

Your Global Automation Partner

TURCK

ARGEE Reference Manual

MA1019



Table of Contents

Preface	6
Why use ARGEE?	7
What are ARGEE's advantages and limitations?	7
Who should use this manual?	7
What is the purpose of this manual?	7
What content is in this manual?	8
Chapter 1: Logging into ARGEE	9
Opening the Environment.....	9
Logging into Program Mode.....	9
Welcome to Flow Chart.....	9
Chapter 2: ARGEE Menu Bar	10
Run.....	10
Debug.....	11
Reset.....	12
Open/Save As.....	13
New Project.....	14
Erase Project.....	14
Convert to ARGEE PRO.....	15
About.....	15
Set Title.....	15
Print.....	16
Edit HMI.....	16
View HMI.....	16
Edit Code.....	17
Project.....	17
Settings.....	18
Run Without Source.....	18
Modify Variables.....	19
Finish Modifications.....	19
Show Trace, Stop Trace, Resume Trace.....	20
Show Variables.....	20
Chapter 3: Getting Familiar with Flow Chart	21
The Basics.....	21
Conditions.....	21
Operations.....	22
Actions.....	22
Clean Empty Rungs.....	23
Add Empty Rungs.....	24
Delete All Rungs.....	25
Timers.....	26
Counters.....	26
Chapter 4 – Getting Familiar with the ARGEE PRO	27
The Basics.....	27
Conditions.....	27
Actions.....	28
Embedded Webserver.....	29
Program Variables.....	30
<i>PLC_connected & PROG_cycle_time</i>	31
<i>Variable Name</i>	31
<i>Delete</i>	32
<i>Add Above</i>	33
<i>Init (Initialize)</i>	34

Integer.....	35
Timer/Counter.....	35
State.....	36
Retain Integer.....	36
Add Variable.....	37
PLC Variables.....	38
Direction.....	39
Word Index.....	39
Bit Offset.....	39
Size.....	40
Signed.....	40
State Names.....	41
Add State.....	42
Keyboard Shortcuts.....	43
Program Variables (Control + Q).....	44
I/O Variables (Control + I).....	44
Operations (Control + F).....	45
State Names (Control + S).....	45
Chapter 5: Conditions & Actions.....	46
Conditions.....	46
Actions.....	47
Assignment.....	47
Timer Start.....	48
Coil.....	49
Timer On.....	50
Timer Off.....	51
Trace.....	52
Comment.....	54
Count Up.....	55
Count Down.....	56
Reset Counter.....	57
Chapter 6 - Operations.....	58
Math.....	58
Addition.....	58
Subtraction.....	58
Multiplication.....	59
Division.....	59
Modulo.....	60
Absolute Value.....	60
Minimum Value.....	61
Maximum Value.....	62
Brackets.....	63
Boolean AND.....	63
Boolean OR.....	64
Boolean NOT.....	64
Greater Than.....	65
Greater Than or Equal to.....	65
Less Than.....	66
Less Than or Equal to.....	66
Equal.....	67
Not Equal.....	67
If_Then_Else.....	68
Change of State.....	69
Count.....	71
Expired.....	72
Chapter 7 - ARGEE Simulation Mode.....	73
Opening the Environment.....	73
Logging into Simulation Mode.....	73
Select Device to Simulate.....	74
Welcome to ARGEE Simulation Mode.....	74

Chapter 8 - ARGEE Security	76
General Security	76
<i>Visual Behavior</i>	76
<i>Connection Behavior</i>	76
Password Protection – ARGEE Environment.....	77
Source Code Protection – Run Without Source	79
Chapter 9 – System Performance	81
Scan Cycle Information	81
IO Variable Formats	81
How Actions Respond to Conditions	82
Defining Variable Types – (Advanced Definitions)	83
Chapter 10 - Common Applications	84
Communicating with an EtherNet/IP Master – Allen Bradley	84
Communicating with a Modbus TCP/IP Master – Red Lion	86
Communicating with a PROFINET Master – Siemens	89
Using State Variables	92
ARGEE HMI	95
<i>General Buttons</i>	95
<i>Editable Fields</i>	97
<i>Display Fields</i>	106
Working with IO-Link	115
Working with RFID.....	117
Working with Analog.....	125
Appendix.....	126
I/O Variable Definitions.....	126
<i>Slot “0” Diagnostics Definitions</i>	126
<i>Slot 1 or 2 Input Definitions</i>	126
<i>Diagnostics Definitions</i>	126
<i>Slot 1 or 2 Output Definitions</i>	127

Preface

Read this preface to familiarize yourself with the rest of the manual. It provides answers to the following questions:

- Why use ARGEE?
- What are ARGEE's advantages and limitations?
- Who should use this manual?
- What is the purpose of this manual?
- What content is in the ARGEE reference manual?

Why use ARGEE?

Imagine that a customer is trying to solve a simple application. This customer does not need a PLC, but they do need some logic. ARGEE was created specifically to solve this problem.

What are ARGEE's advantages and limitations?

ARGEE advantages

- ARGEE stands alone
 - Standalone application (No PLC needed to perform logic)
- ARGEE backs up the PLC
 - PLC back-up (If the application loses communication with the PLC, ARGEE can take over and safely shut down the process)
- ARGEE and the PLC work together
 - Local Control (ARGEE can monitor an application and send updates back to the PLC)

ARGEE limitations

- One ARGEE block cannot control another ARGEE block
- ARGEE is not suited for motion applications

Who should use this manual?

Use this manual if you are responsible for designing, installing, programming or trouble shooting a Turck multiprotocol block that is using the ARGEE programmable functionality.

You should have a basic understanding of networking knowledge, Boolean algebra, and ladder logic. If you do not possess these skills, contact your local Turck representative for proper training before using ARGEE.

What is the purpose of this manual?

This manual is a reference guide for the ARGEE Programming Environment. This manual:

- Teaches the user how to use the ARGEE Flow Chart
- Teaches the user about syntax in ARGEE PRO
- Provides code for common applications
- Defines all the tag names associated with Turck I/O cards

What content is in this manual?

Chapter	Title	Content
	Preface	Overview of the ARGEE Manual content
1	Logging into ARGEE	How to access the ARGEE Environment
2	ARGEE Menu Bar	An explanation of the ARGEE Menu Bar
3	Getting Familiar with Flow Chart	A general overview and walkthrough of the ARGEE Flow Chart
4	Getting Familiar with ARGEE PRO	A general overview and walkthrough of ARGEE PRO
5	Conditions & Actions	A detailed explanation of the ARGEE Condition & Action statements
6	Operations	A detailed explanation of the Operations offered in ARGEE PRO
7	ARGEE Simulation Mode	A general overview and walkthrough of the ARGEE Simulation Mode
8	ARGEE Security	A detailed explanation about ARGEE Security
9	System Performance	A general overview of ARGEE system behavior
10	Common Application	Sample code for many common applications
	Appendix	Defining I/O Variable Names

Chapter 1: Logging into ARGEE

Opening the Environment

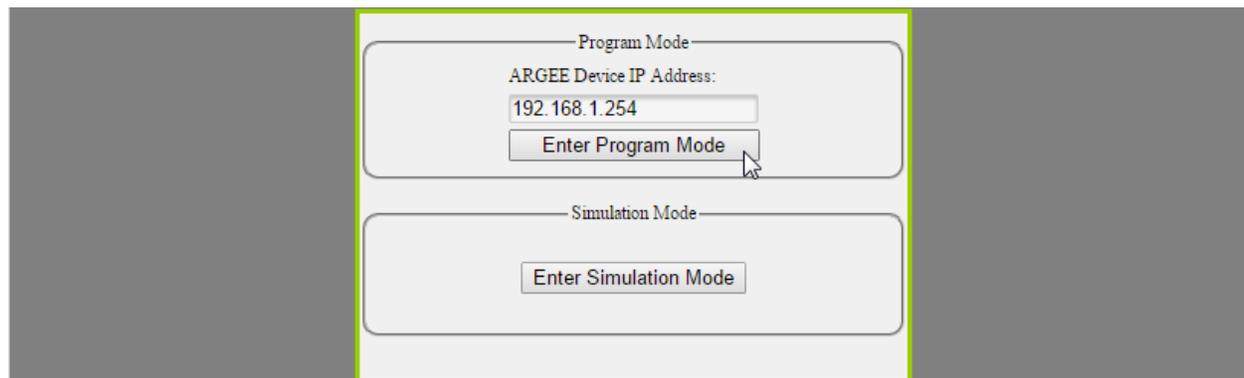
Open the ARGEE Environment and double click on pg.html.



NOTE: ARGEE only opens up in HTML 5 compliant web browsers such as Google Chrome or Firefox.

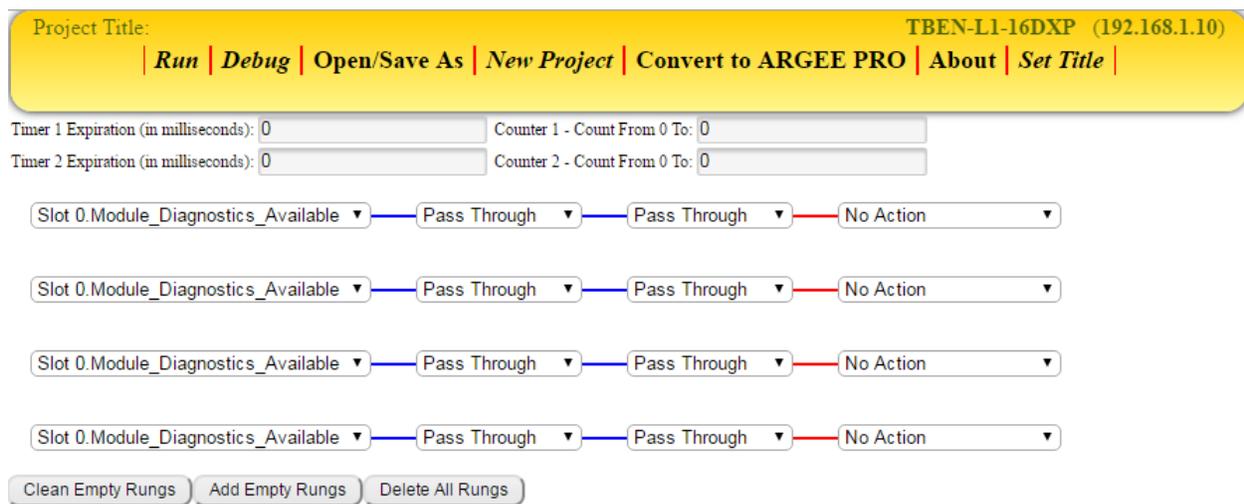
Logging into Program Mode

Type your devices IP Address into the ARGEE Device IP Address text box, and then click *Enter Program Mode*.



NOTE: Simulation Mode is explained on page 73.

Welcome to Flow Chart



Chapter 2: ARGEE Menu Bar

Run

When the user clicks *Run*, several things happen. First, ARGEE checks the code for errors. If the code has no errors, ARGEE downloads the code to the block. It also calculates and displays how much memory the code has used and how much memory is still available. Next, ARGEE transitions over to the Debug screen.



If the code has errors, ARGEE will display an error message and tell the user where the error is located in the code.

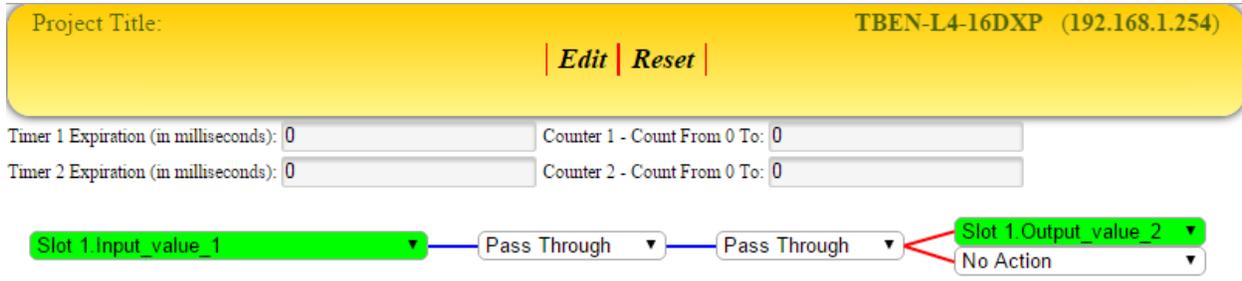


Debug

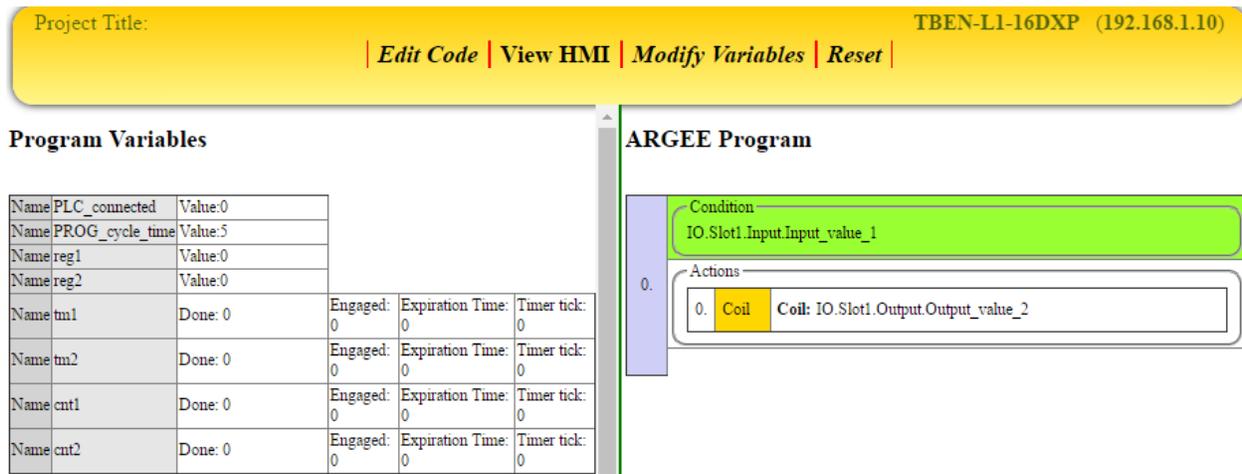
When the user clicks *Debug*, different things happen depending on whether the user is in Flow Chart or ARGEE PRO.



If the user clicks *Debug* while in Flow Chart, the first thing the user will notice is that the Flow Chart will enter *Debug* mode. As conditions become true, the user can visually observe code progression.



When the user clicks *Debug* while in the ARGEE PRO, the user can visually observe code progression from a more advanced screen.



Reset

The user can view the *Reset* button while in *Debug* mode or while viewing an HMI screen. Reset sets the program's timers and counters to zero. If the user clicks Reset while in Flow Chart, the user can visually observe the timers and counters being reset.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Edit* | **Reset** |

Timer 1 Expiration (in milliseconds): 0 Counter 1 - Count From 0 To: 10

Timer 2 Expiration (in milliseconds): 0 Counter 2 - Count From 0 To: 0

Slot 1.Input_value_1 → Pass Through → Pass Through → CTU Counter 1 : 5
No Action

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Edit* | **Reset** |

Timer 1 Expiration (in milliseconds): 0 Counter 1 - Count From 0 To: 10

Timer 2 Expiration (in milliseconds): 0 Counter 2 - Count From 0 To: 0

Slot 1.Input_value_1 → Pass Through → Pass Through → CTU Counter 1 : 0
No Action

When the user clicks *Reset* while in the ARGEE PRO, the user can visually observe timers and counters resetting to zero from a more advanced screen.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Edit Code* | *View HMI* | *Modify Variables* | **Reset** |

Program Variables

Name:PLC_connected	Value:0			
Name:PROG_cycle_time	Value:5			
Name:reg1	Value:0			
Name:reg2	Value:0			
Name:tm1	Done: 0	Engaged: 0	Expiration Time: 0	Timer tick: 0
Name:tm2	Done: 0	Engaged: 0	Expiration Time: 0	Timer tick: 0
Name:cnt1	Done: 0	Count Preset: 10	Current Count: 5	

ARGEE Program

Condition: IO.Slot1.InputInput_value_1

Actions:

0. Count Up Counter: cnt1 Preset: 10

↓

Name:PLC_connected	Value:0			
Name:PROG_cycle_time	Value:5			
Name:reg1	Value:0			
Name:reg2	Value:0			
Name:tm1	Done: 0	Engaged: 0	Expiration Time: 0	Timer tick: 0
Name:tm2	Done: 0	Engaged: 0	Expiration Time: 0	Timer tick: 0
Name:cnt1	Done: 0	Count Preset: 10	Current Count: 0	

ARGEE Program

Condition: IO.Slot1.InputInput_value_1

Actions:

0. Count Up Counter: cnt1 Preset: 10

Open/Save As

The Open/Save feature allows the user to save a current project or load a previous project. The user accesses the Open/Save As feature from different places depending on if they are in Flow Chart or ARGEE Pro. From Flow Chart, the Open/Save As tab is available in the ARGEE Menu Bar.



While in ARGEE PRO, the user can access the Open/Save As screen by clicking on the Project tab and then selecting the Open/Save As tab.



On the Open/Save As screen, the user can perform several actions:

- The *Import Text Above* button imports the above text into the project
- Under the *Open Project* text, the *Choose Files* button allows the user to browse their computer for a previously saved project. Once the file is selected, ARGEE will automatically load the project
- Under the *Save Project* text, the *Save Project With Source* button allows the user to save their project as an .arg file. The user also has the option to *Save Project Without Source Code*.
- The *Edit* tab brings the user back to ARGEE PRO.



Open Project

Choose Files No file chosen

Save Project

Project Name: Save Project With Source Code Save Project Without Source Code

New Project

The user clicks on New Project to start a new project.

Project Title: TBEN-L1-16DXP (192.168.1.10)
[Run](#) | [Debug](#) | [Open/Save As](#) | **[New Project](#)** | [Convert to ARGEE PRO](#) | [About](#) | [Set Title](#) |
Project Checksum:401DA

Erase Project

Starting a new project does not erase the code on your block. If the user wants to erase the code on the block, they need to first start a *New Project* and then click *Run*. This action will load an empty project the block.

Project Title: TBEN-L1-16DXP (192.168.1.10)
[Run](#) | [Debug](#) | [Open/Save As](#) | [New Project](#) | [Convert to ARGEE PRO](#) | [About](#) | [Set Title](#) |
Empty project loaded - Boot project stopped!!!! Project Checksum:401DA

The user can also remove the ARGEE code by selecting *Erase ARGEE Program* or *Reset to Factory Defaults* inside the webserver page.

TBEN-L1-16DXP
Embedded Website of TBEN Block I/O Module

TURCK
Industrial Automation

admin@192.168.1.99 [Logout]

Station Configuration >

- Station Information
- ! Station Diagnostics
- Event Log
- Ethernet Statistics
- EtherNet/IP™ Memory Map
- Modbus TCP Memory Map
- Links
- Station Configuration**
- Network Configuration
- Change Admin Password

16DXP

Protocols

- Deactivate EtherNet/IP™
- Deactivate Modbus TCP
- Deactivate PROFINET
- Deactivate Web Server

EtherNet/IP™ Configuration

- Activate GW Control Word
- Activate GW Status Word
- Activate Scheduled Diagnostics
- Activate Summarized Diagnostics
- Activate Quick Connect

PROFINET Configuration

PROFINET Station Name

Modbus Configuration

NOTE: To disable the watchdog timer, enter 0. Also, the value is in millisecond (ms).

Watchdog Timer

Note: Getting to the webserver is discussed on page 29.

Convert to ARGEE PRO

The user will click *Convert to ARGEE PRO* when they want to leave the Flow Chart mode and enter the ARGEE PRO Programming Environment.



NOTE: Once the user selects *Convert to ARGEE PRO*, they cannot convert back to Flow Chart.

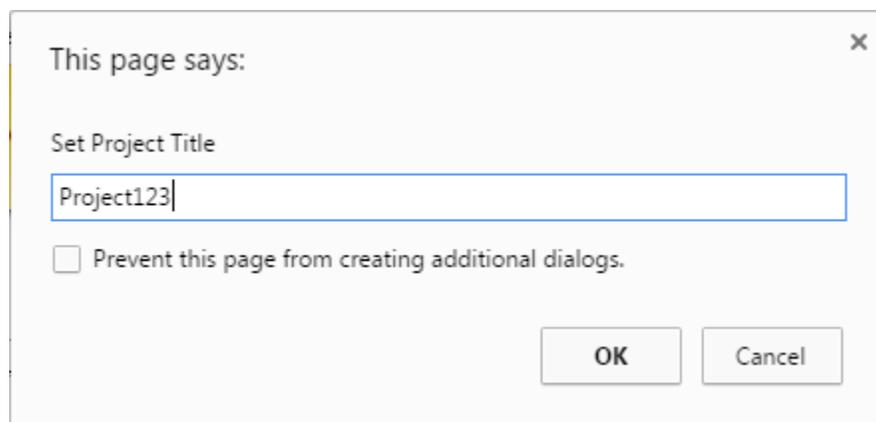
About

The user can click *About* if they want to view the ARGEE environment and kernel firmware revisions.



Set Title

The user can click *Set Title* to add a name to the project.



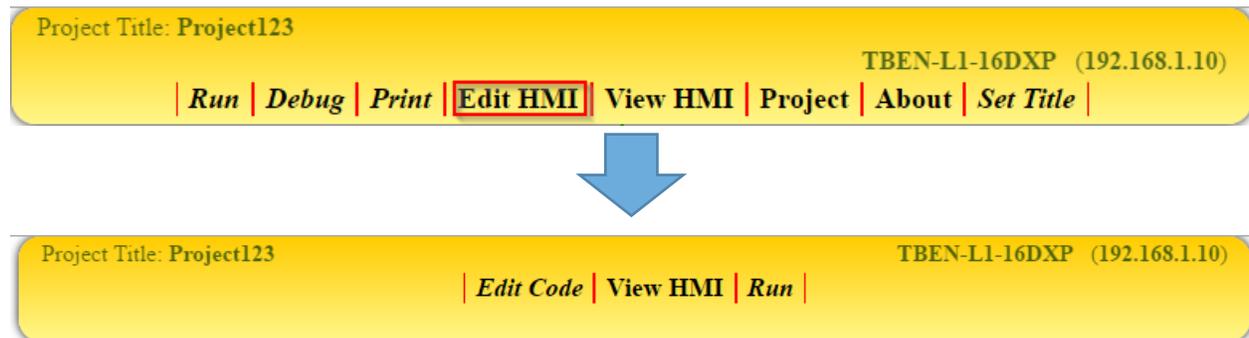
Print

The Print option is available in the ARGEE PRO menu bar. The user can click *Print* if they want to print out a copy of their project.



Edit HMI

The *Edit HMI* tab brings the user to the edit HMI screen.



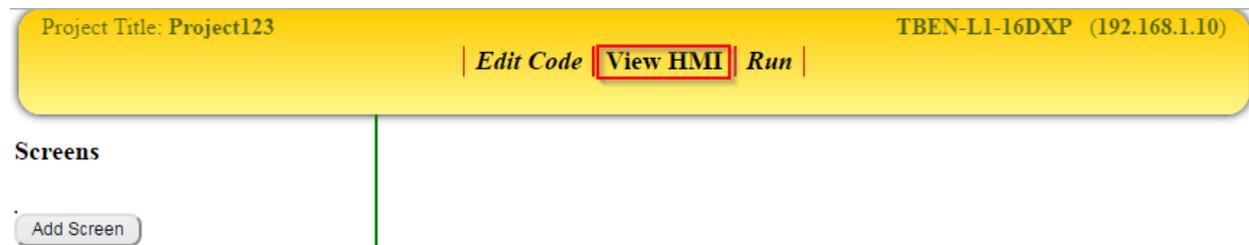
Screens



NOTE: Instructions for how to build an HMI are located on page 94.

View HMI

The *View HMI* tab allows the user to view their HMI screen. This tab becomes active after the user has already built an HMI.



Screens



Edit Code

When the user clicks on the *Edit Code* tab they will return to the main ARGEE PRO page.

Project Title: Project 123 TBEN-L4-16DXP (192.168.1.254)

Edit Code | View HMI | Run |

Project Checksum:160D9

Screens

Add Screen

Project

When the user clicks on the *Project* tab, they will have access to a second ARGEE Menu Bar.

Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)

Run | Debug | Print | Edit HMI | View HMI | **Project** | About | Set Title |



Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)

Edit Code | Open/Save As | New Project | Settings | Run Without Source |

Settings

From the *Settings* screen, the user can set the “percentage” of their screen that displays their Variables and State Names. The remaining “percentage” of the screen displays the ARGEE Program.



Settings

left column width in %
 (for tablet and Computer interfaces only):

TextArea Height :



Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)
 | Run | Debug | Print | Edit HMI | View HMI | Project | About | Set Title |

Program Variables

Name	Type	Actions		
PLC_connected	Integer			
PROC_cycle_time	Integer			
reg1	Integer	Delete	Add Above	Init
reg2	Integer	Delete	Add Above	Init
tm1	Timer/Counter	Delete	Add Above	
tm2	Timer/Counter	Delete	Add Above	
cnt1	Timer/Counter	Delete	Add Above	
cnt2	Timer/Counter	Delete	Add Above	

ARGEE Program

Keyboard shortcuts:
 Press Ctrl-g for list of program variables
 Press Ctrl-i for list of IO variables
 Press Ctrl-f for list of operations
 Press Ctrl-s for list of State Names
 These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition:
 IO.Slot1.Input.Input_value_1

Action:
 0. Cod: IO.Slot1.Output.Output_value_1

Assignment: Add Action

Add Condition

35 percent width

Name	Direction	Word index	Bit offset	Size	Signed
plc_in_reg1	ARGEE->PLC	0	0	Word (16 bit)	unsigned
plc_in_reg2	ARGEE->PLC	1	0	Word (16 bit)	unsigned
plc_out_reg1	PLC->ARGEE	0	0	Word (16 bit)	unsigned
plc_out_reg2	PLC->ARGEE	1	0	Word (16 bit)	unsigned

Add Variable

Run Without Source

Selecting *Run Without Source* will allow the user to run a program without displaying the actual code. The end user will not be able to access source code by loading the ARGEE environment. *Run Without Source* is one of ARGEE’s security protocols.



Very Important Note: The user needs to save a Master Copy of the program before the user logs out of the environment if the user wants to view/edit the code in the future. Security protocols are discussed in Chapter 8 - ARGEE Security.

Modify Variables

The *Modify Variables* tab is available in ARGEE PRO and in the ARGEE Simulation Mode. It only becomes visible when the user is in Debug mode. From the *Modify Variables* screen, the user can manually change register and variable values.

Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | **[Modify Variables](#)** | [Reset](#) |

Code loaded into the station: Loadable size: 120 bytes (out of 6144 bytes). Total Project size:1411 bytes(out of 262144 bytes). Project Checksum:189BB



Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)

| **[Finish Modifications](#)** |

Code loaded into the station: Loadable size: 120 bytes (out of 6144 bytes). Total Project size:1411 bytes(out of 262144 bytes). Project Checksum:189BB

Program Variables

Name	PLC_connected	Value:0
Name	PROG_cycle_time	Value:5
Name	reg1	Value:0
Name	reg2	Value:0
Name	tm1	Done: 0
		Engaged: 0
		Expiration Time: 0
		Timer tick: 0
Name	tm2	Done: 0
		Engaged: 0
		Expiration Time: 0
		Timer tick: 0

ARGEE Program

Condition
IO.Slot1.Input.Input_value_1

Actions

0. **Coil** Coil: IO.Slot1.Output.Output_value_1



Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)

| **[Finish Modifications](#)** |

Code loaded into the station: Loadable size: 120 bytes (out of 6144 bytes). Total Project size:1411 bytes(out of 262144 bytes). Project Checksum:189BB

Program Variables

Name	PLC_connected	Value:0
Name	PROG_cycle_time	Value:5
Name	reg1	Value: <input style="border: 1px solid red;" type="text" value="1"/>
Name	reg2	Value:0
Name	tm1	Done: 0
		Engaged: 0
		Expiration Time: 0
		Timer tick: 0
Name	tm2	Done: 0
		Engaged: 0
		Expiration Time: 0
		Timer tick: 0

ARGEE Program

Condition
IO.Slot1.Input.Input_value_1

Actions

0. **Coil** Coil: IO.Slot1.Output.Output_value_1

Finish Modifications

The user can select *Finish Modification* to exit *Modify Variables* mode.

