

Your Global Automation Partner

**TURCK**

# ARGE 3

# Reference Manual

MA3000  
0521B

<b>1</b>	<b>General Information</b>	<b>9</b>
1.1	About these instructions	9
1.2	Explanation of symbols used	9
1.3	Contents	10
1.4	Feedback about these instructions	10
1.5	Technical support	10
<b>2</b>	<b>Preface</b>	<b>11</b>
2.1	What is ARGEE 3?	11
2.2	Features of ARGEE 3	11
2.3	What are ARGEE's advantages and limitations?	11
2.4	What products support ARGEE?	12
2.5	Who should use this manual?	12
2.6	What is the purpose of this manual?	12
<b>3</b>	<b>Logging into ARGEE</b>	<b>13</b>
3.1	Opening the Environment	13
3.2	Logging into the Program Mode	13
3.3	Welcome to Flow Chart	13
<b>4</b>	<b>Flow Chart</b>	<b>14</b>
4.1	The Basics	14
4.2	Condition	14
4.3	Operations	14
4.4	Actions	15
4.5	Clean Empty Rungs	15
4.6	Add Empty Rungs	16
4.7	Delete All Rungs	16
4.8	Timers	17
4.9	Counters	17
4.10	Internal Reg	18
4.11	Flow Chart Menu Bar	18
4.11.1	Run	18
4.11.2	Debug (ARGEE Flow)	19
4.11.3	Open/Save As	19
4.11.4	New Project	20
4.11.5	Convert to ARGEE PRO	20
4.11.6	Set Title	20
4.11.7	About	21
4.11.8	Flowchart	21
<b>5</b>	<b>ARGEE PRO</b>	<b>22</b>
5.1	The Basics	22
5.2	Variables and Expressions	22

5.3	Condition	23
5.4	Actions	24
5.4.1	Assignment	24
5.4.2	Coil	24
5.4.3	Timer Start	25
5.4.4	Timer On	26
5.4.5	Timer Off	27
5.4.6	Trace	28
5.4.7	Comment	29
5.4.8	Count Up	29
5.4.9	Count Down	29
5.4.10	Reset Counter	30
5.4.11	Call	31
5.4.12	How Actions respond to Conditions	32
5.5	Program Variables	33
5.5.1	Variable Name	33
5.5.2	Variable Types	33
5.5.3	Add Variable	36
5.5.4	Program Variables Context Menu	36
5.6	Alias Variables	38
5.7	Main Task	38
5.7.1	Adding Conditions to the Main Task	39
5.7.2	Adding Actions to the Main Task	40
5.7.3	Main Task Context Menu	41
5.8	Function Blocks	43
5.8.1	The Basics	43
5.8.2	Function Block Options	43
5.8.3	Function Block Segments	44
5.8.4	Function Block Statements	44
5.8.4.1	While	45
5.8.4.2	For	46
5.8.4.3	If	47
5.8.4.4	Else If	47
5.8.4.5	Else	48
5.9	Libraries	48
5.9.1	What is a Library?	48
5.9.2	Creating a Library	48
5.9.3	Importing a Library	49
5.10	HMI Screens	50
5.11	Keyboard Shortcuts	51
5.11.1	List of Keyboard Shortcuts:	51
5.12	ARGEE PRO Menu Bar	53
5.12.1	Debug (ARGEE PRO)	53
5.12.2	Print	53
5.12.3	IO Config (I/O Configuration)	53
5.12.4	HMI	53
5.12.5	Project	54
5.12.6	Edit Code	54
5.12.7	Delete Project	54

5.12.8	Run Without Source	54
5.12.9	ARGEE PRO Advanced Mode	55
<b>6</b>	<b>ARGEE PRO Advanced Mode</b>	<b>56</b>
6.1	The Basics	56
6.2	Function Block Types	57
6.2.1	Regular	57
6.2.2	Task (Multitasking)	57
6.3	Wait Until	57
<b>7</b>	<b>Debugger</b>	<b>58</b>
7.1	Debugger Information	58
7.1.1	Single Task	58
7.1.2	Multiple Tasks	58
7.1.3	Break Points	58
7.1.4	Trace	58
7.1.5	Order of Operation – Calls & Function Blocks	59
7.2	Debug Menu Bar (ARGEE PRO)	60
7.2.1	Halt	60
7.2.2	Step	60
7.2.3	Continue	61
7.2.4	Modify Vars (Modify Variables)	61
7.2.5	Finish Modifications	61
<b>8</b>	<b>ARGEE Simulation Mode</b>	<b>62</b>
8.1	Opening the Environment	62
8.2	Logging into Simulation Mode	62
8.3	Selecting Device to Simulate	62
8.3.1	Flow Chart Simulation Mode	63
8.3.2	Pro Simulation Mode	63
<b>9</b>	<b>ARGEE HMI</b>	<b>64</b>
9.1	The Basics	64
9.2	HMI Screen	64
9.2.1	Sections	64
9.2.1.1	Display Number/State/String	65
9.2.1.2	Display Number with Valid Range	66
9.2.1.3	Enter Number/String	67
9.2.1.4	Enter State	69
9.2.1.5	Submit Action	71
9.3	HMI Grid Screen	72
9.3.1	HMI Grid Screen	72
9.3.2	Grid Row	73
9.3.3	Grid Cell	73
9.3.4	Grid Element	74
9.3.4.1	Display Value	75
9.3.4.2	Enter Value	75
9.3.4.3	Button	77
9.3.4.4	Static Text	77
9.3.4.5	Screen List	78



