by clicking Play, or by pressing the F5 key.



8. Double-click **Global Variables** in the Project Organizer.



9. Locate the variable **\_IO\_P2\_AI\_00**. This is the raw data value in relation to the voltage that is wired to Channel 0. The value should range from 0 to 65535 in relation to a 0 to 10 volt input.

	_IO_EM_DI_10		
	_IO_EM_DI_11		
	_IO_EM_DI_12	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	_IO_EM_DI_13		
▶	<b>_IO_P2_AI_00</b>	22	22
	_IO_P2_AI_01	20	20
	_IO_P2_AI_02	20	20

10. On your demo hardware, turn the potentiometer labeled **Speed Command** and notice the value of **\_IO\_P2\_AI\_00** change.
11. Stop debugging by clicking the Stop button, or by pressing Shift+F5.
12. You have completed this section of the lab.

---

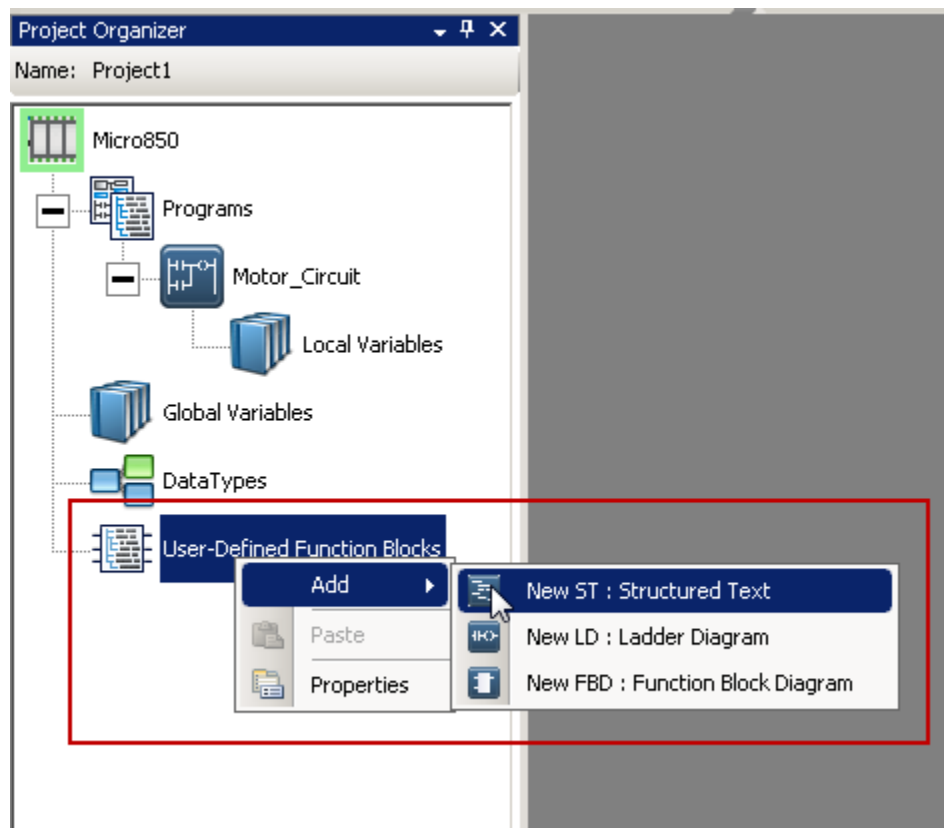
## Learn about User Defined Function Blocks

In this section of the lab, you will learn about a User Defined Function Block (UDFB), and how to create one using Structured Text.