Project Title: Project123 TBEN-L1-16DXP (192.168.1.10)

[Finish Modifications](#)

Code loaded into the station: Loadable size: 120 bytes (out of 6144 bytes). Total Project size:1411 bytes(out of 262144 bytes). Project Checksum:189BB

Show Trace, Stop Trace, Resume Trace

The user will use Trace when they want to measure the run-time behavior of the program. Trace allows the user to measure how long each state takes as well as which states were visited in which order. Trace is an Action that must be inserted into the code before Trace will appear in the menu bar.

The user can click on Show Trace to view the active Trace data.



The user can click on Stop Trace to easily view the programs historical run-time data.



The user can click on Resume Trace to resume tracing the programs run-time.



NOTE: More information about Trace can be found on page 52.

Show Variables

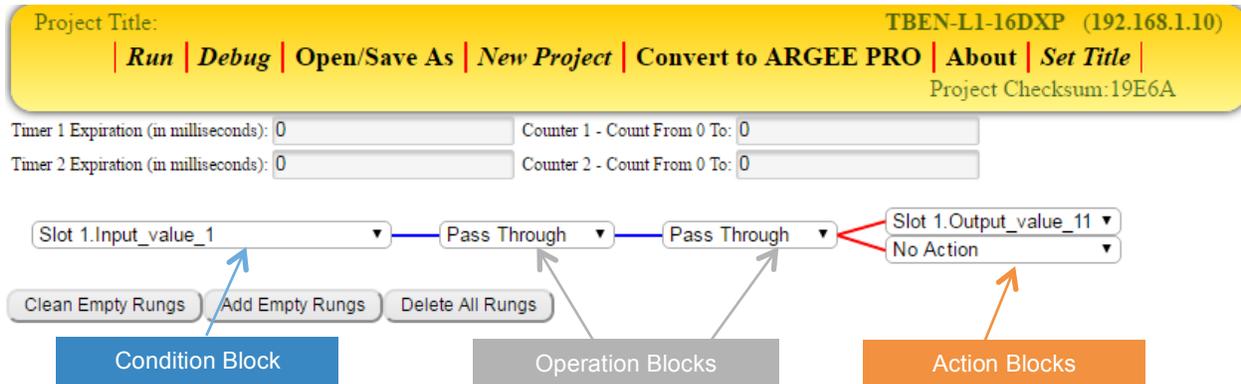
The user can click on *Show Variables* if the user wants to leave Trace Mode.



Chapter 3: Getting Familiar with Flow Chart

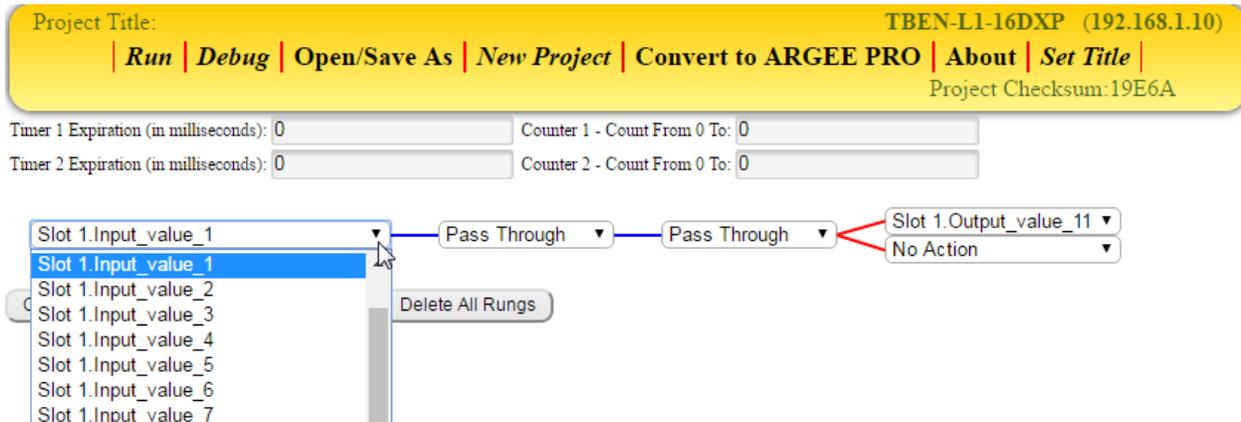
The Basics

The Flow Chart Editor is made up of Condition, Operation, and Action Blocks. Conditions, Operations and Actions are selected by clicking their respective drop down arrows. The Flow Chart Editor also provides the user with two timers and two counters.



Conditions

The Condition Block contains Input conditions. The input conditions the user sees corresponds to the block the user is connected to. Other included input conditions are: Timer X expired, Counter X expired, Internal Reg X, and PLC In Reg X.



NOTE: Expired functions are discussed on page 72. Internal Reg's are discussed on page 30 (Reg = Register). PLC In Reg's are discussed on page 38.

Operations

The Operation Blocks contain various Boolean operations. If no Operations are desired, select Pass Through.

Project Title: **TBEN-L1-16DXP (192.168.1.10)**
Project Checksum: 19E6A

Run | Debug | Open/Save As | New Project | Convert to ARGEE PRO | About | Set Title

Timer 1 Expiration (in milliseconds): 0 Counter 1 - Count From 0 To: 0
Timer 2 Expiration (in milliseconds): 0 Counter 2 - Count From 0 To: 0

Slot 1.Input_value_1 → Pass Through → Pass Through → Slot 1.Output_value_11
No Action

Clean Empty Rungs Add Empty Rungs Delete

Pass Through
Pass Through
AND
OR
NOT
AND of 3 Inputs

NOTE: Boolean Logic is discussed on pages 63 and 64.

Actions

The Action Block contains Output conditions, The Output conditions the user sees corresponds to the block the user is connected to. Other included Output conditions are: TON Timer X (**T**urn **O**N Timer X), CTU Counter X (**C**oun**T** **U**p Counter X), RESET Counter X, Internal Reg X, PLC Out Reg X.

Project Title: **TBEN-L1-16DXP (192.168.1.10)**
Project Checksum: 19E6A

Run | Debug | Open/Save As | New Project | Convert to ARGEE PRO | About | Set Title

Timer 1 Expiration (in milliseconds): 0 Counter 1 - Count From 0 To: 0
Timer 2 Expiration (in milliseconds): 0 Counter 2 - Count From 0 To: 0

Slot 1.Input_value_1 → Pass Through → Pass Through → Slot 1.Output_value_11
Slot 1.Output_value_7
Slot 1.Output_value_8
Slot 1.Output_value_9
Slot 1.Output_value_10
Slot 1.Output_value_11
Slot 1.Output_value_12
Slot 1.Output_value_13

Clean Empty Rungs Add Empty Rungs Delete All Rungs

NOTE: TON is discussed on page 50. CTU is discussed on page 55. RESET is discussed on page 57. Internal Reg's are discussed on page 30 (Reg = Register). PLC Reg's are discussed on page 38.

Clean Empty Rungs

The *Clean Empty Rungs* button will remove all unused rungs from the Flow Chart Editor.

The image shows two screenshots of the Flow Chart Editor interface, illustrating the effect of the 'Clean Empty Rungs' button. The interface includes a project title bar, navigation menu, timer and counter settings, and a ladder logic diagram.

Initial State (Top Screenshot):

- Project Title: TBEN-L1-16DXP (192.168.1.10)
- Navigation: Run | Debug | Open/Save As | New Project | Convert to ARGEE PRO | About | Set Title
- Project Checksum: 19E6A
- Timer 1 Expiration (in milliseconds): 0
- Counter 1 - Count From 0 To: 0
- Timer 2 Expiration (in milliseconds): 0
- Counter 2 - Count From 0 To: 0
- Ladder Logic:
 - Rung 1: Slot 1.Input_value_1 (NO) -> Pass Through (NC) -> Pass Through (NO) -> Slot 1.Output_value_11 (NO)
 - Rung 2: Slot 0.Module_Diagnostics_Available (NO) -> Pass Through (NC) -> Pass Through (NO) -> No Action (NO)
- Buttons: Clean Empty Rungs (highlighted with a red box), Add Empty Rungs, Delete All Rungs

Final State (Bottom Screenshot):

- Project Title: TBEN-L1-16DXP (192.168.1.10)
- Navigation: Run | Debug | Open/Save As | New Project | Convert to ARGEE PRO | About | Set Title
- Project Checksum: 19E6A
- Timer 1 Expiration (in milliseconds): 0
- Counter 1 - Count From 0 To: 0
- Timer 2 Expiration (in milliseconds): 0
- Counter 2 - Count From 0 To: 0
- Ladder Logic:
 - Rung 1: Slot 1.Input_value_1 (NO) -> Pass Through (NC) -> Pass Through (NO) -> Slot 1.Output_value_11 (NO)
 - Rung 2: No Action (NO)
- Buttons: Clean Empty Rungs, Add Empty Rungs, Delete All Rungs

A large blue arrow points from the initial state to the final state, indicating the result of clicking the 'Clean Empty Rungs' button. The 'Clean Empty Rungs' button is highlighted with a red box in both screenshots.

Add Empty Rungs

The *Add Empty Rungs* button will add four empty rungs to Flow Chart Editor.

Project Title: TBEN-L1-16DXP (192.168.1.10)
Run | Debug | Open/Save As | New Project | Convert to ARGEE PRO | About | Set Title
Project Checksum: 19E6A

Timer 1 Expiration (in milliseconds): Counter 1 - Count From 0 To:
Timer 2 Expiration (in milliseconds): Counter 2 - Count From 0 To:

Slot 1.Input_value_1 — Pass Through — Pass Through — Slot 1.Output_value_11
No Action

Clean Empty Rungs **Add Empty Rungs** Delete All Rungs



Project Title: TBEN-L1-16DXP (192.168.1.10)
Run | Debug | Open/Save As | New Project | Convert to ARGEE PRO | About | Set Title
Project Checksum: 19E6A

Timer 1 Expiration (in milliseconds): Counter 1 - Count From 0 To:
Timer 2 Expiration (in milliseconds): Counter 2 - Count From 0 To:

Slot 1.Input_value_1 — Pass Through — Pass Through — Slot 1.Output_value_11
No Action

Slot 0.Module_Diagnostics_Available — Pass Through — Pass Through — No Action

Slot 0.Module_Diagnostics_Available — Pass Through — Pass Through — No Action

Slot 0.Module_Diagnostics_Available — Pass Through — Pass Through — No Action

Slot 0.Module_Diagnostics_Available — Pass Through — Pass Through — No Action

Clean Empty Rungs **Add Empty Rungs** Delete All Rungs

Delete All Rungs

The *Delete All Rungs* button will remove all rungs from Flow Chart Editor.

The screenshot shows the Flow Chart Editor interface for project TBEN-L1-16DXP (192.168.1.10). The top menu bar includes options: Run, Debug, Open/Save As, New Project, Convert to ARGEE PRO, About, and Set Title. The Project Checksum is 19E6A. Below the menu, there are timer and counter settings for Timer 1 and Timer 2, both set to 0 milliseconds and 0 counts. The main workspace contains five rungs. Each rung consists of an input slot, two 'Pass Through' action slots, and an output slot. The first rung has Slot 1 as input and Slot 1 as output. The other four rungs have Slot 0 as input and No Action as output. At the bottom, there are three buttons: Clean Empty Rungs, Add Empty Rungs, and Delete All Rungs. The Delete All Rungs button is highlighted with a red border. A large blue arrow points downwards from this button to the second screenshot, which shows the same interface but with all rungs removed, leaving only the buttons and settings area.

Note: Used and unused rungs will both be deleted from the project.

Timers

Flow Chart Editor contains two *Timers*. The user can set the Timers by typing a value into the Timer text box. Timer values are in milliseconds (1000 Milliseconds = 1 Second).

Project Title: **TBEN-L1-16DXP (192.168.1.10)**
| **Run** | **Debug** | **Open/Save As** | **New Project** | **Convert to ARGEE PRO** | **About** | **Set Title** |
Project Checksum: 19E6A

Timer 1 Expiration (in milliseconds): 0 Counter 1 - Count From 0 To: 0
Timer 2 Expiration (in milliseconds): 0 Counter 2 - Count From 0 To: 0

Slot 1.Input_value_1 Pass Through Pass Through Slot 1.Output_value_11
No Action

Clean Empty Rungs Add Empty Rungs Delete All Rungs

NOTE: Timer examples can be seen on page 48, 50 and 51.

Counters

Flow Chart Editor contains two *Counters*. The user can set the Counters by typing a value into the Counter text box.

Project Title: **TBEN-L1-16DXP (192.168.1.10)**
| **Run** | **Debug** | **Open/Save As** | **New Project** | **Convert to ARGEE PRO** | **About** | **Set Title** |
Project Checksum: 19E6A

Timer 1 Expiration (in milliseconds): 0 Counter 1 - Count From 0 To: 0
Timer 2 Expiration (in milliseconds): 0 Counter 2 - Count From 0 To: 0

Slot 1.Input_value_1 Pass Through Pass Through Slot 1.Output_value_11
No Action

Clean Empty Rungs Add Empty Rungs Delete All Rungs

NOTE: Counter examples can be seen on page 55, 56 and 57.

Chapter 4 – Getting Familiar with the ARGEE PRO

The Basics

The ARGEE PRO home page is made up of Conditions & Actions, An Embedded Webserver Link, Variables & State Names, and Keyboard Shortcuts.

The screenshot shows the ARGEE PRO interface with several callouts:

- Keyboard Shortcuts:** A green callout box pointing to the 'Keyboard shortcuts' section in the ARGEE Program area.
- Embedded Webserver Link:** A purple callout box pointing to the 'In order to configure the IO of the station, follow the Link' text.
- Conditions & Actions:** A blue callout box pointing to the 'Add Condition' button.
- Variables & State Names:** An orange callout box pointing to the 'Program Variables', 'PLC Variables', and 'State Names' sections.

Conditions

The *Add Condition* button will add one blank condition to the ARGEE project. This environment is executed in the manner of IF / THEN statements. ARGEE calls them Conditions (IF) and Actions (THEN).

This screenshot shows the same ARGEE PRO interface, but with the 'Add Condition' button highlighted in red. The 'ARGEE Program' section now displays a 'Condition' input field and an 'Add Action' button, indicating that a new condition has been added to the project.

Actions

Actions are selected from a pull down menu. Users select the desired action, and then select the *Add Action* button.

Project Title: TBEN-L1-16DXP (192.168.1.10)

[Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#)

Project Checksum:19E6A

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer	Delete Add Above Init
reg2	Integer	Delete Add Above Init
tm1	Timer/Counter	Delete Add Above
tm2	Timer/Counter	Delete Add Above
cnt1	Timer/Counter	Delete Add Above
cnt2	Timer/Counter	Delete Add Above

Add Variable

PLC Variables

Name	Direction	Word index	Bit offset	Size
plc_in_reg1	ARGEE->PLC	0	0	Word (16 bit)
plc_in_reg2	ARGEE->PLC	1	0	Word (16 bit)
plc_out_reg1	PLC->ARGEE	0	0	Word (16 bit)
plc_out_reg2	PLC->ARGEE	1	0	Word (16 bit)

Add Variable

State Names

Add State

ARGEE Program

Keyboard shortcuts:
 Press Ctrl-q for list of program variables
 Press Ctrl-i for list of I/O variables
 Press Ctrl-f for list of operations
 Press Ctrl-s for list of State Names
 These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition:

0.

Assignment

Assignment

- Timer start
- Coil
- Timer On
- Timer Off
- Trace
- Comment
- Count Up
- Count Down
- Reset Counter

Project Title: TBEN-L1-16DXP (192.168.1.10)

[Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#)

Project Checksum:19E6A

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer	Delete Add Above Init
reg2	Integer	Delete Add Above Init
tm1	Timer/Counter	Delete Add Above
tm2	Timer/Counter	Delete Add Above
cnt1	Timer/Counter	Delete Add Above
cnt2	Timer/Counter	Delete Add Above

Add Variable

PLC Variables

Name	Direction	Word index	Bit offset	Size
plc_in_reg1	ARGEE->PLC	0	0	Word (16 bit)
plc_in_reg2	ARGEE->PLC	1	0	Word (16 bit)
plc_out_reg1	PLC->ARGEE	0	0	Word (16 bit)
plc_out_reg2	PLC->ARGEE	1	0	Word (16 bit)

Add Variable

State Names

Add State

ARGEE Program

Keyboard shortcuts:
 Press Ctrl-q for list of program variables
 Press Ctrl-i for list of I/O variables
 Press Ctrl-f for list of operations
 Press Ctrl-s for list of State Names
 These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition:

0.

Actions:

0. Assignment Destination: Expression:

Assignment

Add Condition

Note: Some Actions are executed even if the Condition is false. See [Chapter 9 – System Performance](#) for more information in this topic.

28

Turck Inc. | 3000 Campus Drive, Minneapolis, MN 55441 | T +1 800 544 7769 | F +1 763 553 0708 | www.turck.com

Embedded Webserver

The user can click [Link](#) to access the connected blocks webserver. Once the user is in the webserver, they can view the device status and even set device parameters.

Project Title: TBEN-L1-16DXP (192.168.1.10)
Project Checksum: 19E6A

| [Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#) |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer	Delete Add Above Init
reg2	Integer	Delete Add Above Init
tm1	Timer/Counter	Delete Add Above
tm2	Timer/Counter	Delete Add Above
cnt1	Timer/Counter	Delete Add Above
cnt2	Timer/Counter	Delete Add Above

[Add Variable](#)

ARGEE Program

Keyboard shortcuts:
 Press Ctrl-q for list of program variables
 Press Ctrl-i for list of I/O variables
 Press Ctrl-f for list of operations
 Press Ctrl-s for list of State Names
 These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition:

Assignment: Add Action

[Add Condition](#)

PLC Variables

Name	Direction	Word index	Bit offset	Size
plc_in_reg1	ARGEE->PLC	0	0	Word (16 bit)
plc_in_reg2	ARGEE->PLC	1	0	Word (16 bit)
plc_out_reg1	PLC->ARGEE	0	0	Word (16 bit)
plc_out_reg2	PLC->ARGEE	1	0	Word (16 bit)

[Add Variable](#)

State Names

[Add State](#)

TBEN-L1-16DXP
 Embedded Website of TBEN Block I/O Module

TURCK
Industrial Automation

Password [\[Login\]](#)

Station Information >

- Station Information
- ! Station Diagnostics
- Event Log
- Ethernet Statistics
- EtherNet/IP™ Memory Map
- Modbus TCP Memory Map
- Links
- 16DXP

Station Information

Type	TBEN-L1-16DXP
Identification Number	6814008
Firmware Revision	V3.2.5.123
Bootloader Revision	V8.0.1.0
EtherNet/IP™ Revision	V2.7.13.0
PROFINET Revision	V1.4.0.1
Modbus TCP Revision	V2.1.5.0
Addressing Mode	Rotary
PROFINET Station Name	
ARGEE Boot Project	Running
ARGEE Project Title	Project123
ARGEE Factory Programmed	No

NOTE: The default password for the webserver is “password”.

Program Variables

Program Variables can be added, deleted and renamed. The user can also change the variable type by using the drop down arrow.

Project Title:

TBEN-L1-16DXP (192.168.1.10)

| [Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#) |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

PLC Variables

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_in_reg2	ARGEE->PLC ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg2	PLC->ARGEE ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above

Add Variable

State Names

Add State

PLC_connected & PROG_cycle_time

The PLC_connected bit is true when a PLC is connected to the device. The PROG_cycle_time displays the time it takes to execute the entire program.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▾	Delete Add Above Init
reg2	Integer ▾	Delete Add Above Init
tm1	Timer/Counter ▾	Delete Add Above
tm2	Timer/Counter ▾	Delete Add Above
cnt1	Timer/Counter ▾	Delete Add Above
cnt2	Timer/Counter ▾	Delete Add Above

Add Variable

Variable Name

Variable Names are the names of variables in the users program.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▾	Delete Add Above Init
reg2	Integer ▾	Delete Add Above Init
tm1	Timer/Counter ▾	Delete Add Above
tm2	Timer/Counter ▾	Delete Add Above
cnt1	Timer/Counter ▾	Delete Add Above
cnt2	Timer/Counter ▾	Delete Add Above

Add Variable

Delete

The *Delete* button will delete the program variable.

Project Title:

TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

Add Above

The *Add Above* button will add a Program Variable above the selected variable.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable



Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
	Integer ▼	Delete Add Above Init
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

Init (Initialize)

The user will use *Initialize* if they want to pre-set the value in a Program Variable's register.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable



Project Title:

TBEN-L1-16DXP (192.168.1.10)

| *Apply and Back to Code* |

reg1

1

Apply Cancel



Project Title:

TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	init:1 Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

Integer

If the user selects *Integer*, the Program Variable will be stored in 32 bit signed register. This allows the user to store an Integer value between +2,147,483,647 and -2,147,483,647 in the Program Variable's register.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer	Delete Add Above Init
tm1	Timer/Counter	Delete Add Above
tm2	State	Delete Add Above
cnt1	Retain Integer	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

NOTE: Integer values are stored in four 8-bit registers.

Timer/Counter

The user can select *Timer/Counter* if they want to add a Timer or Counter variable to their program.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Integer	Delete Add Above
cnt1	Timer/Counter	Delete Add Above
cnt2	State	Delete Add Above
cnt2	Retain Integer	Delete Add Above

Add Variable

State

The user would select *State* if they wanted to create a State Variable. State Variables are used in State Machines. An example of a State Machine is shown on page 91.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above
	Integer ▼	Delete Add Above Init

Add Variable

- Integer
- Timer/Counter
- State**
- Retain Integer

Retain Integer

The user would use *Retain Integer* if they wanted to save the value in a Program Variable through a power cycle. The value is saved into flash memory once every three minutes if the value has been changed.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above
	Retain Integer ▼	Delete Add Above Init

Add Variable

- Integer
- Timer/Counter
- State
- Retain Integer**

Add Variable

The *Add Variable* button will add a Program Variable to the program.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable



Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above
	Integer ▼	Delete Add Above Init

Add Variable

PLC Variables

PLC Variables are used to define communication between the ARGEE block and the PLC. PLC examples are shown in [Chapter 10 – Common Applications](#).

Project Title:

TBEN-L1-16DXP (192.168.1.10)

| [Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#) |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

PLC Variables

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_in_reg2	ARGEE->PLC ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg2	PLC->ARGEE ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above

Add Variable

State Names

Add State

Direction

The user will use the *Direction* dropdown arrow to assign which direction the data is traveling. *ARGEE->PLC* means the data is traveling from the ARGEE block to the PLC. *PLC->ARGEE* means the data is traveling from the PLC to the ARGEE block.

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_in_reg2	ARGEE->PLC ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg2	PLC->ARGEE ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above

Add Variable

- ARGEE->PLC
- PLC->ARGEE

Word Index

The user will use the *Word index* to assign the data to a specific register in the PLC or the ARGEE block.

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_in_reg2	ARGEE->PLC ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg2	PLC->ARGEE ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above

Add Variable

Bit Offset

The user will use the *Bit offset* to assign the data to a specific bit in a register.

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_in_reg2	ARGEE->PLC ▼	1	0	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	1	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg2	PLC->ARGEE ▼	1	3	Word (16 bit) ▼	unsigned ▼	Delete Add Above

Add Variable

- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

State Names

Add State

Size

The user will use the *Size* drop down to set the size of the data being transferred.

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▾	0	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_in_reg2	ARGEE->PLC ▾	1	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_out_reg1	PLC->ARGEE ▾	0	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_out_reg2	PLC->ARGEE ▾	1	0 ▾	Word (16 bit) ▾ Word (16 bit) Bool (1 bit)	unsigned ▾	Delete Add Above

Add Variable

Signed

The user will use the *Signed* drop down to indicate whether the data being transferred is a signed or unsigned integer.

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▾	0	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_in_reg2	ARGEE->PLC ▾	1	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_out_reg1	PLC->ARGEE ▾	0	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_out_reg2	PLC->ARGEE ▾	1	0 ▾	Word (16 bit) ▾	unsigned ▾ unsigned signed	Delete Add Above

Add Variable

State Names

State Names are used to make it easier to identify which State the users program is in. "State" is the term used to identify a specific program operation at a specific moment.

Project Title:

TBEN-L1-16DXP (192.168.1.10)

| [Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#) |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
reg1	Integer ▼	Delete Add Above Init
reg2	Integer ▼	Delete Add Above Init
tm1	Timer/Counter ▼	Delete Add Above
tm2	Timer/Counter ▼	Delete Add Above
cnt1	Timer/Counter ▼	Delete Add Above
cnt2	Timer/Counter ▼	Delete Add Above

Add Variable

PLC Variables

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_in_reg2	ARGEE->PLC ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg2	PLC->ARGEE ▼	1	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above

Add Variable

State Names

Add State

Add State

The user would click *Add State* if they wanted to add State Names to their program.

State Names



State Names

Name	Actions
<input type="text"/>	<input type="button" value="Delete"/> <input type="button" value="Add Above"/>

Note: An example that uses State Names is shown on page 91 and page 116.

Keyboard Shortcuts

Keyboard shortcuts provide the user with a quick way to access Program Variables, I/O Variables, Operations and State Names.

- Ctrl-q, Program Variables.
- Ctrl-i, I/O Variables.
- Ctrl-f, Operations
- Ctrl-s, State names

Project Title:

TBEN-L1-16DXP (192.168.1.10)

| **Run** | **Debug** | **Print** | **Edit HMI** | **View HMI** | **Project** | **About** | **Set Title** |

ARGEE Program

Keyboard shortcuts:

Press Ctrl-q for list of program variables

Press Ctrl-i for list of I/O variables

Press Ctrl-f for list of operations

Press Ctrl-s for list of State Names

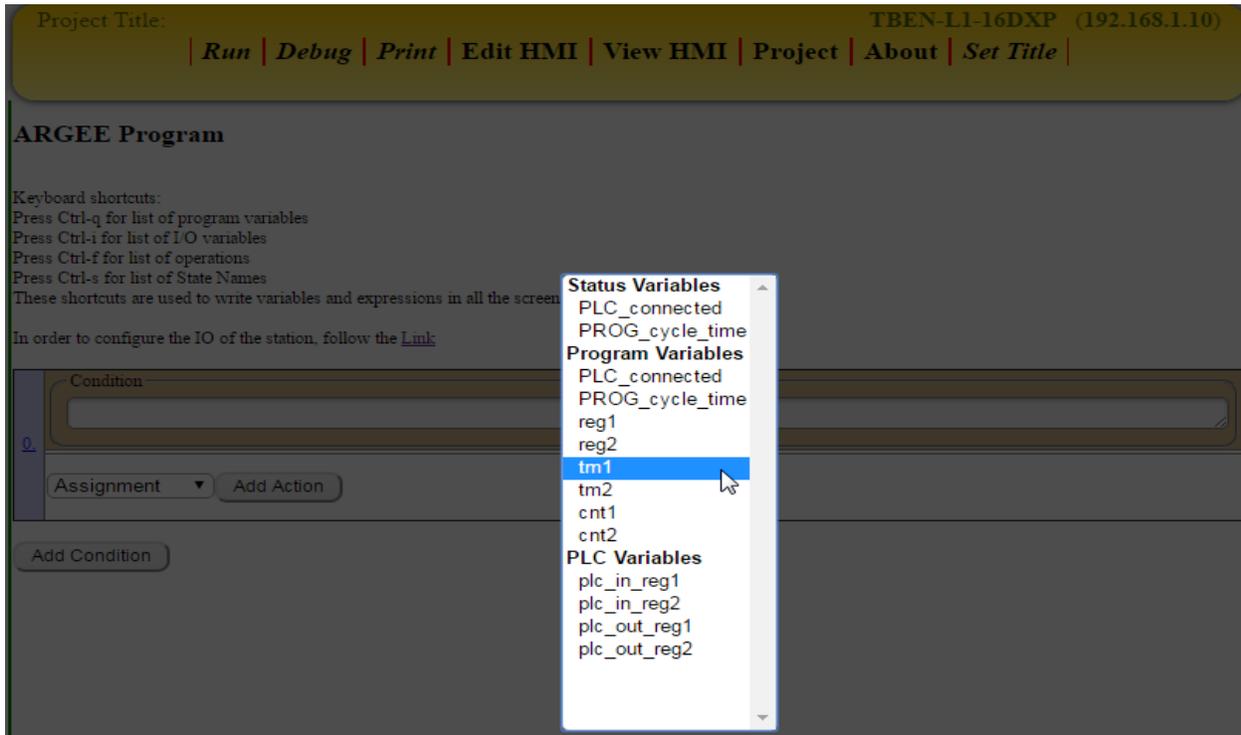
These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

The screenshot shows a configuration window for the ARGEE Program. At the top, there is a "Condition" field with a text input area. Below this, there is a blue vertical bar on the left side with a small "0." label. To the right of the bar, there is an "Assignment" dropdown menu and an "Add Action" button. At the bottom of the window, there is an "Add Condition" button.