9.3.4.6	Static Graphics	79
9.3.4.7	Multi-State Display String	80
9.3.4.8	Multi-State Display Graphics	81
9.3.4.9	Dropdown List	82
9.3.4.10	Display Value with Health	83
9.3.4.11	Link	84
9.4	HMI Image Group	86
9.5	HMI Formatting Tips	87
9.5.1	Cell Spacing in a HMI	87
9.5.2	Row Spacing in a HMI	88
<b>10</b>	<b>PLC Connectivity</b>	<b>90</b>
10.1	Communicating with EtherNet/IP Master – RSLogix5000 / Studio5000	90
10.2	Communicating with a PROFINET Master – SIMATIC STEP 7	92
10.3	Communicating with a Modbus TCP/IP Master – Crimson 3	94
10.4	Communicating with a Turck PLC or TX500 Series HMI – CODESYS 3	96
10.4.1	EtherNet IP	96
10.4.2	PROFINET	98
10.4.3	Modbus TCP/IP	100
<b>11</b>	<b>Appendix I - Definitions</b>	<b>102</b>
11.1	Built-in Functions (Ctrl-f)	102
11.2	Built-in Functions Menu	103
11.2.1	Strings/Arrays	103
11.2.1.1	String Length	104
11.2.1.2	String Left	105
11.2.1.3	String Right	106
11.2.1.4	String Middle	107
11.2.1.5	String Copy	108
11.2.1.6	String Concatenate	108
11.2.1.7	String Compare	109
11.2.1.8	String to Integer	110
11.2.1.9	Integer to String	112
11.2.1.10	Array Initialize	114
11.2.2	Timer	114
11.2.2.1	Start Timer	114
11.2.2.2	Timer Expired	115
11.2.2.3	Timer Count	116
11.2.3	Counter	116
11.2.3.1	Counter Expired	117
11.2.3.2	Counter Count	118
11.2.4	Math	118
11.2.4.1	Addition	119
11.2.4.2	Subtraction	119
11.2.4.3	Multiplication	119
11.2.4.4	Division	119
11.2.4.5	Modulo	120
11.2.4.6	Absolute Value	121
11.2.4.7	Minimum Value	121
11.2.4.8	Maximum Value	122
11.2.5	Brackets	123
11.2.6	Boolean Logic	123

11.2.6.1	Boolean AND	124
11.2.6.2	Boolean OR	124
11.2.6.3	Boolean NOT	124
11.2.7	Compare	125
11.2.7.1	Greater Than	125
11.2.7.2	Greater Than or Equal to	125
11.2.7.3	Less Than	126
11.2.7.4	Less Than or Equal to	126
11.2.7.5	Equal	126
11.2.7.6	Not Equal	127
11.2.8	Trigger	127
11.2.8.1	Change of State (F_COS)	127
11.2.8.2	Rising Edge Trigger (R_TRIG)	128
11.2.8.3	Falling Edge Trigger (F_TRIG)	129
11.2.9	Bit Operations	129
11.2.9.1	Get Bits	129
11.2.9.2	Set Bits	130
11.2.10	Advanced IO/PLC Array	131
11.2.10.1	Get IO Input Integer	132
11.2.10.2	Set IO Output Integer	133
11.2.10.3	Set IO Parameters Integer	133
11.2.10.4	Get IO Diagnostics Integer	134
11.2.10.5	Get IO Input Array	135
11.2.10.6	Set IO Output Array	135
11.2.10.7	Get IO Diagnostics Array	135
11.2.10.8	Get PLC Input Array	135
11.2.10.9	Set PLC Output Array	136
11.2.10.10	Write Data Stream	136
11.2.10.11	Read Data Stream	136
11.2.11	Protocol Conversion	137
11.2.11.1	Little-endian, Get 16 Bits	137
11.2.11.2	Big-endian, Get 16 Bits	138
11.2.11.3	Little-endian, Get 32 Bits	139
11.2.11.4	Big-endian, Get 32 Bits	140
11.2.11.5	Little-endian, Set 16 Bits	141
11.2.11.6	Big-endian, Set 16 Bits	141
11.2.11.7	Little-endian, Set 32 Bits	142
11.2.11.8	Big-endian, Set 32 Bits	142
11.3	ARGEE Security Features	142
11.3.1	Visual Behavior	142
11.3.2	Connection Behavior	143
11.3.2.1	EtherNet IP Master	143
11.3.2.2	Modbus TCP Master	143
11.3.2.3	PROFINET Master	143
11.3.3	Password Protection – ARGEE Environment	144
11.3.4	Source Code Protection – Run Without Source	146
11.4	System Performance	147
11.4.1	Scan Cycle Information	147
11.4.2	IO Variable Formats	147
11.4.3	Defining Variable Types – (Advanced Definitions)	149
11.5	I/O Variable Definitions	151
11.5.1	Slot “0” Diagnostics Definitions	151
11.5.2	Slot 1 or 2 Input Definitions	151
11.5.3	Slot 1 or 2 Output Definitions	152

