ECET 4530

Industrial Motor Control

Introduction to
Ladder Logic Programming
(in the RSLogix environment)

Part II

Tags

Tags contain information that identifies and characterizes data stored in memory, allowing that data to be linked to the operation of a specific instruction.

For example, given the following XIC (Logic Instruction):

the "A" displayed above the XIC icon is actually the name of a Tag that identifies a specific bit in memory, the value of which determines the state of the XIC.

Base Tags vs. Alias Tags

A Base Tag is a tag that, when initially created, results in the allocation of memory and provides a reference to the data stored at that memory location, thus allowing the data to be directly linked to the operation of an instruction.

An Alias Tag is a tag that is used to provide a different name (i.e. – an alias) to a previously defined Base Tag.

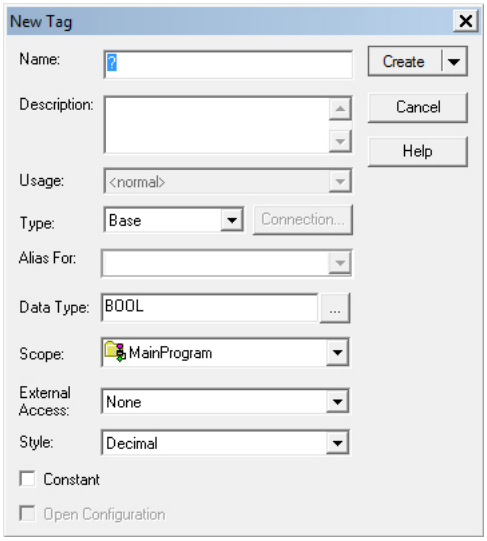
Note – Alias tags allow the assignment of an arbitrary name to a previously defined Base Tag. This can be useful, especially when an existing Base Tag has a complex or potentially confusing name, such as those automatically created by the RSLogix software when the type and configuration of the PLC is initially defined.

Creating a Base Tag

When a Base Tag is initially created, a variety of information must be provided, including:

- **NAME**
- **TYPE**
- **DATA TYPE**
- **SCOPE**
- **EXTERNAL ACCESS**
- **STYLE**

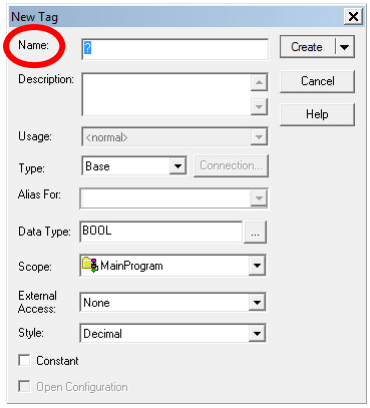
A "text" Description of the tag may also be provided if desired.



Base Tag Information Fields

NAME: an alpha/numeric identifier (including underscores) that is assigned to the tag and thus to the data stored in memory.

Names must begin with an alphabetic character, cannot be greater than 40 characters in length, and are not case sensitive.

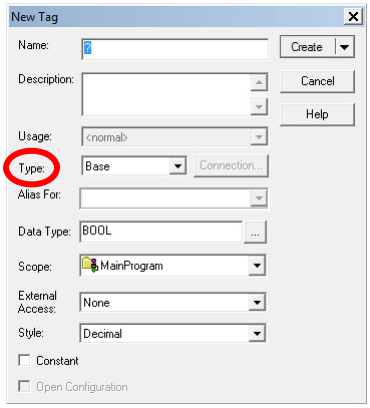


Base Tag Information Fields

TYPE: Base – this results in an allocation of memory, the amount of which is determined by the Data Type field.

Note – the "Type" field may also be set to Alias. This is used when defining an Alias Tag.

When an Alias Tag is selected, the "Alias For" field becomes active. This field is used to select the Base Tag for which the Alias Tag provides a new name.



Base Tag Information Fields

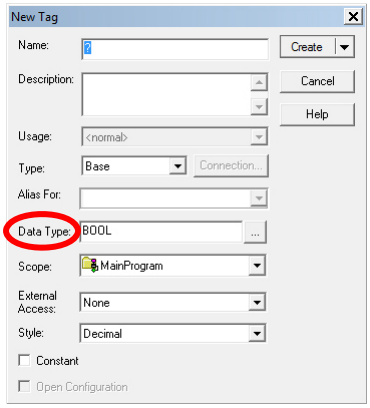
DATA TYPE:

BOOL (Boolean) – Boolean tags require only a single bit of memory, the value of which can be either

0 or 1

INT (Integer) – Integer tags require sixteen bits of memory, within which integer numbers may be stored ranging from

–32768 to +32767

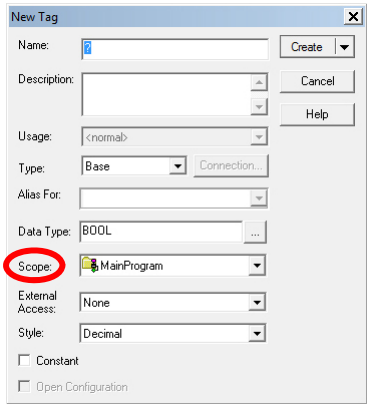


Base Tag Information Fields

SCOPE: this field defines whether a tag is global or local.

The RSLogix 5000 software allows an application to be broken down into multiple programs (subroutines).

A global tag is available to all of the programs, while a local tag can only be utilized within a specific program, such as "Main Program".



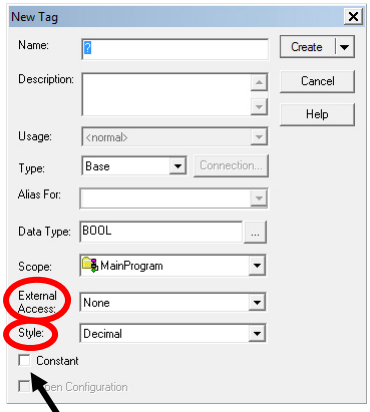
Base Tag Information Fields

EXTERNAL ACCESS:

This field defines whether a tag is allowed Read/Write, Read Only, or no access (None) from programs running on external devices such as HMI (Human Machine Interface) Panels.

STYLE: this field only defines how the tag data is displayed; it does not affect the Data Type.

Binary – Base 2 or Decimal – Base 10



If checked, CONSTANT prevents Instructions from changing the value of the data that is identified by the tag.

Initial Software Configuration

When first entering the RSLogix environment with the goal of creating a new ladder diagram (program), the type and configuration of the PLC must be specified.