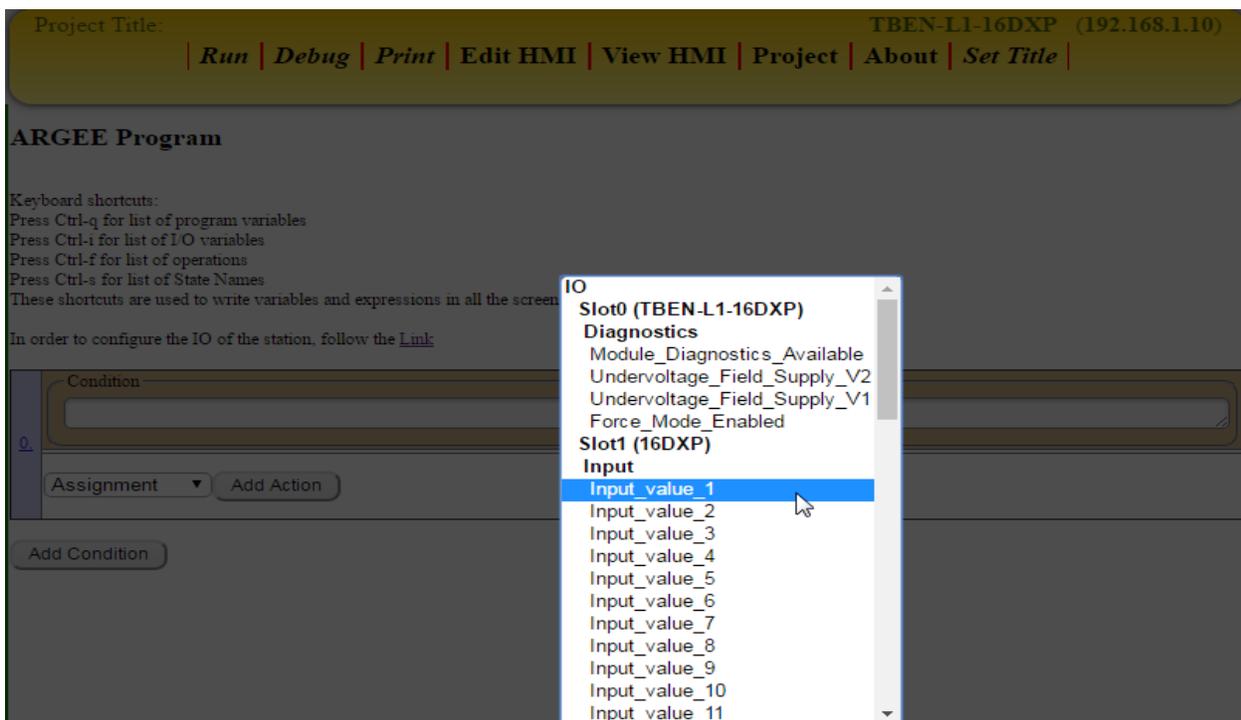
Program Variables (Control + Q)

If the user presses *Ctrl + q* while in a Condition or Action box, the Program Variable List will pop up. The user can select their desired variable and it will be added to their respective Condition or Action box.



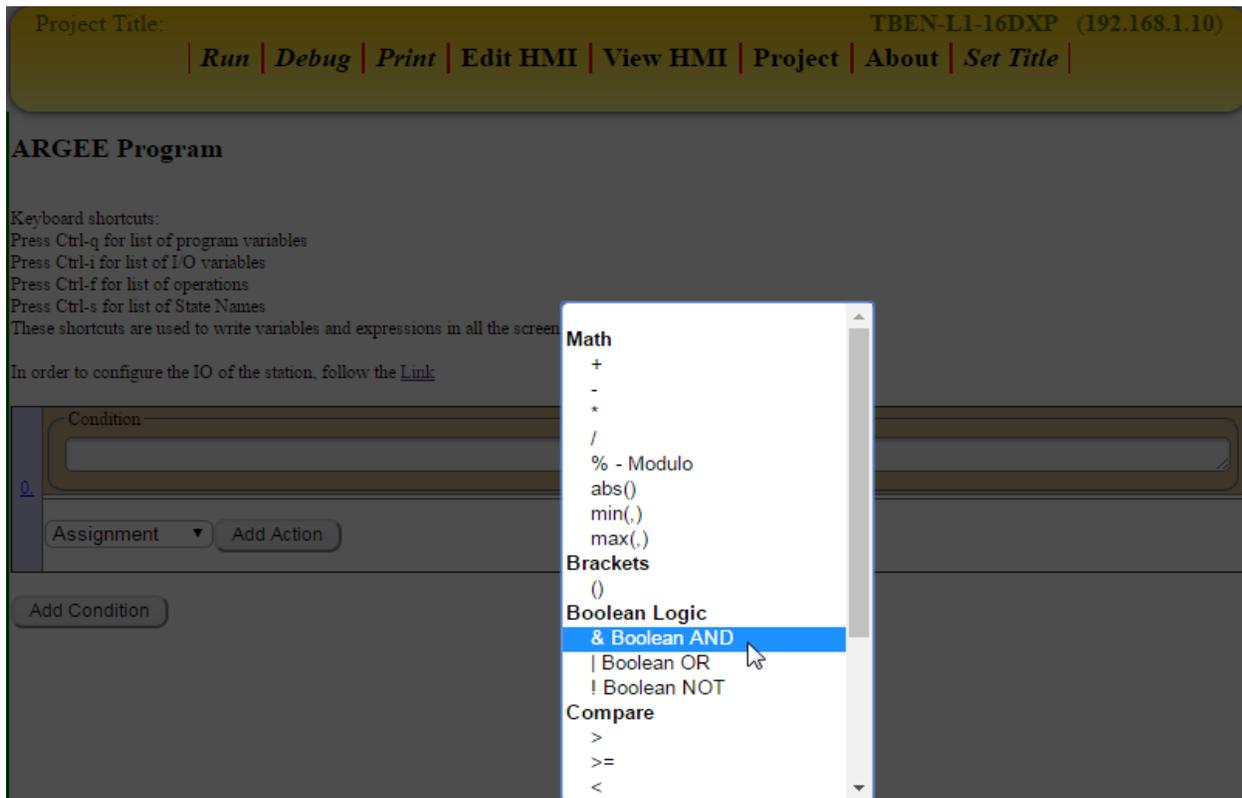
I/O Variables (Control + I)

If the user presses *Ctrl + i* while in a Condition or Action box, the I/O Variable List will pop up. The user can select their desired variable and it will be added to their respective Condition or Action box.



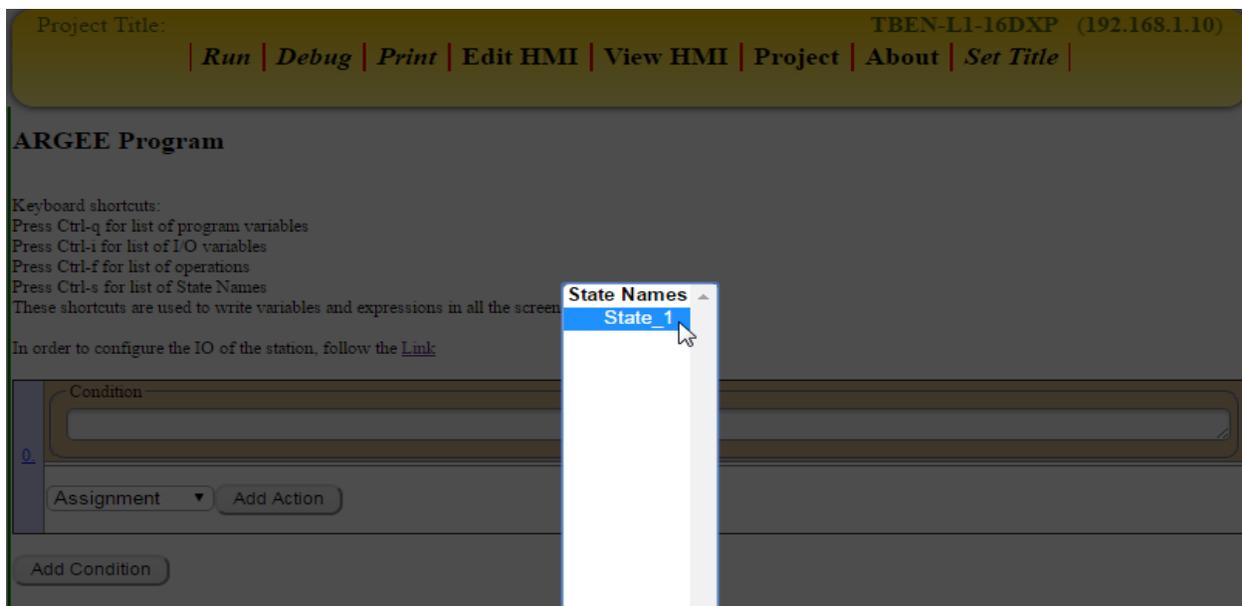
Operations (Control + F)

If the user presses *Ctrl + f* while in a Condition or Action box, the Operations List will pop up. The user can select their desired operation and it will be added to their respective Condition or Action box.



State Names (Control + S)

If the user presses *Ctrl + s* while in a Condition or Action box, the State Name List will pop up. The user can select their desired State Name and it will be added to their respective Condition or Action box.



NOTE: If the user has not added a State Name to their project, the State Name List will be empty.

Chapter 5: Conditions & Actions

Conditions

The Condition box is where the user puts their input conditions. An example of an Input condition could be:

- A Timer expiring
- A Counter reaching a specific value
- A Counter expiring
- A register value changing from 0 to 1
- A register value changing from 1 to 0
- An Input from a sensor becoming true
- ...many other things can also be used as an input condition

Project Title: TBEN-L1-16DXP (192.168.1.10)

[Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#)

ARGEE Program

Keyboard shortcuts:
Press Ctrl-q for list of program variables
Press Ctrl-i for list of I/O variables
Press Ctrl-f for list of operations
Press Ctrl-s for list of State Names
These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition

0.

Assignment ▼ Add Action

Add Condition

The Condition box also allows the user to combine different types of Inputs.

Condition

`expired(tm1) & IO.Slot1.Input.Input_value_1`

0.

Assignment ▼ Add Action

Add Condition

Explaining the Screenshot: The above Condition will only become true when timer 1 expires and Input_value_1 goes true.

Actions

The Actions box is where the user puts their Output conditions. The user can execute several Actions under a single Condition statement. An Action could be:

- Loading a value into a register
- Stating a Timer
- Stopping a Timer
- Signal Tracing
- Incrementing a Counter
- Decrementing a Counter
- Resetting a Counter

The screenshot shows a configuration window with a 'Condition' field containing 'true'. Below it is an 'Assignment' dropdown menu with an 'Add Action' button. The dropdown menu is open, showing the following options: Assignment, Timer start, Coil, Timer On, Timer Off, Trace, Comment, Count Up, Count Down, and Reset Counter.

Assignment

The user would use the *Assignment* action if they want to load a value into a register.

The screenshot shows the same configuration window as above, but now with an 'Assignment' action added to the 'Actions' list. The 'Assignment' action is highlighted in cyan. Its 'Destination' is set to 'IO.Slot1.Output.Output_value_1' and its 'Expression' is set to '1'. Below the list is another 'Assignment' dropdown menu and an 'Add Action' button.

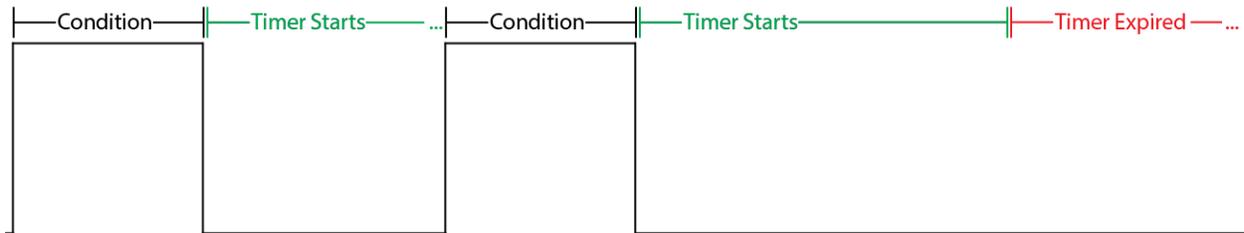
Explaining the Example: The Condition in the above statement is always “true”. The value “1” is loaded into register Output_value_1. In other words, this means that the user’s Output 1 will always be on.

Timer Start

The user will use the *Timer Start* action if they want to start a timer after the Condition has occurred.



If the Condition occurs again before the timer expires, the timer will restart.



Example of Timer Start

The screenshot shows a configuration interface for a PLC ladder logic network. It consists of two rungs. The first rung (labeled '0') has a condition 'IO.Slot1.Input.Input_value_1' and an action 'Timer start' with parameters 'Timer: tm1' and 'Expires (ms): 5000'. Below the rung is a dropdown menu set to 'Assignment' and an 'Add Action' button. The second rung (labeled '1') has a condition 'expired(tm1)' and an action 'Assignment' with parameters 'Destination: IO.Slot1.Output.Output_value_2' and 'Expression: 1'. Below the rung is another dropdown menu set to 'Assignment' and an 'Add Action' button.

Explaining the Example: When Input_value_1 goes true and then false, start timer 1. When timer 1 expires, load the value "1" into register Output_value_2 (or turn on Output 2).

Coil

The user will use the *Coil* action if they want an Output to be “set” if the Condition is true and “cleared” when the Condition is false.

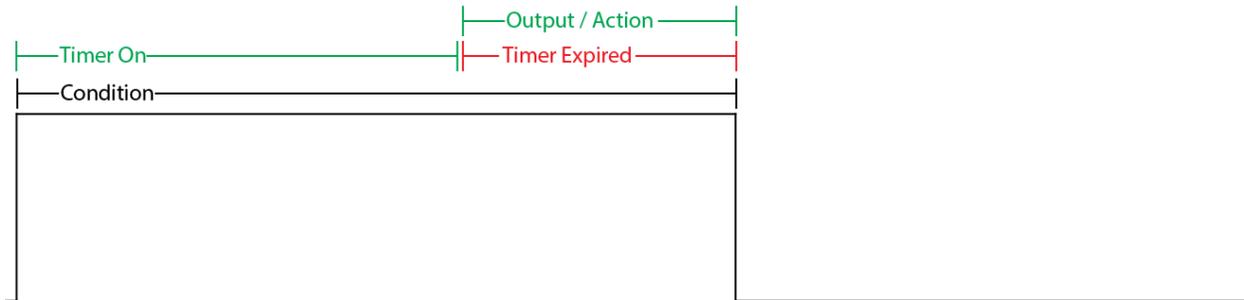


Example of Coil

Explaining the Example: When Input_value_1 is true, Output_value_2 is true. When Input_value_1 is false, Output_value_2 is false.

Timer On

The user will use the Timer On action if they want a timer to run while a Condition is true. The user will normally tie an additional Action or Output to the timer expired Condition.



If the Condition ends before the timer expires, the Action tied to the expired timer will not occur.



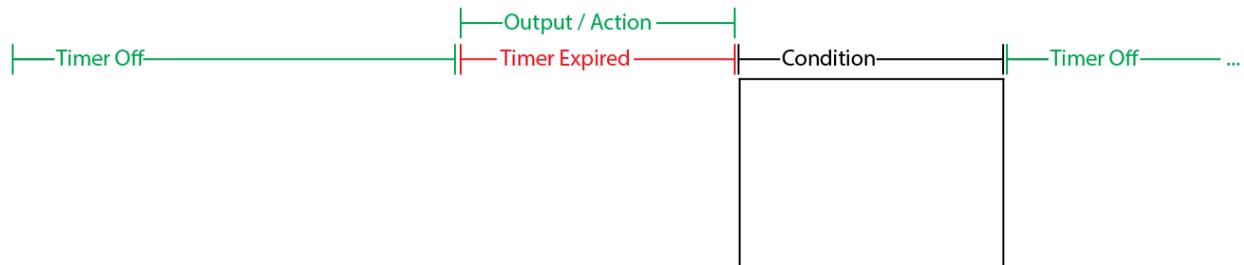
Example of Timer On

The screenshot shows a configuration interface for a PLC. It consists of two main sections. The top section has a 'Condition' field containing 'IO.Slot1.Input.Input_value_1' and an 'Actions' field containing a 'Timer On' action. The 'Timer On' action is configured with 'Timer: tm1' and 'Expires (ms): 5000'. Below this section is an 'Assignment' dropdown and an 'Add Action' button. The bottom section has a 'Condition' field containing 'expired(tm1)' and an 'Actions' field containing a 'Coil' action. The 'Coil' action is configured with 'Coil: IO.Slot1.Output.Output_value_2'. Below this section is another 'Assignment' dropdown and an 'Add Action' button.

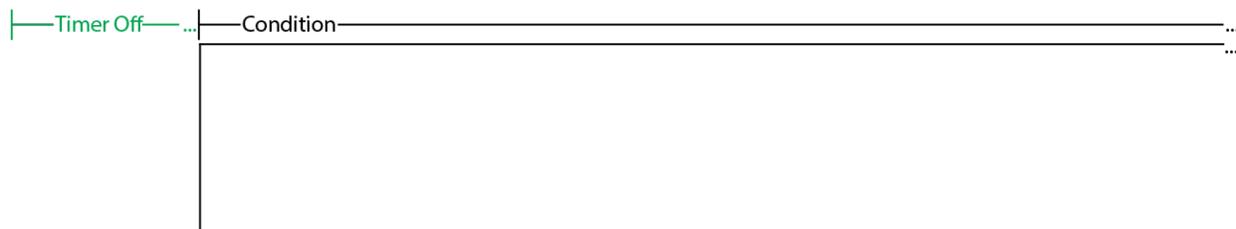
Explaining the Example: When Input_value_1 is true, start timer 1. When timer 1 expires, coil Output_value_2. When Input_value_1 is false, Output_value_2 will be false.

Timer Off

The user will use the Timer Off action if they want a timer to run while a Condition is false. The user will normally tie an additional Action or Output to the timer expired Condition.



If the Condition starts before the timer expires, the Action tied to the expired timer will not occur.



Example of Timer Off

Condition

IO.Slot1.Input.Input_value_1

Actions

0.	Timer Off	Timer: tm1
		Expires (ms): 5000

Assignment Add Action

Condition

expired(tm1)

Actions

0.	Coil	Coil: IO.Slot1.Output.Output_value_2
----	------	--------------------------------------

Assignment Add Action

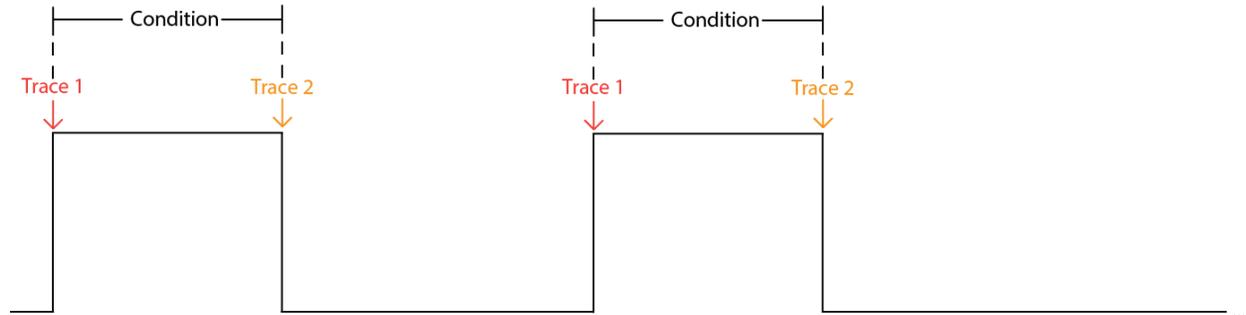
Explaining the Example: Timer 1 starts counting as soon as the program starts. When timer 1 expires, Output_value_2 is coiled on. When Input_value_1 is true, timer 1 is reset to zero and Output_value_2 goes false. When Input_value_1 is false, timer 1 starts counting again.

Trace

The user will use the Trace function if they want to time stamp exactly when an event occurred. Trace can be used to measure a programs run-time behavior, how long each state takes and even which states were visited in which order.

Example of Trace

The user wants to use Trace to measure how long the condition is true.



Note: The below example uses the Change of State Operation (F_COS) in the Condition block. The Change of State Operation is discussed on page 69.

The screenshot shows the configuration interface for the Trace function. It consists of two sections, each with a 'Condition' block and an 'Actions' block. The first section has a condition expression `(F_COS(IO.Slot1.Input.Input_value_1,Temp_1)& IO.Slot1.Input.Input_value_1=1)` and an action with Prefix String 'Trace_1', Display As 'Unsigned', and Expression '0'. The second section has a condition expression `(F_COS(IO.Slot1.Input.Input_value_1,Temp_2)& IO.Slot1.Input.Input_value_1=0)` and an action with Prefix String 'Trace_2', Display As 'Unsigned', and Expression '1'. Both sections include an 'Assignment' dropdown and an 'Add Action' button.

Explaining the Example: When Input_value_1 is true, Trace_1 time stamps that event. When Input_value_1 goes false, Trace_2 time stamps that event. The Prefix String is a name that makes sense to the user. The Expression can be any value or even another variable name that makes sense to the user.

The Trace example is continued on the next page.



Trace Example (Continued): Once the user has written the code the user will click *Run*.

Project Title: TBEN-L1-16DXP (192.168.1.10)

Run | Debug | Print | Edit HMI | View HMI | Project | About | Set Title



To view the Trace, the user will click *Show Trace*.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| Edit Code | View HMI | Modify Variables | **Show Trace** | Reset



The user will trigger their Condition true then false to show a transition in the Trace data.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| Edit Code | View HMI | Stop Trace | Show Variables

Trace Data:				
1	time:37240	cond:1	act:0	Trace_2:1
0	time:36805	cond:0	act:0	Trace_1:0

ARGEE Program

Condition
(F_COS(IO.Slot1.Input.Input_value_1,Temp_1) & (IO.Slot1.Input.Input_value_1 = 1))

Actions

0. Trace
Prefix: Trace_1
String:
Display As: Unsigned
Expression: 0

Condition
(F_COS(IO.Slot1.Input.Input_value_1,Temp_2) & (IO.Slot1.Input.Input_value_1 = 0))

Actions

1. Trace
Prefix: Trace_2
String:
Display As: Unsigned
Expression: 1

To calculate how long the users Condition is true, the user must subtract the two time stamps from one another:
 $37240 - 36805 = 435\text{ms}$.

Comment

The user can use a Comment to explain the Condition and Action statements.

The screenshot displays a user interface for configuring a rule. It features a 'Condition' section with a text input field containing the word 'true'. Below this is an 'Actions' section containing a table with one row. The table has two columns: 'Comment' and 'Comment:'. The 'Comment' column contains a small blue icon, and the 'Comment:' column contains the text 'This Condition is always true.'. At the bottom of the interface, there is a dropdown menu labeled 'Assignment' and a button labeled 'Add Action'.

Condition	
true	

Actions	
	Comment: This Condition is always true.

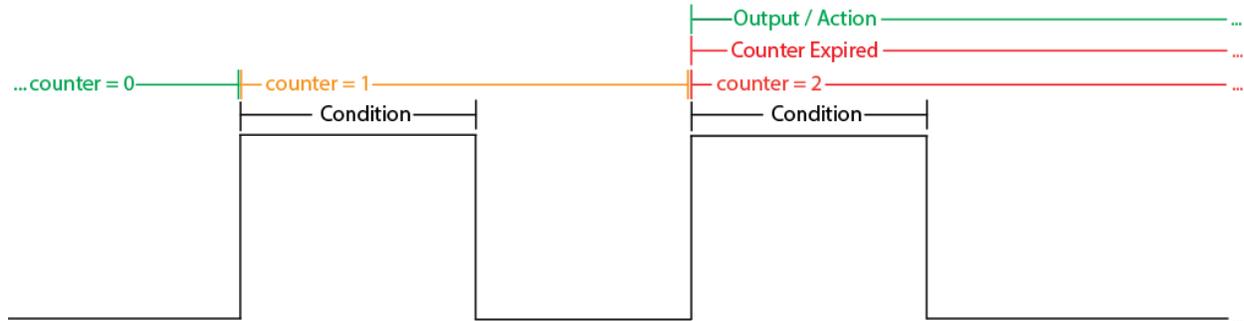
Assignment ▼ Add Action

Count Up

The user will use *Count Up* if they want to count the number of times their condition is true. The user will normally tie an additional Action or Output to the counter expired Condition.

Example of Count Up

The user wants to do an Action after the same Condition has occurred two times.



Condition

IO.Slot1.Input.Input_value_1

Actions

0. Count Up Counter: cnt1
Preset: 2

Assignment ▼ Add Action

Condition

expired(cnt1)

Actions

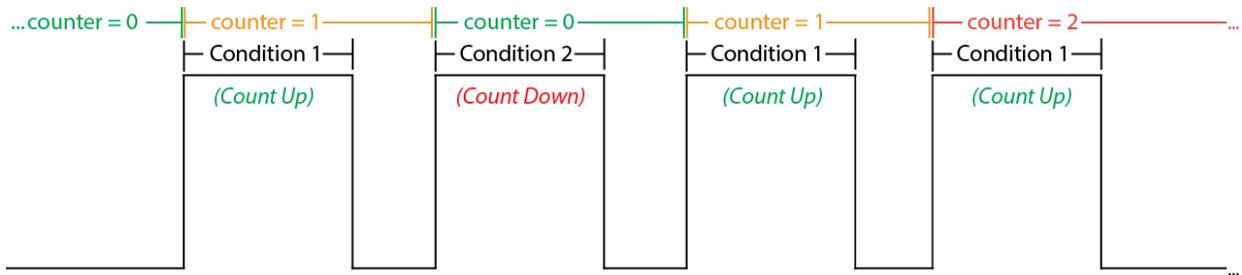
1. Coil Coil: IO.Slot1.Output.Output_value_2

Assignment ▼ Add Action

Explaining the Example: Each time Input_value_1 is true, counter 1 counts up one time. Counter 1 expires after two counts. When counter 1 expires, Output_value_2 is coiled on.

Count Down

The user will use *Count Down* if they want to count down when a condition is true. Count Down is normally used to counter the *Count Up* Action.



Example of Count Down

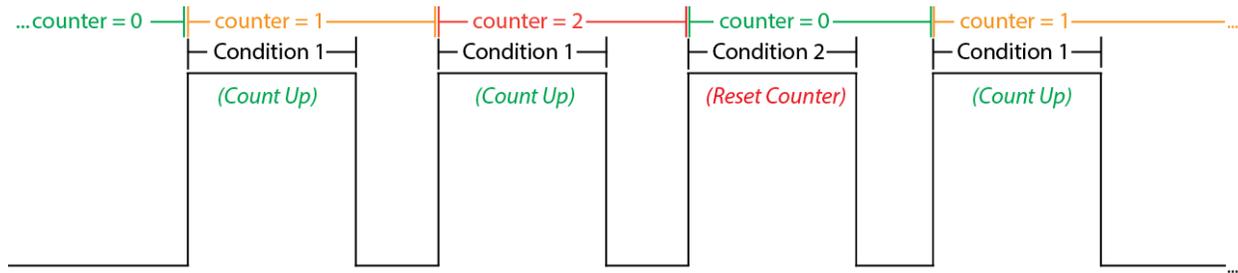
The user wants to keep track of the number of guests in the store. When a guest walks in the store the counter goes up, but when a guest walks out of the store the counter goes down.