<b>12</b>	<b>Appendix II – Example Code</b>	<b>153</b>
12.1	How to Erase a Project from a Device	153
12.1.1	Running an empty Project	153
12.1.2	Using the Webserver Page	153
12.1.3	Using the Turck Service Tool	154
12.2	Trace Example	155
12.3	How to Call a Function Block	156
12.4	Creating and Importing Structure Text (ST View)	157
12.4.1	Example of Exporting an ARGEE Project as Structure Text	157
12.4.2	Example of Importing Structure Text and Converting it into an ARGEE Project.	158
12.5	How to Export a CSV File	160
12.5.1	HMI export of arrays	160
12.5.2	Example of Exporting a CSV	160
12.6	Advanced Application Examples	162
12.6.1	Working with IO-Link	162
12.6.1.1	Working with IO-Link	162
12.6.1.2	Acyclic Communication – Read	163
12.6.1.3	Acyclic Communication – Write	164
12.6.2	Working with RFID	165
12.6.3	Working with Analog	168
12.7	Advanced Analog Example – Inclinator	169
12.7.1	Working with Encoders	171
12.7.2	Working with State Variables	172
12.7.2.1	State Machine	172
12.7.2.2	State Variables	173
12.7.3	Working with User-Defined Data Types	177
12.7.3.1	Referencing Internal Function Block Variables	178
12.7.3.2	User-Defined Data Types as Arguments to other Function Blocks	179
12.7.4	Working with Hex Values	180
12.7.5	Advanced Bitwise Operations – Bit Masking	181
12.7.5.1	What are Bitwise Operations?	181
12.7.5.2	What is Bit Masking?	181
12.7.5.3	Example of Bit Masking	181
12.7.6	Nesting Function Blocks	181
12.7.7	Advanced HMI Example – Tank monitoring with graphics	182
<b>13</b>	<b>Appendix III – Libraries</b>	<b>188</b>
13.1	MISC	188
13.1.1	MISC_wait_ms	188
13.1.2	MISC_array_to_string	188
13.1.3	MISC_sort	189
13.1.4	MISC_filter_sample_into_array	190
13.1.5	MISC_reset_filter	191
13.1.6	MISC_NUMBER_st	191
13.1.7	MISC_copy_byte_to_array	192
13.1.8	Float_to_String	193
13.2	Technology	194
13.2.1	BLCEN_RFIDS_Routines	194
13.2.2	BLCEN_RFIDS_Read	194
13.2.3	BLCEN_RFIDS_Write	195

13.2.4	TBEN_S2_RFID_READ	196
13.2.5	TBEN_S2_RFID_WRITE	197
13.2.6	TBEN_IOL_AsyncRead	198
13.2.7	TBEN_IOL_AsyncWrite	199

# 1 General Information

## 1.1 About these instructions

The following user manual describes the setup, functions, and use of the system. It helps you to plan, design, and implement the system for its intended purpose.

**Note\*:** Please read this manual carefully before using the system. This will prevent the risk of personal injury or damage to property or equipment. Keep this manual safe during the service life of the system. If the system is passed on, be sure to transfer this manual to the new owner as well.

