L04 - Basic PLC Programming with Micro800™ Controller Family

For Classroom Use Only!
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.
# Basic PLC Programming with Micro800™ Controller Family

## Contents

- Before you begin ........................................................................................................................................... 4
- About this lab ............................................................................................................................................... 4
- Tools & prerequisites ................................................................................................................................. 5
- Get familiar with the Connected Components Workbench design environment ................................. 6
- Create a CCW project and program a Micro850 controller ................................................................. 12
- Build and Download your Micro850 Application .................................................................................. 23
- Debug your Micro850 program ............................................................................................................... 27
- Learn about Variables and Data Types .................................................................................................. 32
  - Data types ................................................................................................................................................ 33
- Learn how to create variables ................................................................................................................ 34
- Learn how to Implement an Instruction Block ...................................................................................... 36
- PanelView Component HMI Design using Connected Components Workbench ............................ 55
  - About This Lab ...................................................................................................................................... 55
  - Add the Terminal and Setup Communications .................................................................................... 55
  - Create and Use Tags ............................................................................................................................. 57
  - Create the Motor Control Screen ........................................................................................................ 60
  - Create a Push Button ........................................................................................................................... 63
  - Create a Goto Config Button ............................................................................................................... 68
  - Create a Multistate Indicator .............................................................................................................. 69
  - Create a Numeric Entry ......................................................................................................................... 72
  - Generate a Report ................................................................................................................................. 73
  - Validate and Transfer the Application to the Terminal ........................................................................... 75
  - (OPTIONAL) Learn about User Defined Function Blocks ................................................................. 77
Before you begin

The following should have already been verified with your demo kit by the lab instructor prior to the lab:

1. USB cable connected between PC and Micro850 controller.
3. PanelView Component terminal firmware at v1.80 (updated via USB drive if necessary) with static IP address configured for 192.168.1.2 and subnet mask 255.255.255.0.
4. PC Ethernet port configured for 192.168.1.1, subnet mask 255.255.255.0 with PVc Ethernet cable connected directly to it.
5. PowerFlex 4M drive parameter settings:
   - P106=2
   - P108=0

To verify:
1. Press Esc on the PowerFlex 4M drive keypad multiple times until 0.0 is displayed.
2. Press Sel until the leftmost alphanumeric character is flashing.
3. Press the up or down arrow until the leftmost alphanumeric character being displayed is a flashing P.
4. Press the enter key. The rightmost digit will be flashing.
5. Press the up or down arrow until P106 is displayed.
6. Press the enter key. The value of P106 is displayed.
7. If the value displayed for P106 is 2, jump to step 10.
8. Press the enter key. The value displayed for P106 will be flashing.
9. Press the up or down arrow until a flashing 2 is displayed. Press the enter key to accept this new value.
10. Press Esc – P106 is displayed and the 6 will be flashing.
11. Press the up arrow two times so that P108 is displayed.
12. Press the enter key. The value of P108 is displayed.
13. If the value displayed for P108 is 0, jump to step 16.
14. Press the enter key. The value displayed for P108 will be flashing.
15. Press the up or down arrow until a flashing 0 is displayed. Press the enter key to accept this new value.
16. Press Esc multiple times until 0.0 is displayed.

6. Demo kit power cycled off and back on.

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 Version 7.00.00 – Standard or Developer Edition (Developer Edition is only required for password protecting the UDFB in optional exercise on pages 84-85)
- Hardware: Connected Components demo kit – DEMO-CCMICRO1

**Lab setup**
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.

(If Connected Components Workbench software is already open, jump directly to Step 2.)

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 project layout. Below are descriptions of each of the panels' contents and the general task the pane is used for.
3. 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.
4. Device Toolbox

The Device Toolbox displays all of the devices that you can add to your Connected Components Workbench™ project. 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.
5. 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.
6. Output

In the Output window, you can view and manage general purpose and debug messages generated by the various features of Connected Components Workbench. From the Output window, you can do the following:

- Review status messages
- Locate errors within programs
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. 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 folder. Expand the folder and select catalog **2080-LC50-24QBB**. Double-click this Micro850 controller catalog. This will add a Micro850 controller to your Project.