The screenshot shows a configuration interface for a counter. It consists of two rows, each representing a different condition and action. The first row has a condition 'IO.Slot1.Input.Input_value_1' and an action 'Count Up' for counter 'cnt1' with a preset of 1000. The second row has a condition 'IO.Slot1.Input.Input_value_2' and an action 'Count Down' for counter 'cnt1' with a preset of 1000. Each row includes an 'Assignment' dropdown and an 'Add Action' button.

Explaining the Example: Each time `Input_value_1` is true (or a guest walks in the store), counter 1 counts up one time. Each time `Input_value_2` is true (or a guest walks out of the store), counter 1 counts down one time.

Reset Counter

The user will use *Reset Counter* if they want to reset a counter to zero.



Example of Reset Counter

The user wants the ability to reset the counter at any time.

The screenshot shows a configuration interface for a counter with two conditions and their associated actions:

- Condition 0:** IO.Slot1.Input.Input_value_1. Action: Count Up. Counter: cnt1. Preset: 5.
- Condition 1:** IO.Slot1.Input.Input_value_2. Action: Reset Counter. Counter: cnt1.

Each condition has an 'Assignment' dropdown and an 'Add Action' button.

Explaining the Example: Each time Input_value_1 is true, counter 1 counts up one time. Each time Input_value_2 is true, counter 1 resets to zero.

Chapter 6 - Operations

Math

The user will use Math Operations if they want to monitor, compare or combine data from different registers. Math Operations can be used in both Condition and Action expressions.

Addition

The user will use the Add Operation (+) to add one value to another value.

Example of Add Operation

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text box containing the expression `IO.Slot1.Input.Input_value_1`.
- Actions:** A table with one row:

0	Assignment	Destination: Temporary_Register	Expression: Register_A + Register_B
---	------------	---------------------------------	-------------------------------------

At the bottom, there is a dropdown menu set to "Assignment" and a button labeled "Add Action".

Explaining the Example: When `Input_value_1` is true, the value in `Register_A` will be added to the value in `Register_B`. The result is placed in `Temporary_Register`.

Subtraction

The user will use the Subtraction Operation (-) to subtract one value from another value.

Example of Subtraction Operation

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text box containing the expression `(Register_A - Register_B) > 1`.
- Actions:** A table with one row:

0	Coil	Coil: IO.Slot1.Output.Output_value_1
---	------	--------------------------------------

At the bottom, there is a dropdown menu set to "Assignment" and a button labeled "Add Action".

Explaining the Example: The user is subtracting the value in `Register_A` from the value in `Register_B`. When `Register A` minus `Register B` is greater than 1, the user Coils on `Output_value_1`.

Multiplication

The user will use the Multiplication Operation (*) to multiply one value with another value.

Example of Multiplication Operation

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text box containing the expression `(Register_A * Register_B) < 1000`.
- Actions:** A list of actions where the first action is selected. It is labeled "Coil" and has a value of `IO.Slot1.Output.Output_value_1`.
- Bottom:** A dropdown menu set to "Assignment" and an "Add Action" button.

Explaining the Example: The user is multiplying the value in Register_A with the value in Register_B. If Register A times Register B is less than 1000, the user Coils on Output_value_1.

Division

The user will use the Division Operation (/) to divide one value into another value.

Example of Division Operation

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text box containing the expression `IO.Slot1.Input.Input_value_1`.
- Actions:** A list of actions where the first action is selected. It is labeled "Assignment" and has two fields: "Destination" set to `Temporary_Register` and "Expression" set to `Register_A / Register_B`.
- Bottom:** A dropdown menu set to "Assignment" and an "Add Action" button.

Explaining the Example: When Input_value_1 is true, the value in Register_A will be divided by the value in Register_B. The result is placed in Temporary_Register.

VERY IMPORTANT NOTE: ARGEE does not currently support floating point math (or fractions). If the result has a fraction in it, ARGEE will drop the fraction and just display the whole number.

For example:

$36 / 6 = 6$ ---> ARGEE displays "6"

$34 / 6 = 5\frac{4}{6}$ ---> ARGEE displays "5"

$6 / 36 = \frac{1}{6}$ ---> ARGEE displays "0"

Modulo

The user will use the Modulo Operation (%) if they want to capture the “remainder” after a Division Operation has occurred.

Example of Modulo Operation

The screenshot shows a configuration window with two main sections: 'Condition' and 'Actions'. The 'Condition' section contains a text field with the value 'IO.Slot1.Input.Input_value_1'. The 'Actions' section contains a table with one row: 'Assignment' (highlighted in cyan) with 'Destination: Temporary_Register' and 'Expression: Register_A % Register_B'. Below the table is a dropdown menu set to 'Assignment' and an 'Add Action' button.

Explaining the Example: When Input_value_1 is true, the value in Register_A will be divided by the value in Register_B. The “remainder” from the Division Operation is placed in Temporary_Register.

For example:

36 / 6 = “6” with a remainder of “0” ---> ARGEE displays “0”

34 / 6 = “5” with a remainder of “4” ---> ARGEE displays “4”

6 / 36 = “0” with a remainder of “6” ---> ARGEE displays “6”

Absolute Value

The user will use the Absolute Value Operation (abs) to capture the magnitude of a real number without regard to its sign.

Example of Absolute Value Operation

The screenshot shows a configuration window with two main sections: 'Condition' and 'Actions'. The 'Condition' section contains a text field with the value 'IO.Slot1.Input.Input_value_1'. The 'Actions' section contains a table with one row: 'Assignment' (highlighted in cyan) with 'Destination: Register_A' and 'Expression: abs(Register_A)'. Below the table is a dropdown menu set to 'Assignment' and an 'Add Action' button.

Explaining the Example: When Input_value_1 is true, ARGEE will take the value in Register_A, find its Absolute Value, and place that value back in Register_A.

Minimum Value

The user will use the Minimum Value Operation (min) to compare multiple registers and place the smallest value in to the Destination Register. The user can also use the Minimum Value Operation (min) to compare multiple registers and use the smallest value in a Math Operation.

Example of Minimum Value Operation

Condition	
IO.Slot1.Input.Input_value_1	
Actions	
0. Assignment	Destination: Temporary_Register Expression: min(Register_A , Register_B)
Assignment	Add Action

Explaining the Example: When Input_value_1 is true, ARGEE will take the smallest value between Register_A and Register_B and place that value into Temporary_Register

OR

Condition	
IO.Slot1.Input.Input_value_1	
Actions	
0. Assignment	Destination: Temporary_Register Expression: Register_C + min(Register_A , Register_B)
Assignment	Add Action

Explaining the Example: When Input_value_1 is true, ARGEE will take the smallest value between Register_A and Register_B and place that value into the Math Operation. The result will be stored in Temporary_Register.

Maximum Value

The user will use the Maximum Value Operation (max) to compare multiple registers and place the largest value in to the Destination Register. The user can also use the Maximum Value Operation (max) to compare multiple registers and use the largest value in a Math Operation.

Example of Maximum Value Operation

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text field containing the expression `IO.Slot1.Input.Input_value_1`.
- Actions:** A list of actions. The first action is highlighted in cyan and is labeled "Assignment". It has a "Destination" field set to `Temporary_Register` and an "Expression" field set to `max(Register_A , Register_B)`.

At the bottom of the "Actions" section, there is a dropdown menu currently set to "Assignment" and a button labeled "Add Action".

Explaining the Example: When `Input_value_1` is true, ARGEE will take the largest value between `Register_A` and `Register_B` and place that value into `Temporary_Register`

OR

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text field containing the expression `IO.Slot1.Input.Input_value_1`.
- Actions:** A list of actions. The first action is highlighted in cyan and is labeled "Assignment". It has a "Destination" field set to `Temporary_Register` and an "Expression" field set to `Register_C + max(Register_A , Register_B)`.

At the bottom of the "Actions" section, there is a dropdown menu currently set to "Assignment" and a button labeled "Add Action".

Explaining the Example: When `Input_value_1` is true, ARGEE will take the largest value between `Register_A` and `Register_B` and place that value into the Math Operation. The result will be stored in `Temporary_Register`.

Brackets

The user will use Brackets () to show the order of operations while performing Math.

Example of Brackets

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text field containing the expression `IO.Slot1.Input.Input_value_1`.
- Actions:** A table with one row:

0	Assignment	Destination: Temporary_Register	Expression: Register_A / (Register_B + Register_C)
---	------------	---------------------------------	--
- At the bottom, there is a dropdown menu set to "Assignment" and an "Add Action" button.

Explaining the Example: When `Input_value_1` is true, ARGEE will examine the “(Register_B + Register_C)” first and then divide the answer into the value in Register_A. The result will be stored in Temporary_Register.

Boolean AND

The user will use the Boolean AND Operation (&) if the user wants to combine several *Conditions* together before allowing a specific Action to occur.

Example of Boolean AND

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text field containing the expression `IO.Slot1.Input.Input_value_1 & IO.Slot1.Input.Input_value_2`.
- Actions:** A table with one row:

0	Assignment	Destination: Register_A	Expression: 1
---	------------	-------------------------	---------------
- At the bottom, there is a dropdown menu set to "Assignment" and an "Add Action" button.

Explaining the Example: When both `Input_value_1` AND `input_value_2` are true, load the value “1” into Register_A.

Boolean OR

The user will use the Boolean OR Operation (|) if the user wants one of several *Conditions* to cause an Action to occur.

Example of Boolean OR

The screenshot shows a configuration window with two main sections: "Condition" and "Actions".

- Condition:** A text field contains the expression `IO.Slot1.Input.Input_value_1 | IO.Slot1.Input.Input_value_2`.
- Actions:** A table with one row:

0	Assignment	Destination: Register_A	Expression: 1
---	------------	-------------------------	---------------

At the bottom, there is a dropdown menu set to "Assignment" and an "Add Action" button.

Explaining the Example: When either `Input_value_1` OR `input_value_2` are true, load the value "1" into `Register_A`.

Boolean NOT

The user will use the Boolean NOT Operation (!) if the user wants an Action to occur while a *Condition* is false.

Example of Boolean NOT

The screenshot shows two configuration windows stacked vertically, each with "Condition" and "Actions" sections.

Top Configuration:

- Condition:** `IO.Slot1.Input.Input_value_1`
- Actions:**

0	Assignment	Destination: Register_A	Expression: 1
---	------------	-------------------------	---------------

Bottom Configuration:

- Condition:** `!IO.Slot1.Input.Input_value_1`
- Actions:**

0	Assignment	Destination: Register_A	Expression: 0
---	------------	-------------------------	---------------

Each configuration window has a dropdown menu set to "Assignment" and an "Add Action" button.

Explaining the Example: When `Input_value_1` is true, load the value "1" into `Register_A`. When `Input_value_1` is false, load the value "0" into `Register_A`.

Greater Than

The user will use the Greater Than Operation (>) if the user wants a *Condition* to occur when one register value is Greater Than another register value.

Example of Greater Than

Condition	
Register_A > Register_B	

Actions		
0.	Assignment	Destination: Register_C Expression: 1

Assignment ▼ Add Action

Explaining the Example: When the value in Register_A is Greater Than the value in Register_B, the value “1” will be loaded into Register_C.

Greater Than or Equal to

The user will use the Greater Than or Equal to Operation (>=) if the user wants a *Condition* to occur when one register value is Greater Than or Equal to another register value.

Example of Greater Than

Condition	
Register_A >= Register_B	

Actions		
0.	Assignment	Destination: Register_C Expression: 1

Assignment ▼ Add Action

Explaining the Example: When the value in Register_A is Greater Than or Equal to the value in Register_B, the value “1” will be loaded into Register_C.

Less Than

The user will use the Less Than Operation (<) if the user wants a *Condition* to occur when one register value is Less Than another register value.

Example of Greater Than

The screenshot shows a configuration window with two main sections: 'Condition' and 'Actions'. The 'Condition' section contains a text field with the expression 'Register_A < Register_B'. The 'Actions' section contains a table with one row: 'Assignment' (highlighted in cyan), 'Destination: Register_C', and 'Expression: 1'. Below the table is a dropdown menu set to 'Assignment' and an 'Add Action' button.

Explaining the Example: When the value in Register_A is Less Than the value in Register_B, the value “1” will be loaded into Register_C.

Less Than or Equal to

The user will use the Less Than or Equal to Operation (<=) if the user wants a *Condition* to occur when one register value is Less Than or Equal to another register value.

Example of Greater Than

The screenshot shows a configuration window with two main sections: 'Condition' and 'Actions'. The 'Condition' section contains a text field with the expression 'Register_A <= Register_B'. The 'Actions' section contains a table with one row: 'Assignment' (highlighted in cyan), 'Destination: Register_C', and 'Expression: 1'. Below the table is a dropdown menu set to 'Assignment' and an 'Add Action' button.

Explaining the Example: When the value in Register_A is Less Than or Equal to the value in Register_B, the value “1” will be loaded into Register_C.

Equal

The user will use the Equal Operation (=) if the user wants a *Condition* to occur when one register value is Equal to another register value.

Example of Equal

Condition	
Register_A = Register_B	

Actions	
0. Assignment	Destination: Register_C Expression: 1

Assignment ▼ Add Action

Explaining the Example: When the value in Register_A is Equal to the value in Register_B, the value “1” will be loaded into Register_C.

Not Equal

The user will use the Not Equal Operation (<=>) if the user wants a *Condition* to occur when one register value is Not Equal to another register value.

Example of Not Equal

Condition	
Register_A = Register_B	

Actions	
0. Assignment	Destination: Register_C Expression: 1

Assignment ▼ Add Action

Condition	
Register_A <=> Register_B	

Actions	
0. Assignment	Destination: Register_C Expression: 0

Assignment ▼ Add Action

Explaining the Example: When the value in Register_A is Equal to the value in Register_B, the value “1” will be loaded into Register_C. When the value in Register_A is Not Equal to the value in Register_B, the value “0” will be loaded into Register_C.

If_Then_Else

The user will use the If_Then_Else operation if they want an expression to be set *only* if a particular test evaluates as true. If it evaluates as false, a secondary expression is chosen.

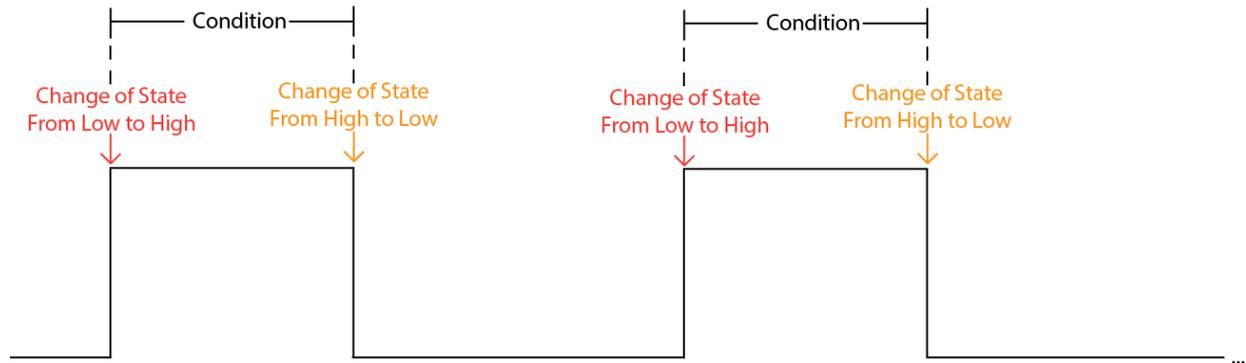
Example of if_then_else

The screenshot displays a configuration window for an 'if_then_else' operation. It is divided into two main sections: 'Condition' and 'Actions'.
- The 'Condition' section has a text input field containing the value 'true'.
- The 'Actions' section contains a list of actions. The first action is an 'Assignment' (highlighted in cyan), with a 'Destination' of 'Temporary_Register' and an 'Expression' of 'if_then_else(Register_A>1000, Register_B, Register_C)'.
- Below the actions list, there is a dropdown menu currently set to 'Assignment' and a button labeled 'Add Action'.

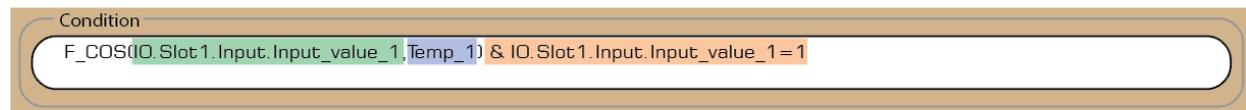
Explaining the Example: If the value in Register_A is below 1000, then the value in Register_B is loaded into the Temporary_Register. If the value in Register_A is above 1000, then the value in Register_C is loaded into the Temporary_Register.

Change of State

The user will use the Change of State Operation (F_COS) if the user wants an Action only to occur when a *Condition* changes state. A *Condition* can change state from either “low to high” or “high to low”.



Change of State Command Structure:



= The *Condition* that is being monitored for a *Change of State*.

= The register that stores the monitored *Condition*'s current state.

= The part of the *Condition* that tells *ARGEE* to monitor the “low to high” *Change of State* or the “high to low” *Change of State*.

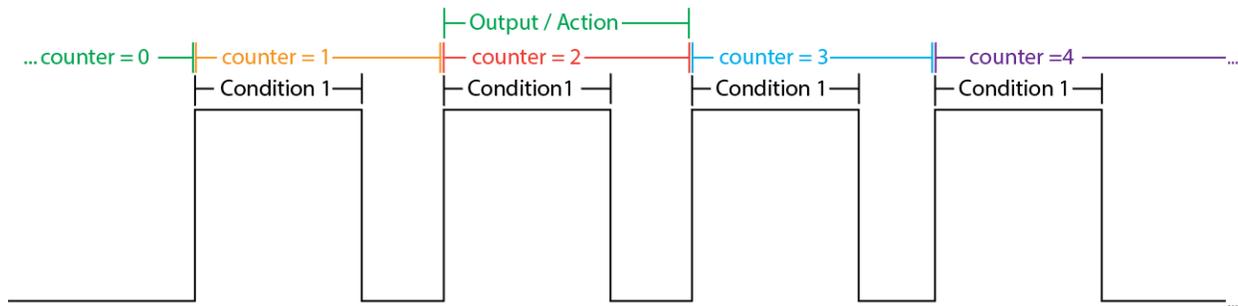
Example of Change of State

Explaining the Example: When *Input_value_1* does a Change of State from low (zero) to high (one), the value “1” is loaded into *Register_A*. When *Input_value_2* does a Change of State from high (one) to low (zero), the value “0” is loaded into *Register_A*.

Note: Each monitored Condition requires its own Current State register. Notice in the example, Temp_1 was used to monitor the Change of State of Input_value_1 and Temp_2 was used to monitor the Change of State of Input_value_2.

Count

The user will use the Count Operation (count) if the user wants to perform an Action when a counter or timer is at a specific value but has not yet expired.



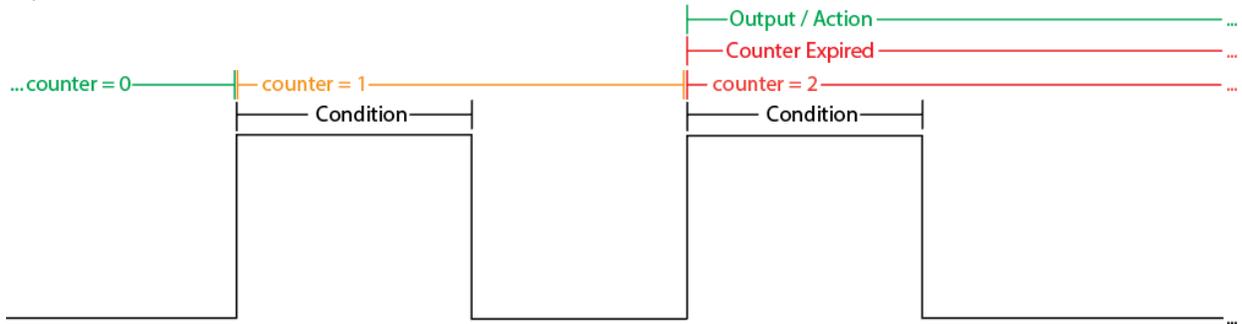
Example of Count

0.	<p>Condition</p> <p>IO.Slot1.Input.Input_value_1</p> <hr/> <p>Actions</p> <p>0. Count Up Counter: cnt1 Preset: 5</p> <p>Assignment Add Action</p>
1.	<p>Condition</p> <p>IO.Slot1.Input.Input_value_2</p> <hr/> <p>Actions</p> <p>0. Timer start Timer: tm1 Expires (ms): 10000</p> <p>Assignment Add Action</p>
2.	<p>Condition</p> <p>count(cnt1)=2 count(tm1)>2000</p> <hr/> <p>Actions</p> <p>0. Coil Coil: IO.Slot1.Output.Output_value_3</p> <p>Assignment Add Action</p>

Explaining the Example: Each time Input_value_1 goes true, counter 1 counts up one time. Counter 1 expires after five counts. After input_value_2 goes true, timer 1 starts. Timer 1 will expire after ten seconds (or 10,000ms). When counter 1 counts to “2” OR timer 1 is Greater Than two seconds (or 2000ms), Output_value_3 is coiled on.

Expired

The user will use the Expired Operation (expired) if they want to perform an Action when a counter or timer has expired.



Example of Expired

The screenshot shows a ladder logic editor with three rungs. Each rung has a 'Condition' field and an 'Actions' field. Below each 'Actions' field are 'Assignment' and 'Add Action' buttons.

- Rung 0:** Condition: `IO.Slot1.Input.Input_value_1`. Action: **Count Up** (green button). Counter: `cnt1`. Preset: `2`.
- Rung 1:** Condition: `IO.Slot1.Input.Input_value_2`. Action: **Timer start** (green button). Timer: `tm1`. Expires (ms): `2000`.
- Rung 2:** Condition: `expired(cnt1) | expired(tm1)`. Action: **Coil** (yellow button). Coil: `IO.Slot1.Output.Output_value_3`.

Explaining the Example: Each time `Input_value_1` goes true, counter 1 counts up one time. Counter 1 expires after two counts. After `input_value_2` it goes true, timer 1 starts. Timer 1 will expire after two seconds (or 2,000ms). When counter 1 OR timer 1 expires, `Output_value_3` is coiled on.

Chapter 7 - ARGEE Simulation Mode

Opening the Environment

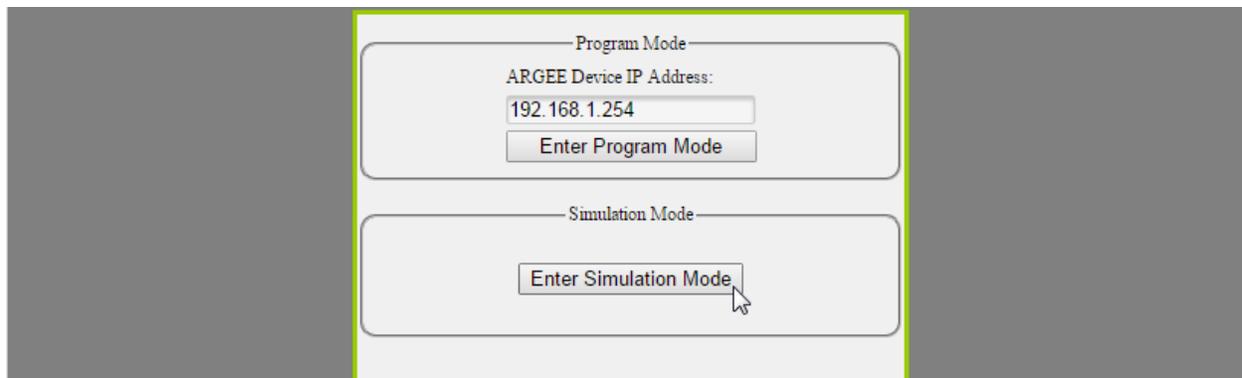
Open the ARGEE Environment and double click on pg.html.



NOTE: ARGEE only opens up in HTML 5 compliant web browsers such as Google Chrome or Firefox.

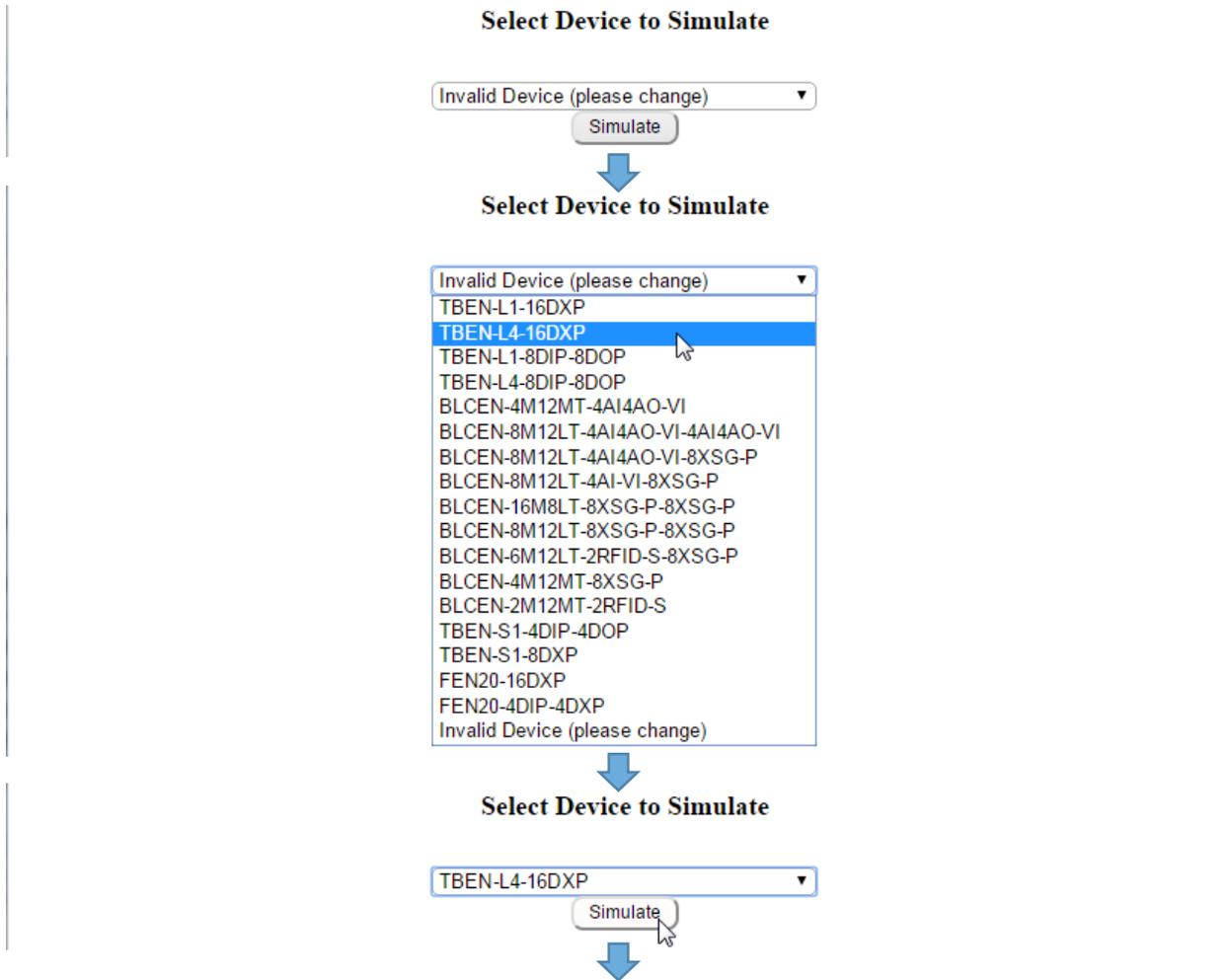
Logging into Simulation Mode

Click *Enter Simulation Mode*.