For Example: The PLCs contained in the Q-215 Machines Lab are:

- **Allen-Bradley, 1769-L32E, Compact Logix 5332E PLCs**

Located in Bus Position #1 of each PLC is a:

- **1769-IQ16/A, 16-Port, 24V DC Input Module**

Located in Bus Position #2 of each PLC is a:

- **1769-OW16/A, 16-Port, Relay Output Module**

(The method for entering the above information is not covered within this presentation)

Initial Software Configuration

After the initial configuration information is entered:

- **Allen-Bradley, 1769-L32E, Compact Logix 5332E PLC**
- **Bus #1: 1769-IQ16/A, 16-Port, 24V DC Input Module**
- **Bus #2: 1769-OW16/A, 16-Port, Relay Output Module**

the RSLogix 5000 software automatically creates a set of Base Tags that are required for proper operation of the PLC and its Input and Output Modules.

The automatically created Base Tags include:

One Boolean-type Base Tag for each Input Port, and

One Boolean-type Base Tag for each Output Port.

1769-IQ16 Input Module

The 1769-IQ16/A, 16-Port, 24V DC Input Module is a 16 port discrete input module that is configured to detect either the presence or the absence of a $24V_{DC}$ signal at the terminals of its inputs.

If $24V_{DC}$ is detected at terminal IN-X, the state of Input Port X becomes TRUE.

If $24V_{DC}$ is not detected at terminal IN-X, the state of Input Port X becomes FALSE.

1769-IQ16 Input Module

When the 1769-IQ16 module is added into an RSLogix project, the software creates 16 Boolean-Type Base Tags, one for each of the module's Input Ports.

The format of the Base Tag names is:

Local:1:I.Data.X

where X ranges from 0 → 15 for each of the 16 Input Ports.

Note that the actual base tag name is "I.Data.X" where X ranges from 0 → 15.

The "Local:1" in front of the tag name instructs the PLC that the tag refers to a memory location that resides within the module that is (directly) connected "locally" to the PLC in the 1st slot position.

1769-IQ16 Input Module

Furthermore, the 1769-IQ16 links each of its ports to the associated memory bits referred to by the I.Data.X Base Tags.

When Input Port X is becomes TRUE, the module sets the bit referenced by tag I.Data.X to a 1.

When Input Port X is becomes FALSE, the module resets the bit referenced by tag I.Data.X to a 0.

1769-IQ16 Input Module

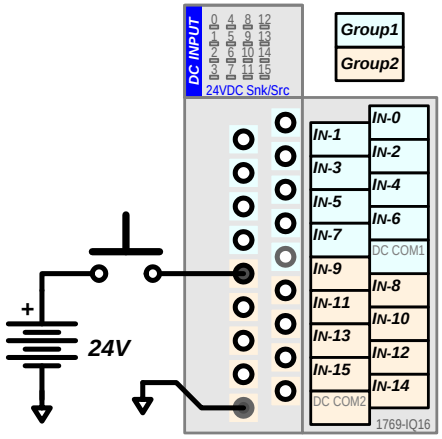
For example, if a <u>NO pushbutton</u> is connected between a $24V_{DC}$ source and terminal IN-9:

When the <u>button is pressed</u>,

- **$24V_{DC}$ is present at IN-9**
- **Input Port 9 ➡ TRUE,**
- **Bit I.Data.9 ➡ 1**

When the <u>button is released</u>,

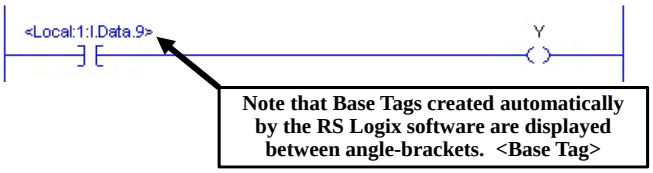
- **$24V_{DC}$ is removed from IN-9**
- **Input Port 9 ➡ FALSE,**
- **Bit I.Data.9 ➡ 0**



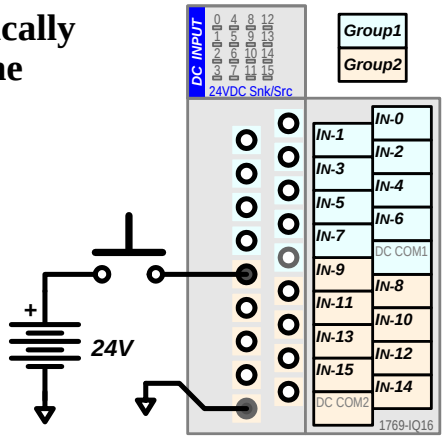
Linking the 1769-IQ16 to Ladder Logic

The 1769-IQ16's inputs can be linked to the Logic Instructions within a ladder by assigning the tags that were automatically created for module's input ports to the Logic Instructions.

For example, tag Local:1:I.Data.9 is assigned to the XIC shown below:



Note that Base Tags created automatically by the RS Logix software are displayed between angle-brackets. <Base Tag>

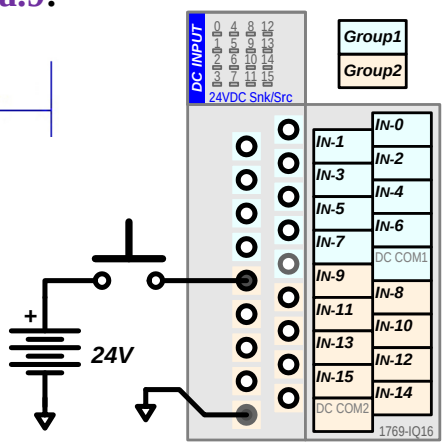


Linking the 1769-IQ16 to Ladder Logic

Given the following ladder rung in which the XIC is assigned the tag Local:1:I.Data.9:

When the button is NOT pressed,

- **$24V_{DC}$ is not present at IN-9**
- **Input Port 9 ➡ FALSE,**
- **Bit I.Data.9 ➡ 0**
- **XIC-Local:1:I.Data.9 ➡ FALSE**
- **OTE-Y ➡ FALSE**

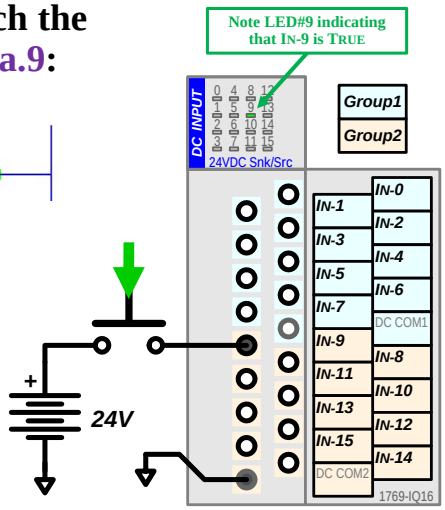


Linking the 1769-IQ16 to Ladder Logic

Given the following ladder rung in which the XIC is assigned the tag Local:1:I.Data.9:

When the button IS pressed,

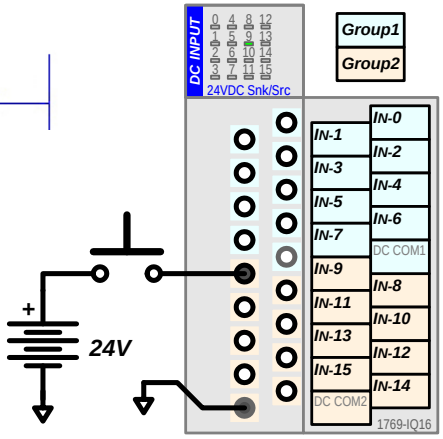
- **$24V_{DC}$ is present at IN-9**
- **Input Port 9 ➡ TRUE,**
- **Bit I.Data.9 ➡ 1**
- **XIC-Local:1:I.Data.9 ➡ TRUE**
- **OTE-Y ➡ TRUE**



Linking the 1769-IQ16 to Ladder Logic

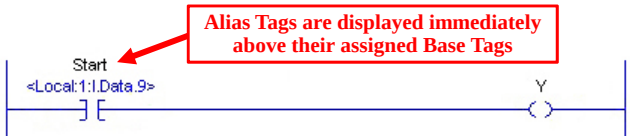
Although the tag Local:1:I.Data.9 is clearly linked to the XIC in the rung shown below:

the function of the pushbutton wired to Input Port 9 is not clearly specified, which may be confusing when trying to create, modify, or debug the ladder diagram.

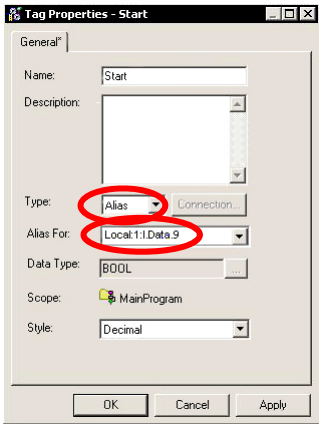


Creating Aliases for the 1769-IQ16's Tags

Alias Tags may be created for the 1769-IQ16's Base Tags to help clarify the ladder diagram:

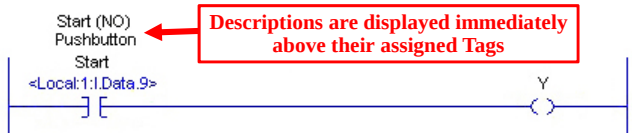


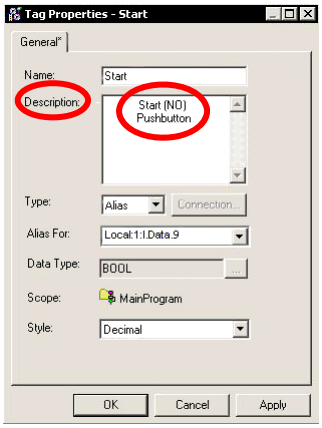
For example, if the intended function of the button wired to Input Port 9 is to "Start" a process, then an Alias Tag named Start may be created for the Local:1:I.Data.9 Base Tag.



Creating Aliases for the 1769-IQ16's Tags

Alias Tags may be created for the 1769-IQ16's Base Tags to help clarify the ladder diagram:





Additionally, if a description is added for the Alias Tag named Start, the description will also appear in the ladder diagram immediately above the Alias Tag name.

1769-OW16 Output Module

The 1769-OW16/A, 16-Port, Relay Output Module is a discrete 16 port output module that has a NO contact wired between each output terminal and a common terminal.

Whenever Output Port X is FALSE, the module opens the contact wired between terminal OUT-X and its common terminal.

Whenever Output Port X is TRUE, the module closes the contact wired between terminal OUT-X and its common terminal.

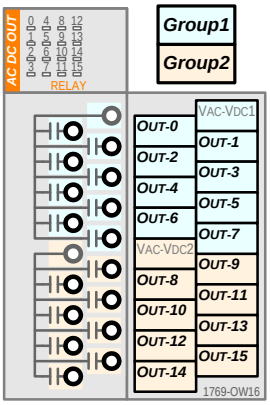
1769-OW16 Output Module

Note that the 1769-IQ16's output ports are separated into two groups:

- **Group 1: OUT-0 → OUT-7**
- **Group 2: OUT-8 → OUT-15**

The NO contacts for Group 1 are connected between the output terminals and common terminal VAC-VDC1.

The NO contacts for Group 2 are connected between the output terminals and common terminal VAC-VDC2.



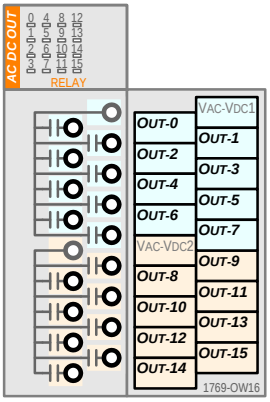
1769-OW16 Output Module

When the 1769-OW16 module is added into an RSLogix project, the software creates 16 Boolean-Type Base Tags, one for each of the module's Output Ports.

The format of the Base Tag names is:

Local:2:O.Data.X

where X ranges from 0 → 15 for each of the 16 Output Ports.



Note that the actual base tag name is "O.Data.X" where X ranges from 0 → 15.

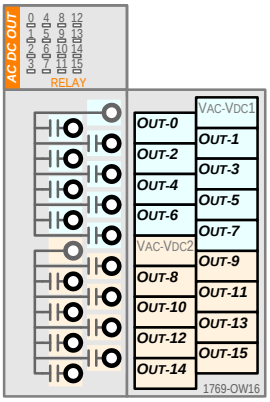
The "Local:2" in front of the tag name instructs the PLC that the tag refers to a memory location that resides within the module that is (directly) connected "locally" to the PLC in the 2nd slot position.

1769-OW16 Output Module

Furthermore, the 1769-OW16 links each of its ports to the associated memory bits referred to by the O.Data.X Base Tags.

When the bit referenced by O.Data.X is 1, Output Port X ➡ TRUE and the contact between OUT-X and VAC-VDC1 closes.

When the bit referenced by O.Data.X is 0, Output Port X ➡ FALSE and the contact between OUT-X and VAC-VDC1 opens.