## 1.2 Explanation of symbols used

### Warnings

Action-related warnings are placed next to potentially dangerous work steps and are marked by graphic symbols. Each warning is initiated by a warning sign and a signal word that expresses the gravity of the danger. The warnings have absolutely to be observed:



#### DANGER!

DANGER indicates an immediately dangerous situation, with high risk, the death or severe injury, if not avoided.



#### WARNING!

WARNING indicates a potentially dangerous situation with medium risk, the death or severe injury, if not avoided.



#### ATTENTION!

ATTENTION indicates a situation that may lead to property damage, if it is not avoided.



#### NOTE

In NOTES you find tips, recommendations and important information. The notes facilitate work, provide more information on specific actions and help to avoid overtime by not following the correct procedure.

---

#### ➤ CALL TO ACTION

This symbol identifies steps that the user has to perform.

#### ➔ RESULTS OF ACTION

This symbol identifies relevant results of steps.

## 1.3 Contents

Contents of this manual/guide:

- Overview of the ARGEE Manual Content
- How to access the ARGEE Environment
- A general overview and walkthrough of the ARGEE Flow Chart
- A general overview and walkthrough of ARGEE PRO
- A general overview and walkthrough of ARGEE PRO Advanced Mode
- A detailed explanation of the ARGEEs Debugger
- A detailed explanation of Simulation Mode
- A detailed explanation of the ARGEE HMI
- A detailed explanation of PLC Connectivity
- Appendix I - Definitions
- Appendix II – Example Code

## 1.4 Feedback about these instructions

We make every effort to ensure that these instructions are as informative and as clear as possible. If you have any suggestions for improving the design or if some information is missing in the document, please send your suggestions to [techdoc@turck.com](mailto:techdoc@turck.com).

## 1.5 Technical support

For additional support, email inquiries to [appsupport@turck.com](mailto:appsupport@turck.com), or call Application Support at 763-553-7300, Monday-Friday 8AM-5PM CST.

## 2 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?
- What products support ARGEE?
- Who should use this manual?
- What is the purpose of this manual?
- What content is in the ARGEE 3 reference manual?

### 2.1 What is ARGEE 3?

ARGEE 3 is the programming software that is used to put logic inside Turck's multi-protocol block I/O devices. This can be done in anyone of three different coding formats Flow Chart, ARGEE PRO, and ARGEE PRO Advanced. 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.

### 2.2 Features of ARGEE 3

The new features in ARGEE 3 include:

- Alias Variables
- Floating Point
- Function Blocks
- Improved HMI
- More Memory
- While, For, If, Else, Wait Until, and Call statements

### 2.3 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
- 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

## 2.4 What products support ARGEE?

Multiprotocol Ethernet Block I/O devices

- TBEN-L Family
- TBEN-S Family
- FEN20 Family
- BL Compact Family

For more information, please contact Turck Application Engineers at: [AppSupport@turck.com](mailto:AppSupport@turck.com)

## 2.5 Who should use this manual?

Use this manual if you are responsible for designing, installing, programming or troubleshooting 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.

## 2.6 What is the purpose of this manual?

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

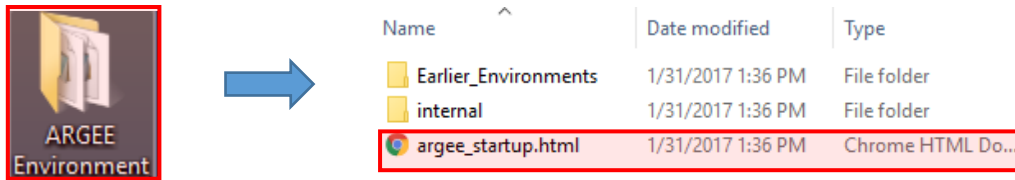
- Teaches the user how to use the ARGEE Flow Chart
- Teaches the user about syntax in ARGEE PRO
- Teaches the user about syntax in ARGEE PRO Advanced Mode
- Provides code for common applications
- Defines all the tag names associated with Turck I/O cards



## 3 Logging into ARGEE

### 3.1 Opening the Environment

- Open the ARGEE Environment and double click on argee\_startup.html.