Select Device to Simulate

If the user has never used Simulation Mode before, the first thing they will have to do is select a device to simulate from the drop down arrow.



Welcome to ARGEE Simulation Mode.

Project Title:

TBEN-L4-16DXP (Simulation)

| Run | Debug | Print | Edit HMI | View HMI | Project | About | Set Title |

Project Checksum:15846

ARGEE Program

Keyboard shortcuts:

Press Ctrl-q for list of program variables

Press Ctrl-i for list of I/O variables

Press Ctrl-f for list of operations

Press Ctrl-s for list of State Names

These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Add Condition

Chapter 8 - ARGEE Security

General Security

Security is a concern to some users. ARGEE provides several security features, the first of which is General Security. General Security is the term used to explain a block's behavior with ARGEE programming versus a block's behavior without ARGEE programming.

Visual Behavior

If there is an ARGEE program running on the block and:

- The BUS LED will flash green three times and then stay off for 1 second.

If there is not an ARGEE program running on the block:

- The block's LED's will behave in accordance with that block's data sheet.

Connection Behavior

Ethernet IP Master (Allen Bradley)

If there is an ARGEE program running on the block before a PLC connection is established:

- The PLC connection point combinations 101,102 or 103,104 will not be allowed
- ARGEE will block any attempt by the PLC to upload parameters from the block
- The PLC will only be able to make connection to the block via the ARGEE connection pair 101, 110

If the PLC makes a connection to the block before an ARGEE program is loaded:

- The PLC connection point combinations 101,102 or 103,104 will be allowed
- The ARGEE connection pair 101, 110 will not be allowed
- The ARGEE environment will not allow upload of new code

Modbus TCP Master (VT500 or Red Lion HMI)

If there is an ARGEE program running on the block before a Modbus connection is established:

- Regular Modbus/TCP registers will not be accessible
- Access to Regular Modbus/TCP registers results in "exception"
- Only ARGEE Modbus/TCP registers can be read/written from:
 - 0x4000 - 0x407F (Registers 16384 - 16512 in decimal) Read only Input Data (ARGEE -> PLC)
 - 0x4400 - 0x447F (Register 17408 - 17536 in decimal) Read/Write Output Data (PLC -> ARGEE)

If a Modbus/TCP connection is established before an ARGEE program is loaded:

- Regular Modbus/TCP registers are accessible
- Access to ARGEE specific registers results in "exception"

PROFINET Master

If there is an ARGEE program running on the block before a PROFINET connection is established:

- Standard IO PROFINET connection is not allowed. ARGEE PROFINET connection is allowed
- Access to the block can be established by installing the ARGEE GSD file to the project

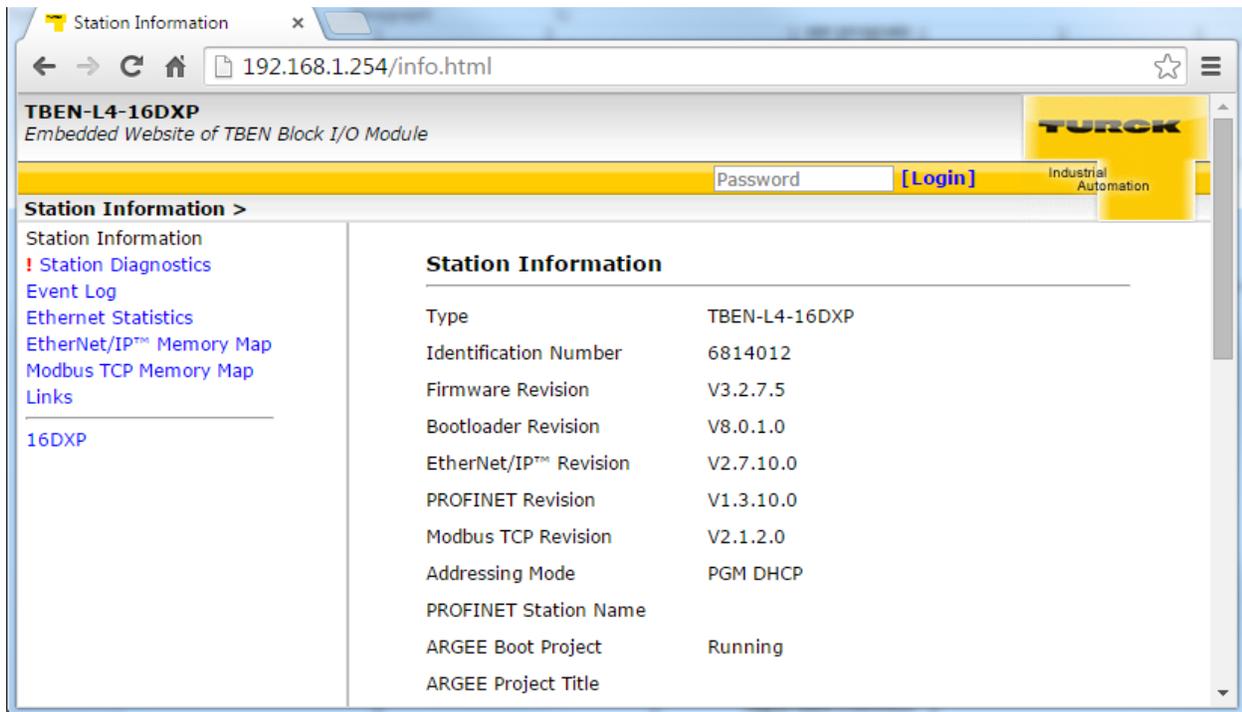
If a PROFINET connection is established before an ARGEE program is loaded:

- The regular PROFINET module ID is accessible. ARGEE PROFINET connection is not allowed. If the ARGEE environment attempts to load an ARGEE code when a standard PROFINET connection is established, the ARGEE environment will block the upload.

Note: PLC Connection examples can be found in Chapter 10 – Common Applications.

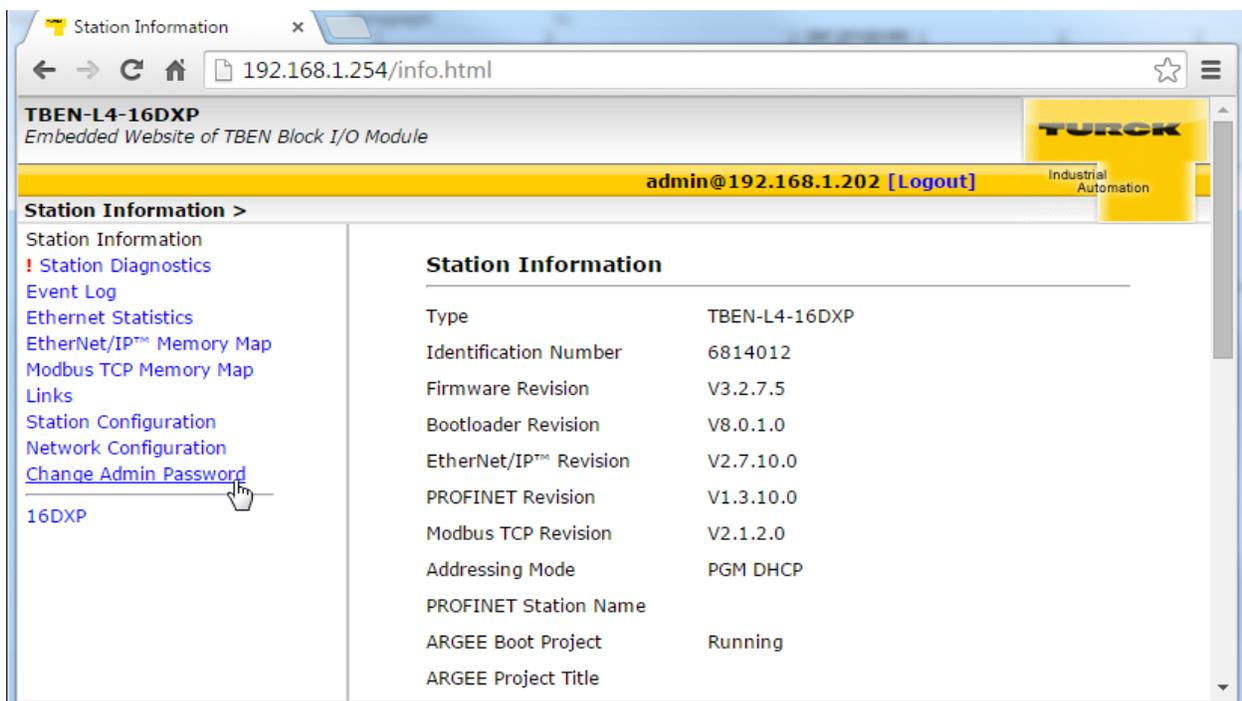
Password Protection – ARGEE Environment

All Turck block devices support a password protected webserver. To access the block’s webserver, the user needs to type the blocks IP address into any web browser.

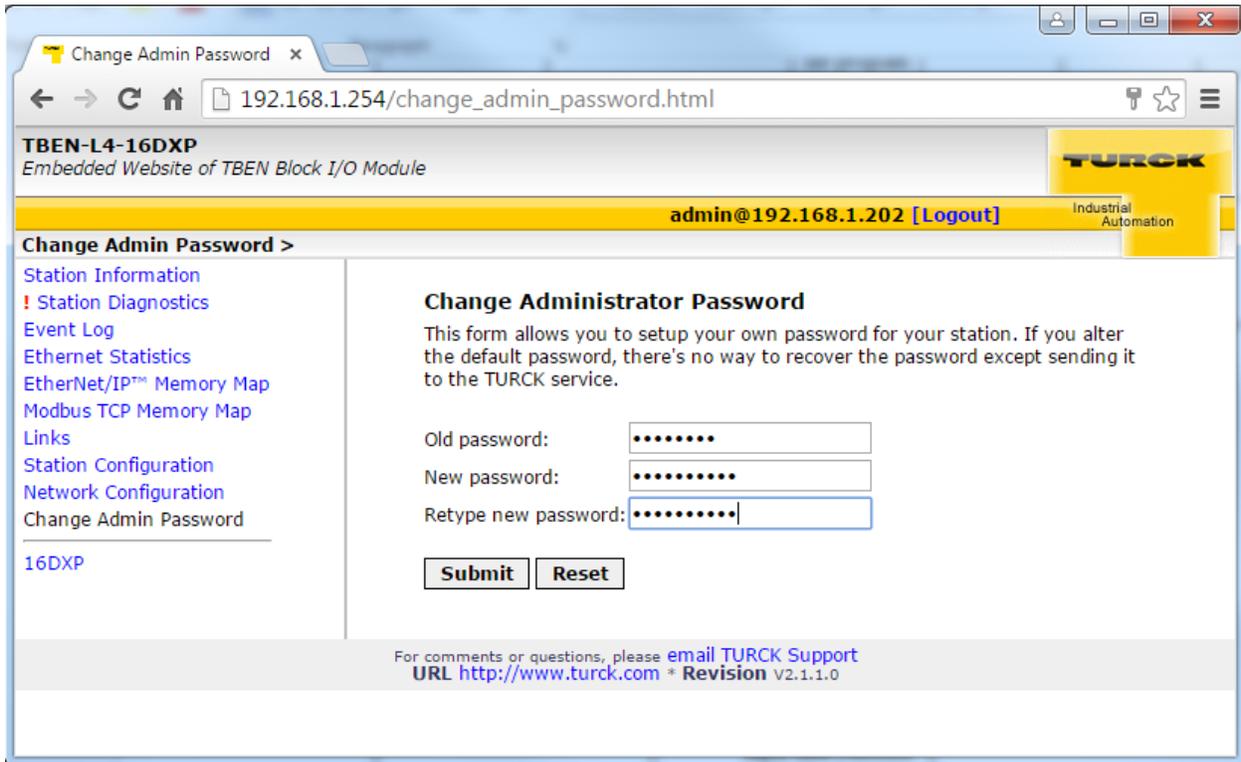


Note: The default password to log into the blocks webserver is “password”.

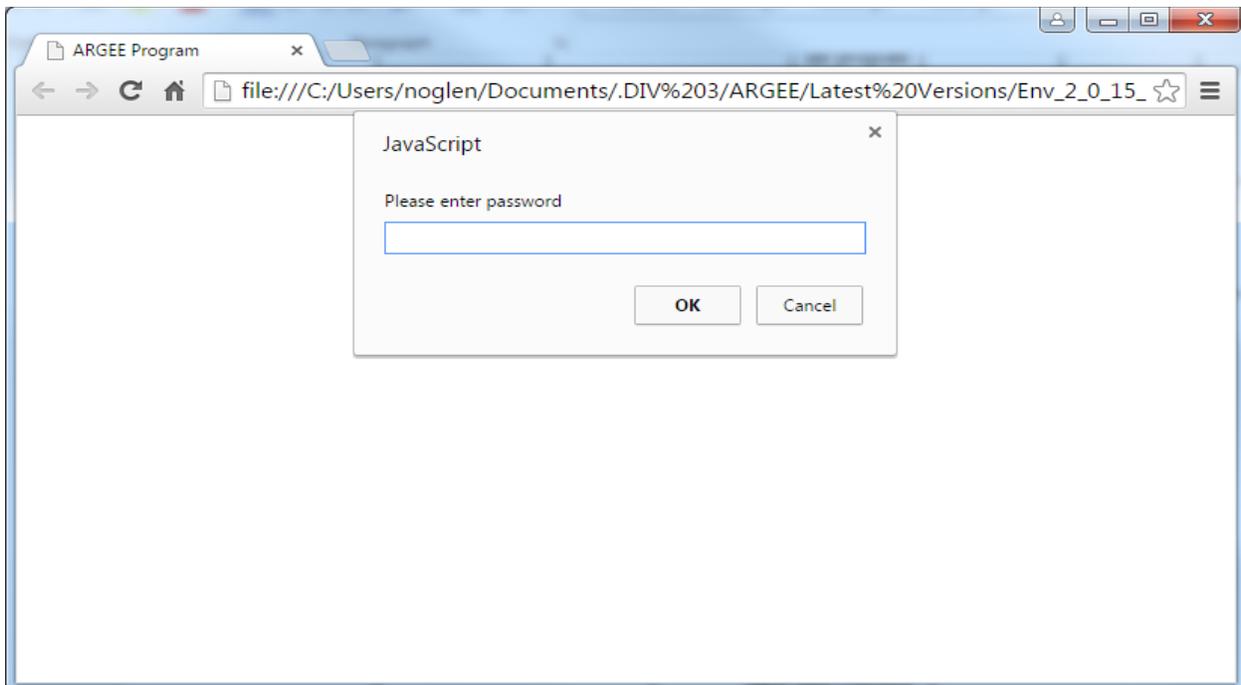
To password protect the users ARGEE environment, the user must change the Admin password on their webserver.



To change the Admin password, select *Change Admin Password* link, follow the instructions, and click *Submit*.



Now every time the user try's to log into the block, they will be prompted to input a password.



Note: To remove this feature, the user can simply change their webserver password back to "password".

Source Code Protection – Run Without Source

If a user wants to prevent “end users” from logging into the block and seeing or modifying code, the user will want to use the *Run Without Source* feature.

To access the *Run Without Source* button, the user must first click on *Project* and navigate to the second ARGEE menu bar.



If the user clicks on *Run Without Source* and then logs out of the environment, the ARGEE program will be hidden the next time anyone logs into the block.

Logging in before clicking Run Without Source

Project Title: TBEN-L1-16DXP (192.168.1.10)
 | *Run* | *Debug* | *Print* | *Edit HMI* | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables		ARGEE Program
Name	Type	
PLC_connected	Integer	Keyboard shortcuts: Press Ctrl-q for list of program variables Press Ctrl-i for list of I/O variables Press Ctrl-f for list of operations Press Ctrl-s for list of State Names These shortcuts are used to write variables and expressions in all the screens In order to configure the IO of the station, follow the Link
PROG_cycle_time	Integer	
reg1	Integer	
reg2	Integer	
tm1	Timer/Counter	
tm2	Timer/Counter	
cnt1	Timer/Counter	
cnt2	Timer/Counter	
Add Variable		

PLC Variables	
Name	Direction
plc_in_reg1	ARGEE->PLC

Condition:

Actions:

Coil:

Assignment



Logging in after the user click Run Without Source

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | **Edit HMI** | **View HMI** | **Project** | **About** | *Set Title* |

Program Variables

Add Variable

PLC Variables

Add Variable

State Names

Add State

ARGEE Program

Add Condition

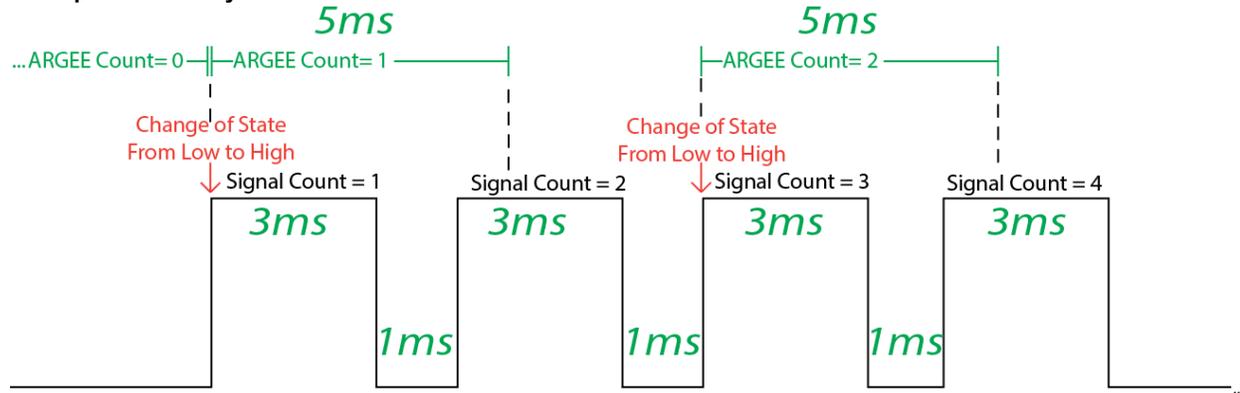
Very Important Note: The user needs to save a Master Copy of the program before the user logs out of the environment if the user wants to view/edit the code in the future.

Chapter 9 – System Performance

Scan Cycle Information

The ARGEE Scan Cycle is typically between 5 – 10 ms depending on the code size. If the user attempts to use ARGEE in an application with scan cycles less than 5 ms, it is possible that ARGEE may miss the signal.

Example of Scan Cycle



Explaining the Example: In this example, the user is hammering ARGEE with repeated 3 ms signals. Notice that ARGEE does not catch all the signals because the signal is occurring faster than ARGEE’s Scan Cycle.

Note: ARGEE is not suited for motion control applications.

IO Variable Formats

IO.Slot2.Output.Output_value_X -> This example loads the value “1” into Output 4 bit 0.

Hardware_failure_5	0
Hardware_failure_6	0
Hardware_failure_7	0
Slot 2:BLC-4AI4AO-V/I Output	
Output_value_4	1
Output_value_5	0
Output_value_6	0

Condition: true

Actions:

0. Assignment Destination: IO.Slot2.Output.Output_value_4 Expression: 1

IO.Slot2.Output.Output_value_X.Y -> In this example, when Input_value_0 Bit_2 equals “1”, a “1” is loading into Output_value_4 Bit_12.

Hardware_failure_5	0
Hardware_failure_6	0
Hardware_failure_7	0
Slot 2:BLC-4AI4AO-V/I Output	
Output_value_4	4096
Output_value_5	0
Output_value_6	0

Condition: IO.Slot2.Input.Input_value_0.2 =1

Actions:

0. Assignment Destination: IO.Slot2.Output.Output_value_4.12 Expression: 1

How Actions Respond to Conditions

Action	Condition=FALSE	Condition=TRUE
Assignment	No action	Assigns a destination variable to a result of expression evaluation.
Coil	Resets a variable to 0	Sets the variable to 1
Timer start	No action	If the timer is not started – it starts the timer. Otherwise it restarts the timer. The timer is executed in the background until the accumulator >= “Expires” Preset value.
Timer On	Resets the timer accumulator and Done flag.	If timer Done flag is 0, run the timer. The timer is accumulated every millisecond until the accumulator >=“Expires” Preset value. In that case the Done flag is raised.
Timer Off	If timer Done flag is 0, run the timer. The timer is accumulated every millisecond until the accumulator >=“Expires” Preset value. In that case, the Done flag is raised.	Resets the timer accumulator and Done flag.
Comment	-	-
Count up	Increments the counter whenever the condition changes from false to true.	
Count down	Decrements the counter whenever the condition changes from false to true. (note - the Preset can be a negative value)	
Reset	-	Restarts the counter to - 0
Trace	-	Record trace information into a trace buffer.

Defining Variable Types – (Advanced Definitions)

Type	Description	Type	Allowed arithmetic expressions	Specific actions	Size in bytes
Integer Variables	Variables are defined in the program.	All these variables are 32 bit signed integers.	All integer arithmetic	Assignment	4
Retain Integer	Variables which are automatically saved to flash.	All these variables are 32 bit signed integers.	All integer arithmetic	Assignment	8 bytes (4 bytes of data 4 bytes of additional information)
PLC Variables	Variables mapping upper level PLC (Modbus/TCP, EtherNet/IP or PROFINET) exchange data to an integer variable accessible in the program.	They are mapped to integer variables in the program	All integer arithmetic	Assignment	20
Timer/Counter	Timers Counters can be used with appropriate functions, such as “expired”, “count” and appropriate actions such as “Timer On”	Complex data types	Only used as argument to functions “expired” and “count”	Specific actions: Timer on, Timer off, Start timer, Count up, Count down	16
State	Integer variable that is used to designate states in state machine. Behaves identically to a regular integer variable except for 2 things: <ol style="list-style-type: none"> 1) Initialize – will list states 2) In the debugger, a state name matching the current value will show up 	32 bit integer	All integer arithmetic	Assignment	4
Local IO	Input/Output/Diagnostic points	They are mapped to integer variables in the program	All integer arithmetic	Assignment	(not allocated out of 1KB of RAM)
System variables	PLC Connected	32 bit integer		Only 1 bit is used to indicate PLC connected state	4
System variables	Max Cycle time (since program start)	32 bit integer indicating time in ms		Time from the previous cycle to the current cycle.	4

Chapter 10 - Common Applications

Communicating with an EtherNet/IP Master – Allen Bradley

ARGEE blocks have the ability to communicate with an EtherNet/IP Master. The E/IP Master can establish communication via connection points 101 & 110.

Example of Communicating with an EtherNet/IP Master

The user wants to check and see if data is being passed back and forth between the ARGEE block and the E/IP Master. The first thing the user does is set up PLC variables.

PLC Variables

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC	0	0	Word (16 bit)	unsigned	Delete Add Above
plc_out_reg1	PLC->ARGEE	0	0	Word (16 bit)	unsigned	Delete Add Above

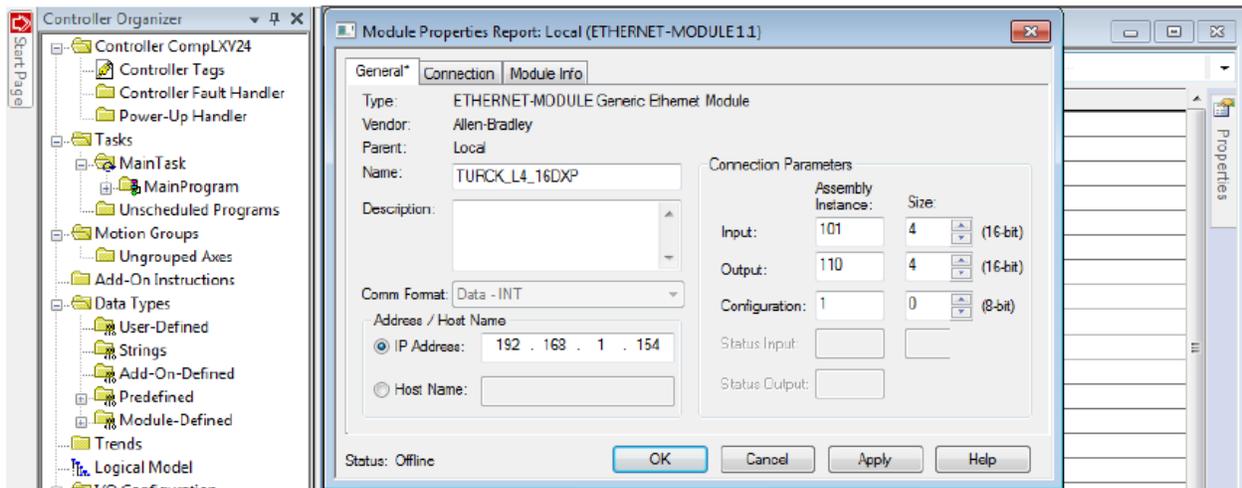
Explaining the Set-up: The user creates two PLC Variables, and they set the direction the data will travel. Data transmitted from ARGEE to the PLC is mapped into AB PLC Instance 101 and the data size is defined in the ARGEE PLC variable section (ARGEE->PLC). Data transmitted from the PLC to ARGEE is mapped into AB PLC Instance 110 and the data size is defined in the ARGEE PLC variable section (PLC->ARGEE).

The next step is to write the ARGEE code.



Explaining the Code: The user wrote the value “1” into plc_in_reg1.

The third step is to set up the connection points inside the PLC.



Explaining the Set up: The user created a Generic Ethernet Device and set the connection points to be 101 & 110. The last step is to connect to the device, place a value in the Output Register and verify data transfer.

The screenshot displays a web-based interface for configuring a PLC system. It is divided into several sections:

- Controller Tags - CompLXV24 (controller):** A table listing various tags with their names, values, styles, and data types. A red arrow points from the 'Value' column of the 'TURCK_L4_16DXP.O' tag to the 'PLC Variables' section.
- Program Variables:** A table listing variables such as 'Name: PLC_connected', 'Name: PFG_cycle_time', and 'Name: reg1' with their respective values.
- PLC Variables:** A list of PLC registers including 'plc_in_reg1', 'plc_in_reg2', 'plc_out_reg1', and 'plc_out_reg2'. A red arrow points from the 'plc_in_reg1' entry to the 'PLC Variables' table.
- ARGE Program:** A logic diagram showing a 'Condition' (true) leading to an 'Action' (Assignment) with the destination 'plc_in_reg1' and expression '1'.

Explaining the Example: The user inserted the value “1” into the PLC’s Output register “0”, bit “0”. The data transfer is verified by observing the PLC registers and the ARGE registers.

Communicating with a Modbus TCP/IP Master – Red Lion

ARGEE blocks have the ability to communicate with a Modbus TCP/IP Master. The Modbus Master can establish communication via registers 0x4000 (register 16384 in decimal) and 0x4400 (register 17408 in decimal). 0x4000 is a read only register, while 0x4400 is a read/write register.

Note: Some Modbus Masters automatically increment the register value by one. For example, register 16384 might be 16385. If the user is having connection issues, the user should try and increment the register value by one.

Example of Communicating with a Modbus TCP/IP Master

The user wants to check and see if data is being passed back and forth between the ARGEE block and the Modbus Master. The first thing the user does is set up the PLC variables.