If windows appear for you to select major revision, select ‘7’, check the ‘Always use the latest revision’ box and click OK.
2. Notice that the Micro850 shows up in your Project Organizer on the left-hand side.

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** has been added under Programs.

Micro800 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 **Prog1** program icon and select **Rename**.
6. Rename the program, **Motor_Circuit**.

![Project Organizer](image)

7. Create a motor seal-in circuit in your Motor_Circuit Ladder Diagram program.

   This circuit will use the DI0 (Discrete Input 0) switch on the Demo box as your Start pushbutton, and the DI1 (Discrete Input 1) switch as your Stop pushbutton. The start motor control is wired to DO9 (Discrete Output 9) so that when DO9 is on, the motor accelerates and runs, and when DO9 is off, the motor decelerates to a stop.
8. Double-click the **Motor Circuit** program icon. A ladder diagram editor will appear in the main project workspace.

You do not need the 'Device Toolbox' window for the time being. Toggle at the top right hand corner of the window to hide it.
9. 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.

10. In the Variable Selector window, select the I/O – Micro850 tab.
11. We will be assigning an embedded I/O point to this instruction. Select _IO_EM_DI_00, then, in the ‘Alias’ column of _IO_EM_DI_00, type ‘Start Motor PB’ and click OK.

12. Your rung should look like the following.
13. 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_09` with alias description ‘Motor’. Your rung should look like the following.

14. Locate the **Reverse Contact** instruction in the Toolbox and drag-and-drop it onto your rung, just to the right of the Direct Contact and assign it to embedded I/O point `_IO_EM_DI_01` with alias description ‘Stop Motor PB’. Your rung should look like the following.

15. 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.

16. Drag-and-drop a **Direct Contact** onto the Branch that you just added, and assign it to the embedded I/O point `_IO_EM_DO_09`. Your rung should look like the following.
17. You’ve completed creating your motor seal-in circuit. When the Start Motor pushbutton is pushed (while the Stop Motor pushbutton is not being pushed), you complete the rung circuit to the output so that the Motor turns on. Once the motor is running, you can release the Start Motor pushbutton because the branch circuit around the pushbutton seals it in and the only way to interrupt the circuit is to push the Stop Motor pushbutton. This breaks the circuit, which turns the motor off and drops out the seal-in branch circuit. The Stop Motor pushbutton can then be released and the Motor remains off until the Start Motor pushbutton is pushed again.
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.
2. When the build is complete, you will see a message in the lower left-hand corner that the build has succeeded. If there were errors in your programming, then they would be listed in the Error List panel 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.

3. Download your program to your Micro850 by right-clicking the Micro850 in your Project Organizer, and selecting Download.
4. The Connection Browser will appear. Browse for your controller by expanding **USB** and selecting **16, Micro850, Micro850**, then clicking **OK**.

![Connection Browser](image)

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

![Download](image)

6. The Download Confirmation will appear. You will be prompted to overwrite the project in the controller. The two choices are **Download** and **Download with Project Values**. The second choice is a new feature in Release 7 of Connected Components Workbench. Now, when a project is uploaded from the controller, a snapshot of the values of all of the user variables is uploaded and saved. These values can be downloaded back into the controller along with the project by selecting **Download with Project Values**. If you clicked the **Help** button, you would find that the following conditions must be met in order for a Project Value to be downloaded to a variable:
   
   - Variables with both a **Project Value** and an **Initial Value** will be initialized with their **Initial Value**.
• Variables with an Attribute of "Read" or "Write" cannot be downloaded. Variables must have an Attribute of "Read/Write".
• Variables with a Direction of "VarInput" cannot be downloaded.
• Complex variables (structures, arrays) that contain members with Initial Values cannot be downloaded.

7. In this lab, we have a new project and none of the variables have a Project Value assigned to them yet. Therefore, just click Download.

![Download Confirmation](image)

8. The download will proceed. When the download is complete, you will be prompted to put the controller back in Remote Run mode. Click Yes.

![Download Confirmation](image)

9. Notice the message in the lower left-hand corner that indicates that the Download has completed successfully.

![Error List](image)

10. 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. Set the speed reference by adjusting the trimpot on the PowerFlex 4M drive.