1769-OW16 Output Module

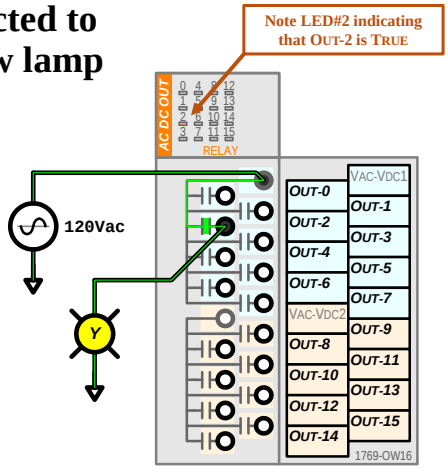
For example, if a $120V_{ac}$ source is connected to terminal VAC-VDC1 and a 120V, yellow lamp is connected to terminal OUT-2:

When bit O.Data.2 = 0,

- Output Port 2 is FALSE,
- Contact OUT-2 is OPEN
- The Yellow Lamp is OFF

When bit O.Data.2 = 1,

- Output Port 2 is TRUE,
- Contact OUT-2 is CLOSED
- The Yellow Lamp is ON

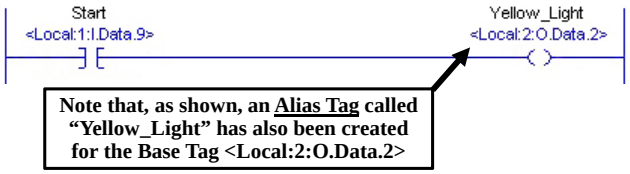


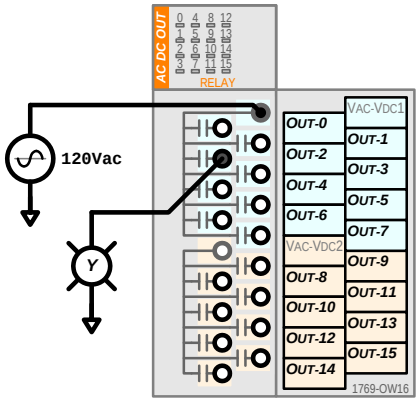
Note that the above figure shows the case when bit O.Data.2 = 1

Linking the 1769-OW16 to Ladder Logic

The 1769-OW16's outputs can be linked to the Output Instructions within a ladder by assigning the tags that were automatically created for module's ports to the Output Instructions.

For example, tag Local:2:O.Data.2 is assigned to the OTE shown below:



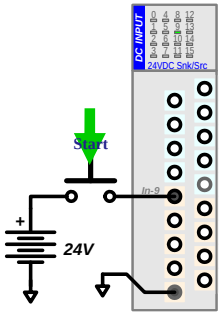


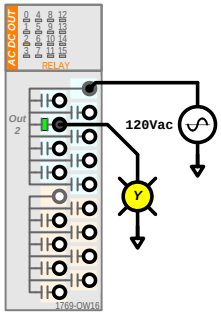
From Input to Output

Given the system shown below, when the button is pressed:

- **$24V_{DC}$ is applied to terminal IN-9**
- **Input Port 9 ➡ TRUE**
- **Bit I.Data.9 ➡ 1**
- **XIC-Start ➡ TRUE**
- **Rung Condition ➡ TRUE**

- **OTE-Yellow_Light Sets Bit O.Data.2 ➡ 1**
- **Contact OUT-2 ➡ CLOSED**
- **Yellow Lamp ➡ ON**

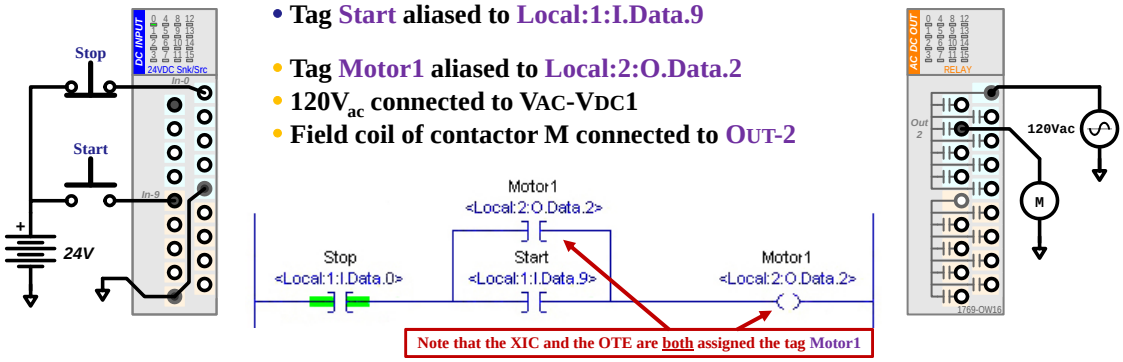




Stop/Start Motor Controller using OTE

Given the a system with the following configuration:

- 24V_{DC} is connected to the NC and NO pushbuttons
- NC pushbutton connected to IN-0
- NO pushbutton connected to IN-9
- DC-COM1 and DC-COM2 connected to DC ground
- Tag Stop aliased to Local:1:I.Data.0
- Tag Start aliased to Local:1:I.Data.9
- Tag Motor1 aliased to Local:2:O.Data.2
- 120V_{ac} connected to VAC-VDC1
- Field coil of contactor M connected to OUT-2

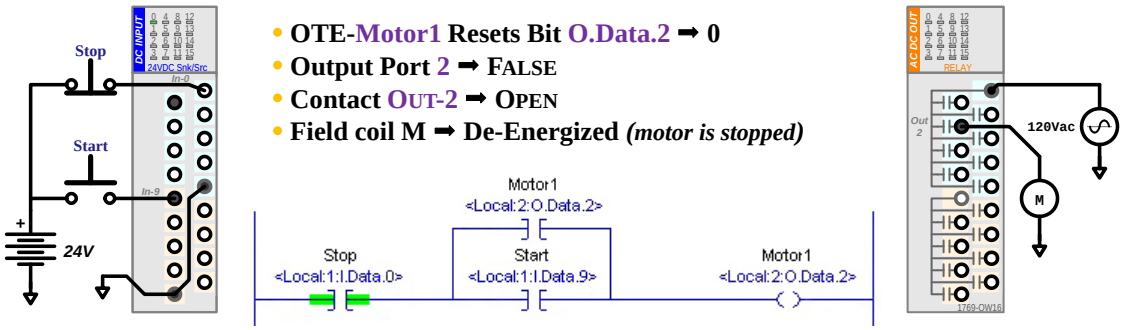


Stop/Start Motor Controller using OTE

Under "normal" conditions when no buttons are pressed:

- 24V_{DC} is applied to IN-0 (by the NC pushbutton)
- Input Port 0 ➡ TRUE — Input Port 9 ➡ FALSE
- Bit I.Data.0 ➡ 1 — Bit I.Data.9 ➡ 0
- XIC-Stop ➡ TRUE — XIC-Start ➡ FALSE
- Rung Condition ➡ FALSE