PLC Variables

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above
plc_out_reg1	PLC->ARGEE ▼	0	0 ▼	Word (16 bit) ▼	unsigned ▼	Delete Add Above

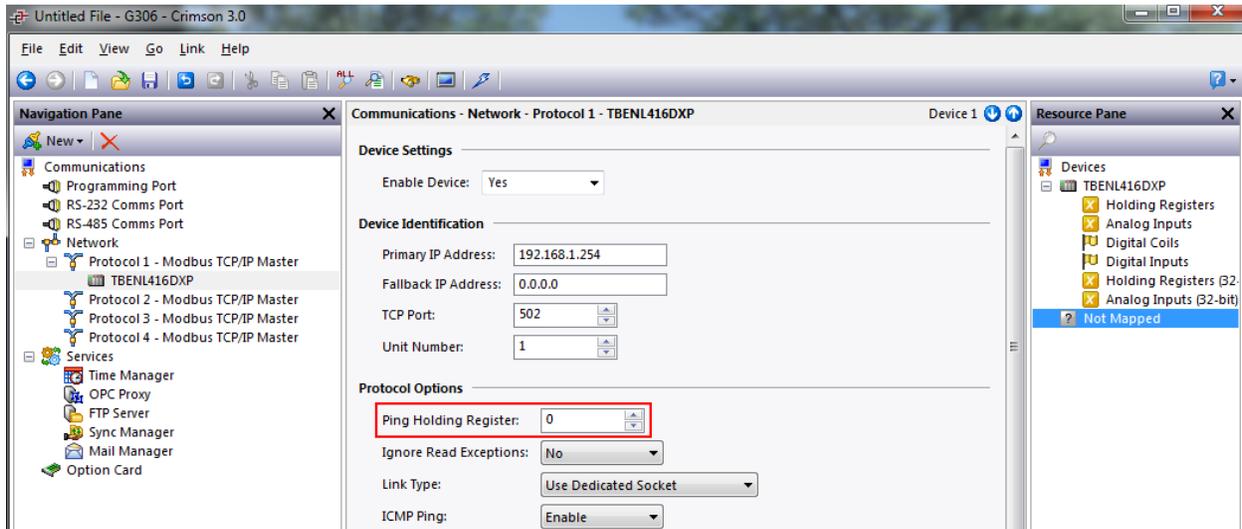
Explaining the Setup: The user creates two PLC Variables, then sets the direction the data will travel. Data transmitted from the ARGEE block to Modbus Master is mapped into register 0x4000(hex) and data size as defined in the ARGEE PLC variable section (ARGEE->PLC). Max input data size is 0x80(hex). Data transmitted from the Modbus Master to the ARGEE block is mapped into register 0x4400(hex), and data size as defined in the ARGEE PLC variable section (PLC->ARGEE). Max output data size is 0x80(hex).

The next step is to write the ARGEE code.

The screenshot shows a graphical user interface for writing ARGEE code. It features a 'Condition' section with a text input field containing the word 'true'. Below this is an 'Actions' section with a table-like structure. The first row is highlighted in cyan and contains an 'Assignment' action. The 'Destination' field is set to 'plc_in_reg1' and the 'Expression' field is set to '1'. At the bottom of the actions list, there is a dropdown menu currently set to 'Assignment' and an 'Add Action' button.

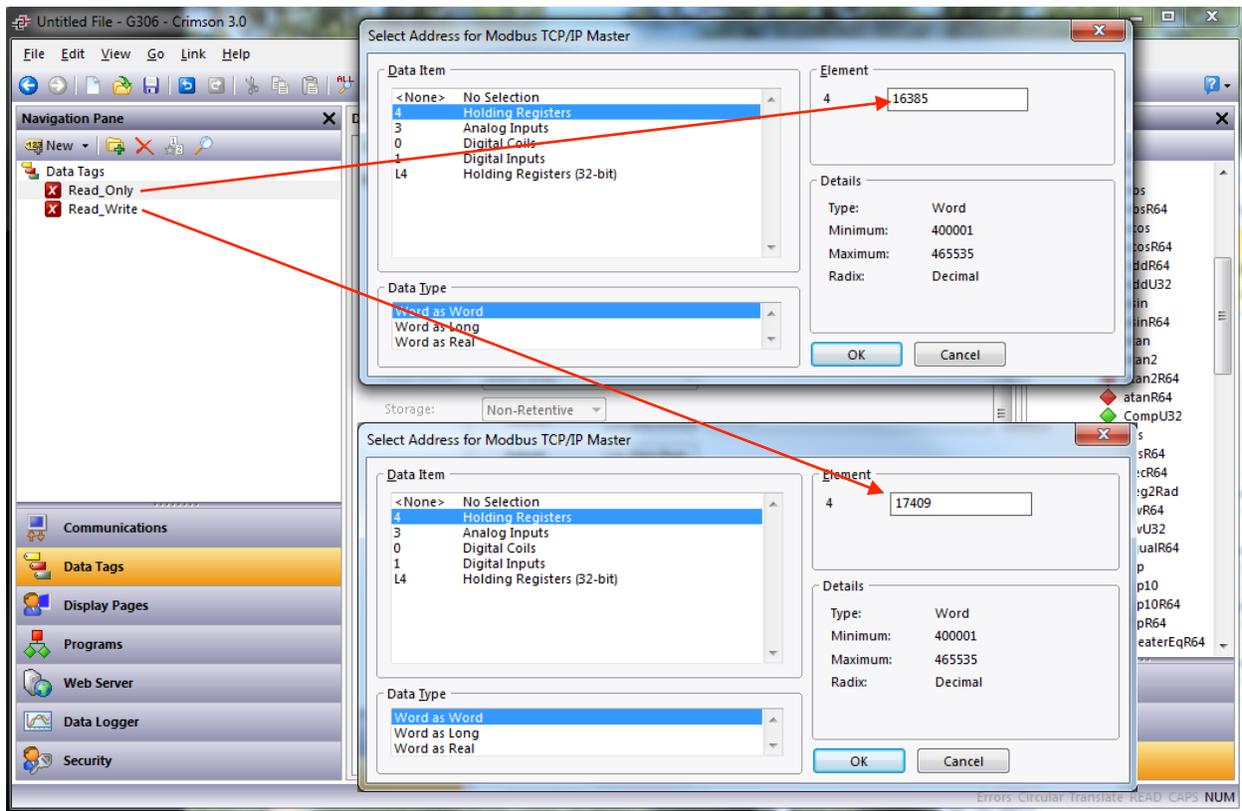
Explaining the Code: The user wrote the value “1” into plc_in_reg1.

The third step is to connect a device to the Modbus Master.



IMPORTANT NOTE: If the user is using a Red Lion HMI, the user needs to set the Ping Holding Register to zero.

The fourth step is to create tags and assign them to the correct registers.



IMPORTANT NOTE: Red Lion Modbus master register addressing = Original address + 1

Example: Original address 0x4000(hex) = 16384
 Red Lion address = 16384 + 1 = 16385

The last step is to connect to the device, place a value in the Modbus TCP Master's Output Register and verify data transfer.

The screenshot shows a red lion HMI on the left and its software interface on the right. The HMI screen displays two variables: 'Read_Only: 1' and 'Read_Write: 1'. The software interface has a yellow header with 'Project Title: TBEN-L4-16DXP (192.168.1.254)' and 'Project Checksum: 13E81'. Below the header are navigation buttons: 'Edit Code', 'View HMI', 'Modify Variables', and 'Reset'. The software interface is divided into three main sections: 'Program Variables', 'PLC Variables', and 'ARGEE Program'. The 'Program Variables' table lists variables like 'Name: PLC_connected', 'Name: PROG_cycle_time', 'Name: reg1', and 'Name: reg2'. The 'PLC Variables' table lists variables like 'Name: tm1', 'Name: tm2', 'Name: cnt1', and 'Name: cnt2'. The 'ARGEE Program' section shows a ladder logic diagram with a 'Condition' set to 'true' and an 'Assignment' action with 'Destination: plc_in_reg1' and 'Expression: 1'. Red arrows point from the '1' values on the HMI screen to the 'plc_in_reg1' variable in the software interface.

Name	Value
PLC_connected	Value: 1
PROG_cycle_time	Value: 5
reg1	Value: 0
reg2	Value: 0

Name	Done	Engaged	Expiration Time	Timer tick
tm1	0	0	0	0
tm2	0	0	0	0
cnt1	0	0	0	0
cnt2	0	0	0	0

Explaining the Example: The user inserted the value “1” into the HMI's Output register “17409”. The data transfer is verified by observing the HMI screen and the ARGEE registers.

Communicating with a PROFINET Master – Siemens

ARGEE blocks have the ability to communicate with a PROFINET Master. The PROFINET Master can establish communication via an ARGEE GSD file.

Example of Communicating with a PROFINET Master

The user wants to check and see if data is being passed back and forth between the ARGEE block and the PROFINET Master. The first thing the user does is set up the PLC variables.

PLC Variables

Name	Direction	Word index	Bit offset	Size	Signed	Actions
plc_in_reg1	ARGEE->PLC ▾	0	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above
plc_out_reg1	PLC->ARGEE ▾	0	0 ▾	Word (16 bit) ▾	unsigned ▾	Delete Add Above

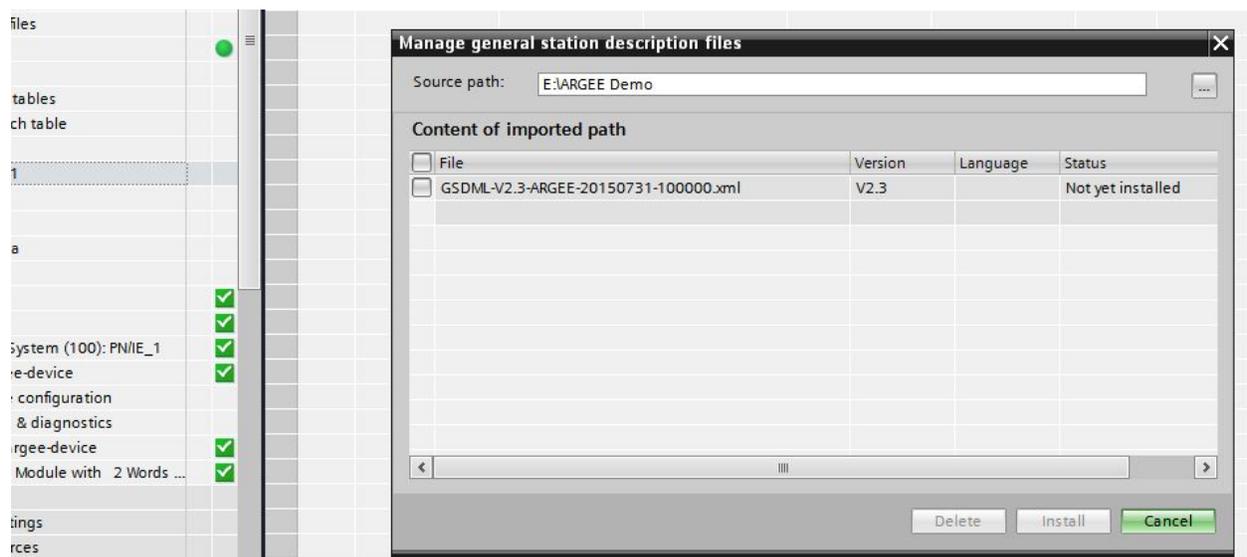
Explaining the Set-up: The user creates two PLC Variables. The user sets the direction the data will travel. Data transmitted from ARGEE to the PLC is mapped into the Siemens input address and the data size is defined in the ARGEE PLC variable section (ARGEE->PLC). Data transmitted from the PLC to ARGEE is mapped into the Siemens output address and the data size is defined in the ARGEE PLC variable section (PLC->ARGEE).

The next step is to write the ARGEE code.

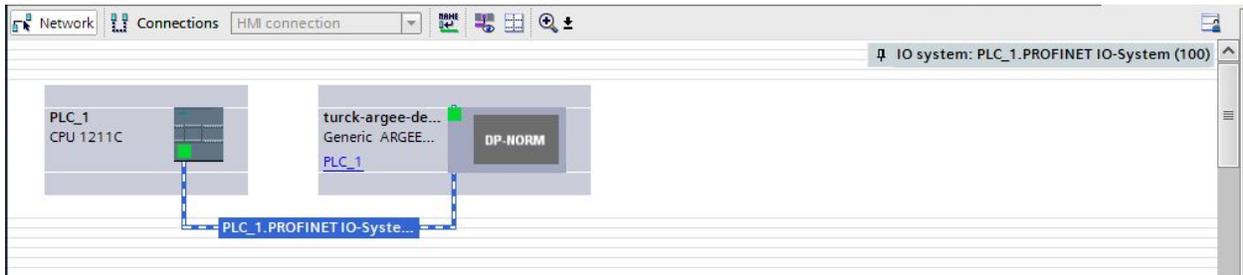
The screenshot shows the ARGEE configuration interface. The 'Condition' field is set to 'true'. Under the 'Actions' section, there is one action: 'Assignment'. The 'Destination' for this action is 'plc_in_reg1' and the 'Expression' is '1'. Below the actions list, there is a dropdown menu set to 'Assignment' and an 'Add Action' button.

Explaining the Code: The user wrote the value “1” into plc_in_reg1.

The third step is to install the ARGEE GSD file.



The fourth step is to add the device to the program.



Explaining the Set up: The user created an ARGEE Device in the devices and networks area.

The fifth step is to set up device addresses.

The screenshot shows the 'Device overview' table in SIMATIC Manager. The table has columns for Module, Rack, Slot, I address, and Q address. The 'I address' and 'Q address' for the ARGEE module are circled in orange.

Module	Rack	Slot	I address	Q address
turck-argee-device	0	0		
PN-IO	0	0 X1		
ARGEE Module with 8 Word...	0	1	68...83	64...79

Explaining the Set up: The user defines the “I address” and “Q address” in the device overview.

The user can now verify the data has been transferred.

The screenshot displays the Siemens SIMATIC Manager interface. The top window shows the 'Watch and force tables' for the project 'ARGEE' on PLC 'PLC_1 [CPU 1211C DGD/DC]'. A table lists variables with their addresses and monitor values. A red arrow points from the 'Monitor value' column to the 'PLC Variables' table in the bottom window. The bottom window shows the 'PLC Variables' table with the value '2' entered for 'plc_out_reg1'.

Name	Address	Display format	Monitor value	Modify value	Comment
%QW68	%QW68	Hex	16#0000	16#0002	
%QW70	%QW70	Hex	16#0000		
%IW68	%IW68	Hex	16#0000		
%IW70	%IW70	Hex	16#0000		

Name	Value
PLC_connected	Value:1
PROG_cycle_time	Value:5
reg1	Value:0
reg2	Value:0
tm1	Done: 0 Engaged: 0 Expiration Time: 0 Timer tick: 0
tm2	Done: 0 Engaged: 0 Expiration Time: 0 Timer tick: 0
cnt1	Done: 0 Engaged: 0 Expiration Time: 0 Timer tick: 0
cnt2	Done: 0 Engaged: 0 Expiration Time: 0 Timer tick: 0
plc_in_reg1	0
plc_in_reg2	0
plc_out_reg1	2
plc_out_reg2	0

Explaining the example: The user inserted the value “2” into the PLC’s Output register and verifies the data transfer.

Using State Variables

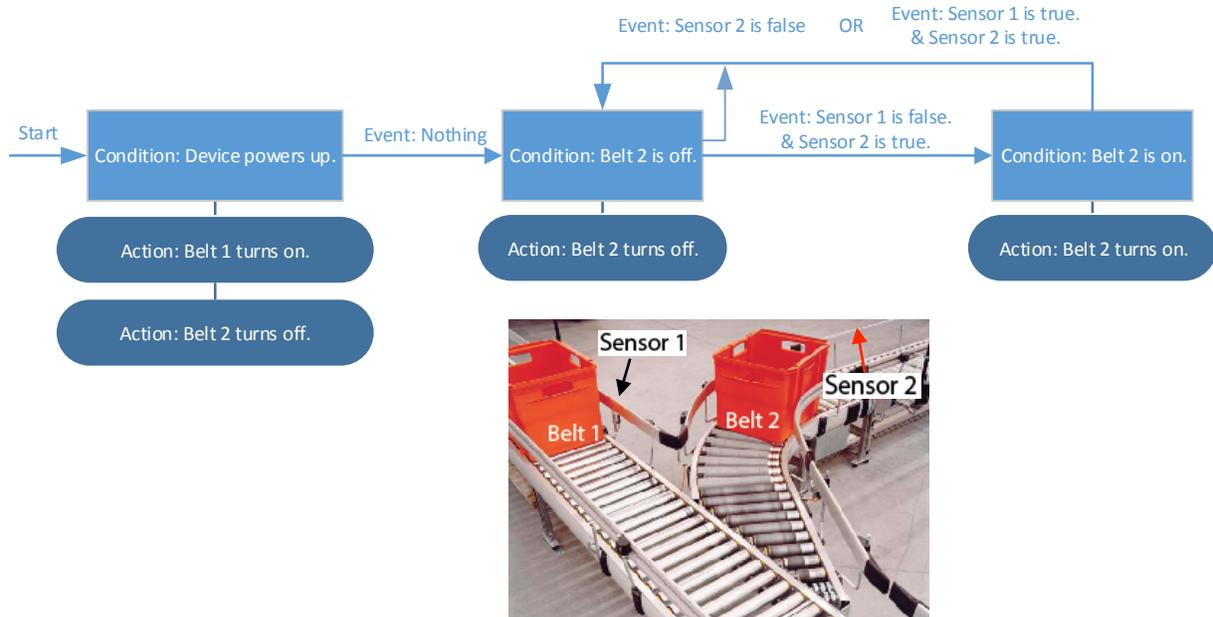
State Variables are helpful in keeping track of the signal as it steps through the code. Before the user creates State Variables, it is a good idea to create a State Machine.

State Machine

A state machine is drawing on a piece of paper that shows how the signal transitions from one state to another.

Example of a State Machine

The user wants to use their ARGEE block to create a Traffic Cop. A Traffic Cop is a device that merges two conveyer belts together without causing a box collision. The first thing the user does is gets out a piece of paper and draws up a State Machine.



Explaining the State Machine: All the States are in light blue boxes. All the Events occur on the arrows. All Actions are in dark blue ovals.

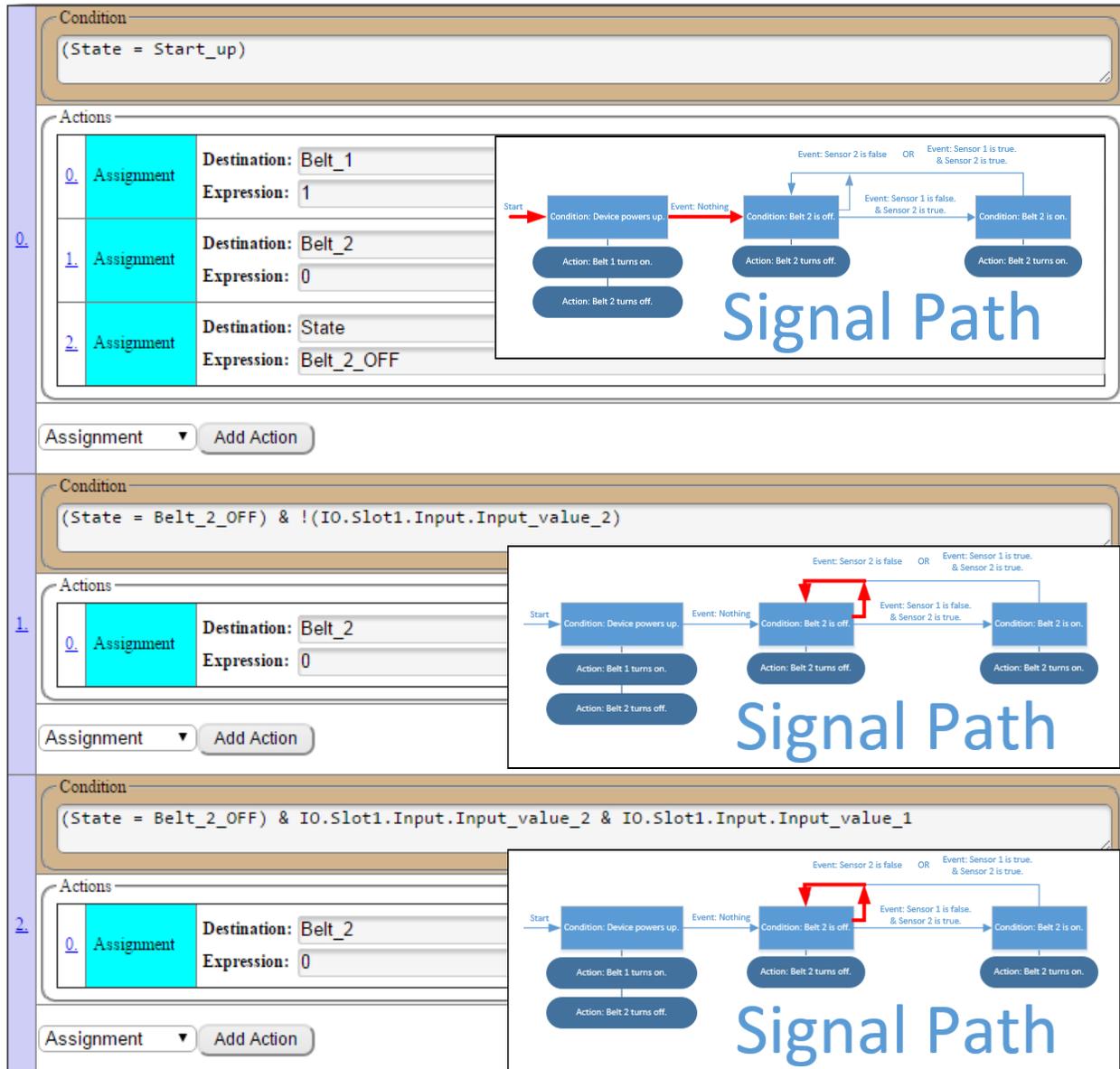
Example of State Variables are on the next page.

Example of a State Variables

The user is satisfied with the Traffic Cop State Machine. The user now creates Program and State Variables.

Program Variables			State Variables		
State	State	init:Start_up	Name	Actions	
		Delete Add Above Init	Start_up	Delete	Add Above
Belt_1	Integer	Delete Add Above Init	Belt_2_OFF	Delete	Add Above
Belt_2	Integer	Delete Add Above Init	Belt_2_ON	Delete	Add Above

Note: Program Variable “State” is initialized to Start-up.



3.

Condition
 $(State = Belt_2_OFF) \& IO.Slot1.Input.Input_value_2 \& !(IO.Slot1.Input.Input_value_1)$

Actions

0.	Assignment	Destination: State	Expression: Belt_2_ON
1.	Assignment	Destination: Belt_2	Expression: 1

Assignment Add Action

Signal Path

4.

Condition
 $(State = Belt_2_ON) \& !(IO.Slot1.Input.Input_value_2)$

Actions

0.	Assignment	Destination: State	Expression: Belt_2_OFF
1.	Assignment	Destination: Belt_2	Expression: 0

Assignment Add Action

Signal Path

5.

Condition
 $(State = Belt_2_ON) \& IO.Slot1.Input.Input_value_1$

Actions

0.	Assignment	Destination: State	Expression: Belt_2_OFF
1.	Assignment	Destination: Belt_2	Expression: 0

Assignment Add Action

Signal Path

Explaining the Example: When the device is powered up, Belt 1 is turned on and Belt 2 is turned off. If Sensor 2 goes true (or a box shows up on Belt 2), ARGEE will check and see if Sensor 1 is true (or if a box is on Belt 1). If Sensor 1 is true then Belt 2 stays off. If Sensor 1 is false, Belt 2 turns on and clears the box on Belt 2.

ARGEE HMI

Many user applications can be enhanced with the use of the ARGEE HMI. The two main ARGEE HMI operations are Editable Fields and Display Fields. General Buttons are used in both types of fields.

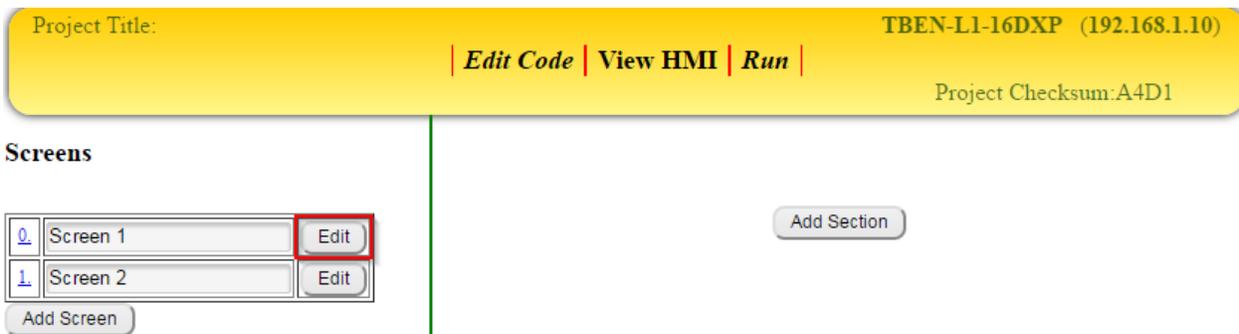
General Buttons

Add Screen

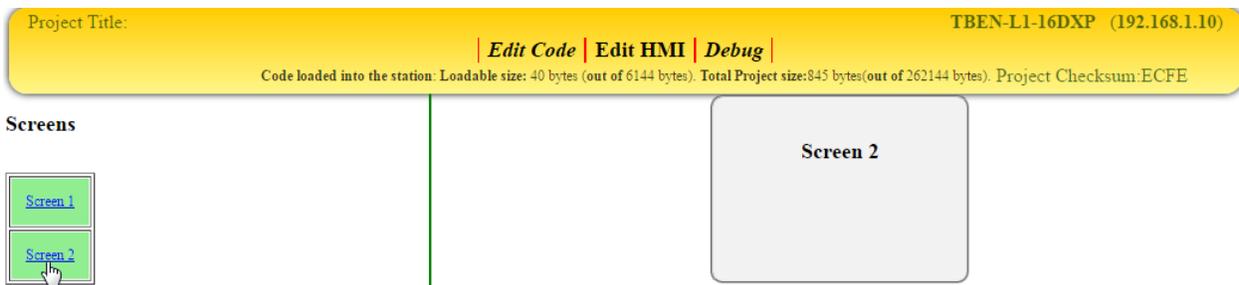
The *Add Screen* button is available under the Edit HMI tab. *Add Screen* allows the user to add several HMI screens to the project.



The user can toggle between multiple screens by clicking on the *Edit* button.



When on the View HMI tab, the user can toggle between screens by clicking on the *Screen* name.



Add Section

The *Add Section* button is available under the Edit HMI tab. The user will click on the *Add Section* button if they want to add more sections to their HMI screen.

The screenshot shows the software interface for editing an HMI screen. At the top, a yellow header bar contains the project title 'Project Title: TBEN-L1-16DXP (192.168.1.10)' and navigation buttons: '| Edit Code | View HMI | Run |'. Below the header, a status bar indicates: 'Code loaded into the station: Loadable size: 40 bytes (out of 6144 bytes). Total Project size:845 bytes(out of 262144 bytes). Project Checksum:ECFE'. The main workspace is split into two panes. The left pane, titled 'Screens', contains a list of screens: 'Screen 1' and 'Screen 2', each with an 'Edit' button, and an 'Add Screen' button at the bottom. The right pane, titled 'Screen 1', shows a configuration window for a section. It has a 'Section Name' field and a dropdown menu currently set to 'Button'. Below the dropdown is an 'Add New Element' button. At the bottom of the right pane, an 'Add Section' button is highlighted with a red rectangular box.

Add New Element

The user can *Add New Elements* to a specific Section of a specific Screen by selecting an *Element* from the drop down arrow and clicking *Add New Element*.

This screenshot is identical to the one above, showing the same software interface. However, in this view, the 'Add New Element' button within the 'Section 1' configuration window is highlighted with a red rectangular box, indicating the step of selecting an element to add to the section.

Editable Fields

The Editable Field elements in the ARGEE HMI are *Enter number*, *Enter state* and *Edit hex number*. The *Button* element is used to submit the new value into the program.

Enter number (and Button)

The *Enter number* element, in conjunction with the *Button* element, allows the user to manually input a value into a register while a program is running.