4. Turn and release the **DI0** switch on the Demo box. 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_09 Direct Contact and Direct Coil instructions turn red. You should also notice that the motor is running.

![Diagram of ladder diagram indicating motor seal-in circuit](image)

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.

5. Turn and release the **DI1** switch on the Demo box to turn off the output. Notice the motor decelerate to a stop and the corresponding changes in your Ladder Diagram.

6. 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 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.
7. 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.

8. Locate the **_IO_EM_DO_09** embedded I/O variable in the Global Variable list, and notice that the logical value checkbox is empty.
9. Turn and release the DI0 switch on the Demo box. Notice there is now a checkmark in the Logical Value for _IO_EM_DO_09, and the motor begin to run. 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Logical Value</th>
<th>Physical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IO_EM_DO_00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_09</td>
<td>Motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DI_00</td>
<td>Start Motor PB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DI_01</td>
<td>Stop Motor PB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. Turn and release the DI1 switch on the Demo box. Notice the checkmark go away in the logical value for _IO_EM_DO_09, and the motor decelerate to a stop.

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Logical Value</th>
<th>Physical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IO_EM_DO_00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DO_09</td>
<td>Motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DI_00</td>
<td>Start Motor PB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_IO_EM_DI_01</td>
<td>Stop Motor PB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
11. 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 troubleshooting process, and Connected Components Workbench makes this design step very 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.
User 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**: Boolean, True or False
- **SINT**: 8-bit signed integer, range -128 to +127
- **USINT**: 8-bit unsigned integer, range 0 to 255
- **BYTE**: 8-bit unsigned integer, range 0 to 255
- **INT**: 16-bit signed integers, range -32768 to 32767
- **UINT**: 16-bit unsigned integers, range 0 to 65535
- **WORD**: 16-bit unsigned integers, range 0 to 65535
- **DINT**: 32-bit signed integers, range -2147483648 to +2147483647
- **UDINT**: 32-bit signed integers, range 0 to 4294967295
- **DWORD**: 32-bit signed integers, range 0 to 4294967295
- **LINT**: 64-bit signed integers, range -9223372036854775808 to 9223372036854775807
- **ULINT**: 64-bit unsigned integers, range 0 to 18446744073709551615
- **LWORD**: 64-bit unsigned integers, range 0 to 18446744073709551615
- **REAL**: 32-bit floating values, range -3.40282347E+38 to 3.40282347E+38
- **LREAL**: 64-bit floating values, range -1.7976931348623157E+308 to 1.7976931348623157E+308
- **TIME**: 32-bit words, range 0 to 1193h2m47s294ms
- **DATE**: 32-bit words, range 1970-01-01 to 2038-01-18
- **STRING**: Up to 252 characters, can contain ASCII codes from 0 to 255
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 Global Variables in your Motor_Circuit program to launch the Variables panel.

![Image of Variables panel]

2. Scroll to the bottom of the list and create a variable called Motor_On_Time of Data Type TIME and press 'ENTER'.

   | __SYSVA_POWERUP_BIT | BOOL |
   | __SYSVA_PROJ_INCOMPLETE | UDINT |
   | Motor_On_Time | TIME |

3. Create a variable called Motor_On_Time_ms of Data Type INT with an Initial Value of 5000.

   | __SYSVA_PROJ_INCOMPLETE | UDINT |
   | Motor_On_Time | TIME |
   | Motor_On_Time_ms | INT |
   | | | 5000 |
4. Create a variable called **Motor_Timer** of Data Type **TON** and press ‘ENTER’.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SYSVA_PROJ_INCOMPLE</td>
<td>UDINT</td>
</tr>
<tr>
<td>Motor_On_Tme</td>
<td>TIME</td>
</tr>
<tr>
<td>Motor_On_Time_ms</td>
<td>INT</td>
</tr>
<tr>
<td>Motor_Timer</td>
<td>TON</td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