- OTE-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized *(motor is stopped)*

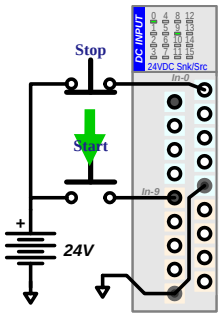


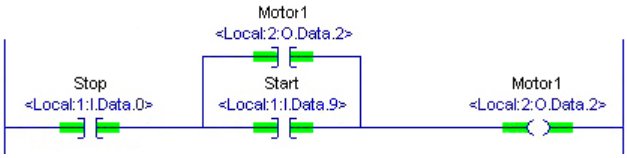
Stop/Start Motor Controller using OTE

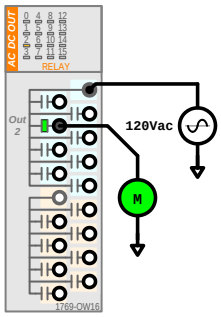
When the "Start" button is pressed:

- $24V_{DC}$ is applied to IN-9 (by the NO pushbutton)
- Input Port 9 ➡ TRUE
- Bit I.Data.9 ➡ 1
- XIC-Start ➡ TRUE
- Rung Condition ➡ TRUE

- OTE-Motor1 Sets Bit O.Data.2 ➡ 1
- Output Port 2 ➡ TRUE XIC-Motor1 ➡ TRUE
- Contact OUT-2 ➡ CLOSED
- Field coil M ➡ Energized *(motor starts)*





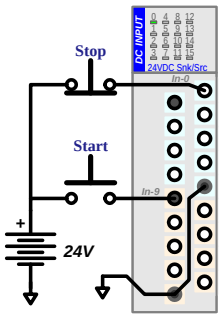


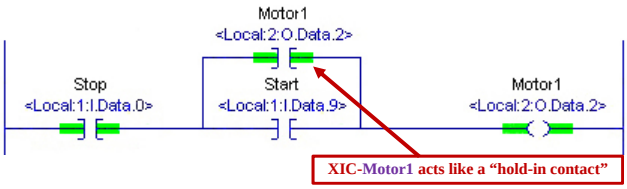
Stop/Start Motor Controller using OTE

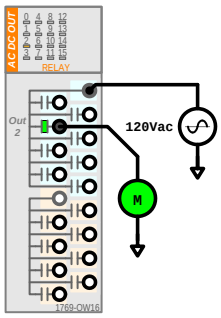
When the "Start" button is released:

- $24V_{DC}$ is removed from IN-9 (by the NO pushbutton)
- Input Port 9 ➡ FALSE
- Bit I.Data.9 ➡ 0
- XIC-Start ➡ FALSE
- Rung Condition remains TRUE due to XIC-Motor1

- OTE-Motor1 Sets Bit O.Data.2 (O.Data.2 remains 1)
- Output Port 2 and XIC-Motor1 remain TRUE
- Contact OUT-2 remains CLOSED
- Field coil M remains Energized *(motor keeps running)*



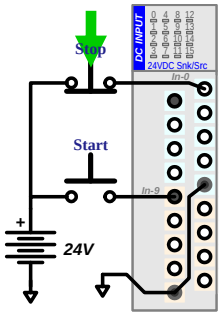




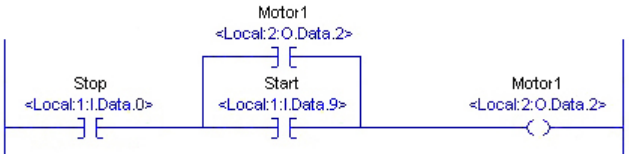
Stop/Start Motor Controller using OTE

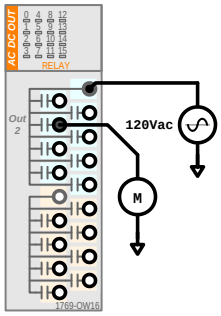
When the "Stop" button is pressed:

- 24V_{DC} is removed from IN-0 (by the NC pushbutton)
- Input Port 0 ➡ FALSE
- Bit I.Data.0 ➡ 0
- XIC-Stop ➡ FALSE
- Rung Condition ➡ FALSE



- OTE-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 ➡ FALSE XIC-Motor1 ➡ FALSE
- Contact OUT-2 ➡ OPENS
- Field coil M ➡ De-Energized *(motor stops)*

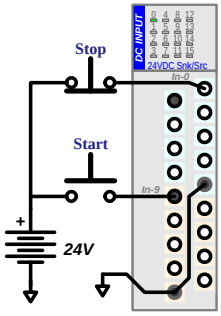




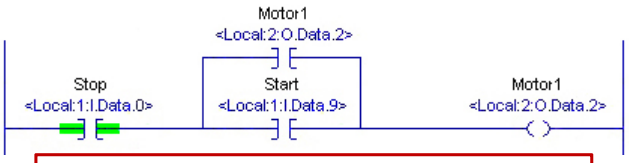
Stop/Start Motor Controller using OTE

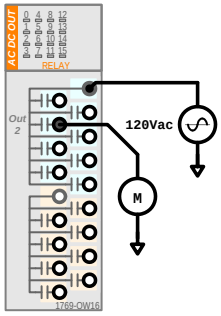
When the "Stop" button is released:

- 24V_{DC} is re-applied to IN-0 (by the NC pushbutton)
- Input Port 0 ➡ TRUE
- Bit I.Data.0 ➡ 1
- XIC-Stop ➡ TRUE
- Rung Condition remains FALSE



- OTE-Motor1 Resets Bit O.Data.2 (O.Data.2 remains 0)
- Output Port 2 and XIC-Motor1 remain FALSE
- Contact OUT-2 remains OPENED
- Field coil M remains De-Energized *(motor is stopped)*



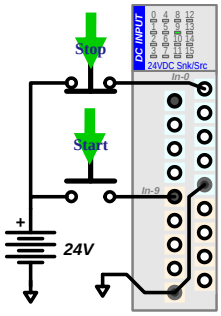


The system has returned to the initial "stopped" condition.