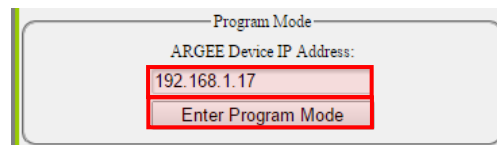


#### NOTE

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

### 3.2 Logging into the Program Mode

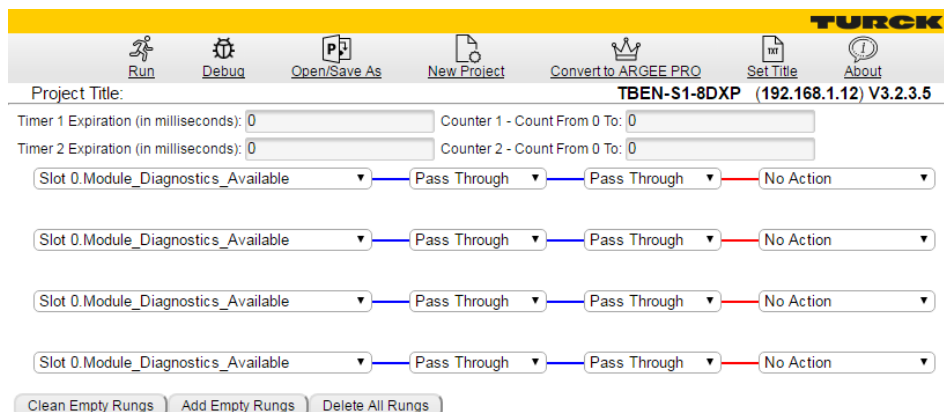
- Type **your** device's IP Address into the ARGEE Device IP Address text box, and then click Enter Program Mode.



#### NOTE

Simulation Mode is explained in chapter [8 ARGEE Simulation Mode](#).

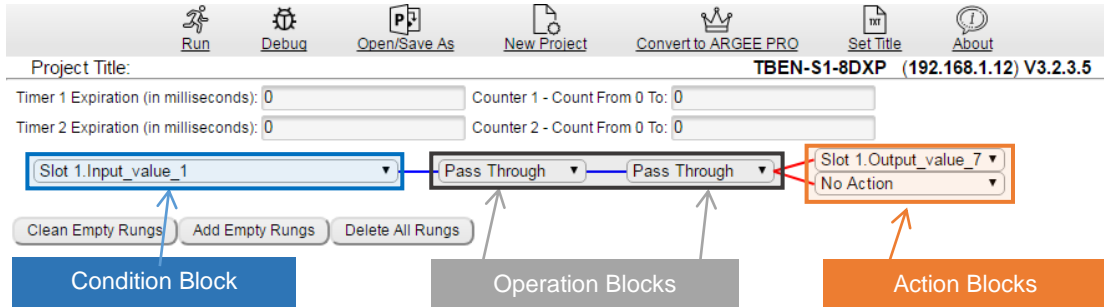
### 3.3 Welcome to Flow Chart



## 4 Flow Chart

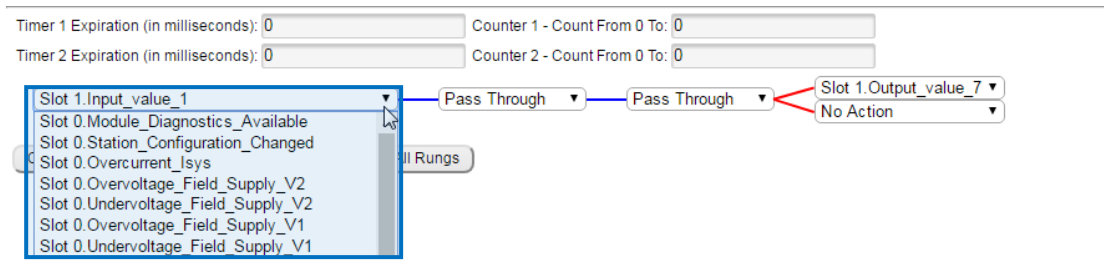
### 4.1 The Basics

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, two counters, and two internal registers.



### 4.2 Condition

The Condition Block contains input conditions. The input conditions that the user sees correspond to the device the user is connected to. Other included input conditions are: Timer X expired, Counter X expired, and Internal Reg X.

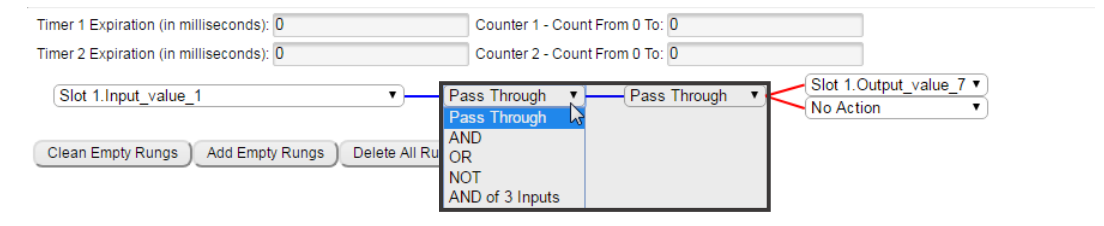


#### NOTE

Expired functions are discussed in chapter [5.3 Condition](#). Internal Regs (Reg = register) are discussed later in this chapter in section [4.9 Internal Reg](#).