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. Create a variable called **Start_Motor** of Data Type **BOOL** and press ‘Enter’.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor_On_Tme</td>
<td>TIME</td>
</tr>
<tr>
<td>Motor_On_Time_ms</td>
<td>INT</td>
</tr>
<tr>
<td>Motor_Timer</td>
<td>TON</td>
</tr>
<tr>
<td>Start_Motor</td>
<td>BOOL</td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

6. Create a variable called **Stop_Motor** of Data Type **BOOL** and press ‘Enter’.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor_On_Tme</td>
<td>TIME</td>
</tr>
<tr>
<td>Motor_On_Time_ms</td>
<td>INT</td>
</tr>
<tr>
<td>Motor_Timer</td>
<td>TON</td>
</tr>
<tr>
<td>Start_Motor</td>
<td>BOOL</td>
</tr>
<tr>
<td>Stop_Motor</td>
<td>BOOL</td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

7. 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 automatically turn off the motor coil after it has been running for five 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. Click on the Motor_Circuit-POU tab to return to the ladder diagram.
2. Drag-and-drop a Branch instruction around the coil instruction.

3. Locate the Instruction Block instruction in the Toolbox.
4. Drag-and-drop this Instruction Block instruction into the branch that you just added.

5. 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.

6. Since we want to use a Timer-On-Delay instruction block, type TON in the ‘Search’ field. This will filter the
choices to only Instruction Blocks that contain TON within their name.

7. Select the TON Instruction Block – this is the Timer-ON-delay. Then select the Instance combo box pull-down, and select the variable instance, Motor_Timer that you created in the previous section, and click OK.
8. Your ladder program should look like the following.

9. Next, hover the mouse cursor over the red box area next to the PT parameter of the Motor_Timer TON instruction. You will notice a light blue highlighted box appear.

10. 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.
11. Your ladder program should look like the following.

![Ladder Diagram](image1)

12. Insert another Reverse Contact after the _IO_EM_DI_01 Reverse Contact, as shown below.

![Reverse Contact](image2)

13. The Variable Selector will display. Select the **Global Variables – Micro850** tab. Click on the first cell in the last row to display the remaining non-BOOL User Global Variables.

![Variable Selector](image3)

14. Expand the variable **Motor_Timer** and select **Motor_Timer.Q**, which goes TRUE as soon as the timer’s Elapsed...
Time (ET) times up to its Preset Time (PT). Then click **OK**.

15. Your ladder program should look like the following.
16. Locate the rung instruction in the Toolbox.

17. Drag and drop the **Rung** instruction below Rung 1.
18. Your program should look like the following.

![Program Diagram]

19. Insert an **Instruction Block** instruction into the rung you just created, type ‘any_to_time’ in the search field and select the **ANY_TO_TIME** Instruction Block. Then click **OK**.
20. Your program should look like the following.

![Program Diagram]

21. Select the variable Motor_On_Time_ms for the i1 parameter.

![Variable Selection Screenshot]
22. Select the variable **Motor_On_Time** for the o1 parameter.

23. 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.

24. From here, you will assign two variables as a set of parallel inputs to start and stop the motor. This will be used for the PanelView Component HMI Design lab in the later section.

Create another parallel branch on the far left of rung 1.
25. Drag and drop a **Direct Contact** onto the branch you inserted in the previous step.

26. The Variable Selector window appears. Select the **User Global Variables –Micro850** tab, select ‘Start_Motor’ and click **OK**.
27. Drag and drop a **Reverse Contact** to the left of _IO_EM_DL_01
28. The Variable Selector window appears. Select the **User Global Variables – Micro850** tab, select ‘**Stop_Motor**’ and click **OK**.

29. Your program should look like the following.
30. 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). Note that if you select Download without building first, the software will automatically build before downloading.

31. After completing the download, put the program into Debug mode by clicking the Play button (or pressing F5).

32. Now test your program. Turn the DI0 switch on, watch the motor run for 5 seconds and then decelerate to a stop.

33. Change the value of the variable, Motor_On_Time_ms. Double click the variable box in debug mode to launch the Variable Monitoring window.

34. Change the amount of time the motor runs to 10 seconds (remember to enter the value in milliseconds). Make sure to press ‘Enter’ after changing the value.

35. Now test your program again. Turn the DI0 switch on and off. The motor runs for 10 seconds and then decelerates to a stop.