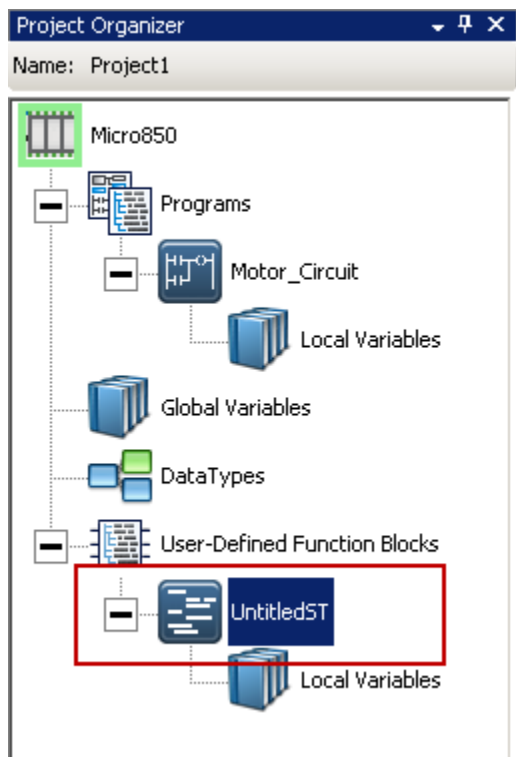
A User-Defined Function Block is a user defined program that can be packaged into an Instruction Block and reused within your Micro800 project. A UDFB can be written in Ladder, Function Block, or Structured Text.

You will be creating a UDFB to calculate the volume of a cylinder based on an inputted radius and height value.

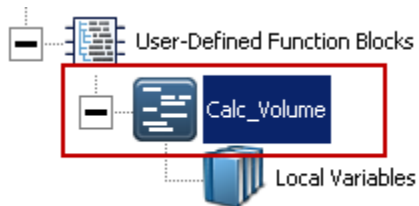
1. In your current project, right click **User-Defined Function Blocks** and select **Add → New ST: Structured Text**.



2. A program called **UntitledST** will be created under User-Defined Function Blocks.



3. Rename this UDFB, **Calc\_Volume**.

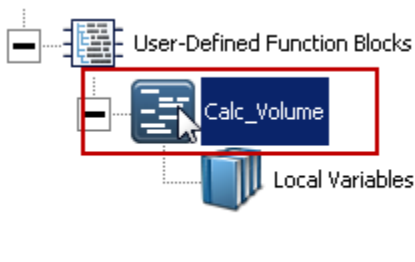


4. Double-click Local Variables under Calc\_Volume.

5. Create the following variables. Take careful note to properly configure the **Direction** property. This property defines whether the variable is an Input, Output, or standard Variable.

Calc_Volume-VAR		
Name	Data Type	Direction
Radius	REAL	VarInput
Height	REAL	VarInput
Volume	REAL	VarOutput
*		