### 4.3 Operations

The Operation Blocks contain various Boolean operations. If no operation is desired, select *Pass Through*.

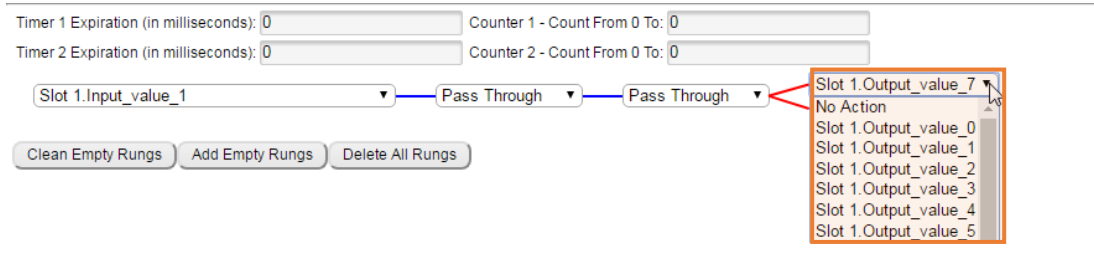


#### NOTE

Boolean Logic is discussed in the Appendix [11.2.7 Boolean Logic](#).

## 4.4 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, CTU Counter X, RESET Counter X, and Internal Reg X.



### NOTE

TON Timer X (**T**imer **O**N Timer X), CTU Counter X (**C**oun**T** Up Counter X), and RESET Counter X are discussed in chapter [5.4 Actions](#).

## 4.5 Clean Empty Rungs

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



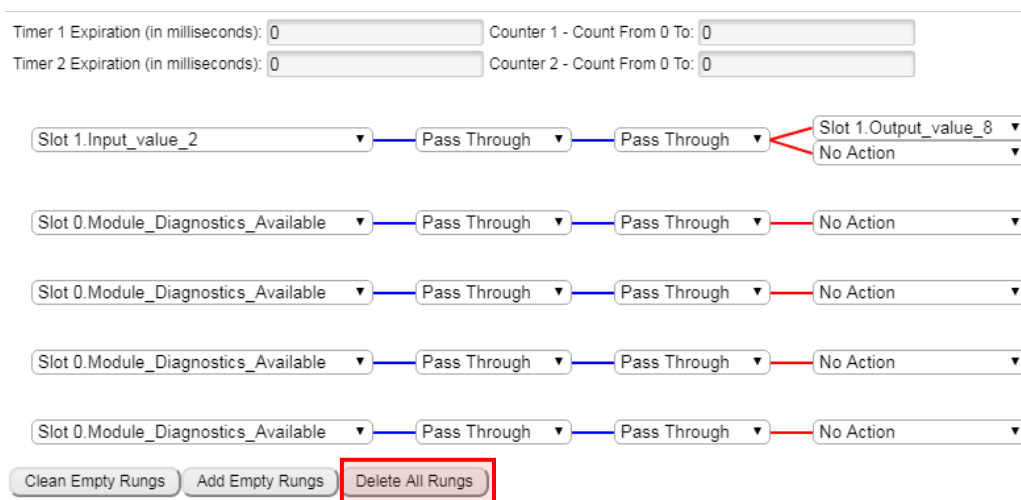
## 4.6 Add Empty Rungs

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



## 4.7 Delete All Rungs

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



Timer 1 Expiration (in milliseconds):	<input type="text" value="0"/>	Counter 1 - Count From 0 To:	<input type="text" value="0"/>
Timer 2 Expiration (in milliseconds):	<input type="text" value="0"/>	Counter 2 - Count From 0 To:	<input type="text" value="0"/>



**NOTE**

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

## 4.8 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).

Timer 1 Expiration (in milliseconds):	<input type="text" value="0"/>	Counter 1 - Count From 0 To:	<input type="text" value="0"/>
Timer 2 Expiration (in milliseconds):	<input type="text" value="0"/>	Counter 2 - Count From 0 To:	<input type="text" value="0"/>



**NOTE**

Timers are discussed further in chapter [5.4 Actions](#).