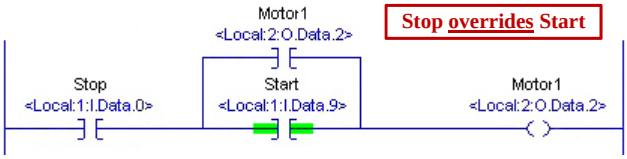
Stop/Start Motor Controller using OTE

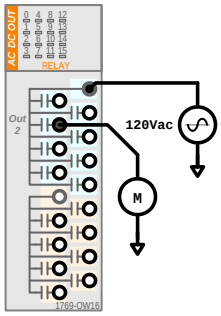
What if "Stop" and "Start" are pressed at the same time:

- $24V_{DC}$ is removed from IN-0 — $24V_{DC}$ is applied to IN-9
- Input Port 0 ➡ FALSE — Input Port 9 ➡ TRUE
- Bit I.Data.0 ➡ 0 — Bit I.Data.9 ➡ 1
- XIC-Stop ➡ FALSE — XIC-Start ➡ TRUE
- Rung Condition ➡ FALSE



- OTE-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 and XIC-Motor1 ➡ FALSE
- Contact OUT-2 ➡ OPENS
- Field coil M ➡ De-Energized *(motor stops)*

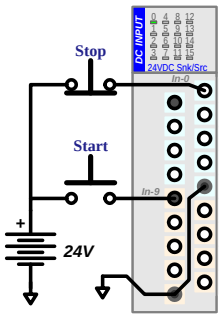




Stop/Start Motor Controller using OTL/OTU

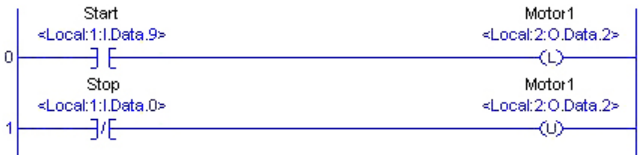
Under "normal" conditions when no buttons are pressed:

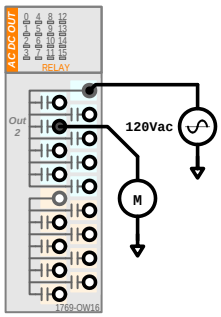
- $24V_{DC}$ is applied to IN-0 (by the NC pushbutton)
- Input Port 0 ➡ TRUE — Input Port 9 ➡ FALSE
- Bit I.Data.0 ➡ 1 — Bit I.Data.9 ➡ 0
- XIC-Stop ➡ FALSE — XIC-Start ➡ FALSE
- Rung Condition 1 ➡ FALSE — Rung Condition 0 ➡ FALSE



Assuming Bit O.Data.2 = 0 ⇦ *(Initial Condition)*

- OTL-Motor1 and OTU-Motor1 do nothing
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized *(motor is stopped)*





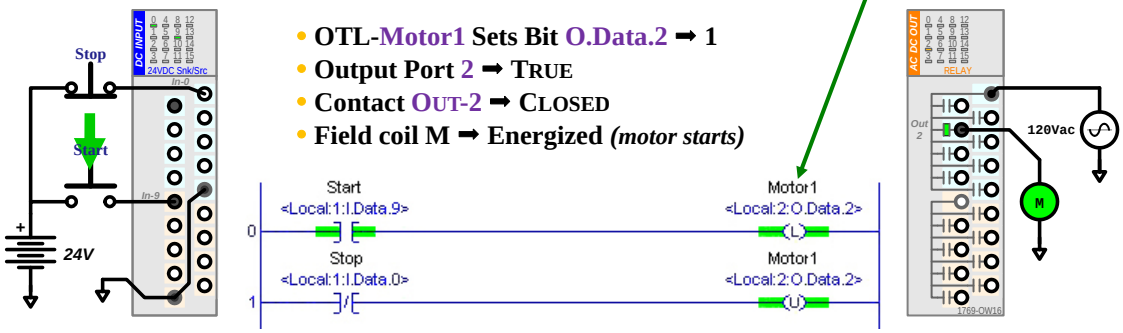
Stop/Start Motor Controller using OTL/OTU

When the "Start" button is pressed:

- $24V_{DC}$ is applied to IN-9 (by the NO pushbutton)
- Input Port 9 ➡ TRUE
- Bit I.Data.9 ➡ 1
- XIC-Start ➡ TRUE
- Rung Condition 0 ➡ TRUE

Green bar behind OTL/OTL denotes that the bit referenced by their tag is a 1
Local:2:O.Data.2 = 1

- OTL-Motor1 Sets Bit O.Data.2 ➡ 1
- Output Port 2 ➡ TRUE
- Contact OUT-2 ➡ CLOSED
- Field coil M ➡ Energized *(motor starts)*



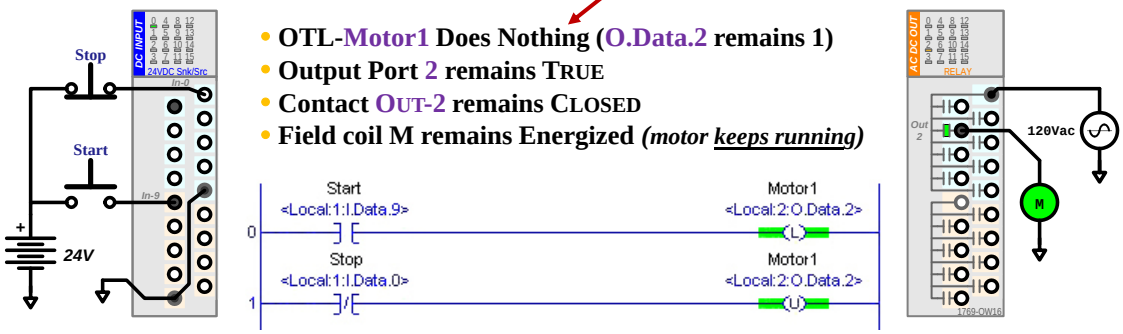
Stop/Start Motor Controller using OTL/OTU

When the "Start" button is released:

- $24V_{DC}$ is removed from IN-9 (by the NO pushbutton)
- Input Port 9 ➡ FALSE
- Bit I.Data.9 ➡ 0
- XIC-Start ➡ FALSE
- Rung Condition 0 ➡ FALSE

OTL-Motor1 cannot reset the bit O.Data.2, therefore when OTL-Motor1 ➡ FALSE, the bit O.Data.2 remains 1

- OTL-Motor1 Does Nothing (O.Data.2 remains 1)
- Output Port 2 remains TRUE
- Contact OUT-2 remains CLOSED
- Field coil M remains Energized *(motor keeps running)*