36. Close the Variable Monitoring window and click the Stop button to exit Debug Mode (or press Shift+F5).

37. 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 to make sure you are displaying the controller configuration window.

2. If your controller is connected, click the ‘Disconnect’ button.
4. Notice that the 2080-IF4 module is now added to your controller graphic. The plug-in configuration properties specific to the 2080-IF4 should also show up in the pane below it.

5. Configure the Input Type for Channel 0 to Voltage.
6. Configure the input State for Channel 1, 2, and 3 to **Disabled**.

![Diagram showing input configuration for Channel 0, 1, 2, and 3]

7. Build and download your program to the Micro850.

8. Start debugging by clicking Play, or by pressing the F5 key.

10. Locate the variable **_IO_P2_AI_00**. This is the raw data value in relation to the voltage that is wired to Channel 0 of the plug-in module. The value should range from ~0 to 65535 in relation to a 0 to 10 volt input.

11. On your demo box, turn the potentiometer labeled **Speed Command** and notice the value of **_IO_P2_AI_00** change.

12. Stop debugging by clicking the Stop button, or by pressing Shift+F5.

13. You have completed this section of the lab.
PanelView Component HMI Design using Connected Components Workbench

About This Lab
Now that you've learned the basics of creating a Micro800 program, let's create a PanelView Component application that will work with it.

Add the Terminal and Setup Communications
Add a PanelView Component terminal to project and then configure the application.

1. From View → Device Toolbox, expand Catalog and Graphic Terminals to bring up the list of PanelView Component HMI catalog numbers. Double-click on 2711C-T6T to add it to the Project Organizer.

2. Double-click the Graphic Terminal (Labeled as PVcApplication1) icon in Project Organizer to bring up the PanelView Component device configuration screen.
3. On the **Settings** tab, go to the **Protocol** menu located under **Communication**. From the drop down menu, select **Serial | Allen-Bradley CIP** as shown.

4. Leave the rest of the communication settings at the default state. Notice that the name of the controller we will be using is PLC-1.

5. Rename the PVCApplication1 as "**MotorStarter**" by right-clicking on the icon and selecting **Rename** from the menu.
Create and Use Tags

This application uses read and write tags to define how objects interact with the addresses of the Micro850 controller. You must create these tags before you can assign them to the objects in the application. Only objects that interact with the controller require a tag. Objects such as screen navigation buttons, drawings, and screen text do not require tags.

1. Double-click on the tags icon to open the Tag Editor.

   ![MotorStarter Tags Icon]

2. Verify that the External tab is selected.

   ![Tag Editor External Tab]

3. Click the Add button to add a tag.

   ![Tag Editor Add Button]

4. Click in the Address field and select the (...).

   ![Tag Editor Address Field]
5. Select **Motor_On_Time_ms** from the list of variables and click **OK**.

![Variable Selector]

6. The **Data Type** is automatically filled in when the tag is selected.

7. Type “**Motor_On_Time_ms**” in the **Tag Name** field.

**Note:** The Tag Name and Address do not have to match, but it is good programming practice.

8. Click in the **Controller** field and select **PLC-1**. This is the controller name defined on the Communication tab.

![Tag Editor]

9. Optionally, enter the tag description.
10. Repeat steps 3 through 9 to add the **Start_Motor** and **Stop_Motor**.

11. Add one more tag by selecting the **Add** button.

12. Select the (...) in the **Address** field.

13. Select the **I/O – Micro850** tab.

14. Select the **_IO_EM_DO_09** tag and select **OK**.

15. Type "**Motor_Ind**" in the **Tag Name** field.

16. Click in the **Controller** field and select **PLC-1** and optionally, enter a description.
17. When done, the Tag Editor should look like this.