## 4.9 Counters

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

Timer 1 Expiration (in milliseconds):	<input type="text" value="0"/>	Counter 1 - Count From 0 To:	<input type="text" value="0"/>
Timer 2 Expiration (in milliseconds):	<input type="text" value="0"/>	Counter 2 - Count From 0 To:	<input type="text" value="0"/>

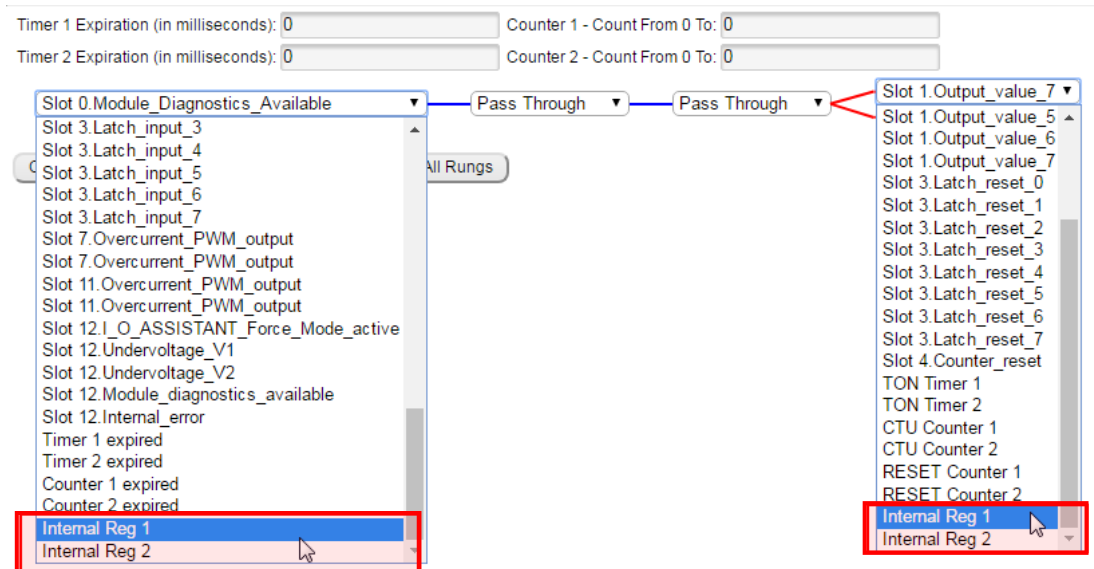


**NOTE**

Counters are discussed further in chapter [5.4 Actions](#).

## 4.10 Internal Reg

Flow Chart Editor contains two *Internal Regs* (*Reg* = *register*). The user can use an internal register as a condition to trigger an action or as an action to trigger a condition.



## 4.11 Flow Chart Menu Bar

### 4.11.1 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. Lastly, ARGEE transitions over to the *Debug* screen.

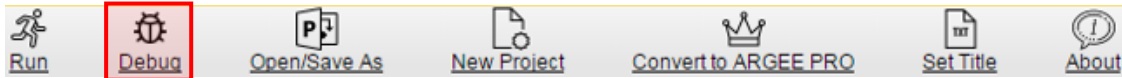


#### NOTE

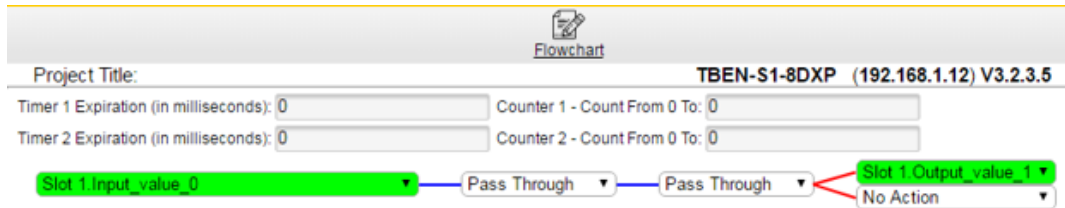
After the run button is pressed, the environment transitions to the Debug screen. More information about Debug can be found in chapter [7 Debugger](#).

### 4.11.2 Debug (ARGEE Flow)

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

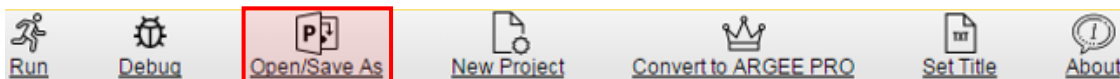


If the user clicks *Debug* while in Flow Chart, the first thing the user will notice is that the Flow Chart will enter *Debug* mode. As conditions become true, the user can visually observe code progression.



### 4.11.3 Open/Save As

The Open/Save feature allows the user to save a current project or load a previous project. The user can access the *Open/Save As* screen 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.