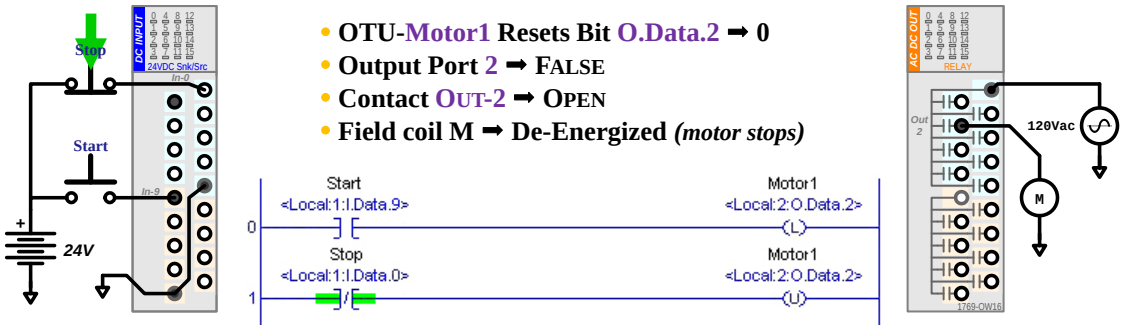


Stop/Start Motor Controller using OTL/OTU

When the "Stop" button is pressed:

- $24V_{DC}$ is removed from IN-0 (by the NC pushbutton)
- Input Port 0 ➡ FALSE
- Bit I.Data.0 ➡ 0
- XIO-Stop ➡ TRUE
- Rung Condition 1 ➡ TRUE

- OTU-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized *(motor stops)*

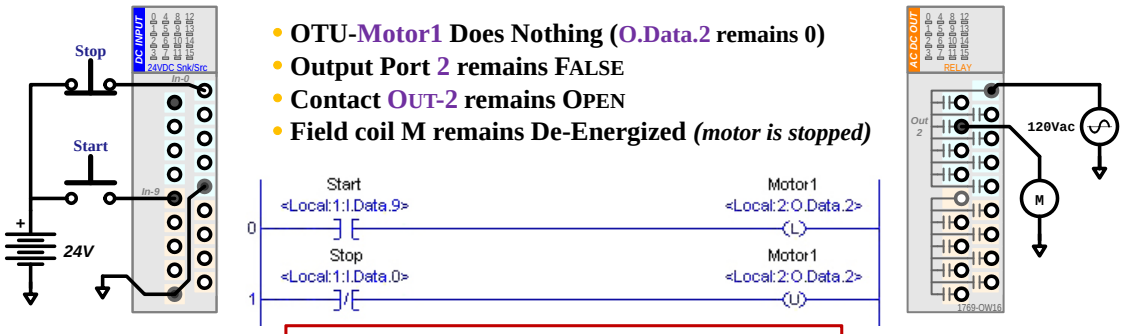


Stop/Start Motor Controller using OTL/OTU

When the "Stop" button is released:

- $24V_{DC}$ is re-applied to IN-0 (by the NC pushbutton)
- Input Port 0 ➡ TRUE
- Bit I.Data.0 ➡ 1
- XIO-Stop ➡ FALSE
- Rung Condition 1 ➡ FALSE

- OTU-Motor1 Does Nothing (O.Data.2 remains 0)
- Output Port 2 remains FALSE
- Contact OUT-2 remains OPEN
- Field coil M remains De-Energized *(motor is stopped)*



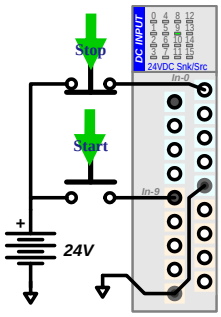
The system has returned to the initial "stopped" condition.

Stop/Start Motor Controller using OTL/OTU

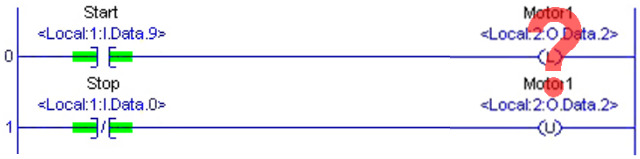
What if "Stop" and "Start" are both pressed at the same time:

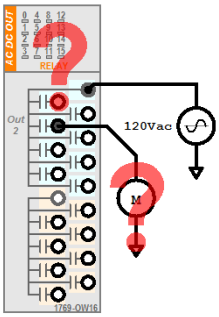
- $24V_{DC}$ is removed from IN-0 — $24V_{DC}$ is applied to IN-9
- Input Port 0 ➡ FALSE — Input Port 9 ➡ TRUE
- Bit I.Data.0 ➡ 0 — Bit I.Data.9 ➡ 1
- XIO-Stop ➡ TRUE — XIC-Start ➡ TRUE
- Rung Condition 1 ➡ TRUE — Rung Condition 1 ➡ FALSE

Actual operation will depend on the placement of the OTL & OTU within the overall ladder diagram



- OTU/OTL Set/Reset Bit O.Data.2 conflicting operations
- Output Port 2 state unknown
- Contact OUT-2 state unknown
- Field coil M state unknown (*motor state unknown*)

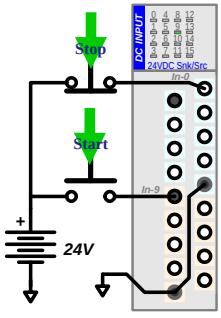




Stop/Start Motor Controller using OTL/OTU

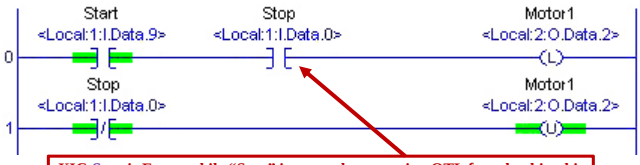
What if "Stop" and "Start" are both pressed at the same time:

- $24V_{DC}$ is removed from IN-0 — $24V_{DC}$ is applied to IN-9
- Input Port 0 ➡ FALSE — Input Port 9 ➡ TRUE
- Bit I.Data.0 ➡ 0 — Bit I.Data.9 ➡ 1
- XIO-Stop ➡ TRUE — XIC-Start ➡ TRUE XIC-Stop ➡ FALSE
- Rung Condition 1 ➡ TRUE — Rung Condition 1 ➡ FALSE

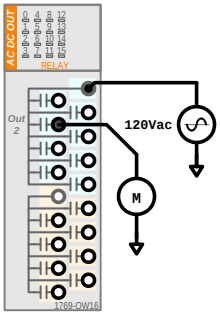


- OTU-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized (*motor stops*)

Stop overrides Start



XIC-Stop is FALSE while "Stop" is pressed, preventing OTL from latching bit



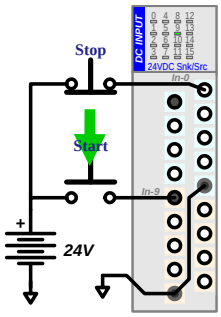
Destructive Bit Reference Error using OTEs

When the "Start" button is pressed:

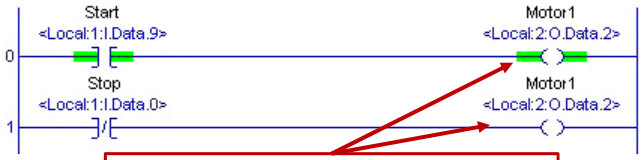
- $24V_{DC}$ was already at IN-0
- Input Port 0 ➡ TRUE
- Bit I.Data.0 ➡ 1
- XIO-Stop ➡ FALSE
- Rung Condition 1 ➡ FALSE

$24V_{DC}$ is applied to IN-9
Input Port 9 ➡ TRUE
Bit I.Data.9 ➡ 1
XIC-Start ➡ TRUE
Rung Condition 1 ➡ TRUE

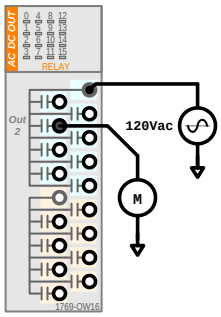
There should never be two or more OTEs with the same tag in a ladder diagram!!