![Tag Editor Interface]

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Data Type</th>
<th>Address</th>
<th>Controller</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor_On_Time_m</td>
<td>16 bit integer</td>
<td>Motor_On_Time_m</td>
<td>PLC-1</td>
<td></td>
</tr>
<tr>
<td>Start_Motor</td>
<td>Boolean</td>
<td>Start_Motor</td>
<td>PLC-1</td>
<td></td>
</tr>
<tr>
<td>Stop_Motor</td>
<td>Boolean</td>
<td>Stop_Motor</td>
<td>PLC-1</td>
<td></td>
</tr>
<tr>
<td>Motor_ind</td>
<td>Boolean</td>
<td>_IO_EM_DIO_09</td>
<td>PLC-1</td>
<td></td>
</tr>
</tbody>
</table>

18. Save the application by selecting the **Save** icon ( ) from the toolbar.

Create the Motor Control Screen

New applications open with the first screen created. Each screen is created with a default name, Screen_NN, where NN is a number from 1 to 99. The first screen is named Screen_1. You can rename the screen using a maximum of 50 characters.

This is what the Motor Control screen will look like when the following sections are complete.

![Motor Control Screen Diagram]

1. Right-click **1- Screen 1** and select **Rename**.
2. Type "MOTOR_CONTROL" into the text field as shown below and press Enter.

![Diagram showing screens and Motor-Control folder]

3. If the screen tab is not open already, double click the screen icon next to the newly named screen to open it for editing. It should look similar to this:

![Image of screen settings and Motor-Control screen]

Verify the screen border is highlighted (in red) indicating the screen is selected. You can modify screen properties as long as nothing on the screen is selected.
4. Select **View → Properties Window** from the tool bar to view the Properties Window. It will show up on the right side of the screen.

Note: The Properties Window can also be activated by right-clicking on the object, in this case the screen, and selecting **Properties**. If you wanted to see the Properties of a push button, right-click on that button and select Properties.

5. Set the **Horizontal Grid Spacing** and **Vertical Grid Spacing** to 5.

Note: From the Properties Window, the Screen Name can also be edited, among other basic settings. Feel free to take the time to explore.
Create a Push Button

1. From the Toolbox, locate **Momentary Push Button**, and drag-and-drop it onto the middle of your screen.

   ![Toolbox Image]

   Note: The “Toolbox” is context sensitive. It changes the selection of objects for HMI development versus the controller program development.

2. Double-click the button to open the States editor.
   Each row is a state with a default value, text, and other format options. Move the scroll bar to see additional options. It should look similar to the one shown below:

   ![States Editor Image]

3. Edit State 1:

   - Click in the **Caption Text** field and type “**START MOTOR**”
• Click (…) button in the **Background Color** field and select *green* from the color palatte.

![Color Palatte Image]

• Click the (…) button in the **Caption Text Color** field and select *black*.

![Color Palatte Image]

• Check the **Caption Font Bold** checkbox.

4. Edit State 2:
   - Click in the **Text** field and type *“MOTOR STARTED”*
   - Select *green* as the **Background Color**
   - Select *black* as the **Caption Text Color**.
   - Check the **Caption Font Bold** checkbox.

5. Click **row 1** so that this is the state displayed on the screen in CCW.
6. Verify your States window matches the image below and click OK to close the window and apply the changes.

![States Window](image)

7. Make sure the button is still selected and go to the Properties window. (View ➔ Properties)

8. Configure the following properties as shown below:

- **Height:** 60
- **Left:** 0
- **Top:** 180
- **Width:** 80

![Properties Window](image)

Note: The Top and Left parameters determine the location of the object. The Height and Width parameters determine the size of the object.
9. Click the **Write Tag** drop down arrow in the **Connections** section and select **Start_Motor** from the list of tags.

The **Start Motor** button is complete. Next, create the **Stop Motor** button by using a copy and paste operation.

11. Verify the **Start Motor** push button is selected.

10. Click the **Copy** icon ( ) on the toolbar or press the Ctrl+C keys on your keyboard.

11. Click the **Paste** icon ( ) on the toolbar or press the Ctrl+ keys.

12. Move the pasted button to the right of the original push button.

13. Double click the **new button** to open the **States** editor.

14. Edit state 1:
   - Replace the **Caption Text** with “STOP MOTOR”
   - Select red as the **Background Color**
   - Select white as the **Caption Text Color**
   - Double check the **Caption Font Bold** checkbox is checked.

