Micro800 Programming Basics
Tutorial 2: Variables and Instruction Blocks

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 temperature.
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.

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.

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 we have 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.

### System Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SYSVA_CYCLECNT</td>
<td>DINT</td>
</tr>
<tr>
<td>__SYSVA_CYCLEDATE</td>
<td>TIME</td>
</tr>
<tr>
<td>__SYSVA_KYBPERR</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_KVBCERR</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_RESNAME</td>
<td>STRING</td>
</tr>
<tr>
<td>__SYSVA_SCANCNT</td>
<td>DINT</td>
</tr>
<tr>
<td>__SYSVA_TCYCYCTIME</td>
<td>TIME</td>
</tr>
<tr>
<td>__SYSVA_TCYCURRENT</td>
<td>TIME</td>
</tr>
<tr>
<td>__SYSVA_TCYMAXIMUM</td>
<td>TIME</td>
</tr>
<tr>
<td>__SYSVA_TCYOVERFLOW</td>
<td>DINT</td>
</tr>
<tr>
<td>__SYSVA_RESMODE</td>
<td>SINT</td>
</tr>
<tr>
<td>__SYSVA_CCEXEC</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_REMOTE</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_SUSPEND_ID</td>
<td>UINT</td>
</tr>
<tr>
<td>__SYSVA_TCYWDG</td>
<td>UDINT</td>
</tr>
<tr>
<td>__SYSVA_MAJ_ERR_HALT</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_ABORT_CYCLE</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_FIRST_SCAN</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_USER_DATA_LOST</td>
<td>BOOL</td>
</tr>
<tr>
<td>__SYSVA_POWERUP_BIT</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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.

- **Bit Strings** – groups of on/off values
  - **BYTE** – 8 bit (1 byte)
  - **WORD** – 16 bit (2 byte)
  - **DWORD** – 32 bit (4 byte)
  - **LWORD** – 64 bit (8 byte)

- **INTEGER** – whole numbers (Considering byte size 8 bits)
  - **SINT** – signed short integer (1 byte)
  - **INT** – signed integer (2 byte)
  - **DINT** – signed double integer (4 byte)
  - **LINT** – signed long integer (8 byte)
  - **USINT** – Unsigned short integer (1 byte)
  - **UINT** – Unsigned integer (2 byte)
  - **UDINT** – Unsigned double integer (4 byte)
  - **ULINT** – Unsigned long integer (8 byte)

- **REAL** – floating point IEC 60559 (same as IEEE 754-2008)
  - **REAL** – (4 byte)
  - **LREAL** – (8 byte)

- **Duration**
  - **TIME** – (Size is not specified)
  - **LTIME** – (8 byte)

- **Date**
  - **DATE** – calendar date (Size is not specified)
  - **LDATE** – calendar date (Size is not specified)

- **Time of day**
  - **TIME_OF_DAY / TOD** – clock time (Size is not specified)
  - **LTIME_OF_DAY / LTOD** – clock time (8 byte)

- **Date and time of Day**
  - **DATE_AND_TIME / DT** – time and date (Size is not specified)
  - **LDATE_AND_TIME / LDT** – time and date (8 byte)

- **Character / Character string**
  - **CHAR** – Single-byte character (1 byte)
  - **WCHAR** – Double-byte character (2 byte)
  - **STRING** – Variable-length single-byte character string
  - **WSTRING** – Variable-length double-byte character 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.

![Local Variables panel](image)

2. Create a variable called **Motor_On_Time** of Data Type **TIME**.

![Variable Motor_On_Time](image)

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

![Variable Motor_On_Time_ms](image)
4. Create a variable called **Motor_Timer** of Data Type **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, mathematical comparisons and turning ON and OFF

1. Open Tutorial Project PLC Basics from Tutorial 1 > Program should look as follows

2. Drag-and-drop a Branch instruction to right side of the rung, wrapping around the coil instruction.
3. Locate the **Block** instruction in the Toolbox.

4. Drag-and-drop this 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. 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.

7. 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.
8. Your ladder program should look like the following.

9. Next, hover the mouse cursor over the PT parameter of the Motor_Timer TON instruction. You will notice a light blue highlighting the box..

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.

Output Side will remain blank until the program is running then the program will show the timer counting up to Motor_On_Time Value.
11. Your ladder program should look like the following.

![Ladder Diagram Image]

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

![Ladder Diagram Image]
13. The Variable Selector will display. Select the **Local Variables – Motor_Circuit** tab, and then click the empty cell shown below.

14. Expand the variable, **Motor_Timer** > Select **Motor_Timer.Q** > click **OK**.
15. Your ladder program should look like the following.

![Ladder Diagram](image)

16. Locate the rung instruction in the Toolbox.

![Toolbox Image](image)
17. Drag and drop the **Rung** instruction below Rung 1. Program should look as follows:

![Diagram showing Rung instruction placement](image)

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

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.

![Instruction Block Selector](image)
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. Your program should look like the following.
   Note: Both Rungs are independent from each other Rung 2 will execute as soon as the program is downloaded. Rung 1 will run when D10 is pressed.

23. 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).

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

25. Now test your program. Turn the DI0 switch on, and watch the DO7 light turn on. After 5 seconds, the light should turn off.

26. Debugging: Double Click on Local Variables under Program 1 > Change the logical 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. The program will update on the fly.
27. Now test your program again. Turn the DI0 switch on. The DO8 light should now stay on for 10 seconds, before turning off.

28. Click the Stop button to exit Debug Mode (or press Shift+F5).

29. You have completed this section of the lab.

**Part 2:** Now we will add a few more lines of code to turn on a separate light after a DI0 has been pressed a certain amount of times.

1. Setting a Variable Alias: Aliases can be used to provide a clearer understanding what the I/O Embedded Address is operating.

   1. Double Click on the Local Variables > I/O Select Micro 850 > Set the following
ii. Create a variable **Counter** > Data Type = Int > Initial Value = 0

iii. Create a variable **CounterReset** > Data Type = Int > Initial Value = 0

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Data Type</th>
<th>Dimension</th>
<th>Project Value</th>
<th>Initial Value</th>
<th>String Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor_On_Time</td>
<td></td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motor_On_Time_ms</td>
<td></td>
<td>INT</td>
<td></td>
<td>5000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motor_Timer</td>
<td></td>
<td>TON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter</td>
<td></td>
<td>INT</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b. Add the following code

- **Reset Coil** will change the Green Light (DO6) to the Off State
- **Timer** allows for the Green Light (DO6) to stay turned off for 3 secs (T#3s) before counter is reset and the machine resumes operation. Scenario inserting or removing part from a mill and inserting a new piece to be operated on

- **CounterReset** is defaulted at 0, thus resetting the process
Assignment 2: Drag Racer Lights

Directions: Create a program that will do the following

1. Toggle Switch DI4 to start program
2. Turn ON the Red Lights (DO0/DO1)
3. Timer 2 seconds
4. Turn ON the Yellow Lights (DO2/DO3); Keep Yellow Lights ON
5. Timer 2 seconds
6. Turn ON the Green Lights (DO4/DO5); Keep Yellow/Red Lights ON
7. Timer 2 seconds
8. Turn OFF all lights and turn ON Blue Lights (DO6/DO7)
9. Timer 5 seconds
10. Turn OFF all lights

Submission:
A. Show/Video program working (Show computer screen and PLC Board) B.

Printout a copy of the program