- OTE/OTE Set/Reset O.Data.2 conflicting operations
- Output Port 2 state unknown
- Contact OUT-2 state unknown
- Field coil M state unknown (*motor state unknown*)



OTE-Motor1 on Rung 0 conflicts with OTE-Motor1 on Rung 1



PowerFlex 40 Variable Frequency Drive

When properly configured* and connected to the PLC via a communications network, the operation of a PowerFlex 40 Variable Frequency Drive (PF40 VFD) can also be controlled and monitored by the PLC.

This is accomplished by either manipulating or monitoring the values stored in specific locations within the PF40's memory.

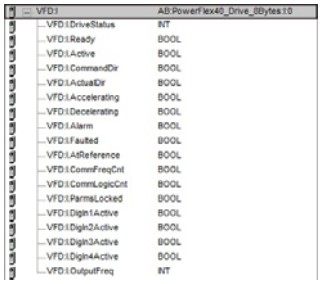
* - detailed information regarding the configuration of the PF40 VFD is not provided in this presentation.

Refer to parts B and C of the CompactLogix-based Motor Control System tutorial lab exercise for wiring and configuration information of the PF40.

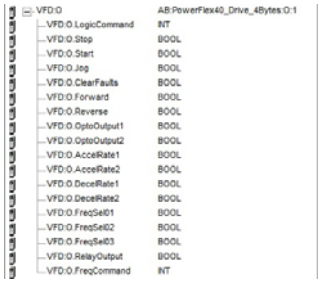
PowerFlex 40 – Tags

When a PF40 is initially added into an RSLogix project, a set of Tags are automatically created by the RS5000 software, similar to those created for the IQ16 and OW16 input and output modules:

Input Tags



Output Tags

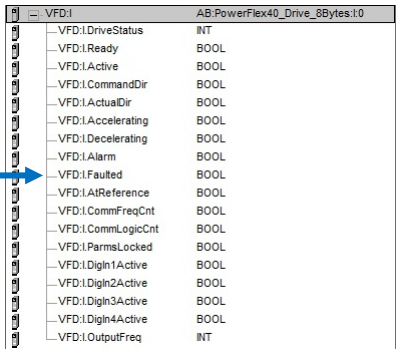


VFD – Input Tags

The <u>Input Tags</u> refer to memory locations within the PF40's memory that contain information relating to the VFD's operational state:

Input Tags

FOR EXAMPLE
When the VFD becomes faulted, the bit referred to by the tag:
I.Faulted
is set to a 1.
Thus, the value of this bit can be monitored by the PLC, and if a fault occurs, the PLC can be programmed to take the appropriate action.



Not that the "VFD" that appears in front of the tag names is the name assigned to the PF40 when added into the RSLogix project.

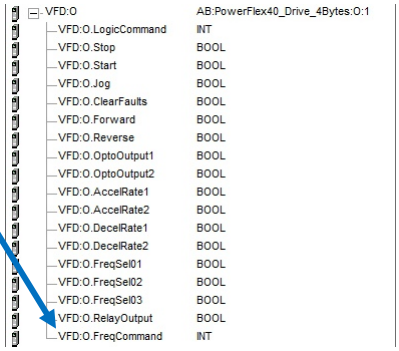
Note that the values stored in the Input Tag locations are set by the VFD based upon its operational state and **<u>cannot</u>** be changed by the PLC (user).

VFD – Output Tags

The <u>Output Tags</u> refer to memory locations within the PF40's memory that contain information that determines the operational state of the VFD:

Output Tags

FOR EXAMPLE
The integer value stored in the location defined by the tag: ***O.FreqCommand*** determines the output frequency of the drive. Thus, a MOV command can be utilized to write a new value to this location in order to change the drive's output frequency.



Not that the "VFD" that appears in front of the tag names is the name assigned to the PF40 when added into the RSLogix project.

Note that the values stored in the Output Tag locations **<u>can</u>** be changed by the PLC (user) in order to change the operational state of the drive.

Starting & Stopping the VFD

Two tags define whether the PF40 is enabled or disabled:

O.Start

O.Stop

The O.Start and O.Stop bits should always contain opposing values.

To <u>enable</u> (start) the drive, the O.START bit should be set ➡ 1 and the O.STOP bit should be reset ➡ 0.

To <u>disable</u> (stop) the drive, the O.STOP bit should be set ➡ 1 and the O.START bit should be reset ➡ 0.

Setting the Direction of the VFD

Two tags also define direction of rotation for the PF40's motor:

O.Forward

O.Reverse

The O.Forward and O.Reverse bits should also always contain opposing values.

To set the drive for <u>forward</u> operation, the O.FORWARD bit should be set ➡ 1 and the O.REVERSE bit should be reset ➡ 0.

To set the drive for <u>reverse</u> operation, the O.REVERSE bit should be set ➡ 1 and the O.FORWARD bit should be reset ➡ 0.

Setting the Output Frequency of the VFD

A single tag defines the <u>output frequency</u> of the PF40:

O.FreqCommand

To change the frequency of the waveforms being output by the drive, a new (integer) value should be written into the location defined by the O.FREQCOMMAND tag.

Although the PF40 sets its output frequency based on the value stored in the O.FreqCommand memory location, the PF40 interprets the last digit of value as tenths of a Hertz.

Thus, in the example shown above, if a value of "200" is moved (stored) in the O.FreqCommand memory location, the PF40 will set its output frequency to 20.0 Hertz.

ClearFaults – VFD

The tag:

O.ClearFaults

can be used to clear the drive when it becomes faulted, similar to physically pushing the "Stop" button on the front panel of the PF40.

Warning – The rung shown above will automatically clear any fault that occurs regardless of the fault type. Although this will be useful in the lab environment while programming the PLC, this could lead to hazardous situations in a live (industrial) setting.