15. Edit State 2:
   - Replace the **Caption Text** with “MOTOR STOPPED”
   - Select red as the **Background Color**
   - Select white as the **Caption Text Color**
   - Double check the **Caption Font Bold** checkbox is checked.

16. Click **row 1** so that this is the state displayed on the screen in CCW.

17. Click **OK** when done.

18. Make sure the button is still selected and go to the **Properties** window.
19. Assign tags to the **Stop Motor** push button by clicking the drop down arrow for the **Write Tag**, on the **Properties** window. Assign tag **Stop_Motor** by clicking on it.

<table>
<thead>
<tr>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicator Tag</td>
</tr>
<tr>
<td>Visibility Tag</td>
</tr>
<tr>
<td>Write Tag</td>
</tr>
</tbody>
</table>

20. Verify your screen appears as shown below.

21. Save the application by selecting the **Save** icon ( ) from the toolbar.
Create a Goto Config Button

The Goto Config button allows access to the configuration mode screens of the terminal when the button is pressed at runtime.

1. From the Toolbox, scroll down to the **Advanced** section, and find the **Goto Config** button. Click and drag it to the center of the screen.

2. Make sure the button is still selected and go to the **Properties** window.

3. We will use the default colors for the object.

4. Configure the following properties as shown below:
   - **Height:** 60
   - **Left:** 160
   - **Top:** 180
   - **Width:** 80
   - **Text:** "CONFIG SCREEN"
   - **Font Bold:** True
5. Verify the screen looks as shown below.

![Multistate Indicator](image)

6. Save the application by selecting the **Save** icon ( ) from the toolbar.

**Create a Multistate Indicator**

The Multistate indicator will show whether the motor is running or stopped. You will be creating the indicator.

1. From the Toolbox, locate the **Multistate Indicator** object listed in the Display section.

![Toolbox](image)

2. Drag the object into the center of the screen.

3. Double-click the object to open the States editor. The indicator is created with five states. This lab only uses two.
4. Delete rows 3 and 4. (Select the row, and click the **Delete** button). Select **OK** when prompted.

![Image of delete confirmation dialog]

5. Edit state 1:
   - Accept **0** as the Value.
   - Replace the MultiState0 Caption Text with **"Stopped"**
   - Select **red** as the Background Color
   - Select **black** as the Caption Text Color
   - Check the Font **Bold** and **Italics** checkboxes

6. Edit state 2:
   - Accept **1** as the Value.
   - Replace the MultiState1 Caption Text with **"Running"**
   - Select **green** as the Background Color
   - Leave **white** as the Caption Text Color
   - Check the Font **Bold** and **Italics** checkboxes

![Image of state editing dialog]
7. Click **row 1** and then click **OK** to close the editor.

8. Make sure the button is still selected and go to the **Properties** window.

9. Configure the following properties as shown below:

- **Height**: 60
- **Left**: 115
- **Top**: 85
- **Width**: 80
- **Read Tag**: Motor_Ind

10. Verify the screen looks as shown below

![](image)

11. Save the application by selecting the **Save** icon ( ) from the toolbar.
Create a Numeric Entry
The Numeric Entry will allow the user to set how long the motor will run for, in milliseconds.

1. From the Toolbox, locate the **Numeric Entry** object.

2. Drag the object into the space above the Multistate Indicator.

3. Make sure the button is still selected and go to the **Properties** window.

4. Configure the following properties as shown below:
   - **Height**: 60
   - **Left**: 100
   - **Top**: 15
   - **Width**: 110
   - **Write Tag**: `Motor_On_Time_ms`
5. Verify the screen looks as shown below

![Screen Screenshot]

6. Save the application by selecting the **Save** icon ( ) from the toolbar.

---

**Generate a Report**

In this section, you will learn how to generate a report. It provides detailed information about the PanelView Component application.

1. Go to the **Settings** tab of the PanelView Component. If it is not open, double click the **Motor Starter** icon in the **Project Organizer**.

![Settings Tab Screenshot]

2. Click the **Generate Report** button.

![Generate Report Button]

3. A pop up window will appear with the results of the report.
4. Select the **Save** icon from the toolbar. There are two options: save it as an excel file, or as a PDF. **Select** an option.