Example of Enter number (and Button)

For the HMI to compile, the user must first create some code and then click *Edit HMI* from the ARGEE menu tab.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| Run | Debug | Print | **Edit HMI** | View HMI | Project | About | Set Title |
Project Checksum:ECFE

Program Variables			ARGEE Program
Name	Type	Action	Keyboard shortcuts:
PLC_connected	Integer		Press Ctrl-q for list of program variables
PROG_cycle_time	Integer		Press Ctrl-i for list of I/O variables
Submit	Integer	Del	Press Ctrl-f for list of operations
Temporary_Register	Integer	Del	Press Ctrl-s for list of State Names

These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition

Submit

Actions

0	Assignment	Destination: Submit	Expression: 0
1	Assignment	Destination: IO.Slot1.Output.Output_value_3	Expression: Temporary_Register

Assignment Add Action

Add Condition

Explaining the code: The user created two Program Variables: Submit and Temporary_Register. When Submit goes true, the code sets Submit false and loads the value that is in Temporary_Register into Output_value_3.



The user creates an HMI screen and adds an *Enter number* and *Button* element to it.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Run](#) |

Project Checksum:112F4

Screens

[Screen 1](#) [Edit](#)

[Add Screen](#)

Screen 1

Section Name:	
Element type:	Enter number
Name:	Value loaded into Temporary
Destination Variable:	Temporary_Register
Units:	
Element type:	Button
Name:	Submit
Destination Variable:	Submit

Button [Add New Element](#)

[Add Section](#)

Explaining the Example: The user named the Enter number element “Value loaded into Temporary Register”. The user then set the destination variable to be Temporary_Register. The user then named the Button element “Submit”. The user then set the destination variable to be Submit.



The user clicks *Run* to download the code to the block and then clicks *View HMI* to view the HMI.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [Edit HMI](#) | [Debug](#) |

Code loaded into the station: Loadable size: 72 bytes (out of 6144 bytes). Total Project size: 1177 bytes (out of 262144 bytes). Project Checksum: 153B3

Screens

Screen 1

Screen 1

Value loaded into Temporary_Register	<input style="width: 90%;" type="text" value="1"/>
<input type="button" value="Submit"/>	

Explaining the Example: The user entered the value “1” into the editable field. The user then clicked the Submit Button to load that value into Temporary_Register.



To observe the bits moving, the user can click on *Debug* and see that Temporary_Register and Output_value_3 have the same values loaded into them.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Modify Variables](#) | [Reset](#) |

Code loaded into the station: Loadable size: 72 bytes (out of 6144 bytes). Total Project size: 1177 bytes (out of 262144 bytes). Project Checksum: 153B3

Program Variables

Name: PLC_connected	Value: 0
Name: PROG_cycle_time	Value: 5
Name: Submit	Value: 0
Name: Temporary_Register	Value: 1

PLC Variables

Local IO

Slot 0: TBEN-L1-16DXP Diagnostics	
Module_Diagnostics_Available	0
Undervoltage_Field_Supply_V2	0
Undervoltage_Field_Supply_V1	0
Force_Mode_Enabled	0
Slot 1: 16DXP Input	
Input_value_1	0
Input_value_2	0
Input_value_3	1
Input_value_4	0

ARGEE Program

Condition

Submit

Actions

0.	Assignment	Destination: Submit Expression: 0
1.	Assignment	Destination: IO.Slot1.Output.Output_value_3 Expression: Temporary_Register

Enter state

The *Enter state* element is used when multiple State Machines are running on the same device. This feature is useful in recipe applications, RFID applications and even pick-to-light applications.

Example of Enter state

The user wants to toggle between the Beef Stew, Vegetable Stew and Tomato Soup state machines. The user must first create the code and then click *Edit HMI*.

The screenshot displays the HMI configuration software interface. At the top, there is a yellow header bar with the project title and navigation buttons: Run, Debug, Print, Edit HMI (highlighted), View HMI, Project, About, and Set Title. The right side of the header shows the project name 'TBEN-L1-16DXP (192.168.1.10)' and the project checksum '23D0F'.

On the left side, there are three sections:

- Program Variables:** A table with columns for Name, Type, and Actions. It lists variables like PLC_connected, PROC_cycle_time, Submit, Program_Mode, Create_Beef_Stew, Create_Vegetable_Stew, and Create_Tomato_Stew.
- PLC Variables:** A section with an 'Add Variable' button.
- State Names:** A table with columns for Name and Actions, listing states like Beef_Stew, Vegetable_Stew, Tomato_Soup, and their sub-states.

The main area on the right shows a ladder logic diagram for the 'Enter state' function. It consists of three rungs, each with a condition and an assignment action:

- Rung 1:** Condition: Submit. Action: Assignment. Destination: Submit, Expression: 0.
- Rung 2:** Condition: Program_Mode = Beef_Stew. Action: Assignment. Destination: Program_Mode, Expression: 1.
- Rung 3:** Condition: Program_Mode = Vegetable_Stew. Action: Assignment. Destination: Create_Vegetable_Stew, Expression: 1.

Below the diagram, there are buttons for 'Add Condition' and 'Add Action'.

Explaining the code: The user created three State Machines (Beef_Stew, Vegetable_Stew and Tomato_Soup). Each State Machine has its own individual Sub-States (Beef_Stew_State_1/2, Vegetable_Stew_State_1/2, Tomato_Soup_State_1/2) associated with it. The user created five Program Variables. When Submit goes true, the code sets Submit false and loads the value "1" into Program Mode. When Program Mode goes true, it loads the value "1" into the selected stews State Machine. The other three Program Variables (Create_Beef_Stew, Create_Vegetable_Stew, and Create_Tomato_Soup) were created to signify the specific type of stews being created. They don't actually do anything in this code.

The user creates an HMI screen and adds an *Enter state* and *Button* element to it.

The screenshot displays the HMI configuration software interface. At the top, a yellow header bar contains the Project Title, navigation buttons for 'Edit Code', 'View HMI', and 'Run', and the project ID 'TBEN-L1-16DXP (192.168.1.10)'. Below the header, a status bar shows 'Code loaded into the station: Loadable size: 184 bytes (out of 6144 bytes). Total Project size: 1953 bytes (out of 262144 bytes). Project Checksum: 23D0F'. The main workspace is divided into two panes. The left pane, titled 'Screens', shows a list with 'Screen 1' selected and an 'Edit' button. The right pane, titled 'Screen 1', shows the configuration for a section. It includes a 'Section Name' field, a list of elements, and an 'Add Section' button. The elements are:

Element type	Value
Enter state	Program Mode
Name	Program Mode
Destination Variable	Program_Mode
StartValue	Beef_Stew
EndValue	Tomato_Soup
Button	Submit
Name	Submit
Destination Variable	Submit

At the bottom of the right pane, there is a 'Button' dropdown menu and an 'Add New Element' button.

Explaining the Example: The user named the *Enter state* element “Program Mode”. The user then set the *Destination Variable* to be Program_Mode. The user used the *StartValue* and *EndValue* to set the limits on the states that the user wants to display in the HMI drop down menu. The user then named the Button element “Submit”. The user then set the *Destination Variable* to be Submit.

The user clicks *Run* to download the code to the block and then clicks *View HMI* to view the HMI.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [Edit HMI](#) | [Debug](#) |

Code loaded into the station: Loadable size: 184 bytes (out of 6144 bytes). Total Project size:1953 bytes(out of 262144 bytes).Project Checksum:23D0F

Screens

Screen 1

Screen 1

Program Mode	Vegetable_Stew ▾
Submit	

Explaining the Example: The user selects the recipe from the drop down arrow and then clicks the Submit Button to execute the Vegetable_Stew State Machine.



To observe the bits moving, the user can click on *Debug* and see that the Create_Vegetable_Stew register is true.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Modify Variables](#) | [Reset](#) |

Code loaded into the station: Loadable size: 184 bytes (out of 6144 bytes). Total Project size:1973 bytes(out of 262144 bytes).Project Checksum:24347

Program Variables

Name:PLC_connected	Value:0
Name:PROG_cycle_time	Value:5
Name:Submit	Value:0
Name:Program_Mode	Value:Vegetable_Stew
Name:Create_Beef_Stew	Value:0
Name:Create_Vegetable_Stew	Value:1
Name:Create_Tomato_Stew	Value:0

PLC Variables

Local IO

Slot 0:TBEN-L1-16DXP Diagnostics	
Module_Diagnostics_Available	0
Undervoltage_Field_Supply_V2	0
Undervoltage_Field_Supply_V1	0
Force_Mode_Enabled	0
Slot 1:16DXP Input	
Input_value_1	0
Input_value_2	0
Input_value_3	0
Input_value_4	0
Input_value_5	0
Input_value_6	0

ARGEE Program

Condition

Submit

Actions

0.	Assignment	Destination: Submit
		Expression: 0
1.	Assignment	Destination: Program_Mode
		Expression: 1

Condition

Program_Mode = Beef_Stew

Actions

0.	Assignment	Destination: Create_Beef_Stew
		Expression: 1

Condition

Program_Mode = Vegetable_Stew

Actions

0.	Assignment	Destination: Create_Vegetable_Stew
		Expression: 1

Edit hex number

The *Edit hex number* element allows the user to manually input a value (in Hex) into a register while a program is running.

Example of Edit hex number

The user wants to load data into the RFID Write Register. After the code is written, the user clicks on the *Edit HMI* tab.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| *Run* | *Debug* | *Print* | **Edit HMI** | *View HMI* | *Project* | *About* | *Set Title* |

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
Submit	Integer	Delete
RFID_Write_Register	Integer	Delete

Add Variable

PLC Variables

Add Variable

State Names

Add State

ARGEE Program

Keyboard shortcuts:
 Press Ctrl-q for list of program variables
 Press Ctrl-i for list of I/O variables
 Press Ctrl-f for list of operations
 Press Ctrl-s for list of State Names
 These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition

Submit

Actions

0.	Assignment	Destination: Submit	Expression: 0
1.	Assignment	Destination: RFID_Write_Register	Expression: RFID_Write_Register

Assignment Add Action

Add Condition

Explaining the code: The user created two Program Variables, Submit and RFID_Write_Register. When Submit goes true, the code sets Submit false and loads the value that is in RFID_Write_Register into RFID_Write_Register.



The user creates an HMI screen and adds an *Edit hex number* and *Button* element to it.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Run](#) |

Screens

0. Screen 1 Edit

Add Screen

Screen 1

Section Name:

0.	Element type:	Edit hex number
	Name	<input type="text" value="RFID Write Register"/>
	Destination Variable	<input type="text" value="RFID_Write_Register"/>

0.	Element type:	Button
1.	Name	<input type="text" value="Submit"/>
	Destination Variable	<input type="text" value="Submit"/>

Button

Add New Element

Add Section

Explaining the Example: The user named the Edit hex number element “RFID Write Register”, and then set the destination variable to be RFID_Write_Register. The user then named the Button element “Submit”, and then set the destination variable to be Submit.



The user clicks *Run* to download the code to the block and then clicks *View HMI* to view the HMI.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [Edit HMI](#) | [Debug](#) |

Code loaded into the station: Loadable size: 72 bytes (out of 6144 bytes). Total Project size: 1141 bytes(out of 262144 bytes). Project Checksum: 14437

Screens

Screen 1

Screen 1

RFID Write Register	A2	FF	34	BD
<input type="button" value="Submit"/>				

Explaining the Example: The user entered the hex value “A2 FF 34 BD” into the editable field. The user then clicked the Submit Button to load that value into Temporary_Register_1.



To observe the bits moving, the user can click *Debug* and see that RFID_Write_Register has the hex number A2 FF 34 BD loaded into it.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Modify Variables](#) | [Reset](#) |

Code loaded into the station: Loadable size: 72 bytes (out of 6144 bytes). Total Project size: 1141 bytes(out of 262144 bytes). Project Checksum: 14437

Program Variables

Name: PLC_connected	Value: 0
Name: PROG_cycle_time	Value: 5
Name: Submit	Value: 0
Name: RFID_Write_Register	Value: -1560333123

PLC Variables

Local IO

ARGEE Program

Condition					
Submit					
Actions					
0.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="background-color: cyan;">Assignment</td> <td>Destination: Submit</td> </tr> <tr> <td></td> <td>Expression: 0</td> </tr> </table>	Assignment	Destination: Submit		Expression: 0
Assignment	Destination: Submit				
	Expression: 0				
1.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="background-color: cyan;">Assignment</td> <td>Destination: RFID_Write_Register</td> </tr> <tr> <td></td> <td>Expression: RFID_Write_Register</td> </tr> </table>	Assignment	Destination: RFID_Write_Register		Expression: RFID_Write_Register
Assignment	Destination: RFID_Write_Register				
	Expression: RFID_Write_Register				

Display Fields

The Display Field elements in the ARGEE HMI are *Display number or state*, *Display number with valid range*, *Display hex number*.

Display number or state

The *Display number or state element* is a feature that allows the user to see the current value in a particular register.

Example of Display number or state

The user wants to monitor the value in Temporary_Register.

Project Title: TBEN-L1-16DXP (192.168.1.10)

[Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#)

Project Checksum:14437

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
Submit	Integer	Delete Add Action
Temporary_Register	Integer	Delete Add Action

Add Variable

ARGEE Program

Keyboard shortcuts:
Press Ctrl-q for list of program variables
Press Ctrl-i for list of I/O variables
Press Ctrl-f for list of operations
Press Ctrl-s for list of State Names
These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

PLC Variables

Add Variable

State Names

Add State

Condition

Submit

Actions

0. Assignment	Destination: Submit
	Expression: 0
1. Assignment	Destination: Temporary_Register
	Expression: 1

Assignment Add Action

Add Condition

Explaining the code: The user created two Program Variables, Submit and Temporary_Register. When Submit goes true, the code sets Submit false and loads the value “1” into Temporary_Register.

The user creates an HMI screen and adds a *Display number or state* and *Button* element to it.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Run](#) |

Project Checksum:14437

Screens

0.	Screen 1	Edit
----	----------	------

Add Screen

Screen 1

Section Name:

0.	Element type:	Display number or state
0.	Name	Value in Temporary Registe
	Expression	Temporary_Register
	Units	<input style="width: 80%;" type="text"/>

0.	Element type:	Button
1.	Name	Submit
	Destination Variable	Submit

Button ▼

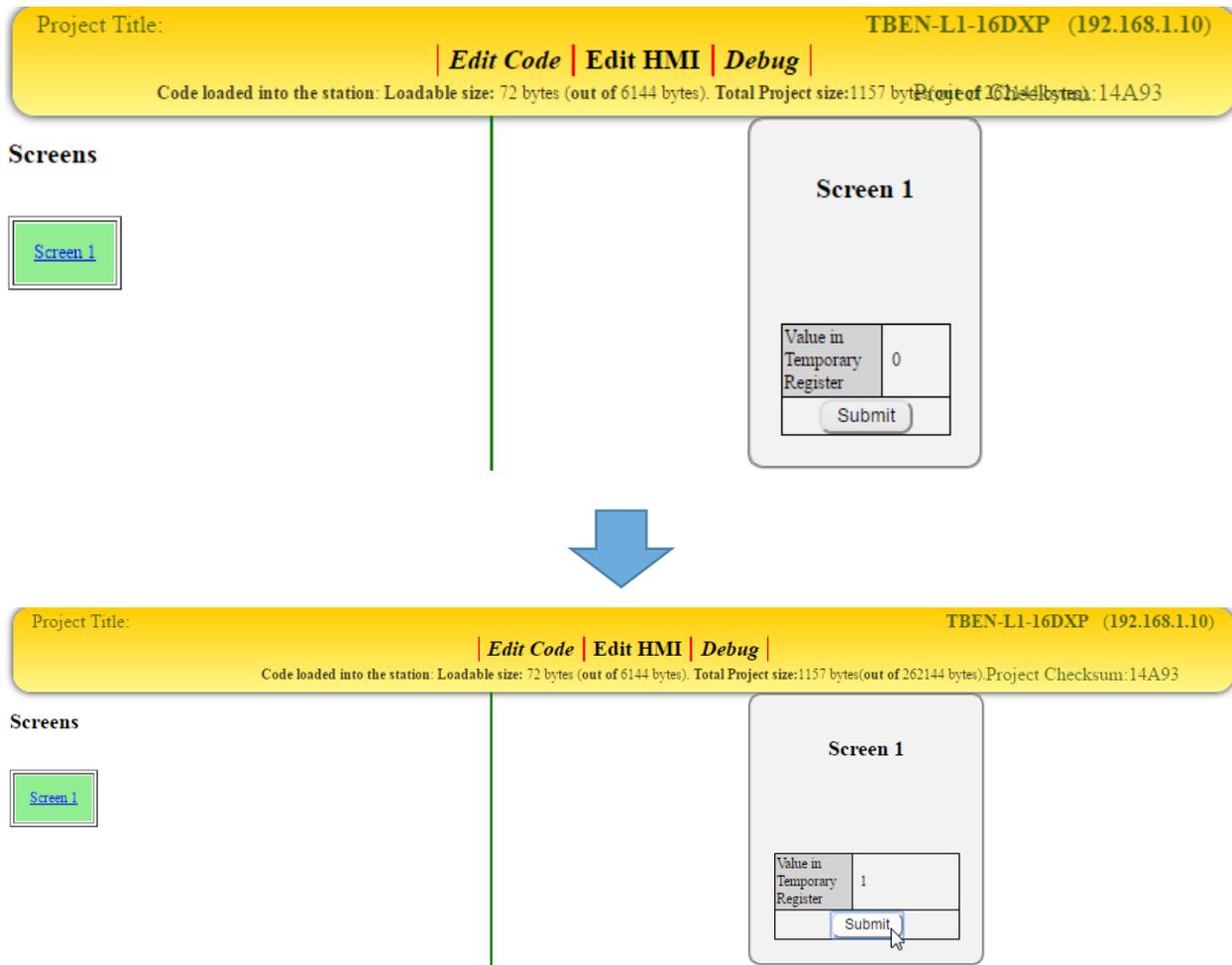
Add New Element

Add Section

Explaining the Example: The user named the Display number or state element “Value in Temporary Register”, and then sets the destination variable to be Temporary_Register. Next, the user named the Button element “Submit”, and then set the destination variable to be Submit.



The user clicks *Run* to download the code to the block and then clicks *View HMI* to view the HMI.



Explaining the Example: When the HMI first loads up, the value “0” is in Temporary_Register. When the user presses the Submit button, the value “1” is loaded in to Temporary_Register.

Display number with valid range

The *Display number with valid range* element is a feature that allows the user to see the current value in a particular register. It also lets the user know when that value has exceeded a preset range.

Example of Display number with valid range

The user wants to monitor the value in Temporary_Register. The user also wants a visual notification when that value has exceeded a preset range.

Project Title:
TBEN-L1-16DXP (192.168.1.10)

| [Run](#) | [Debug](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#) |

Project Checksum:14A93

Program Variables

Name	Type	Actions
PLC_connected	Integer	
PROG_cycle_time	Integer	
Submit	Integer ▾	Delete Add A
Temporary_Register	Integer ▾	Delete Add A

Add Variable

ARGEE Program

Keyboard shortcuts:
 Press Ctrl-q for list of program variables
 Press Ctrl-i for list of I/O variables
 Press Ctrl-f for list of operations
 Press Ctrl-s for list of State Names
 These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition

Submit

Actions

0	Assignment	Destination: Submit	Expression: 0
1	Assignment	Destination: Temporary_Register	Expression: 2

Assignment ▾ Add Action

Add Condition

Explaining the code: The user created two Program Variables, Submit and Temporary_Register. When Submit goes true, the code sets Submit false and loads the value "2" into Temporary_Register.



The user creates an HMI screen and adds a *Display number with valid range* and *Button* element to it.

Project Title: TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Run](#) |

Project Checksum: 14A93

Screens

0 Screen 1 Edit

Add Screen

Screen 1

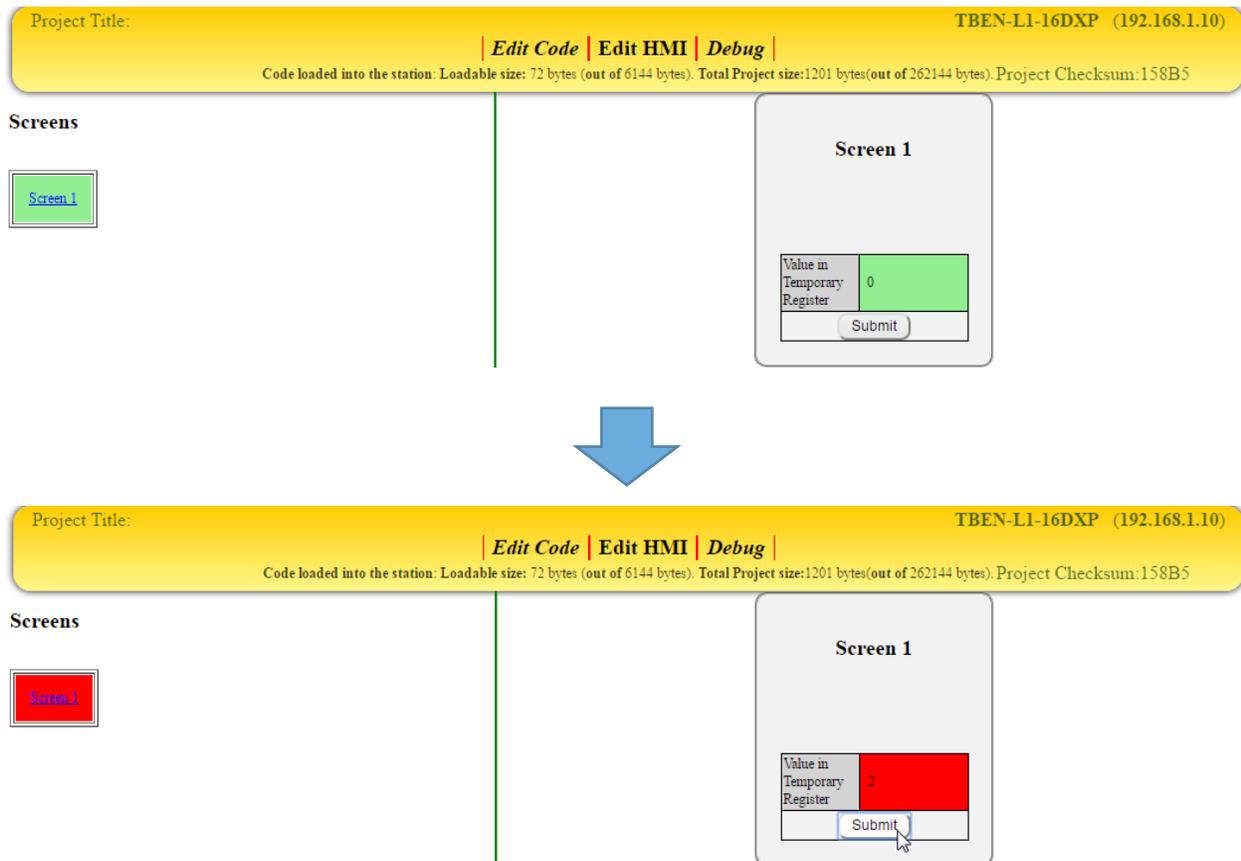
Section Name:	
Element type:	Display number with valid range
Name	Value in Temporary Register
Expression	Temporary_Register
Units	
NormalRangeMin	0
NormalRangeMax	1
Element type:	Button
Name	Submit
Destination Variable	Submit
Button	
Add New Element	

Add Section

Explaining the Example: The user named the Display number with valid range element “Value in Temporary Register”. The user then set the destination variable to be Temporary_Register. The user set the range minimum to be “0” and the range maximum to be “1”. The user then named the Button element “Submit”, and set the destination variable to be Submit.



The user clicks *Run* to download the code to the block and then clicks *View HMI* to view the HMI.



Explaining the Example: When the HMI first loads up, the value “0” is in Temporary_Register. When the user presses the Submit button, the value “2” is loaded in to Temporary_Register. The value “2” exceeds the preset maximum so the HMI goes red.

Display hex number

The *Display hex number* element is a feature that allows the user to see the current value in a particular register displayed in Hex.

Example of Display hex number

The user wants to monitor the value in Temporary_Register. The user also wants to display that value in hex.

Project Title: TBEN-L4-16DXP (192.168.1.254)

| [Run](#) | [Test](#) | [Print](#) | [Edit HMI](#) | [View HMI](#) | [Project](#) | [About](#) | [Set Title](#) |

Project Checksum:14821

Program Variables

Name	Type	Action
PLC_connected	Integer	
PROG_cycle_time	Integer	
Submit	Integer	De
Temporary_Register	Integer	De

[Add Variable](#)

PLC Variables

[Add Variable](#)

State Names

[Add State](#)

ARGEE Program

Keyboard shortcuts:
Press Ctrl-q for list of program variables
Press Ctrl-i for list of I/O variables
Press Ctrl-f for list of operations
Press Ctrl-s for list of State Names
These shortcuts are used to write variables and expressions in all the screens

In order to configure the IO of the station, follow the [Link](#)

Condition

Submit

Actions

0. Assignment	Destination: Submit	Expression: 0
1. Assignment	Destination: Temporary_Register	Expression: 45842

Assignment [Add Action](#)

[Add Condition](#)

Explaining the code: The user created two Program Variables, Submit and Temporary_Register. When Submit goes true, the code sets Submit false and loads the value “45842” into Temporary_Register.



The user creates an HMI screen and adds a *Display number or state* and *Button* element to it.

Project Title:
TBEN-L1-16DXP (192.168.1.10)

| [Edit Code](#) | [View HMI](#) | [Run](#) |

Project Checksum:158B5

Screens

0 Screen 1 Edit

Add Screen

Screen 1

Section Name:

Element type:	Display hex number
0	Name Value in Temporary Registe
	Expression Temporary_Register

1	Element type: Button
	Name Submit
	Destination Variable Submit

Button

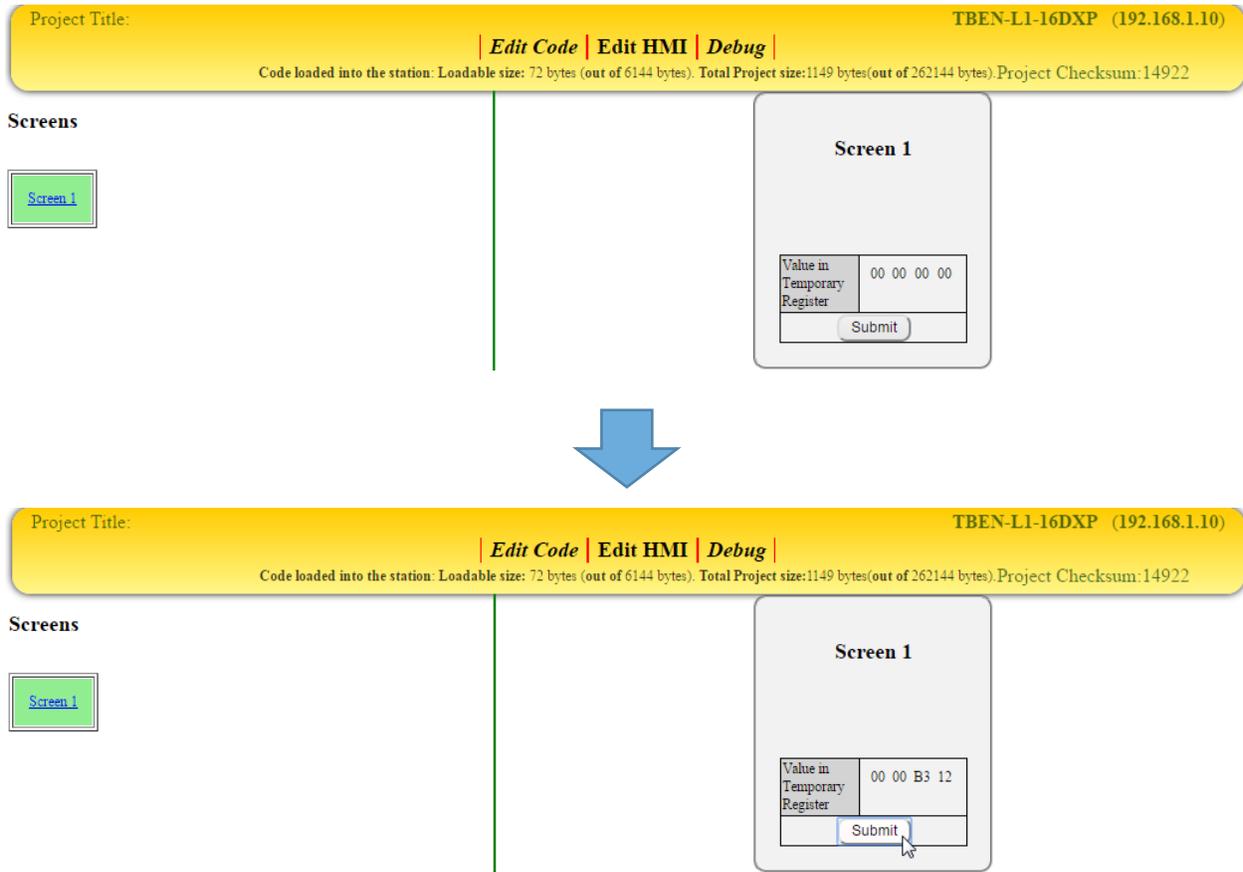
Add New Element

Add Section

Explaining the Example: The user named the Display hex number “Value in Temporary Register”, and then set the destination variable to be Temporary_Register. Next, the user named the Button element “Submit” and set the destination variable to be Submit.