6. Next, double-click the **Calc\_Volume** UDFB to launch the program editor in the main project window.

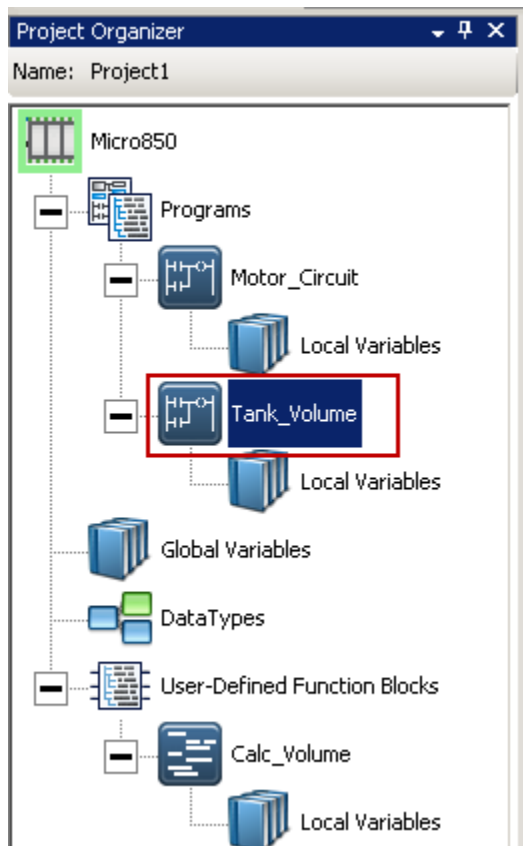


7. Add the following line of code to the program.

```
Volume := 3.14 * Radius * Radius * Height;
```

8. You have completed creating your UDFB.
9. Save your program.

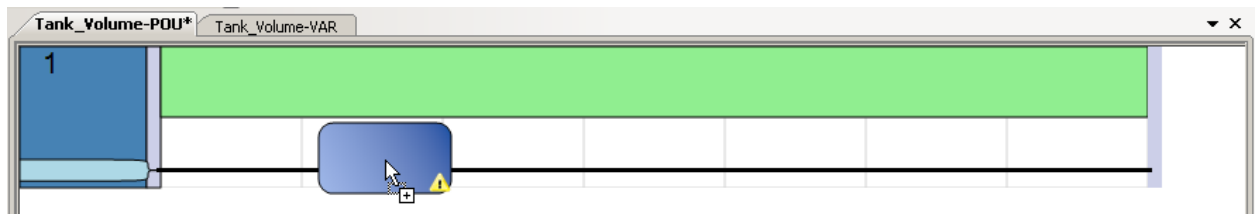
10. Next, create a new ladder diagram program called **Tank\_Volume**.



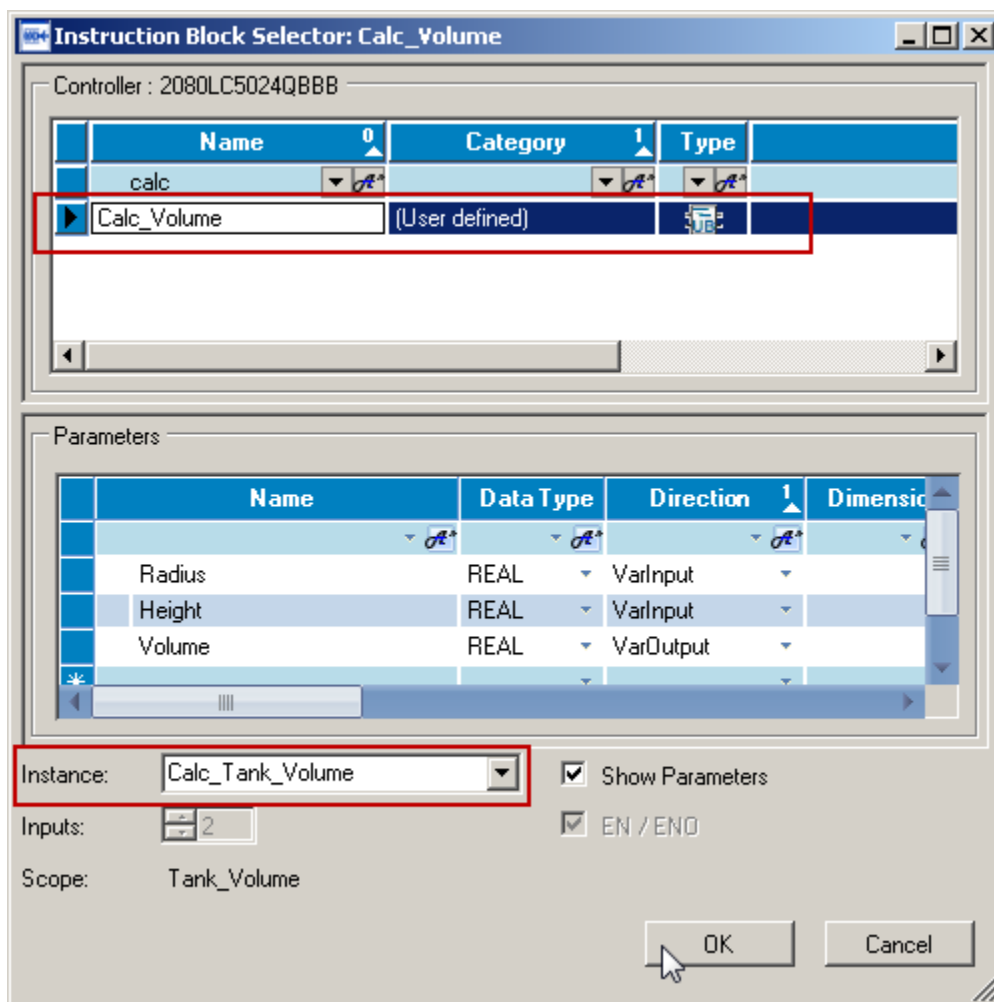
11. Open the Local Variables for the Tank\_Volume program, and create the following variables. Notice the Data Type for the variable **Calc\_Tank\_Volume** is the Calc\_Volume UDFB you created.