   ![Save Options]

   - Excel
   - Acrobat (PDF) file

5. Give the file a name and save it to the desktop.

   ![Save As]

6. Once the file has been saved, locate it on the desktop. **Open** the file and view the report in the preferred format. Notice the report provides images of all the screens in the application, a list of tags used, all the settings configured for the controller, as well as data usage and much more.
Validate and Transfer the Application to the Terminal

Before you run an application on the PanelView Component, it is strongly recommended that you validate the application for errors and warnings. The application does not need to be error-free to run, however, it is good practice to fix both errors and warnings to ensure expected system behavior.

1. Right-click on the MotorStarter icon in the Project Organizer and select Validate.

![Validate button]

2. The Validation Results window opens.

![Validation Results window]

3. Correct any errors if desired, and close the Validation Window.

4. Re-validate the application after the errors have been fixed.
   Warnings still may exist, however they can be ignored as they are not functionality errors. Move onto the next step.

5. Right-click on the MotorStarter icon.

![MotorStarter icon]

6. Select Download

![Download button]
7. Enter the terminals IP address, "192.168.1.2". Then click Download.

8. The Output window will show the progress of the download. Ensure the application downloads successfully before moving forward.

9. Turn to your PanelView Component terminal. It should be on the Main screen.

10. Select the File Manager button.

11. Scroll through the list of applications using the arrow keys to find the MotorStarter application you created.

12. Select Run. If prompted with a warning, select Yes (F1).

13. The application will load. Select the START MOTOR button, and let the motor run. Notice the Multistate Indicator you created now says “Running”.

14. Select the STOP MOTOR button.

15. Select the Numeric Input button you created, enter “5000” into the keypad, and press Enter.

16. Select the START MOTOR button again. Watch the motor run for 5 seconds then stop. Feel free to explore some more.

25. To exit the application, select the CONFIG SCREEN button. This will bring you back to the File Manager screen.
(OPTIONAL) Learn about User Defined Function Blocks

In this section of the lab, you will learn about User Defined Function Blocks (UDFBs) - how to create one using Structured Text and how to password protect it using Connected Components Workbench Developer Edition.

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 **FB1** 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Data Type</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td></td>
<td>REAL</td>
<td>VarInput</td>
</tr>
<tr>
<td>Height</td>
<td></td>
<td>REAL</td>
<td>VarInput</td>
</tr>
<tr>
<td>Volume</td>
<td></td>
<td>REAL</td>
<td>VarOutput</td>
</tr>
</tbody>
</table>
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.

   ```plaintext
   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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td>Height</td>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td>Volume</td>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td>Calc_Tank_Volume</td>
<td>Calc_Volume</td>
<td></td>
</tr>
</tbody>
</table>
12. Next, open the Tank_Volume program, and add an Instruction Block to the first rung.

13. Search for Calc_Volume and specify the Instance Calc_Tank_Volume. Then click OK.
14. Next, specify the variables as shown for each parameter of the Instruction 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.
19. The value of Volume should read 785.0.

20. Change the value of Radius to 7.

21. The value of Volume should read 1538.6.
22. Press **Shift+F5** to exit Debug Mode.

Now that you have proved out the functionality of your UDFB, you may want to password protect it, especially if you are trying to prevent disclosure of any proprietary algorithms you may have used. Starting with version 6.0 of Connected Components Workbench – Developer Edition, you may password protect individual UDFBs and programs so that the logic cannot be viewed without knowing the password.

23. Right click on the **Calc_VOLUME** UDFB and select **Password**.

24. Enter from one to eight alpha-numeric characters into the **New Password** field, then re-type it in the **Confirm Password** field and click **OK**.

25. Notice that the **Calc_VOLUME** UDFB now has a padlock showing on its icon to indicate that it has been password protected.
26. **Save** the project, **Close** the project then **Open** the project again. Double click on the **Calc_Volume** UDFB icon and enter the **Password** you created above in order to display the program.

![Password dialog box](image)

Note that although you need Developer Edition to add or remove password protection for programs and UDFBs, you may still enter passwords using Standard Edition in order to access password-protected files.

Congratulations!! You have now completed the Basic PLC Programming with Micro800™ Controller Family lab!