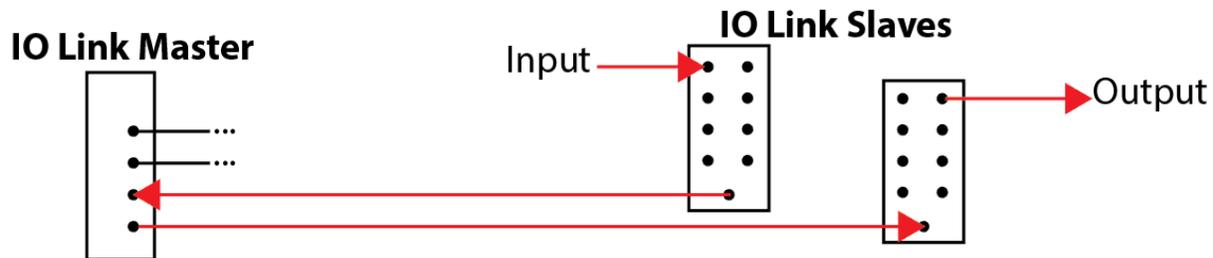
The user clicks *Run* to download the code to the block and then clicks *View HMI* to view the HMI.



Explaining the Example: When the HMI first loads up, the value “00 00 00 00” is in Temporary_Register. When the user presses the Submit button, the value “45842” is loaded in to Temporary_Register. The value “45842” is transformed into hex and is displayed in the HMI.

Working with IO-Link

When a user combines IO-Link technology with AGREE, the application solutions that can be created become endless. IO-Link can support digital and analog signals. Because there are so many IO-Link configurations, it is recommended that the user read the IO-Link user manual before attempting any IO-Link applications.



IMPORTANT NOTE: When using IO-Link over EtherNet/IP or a Modbus/TCP, the user must change the Input and Output Data Mapping parameter from “swap 16 bit” to “direct” in the webserver.

Example of IO-Link

The user wants to use a digital input on an IO-Link slave to turn on a digital output on a different IO-Link slave using EtherNet/IP. The first thing the user has to do is change the Input and Output Data Mapping parameter from “swap 16 bit” to “direct”.

BLCEN-8M12LT-4IOL-4AI4AO-VI
Embedded Website of BLCEN Block I/O Module

admin@192.168.1.202 [Logout]

Slot 1 - BLC-4IOL > Parameters >

Slot 1 - Parameters

IO-Link channel 0 - Mode	IO-Link without validation
IO-Link channel 1 - Mode	IO-Link without validation
IO-Link channel 2 - Mode	IO-Link without validation
IO-Link channel 3 - Mode	IO-Link without validation
IO-Link channel 0 - Data storage mode	deactivated, clear
IO-Link channel 1 - Data storage mode	deactivated, clear
IO-Link channel 2 - Data storage mode	deactivated, clear
IO-Link channel 3 - Data storage mode	deactivated, clear

IO-Link channel 0 - Input data mapping	swap 16 bit
IO-Link channel 1 - Input data mapping	swap 16 bit
IO-Link channel 2 - Input data mapping	swap 16 bit
IO-Link channel 3 - Input data mapping	direct
IO-Link channel 0 - Input data length	2 byte
IO-Link channel 1 - Input data length	2 byte
IO-Link channel 2 - Input data length	2 byte
IO-Link channel 3 - Input data length	2 byte
IO-Link channel 0 - Output data mapping	swap 16 bit
IO-Link channel 1 - Output data mapping	swap 16 bit
IO-Link channel 2 - Output data mapping	swap 16 bit
IO-Link channel 3 - Output data mapping	swap 16 bit

Explaining the Example: The user logged into the webserver and changed the Input and Output Data Mapping parameter from “swap 16 bit” to “direct”, and then clicked submit.

The next thing the user does is look at the data map of the two IO-Link slaves and determine which Input and Output to link together.

Slave 1 Data Map (A TBIL-M1-16DIP Connected to Port 0)

Process Data									
	Byte	Bit 7 MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 LSB
Inputs	0	DI8 C4P2 (B)	DI7 C4P4 (A)	DI6 C3P2 (B)	DI5 C3P4 (A)	DI4 C2P2 (B)	DI3 C2P4 (A)	DI2 C1P2 (B)	DI1 C1P4 (A)
	1	DI16 C8P2 (B)	DI15 C8P4 (A)	DI14 C7P2 (B)	DI13 C7P4 (A)	DI12 C6P2 (B)	DI11 C6P4 (A)	DI10 C5P2 (B)	DI9 C5P4 (A)

C... = slot no., P... = pin no.

Slave 2 Data Map (A TBIL-M1-16DXP Connected to Port 1)

Process Data									
	Byte	Bit 7 MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 LSB
Inputs	0	DI8 C4P2 (B)	DI7 C4P4 (A)	DI6 C3P2 (B)	DI5 C3P4 (A)	DI4 C2P2 (B)	DI3 C2P4 (A)	DI2 C1P2 (B)	DI1 C1P4 (A)
	1	DI16 C8P2 (B)	DI15 C8P4 (A)	DI14 C7P2 (B)	DI13 C7P4 (A)	DI12 C6P2 (B)	DI11 C6P4 (A)	DI10 C5P2 (B)	DI9 C5P4 (A)
Outputs	0	DO8 C4P2 (B)	DO7 C4P4 (A)	DO6 C3P2 (B)	DO5 C3P4 (A)	DO4 C2P2 (B)	DO3 C2P4 (A)	DO2 C1P2 (B)	DO1 C1P4 (A)
	1	DO16 C8P2 (B)	DO15 C8P4 (A)	DO14 C7P2 (B)	DO13 C7P4 (A)	DO12 C6P2 (B)	DO11 C6P4 (A)	DO10 C5P2 (B)	DO9 C5P4 (A)

C... = slot no., P... = pin no.

Explaining the user's decision: The user wants Connector 7 Pin 2 (Port B) on the input block to turn on Connector 8 Pin 4 (Port A) on the output block.

IO-Link Command Structure

Condition

I.O.Slot1.Input.IO_Link_input_data_word_0.2 = 1

- = The I/O location and data word that is being targeted.
- = The specific bit in the data word that is being targeted.
- = The value that is being loaded into that bit location

Condition

I.O.Slot1.Input.IO_Link_input_data_word_0.13 = 1

Actions

0. Assignment Destination: I.O.Slot1.Output.IO_Link_output_data_word_1.14
 Expression: 1

Assignment Add Action

Condition

I.O.Slot1.Input.IO_Link_input_data_word_0.13 = 0

Actions

0. Assignment Destination: I.O.Slot1.Output.IO_Link_output_data_word_1.14
 Expression: 0

Assignment Add Action

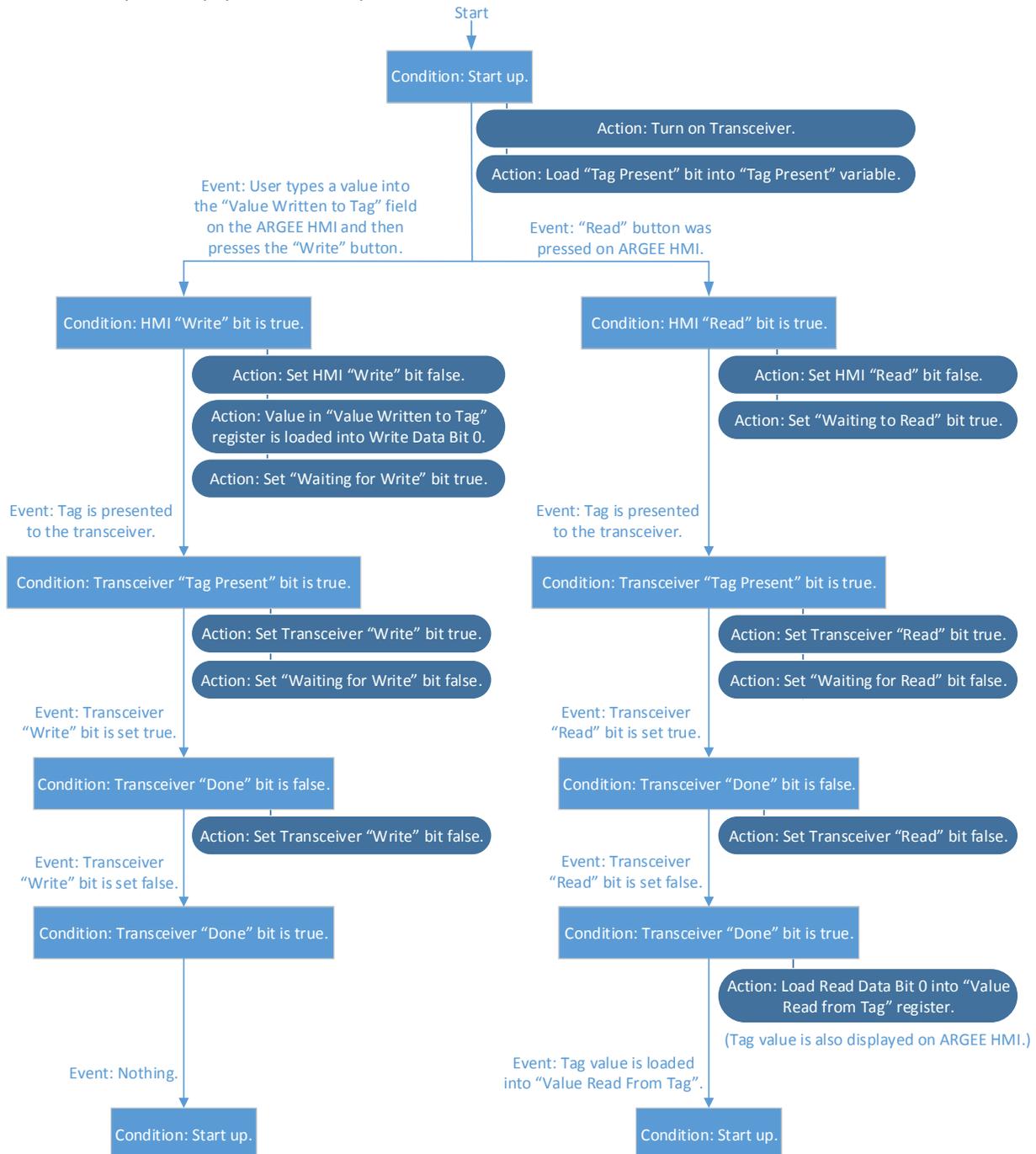
Explaining the example: When Connector 7 Port B on the input slave block is true, Connector 8 Port A on the output slave block is true. When Connector 7 Port B on the input slave block is false, Connector 8 Port A on the output slave block is false.

Working with RFID

If a user needs to solve a simple tracking application, using RFID technology (powered by ARGEE) might be the solution. Many factors influence RFID Read/Write applications. The user can reference the RFID user manual for more information about RFID.

Example of RFID

The user wants to create an ARGEE HMI that can read from and write to RFID tags. The first thing the user must do is break out a pen and paper and draw up a state machine.



Explaining the RFID State Machine: When the device powers up, the transceiver gets turned on. From the ARGEE HMI screen the user can select two paths: *Read from Tag*, or *Write to Tag* path. If the user wants to read a tag value, simply click the *Read* button on the HMI and present a tag to the transceiver. If the user wants to write to a tag, just type a value into the HMI, click *Write* on the HMI, and then present a tag to the transceiver. After a Read/Write has occurred, the program goes back to the “Startup” state and waits for the next command.

Writing the Code:

Program Variables

Read	Integer
Write	Integer
State	State
Value_Read_From_Tag	Integer
Value_Written_To_Tag	Integer
Tag_Present	Integer
Waiting_To_Read	Integer
Waiting_To_Write	Integer

State Variables

Start_up
Read_State_Machine
Read_Bit_Is_High
Read_Wait_For_Done_Bit
Write_State_Machine
Write_Bit_Is_High
Write_Wait_For_Done_Bit

Note: Set initial state for Program Variable “State” to “Start_up”

The screenshot displays a state machine configuration interface with two states. Each state is defined by a condition and a set of actions.

- State 0:**
 - Condition:** State = Start_up
 - Actions:**
 0. Assignment: Destination: IO.Slot1.Output.XCVR_0, Expression: 1
 1. Assignment: Destination: Tag_Present, Expression: IO.Slot1.Input.TP_0
- State 1:**
 - Condition:** State = Read_State_Machine & IO.Slot1.Input.TP_0
 - Actions:**
 0. Assignment: Destination: IO.Slot1.Output.Read_0, Expression: 1
 1. Assignment: Destination: Waiting_To_Read, Expression: 0
 2. Assignment: Destination: State, Expression: Read_Bit_Is_High

Condition

`(State = Read_Bit_Is_High) & (!IO.Slot1.Input.Done_0)`

Actions

0.	Assignment	Destination: IO.Slot1.Output.Read_0	Expression: 0
1.	Assignment	Destination: State	Expression: Read_Wait_For_Done_Bit

Assignment ▾ Add Action

Condition

`(State = Read_Wait_For_Done_Bit) & IO.Slot1.Input.Done_0`

Actions

0.	Assignment	Destination: Value_Read_From_Tag	Expression: IO.Slot1.Input.Read_data_0_0
1.	Assignment	Destination: State	Expression: Start_up

Assignment ▾ Add Action

Condition

`State = Write_State_Machine & IO.Slot1.Input.TP_0`

Actions

0.	Assignment	Destination: IO.Slot1.Output.Write_0	Expression: 1
1.	Assignment	Destination: Waiting_To_Write	Expression: 0
2.	Assignment	Destination: State	Expression: Write_Bit_Is_High

Assignment ▾ Add Action

5.

Condition
 (State = Write_Bit_Is_High) & (!IO.Slot1.Input.Done_0)

Actions

0.	Assignment	Destination: IO.Slot1.Output.Write_0	Expression: 0
1.	Assignment	Destination: State	Expression: Write_Wait_For_Done_Bit

Assignment ▾ Add Action

6.

Condition
 (State = Write_Wait_For_Done_Bit) & IO.Slot1.Input.Done_0

Actions

0.	Assignment	Destination: State	Expression: Start_up
----	------------	--------------------	----------------------

Assignment ▾ Add Action

7.

Condition
 Read

Actions

0.	Assignment	Destination: State	Expression: Read_State_Machine
1.	Assignment	Destination: Read	Expression: 0
2.	Assignment	Destination: Waiting_To_Read	Expression: 1

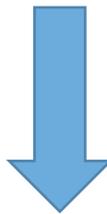
Assignment ▾ Add Action

Condition	
Write	

Actions	
0. Assignment	Destination: State Expression: Write_State_Machine
1. Assignment	Destination: Write Expression: 0
2. Assignment	Destination: IO.Slot1.Output.Write_data_0_0 Expression: Value_Written_To_Tag
3. Assignment	Destination: Waiting_To_Write Expression: 1

Explaining the Code: The user created the “Read” and “Write” program variables so the ARGEE HMI buttons would have a way to start the RFID operation. The “Tag Present”, “Waiting To Read” and “Waiting To Write” program variables are status bits that can be displayed on the HMI. Everything else is identical to the state machine explanation.

The RFID example continues on the next page.



Building the RFID HMI

After the code is written, the user clicks *Edit HMI* from the ARGEE menu bar.

Project Title: **BLCEN-6M12LT-2RFID-S-8XSG-P (192.168.1.254)** | [Edit Code](#) | [View HMI](#) | [Run](#) | Project Checksum:4AB32

Screens

Section Name: Read

Element type:	Display number or state
0. Name	Read Value
Expression	Value_Read_From_Tag
Units	

1. Element type: Button

Name	Read
Destination Variable	Read

Button

Section Name: Write

Element type:	Enter number
0. Name	Write Value
Destination Variable	Value_Written_To_Tag
Units	

1. Element type: Button

Name	Write
Destination Variable	Write

Button

Section Name: Status

Element type:	Display number or state
0. Name	Tag Present
Expression	Tag_Present
Units	

1. Element type: Display number or state

Name	Ready to Read
Expression	Waiting_To_Read
Units	

2. Element type: Display number or state

Name	ReadyTo Write
Expression	Waiting_To_Write
Units	

Button

Explaining the RFID HMI: The user creates a “Read” section with two elements. One element displays the tag value and the other is a button that starts the “Read” operation. They also create a “Write” section with two elements. One element allows the user to enter a value and the other is a button that starts the “Write” operation. The “Status” section just displays status bits.

Working with the RFID HMI

To start working with the RFID HMI, the users clicks *Run* and then *View HMI* from the ARGEE menu bar.

Project Title: **BLCEN-6M12LT-2RFID-S-8XSG-P (192.168.1.254)** | [Edit Code](#) | [Edit HMI](#) | [Test](#) | Project Checksum:4ADBE

Screens

[RFID Sample](#)

RFID Sample

Read

Read Value	0
<input type="button" value="Read"/>	

Write

Write Value	0
<input type="button" value="Write"/>	

Status

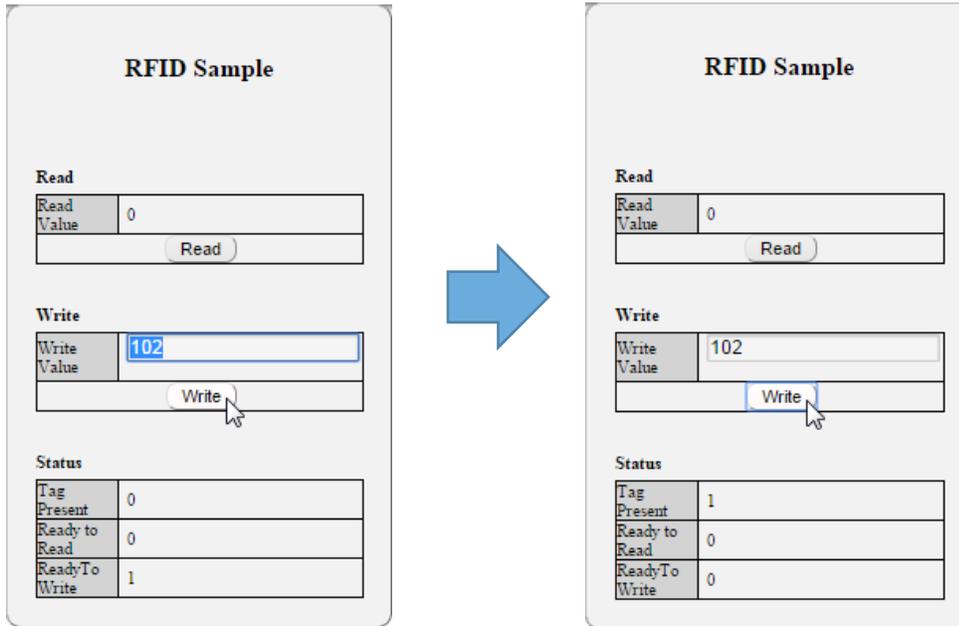
Tag Present	0
Ready to Read	0
ReadyTo Write	0

The RFID example continues on the next page.



Writing to a tag

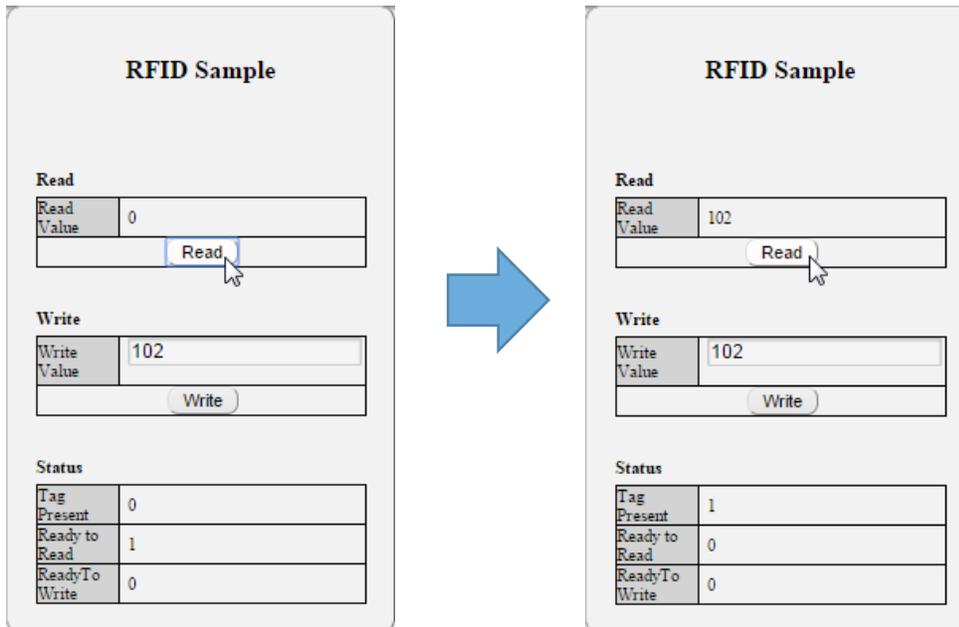
If the user wants to write to a tag, they must first type a value into the *Write Value* field and click the *Write* button.



Explaining “Write”: When the user types a value into the *Write Value* field and clicks *Write*, the “Ready To Write” status bit is true. When a tag is presented to the transceiver, the “Ready To Write” status bit goes false and the “Tag Present” bit goes true.

Reading from a tag

If the user wants to read tag, simply click the *Read* button and then present a tag to the transceiver.



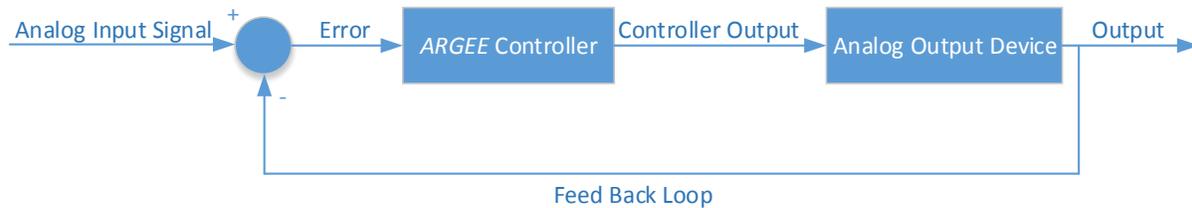
Explaining “Read”: When the user clicks “Read”, the “Ready To Read” status bit is true. When a tag is presented to the transceiver, the “Ready To Read” status bit goes false and the “Tag Present” bit goes true. The tag value is displayed in the “Read Value” field.

Working with Analog

If the user wants to use an Analog input signal to track errors and make corrections to an Analog output signal (Similar to a proportional Controller), they no longer need a PLC. ARGEE has the ability to apply logic and math to analog signals.

Explaining a Proportional Controller: A proportional controller continuously calculates the difference between the output and the input. The purpose of a proportional controller is to minimize the difference (error) by adjusting the controller's output.

Proportional Controller Example



Example of Analog - Proportional Controller

The user wants to create a simple proportional controller, where an Analog input signal inversely controls an Analog output signal. The user must first teach the analog input sensor what the minimum and maximum ranges of the application are (Read sensor data sheet to learn how to teach minimum and maximum ranges). The user is using a 4-20 mA input signal and outputting a 0-10 VDC signal.

Write the Code

Condition	
true	
Actions	
0. Assignment	Destination: IO.Slot2.Output.Output_value_4 Expression: 32767 - IO.Slot2.Input.Input_value_0
Assignment	Add Action

Explaining the Code: Analog sensors use 16 bit signed integers. Therefore the range of the analog input signal is from -32767 -> +32767. The user want's an inversely proportional controller, so they are taking 32767 – Input_value_0 and loading that value into Output_value_4.

Appendix

I/O Variable Definitions

Slot “0” Diagnostics Definitions

Module_Diagnostics_Available : Module Diagnostics Bit
Station_Configuration_Changed : Station Configuration Changed Bit.
Overcurrent_Isys : Station Overcurrent Register Bit
Overvoltage_Field_Supply_V1 - Overvoltage_Field_Supply_V2 : Station Overvoltage Register Bit
Undervoltage_Field_Supply_V1 - Undervoltage_Field_Supply_V1 : Station Under Voltage Register Bit
Modulebus_Communication_Lost : Module communication register Bit
Modulebus_Configuration_Error : Module Error Bit
Force_Mode_Enabled : Force Mode Enabled Bit

Slot 1 or 2 Input Definitions

Input_Value_0 – Input_Value_7 : Input Channel Registers
XCVR_DETUNED_0 - XCVR_DETUNED_1 : Transceiver Detuned Bit
TFR_0 – TFR_1 : Transfer Data Bit
TP_0 – TP_1 : Tag Present Bit
XCVR_ON_0 - XCVR_ON_1 : Transceiver On Bit
XCVR_CON_0 - XCVR_CON_1 : Transceiver Connected Bit
Error_0 – Error_1 : Error Bit
Busy_0 – Busy_1 : Busy Bit
Done_0 – Done_1 : Done Bit
Error_code_0_0 - Error_code_2_0 : Error Code Bits
Read_data_0_0 – Read_data_7_0 : Read Data Registers

Diagnostics Definitions

Output_signal_overcurrent_1 - Output_signal_overcurrent_16 : Signal Overcurrent Error Bit
Overcurrent_on_sensor_group : Sensor Overcurrent Error Bit
Overcurrent_supply_VAUX1/2_at_channels_1-7 : Supply Overcurrent Error Bit
Overcurrent_VAUX1/2_Digital_In_CH1-16: AUX Power Overcurrent Error Bit
Measued_value_out_of_range_0 - Measued_value_out_of_range_3 : Measured Value Out of Range Bit
Wire_break_0 – Wire_break_3 : Wire Break Bit. Used for wire break detection.
Hardware_failure_0 – Hardware_failure_7 : Hardware Failure Bit
Output_value_out_of_range_4 - Output_value_out_of_range_7: Output Value Out of Range Bit
Output_signal_overcurrent_0 - Output_signal_overcurrent_16 : Output Signal Overcurrent Bit
Transc_param_not_supported_0/1: Transceiver Parameter Not Supported Bit
Module_parameter_invalid_0/1: Module Parameter Invalid Bit
Hardware_failure_transceiver_0/1: Transceiver Hardware Failure Bit
Transc_power_supply_error_0/1: Transceiver Power Supply Error Bit

Slot 1 or 2 Output Definitions

Output_value_0 – Output_value_7 : Output channel register.

Reset_0 – Reset_1 : Transceiver Reset Bit

XCVR_Info_0 - XCVR_Info_1 : Transceiver Information Bit

TAG_Info_0 - TAG_Info_1 : Tag Information Bit

Write_0 – Write_1 : Write Bit

Read_0 – Read_1 : Read Bit

Tag_ID_0 – Tag_ID_1 : Tag ID Bit

Next_0 – Next_1 : Next Bit

XCVR_0 – XCVR_1 : Turn Transceiver On Bit

Byte_count_0 – Byte_count_2 : The Byte Count Bytes.

Domain_0 – Domain_1 : Domain Bit

Address_0 – Address_1 : Set Read/Write Address Bit

Write_data_0_0 - Write_data_7_0 : Write Registers