Tank_Volume-VAR		
	Name	Data Type
+	Calc_Tank_Volume	Calc_Volume
	Radius	REAL
	Height	REAL
	Volume	REAL
*		

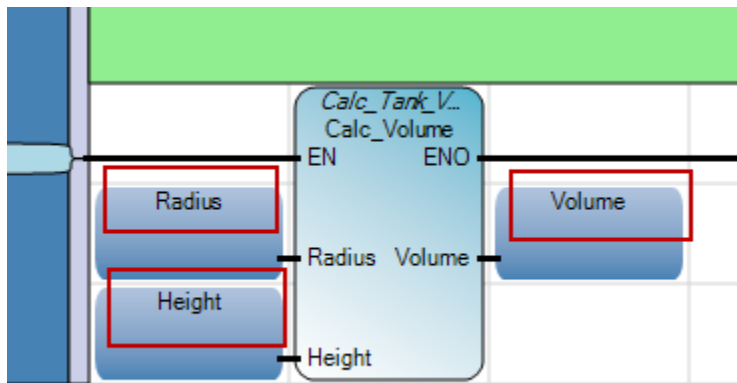
12. Next, open the **Tank\_Volume** program, and add a **Block** Instruction to the first rung.



13. Select the **Calc\_Volume** UDFB, and specify the Instance **Calc\_Tank\_Volume**. Then click **OK**.



14. Next, specify the following variables for each parameter of the Block.



15. Save your project.

16. Build and download your program to your Micro850 controller.

17. Once your download is complete, press the F5 key to enter Debug Mode.

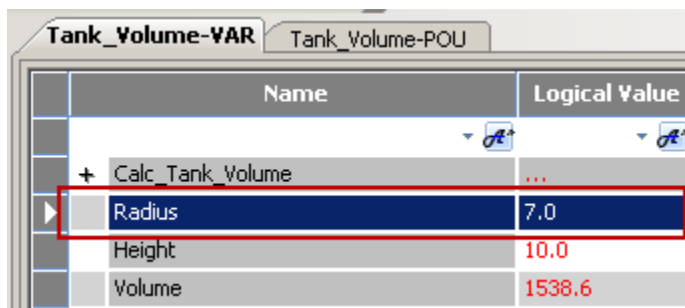
18. Open the Local Variables of your Tank\_Volume program, and set the value of Radius to 5, and Height to 10.

Tank_Volume-VAR	
Tank_Volume-POU	
Name	Logical Value
+	...
Calc Tank Volume	...
Radius	5.0
Height	10.0
Volume	785.0

19. The value of Volume should read 785.0.

Tank_Volume-VAR	
Tank_Volume-POU	
Name	Logical Value
+	...
Calc_Tank_Volume	...
Radius	5.0
Height	10.0
Volume	785.0

20. Change the value of Radius to 7.



The screenshot shows the 'Tank\_Volume-POU' window with a table of variables. The 'Radius' variable is highlighted with a red box.

Name	Logical Value
Calc_Tank_Volume	...
Radius	7.0
Height	10.0
Volume	1538.6

21. The value of Volume should read 1538.6.



The screenshot shows the 'Tank\_Volume-POU' window with a table of variables. The 'Volume' variable is highlighted with a red box.

Name	Logical Value
Calc_Tank_Volume	...
Radius	7.0
Height	10.0
Volume	1538.6

22. Press Shift+F5 to exit Debug Mode.

23. You have completed this section of the lab.



**[www.rockwellautomation.com](http://www.rockwellautomation.com)**

**Power, Control and Information Solutions Headquarters**

Publication CE-DM246A-EN-P — November 2012

Copyright© 2012 Rockwell Automation, Inc. All rights reserved.