## Open Project/Library

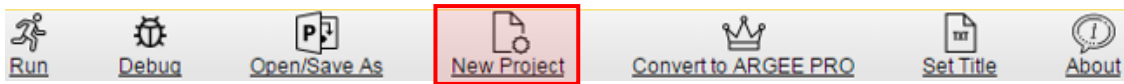
No file chosen

## Save Project/Library

Project Name:

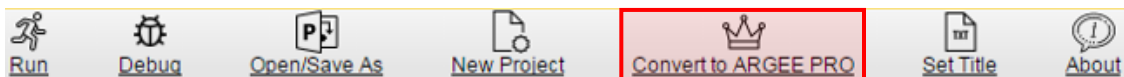
#### 4.11.4 New Project

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



#### 4.11.5 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. ARGEE PRO functions are discussed in Chapter 7.

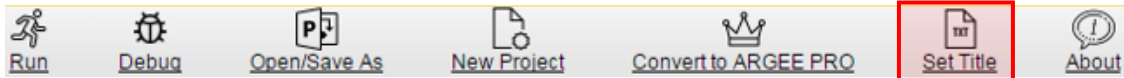


#### NOTE

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

#### 4.11.6 Set Title

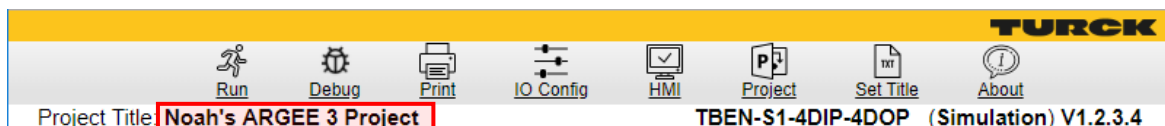
The user can click *Set Title* to add a name to the project.



This page says

Set Project Title

OK Cancel





#### 4.11.7 About

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



#### ↓ Versions and Links:

Environment Version:	3.2.72.5
ARGEE Kernel Version:	3.5.2.0
Download link to the latest version of the environment:	<a href="#">Click Here</a>

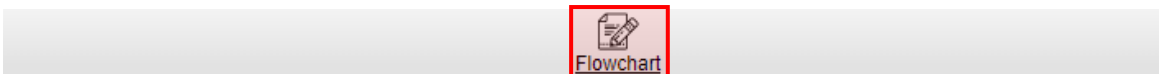


#### NOTE

The user can use the "[Click Here](#)" hyperlink to download the latest ARGEE environment.

#### 4.11.8 Flowchart

The user will click *flowchart* when they want to leave the debug page and return to the ARGEE Flow Chart screen.



## 5 ARGEE PRO

### 5.1 The Basics

The ARGEE PRO home page is made up of Keyboard Shortcuts, Main Task, Condition, Actions, HMI Information, Program Variables, Alias Variables, Function Blocks, Statements and Libraries.

The screenshot shows the ARGEE PRO home page interface. The top bar includes icons for Run, Debug, Print, IO Config, HMI, Project, Set Title, and About. The Project Title is "TBEN-S1-4DIP-4DOP (Simulation) V1.2.3.4".

The interface is divided into two main sections: "Variables and Definitions" on the left and "ARGEE Program" on the right.

**Variables and Definitions:**

- Program Variables:** A table with columns "Name" and "Type". It lists four variables: reg1 (Number), reg2 (Number), tm1 (Timer/Counter), and tm2 (Timer/Counter). There is an "Add Variable" button.
- Alias Variables (hidden):** A section with a "Function Block" dropdown and an "Add" button.
- Import Library:** A section with a "Choose Files" button and a "No file chosen" status.

**ARGEE Program:**

- Keyboard Shortcuts:** A section with a list of shortcuts: Press Ctrl-q for list of program variables, Press Ctrl-l for list of function block variables, Press Ctrl-i for list of I/O variables, Press Ctrl-f for list of built-in functions, and Press Ctrl-s for list of State Names.
- Main Task:** A section with a "Task - MainTask" dropdown and an "Add Condition" button.
- Condition & Actions:** A section with a "Condition" dropdown and an "Add Condition" button.
- HMI Screens:** A section with a "HMI Screens (hidden)" dropdown.
- Program Variables & Alias Variables:** A section with a "Function Block" dropdown and an "Add" button.
- Function Blocks, Statements & Libraries:** A section with a "Choose Files" button and a "No file chosen" status.

Callouts from the right side of the image point to the following components:

- Keyboard Shortcuts
- Main Task
- Condition & Actions
- HMI Screens
- Program Variables & Alias Variables
- Function Blocks, Statements & Libraries

### 5.2 Variables and Expressions

Variables are named storage locations for changing information. Expressions are a combination of values, variables, conditions, actions, and functions that are interpreted in a predictable way by the program. The user must understand how the expressions of the program work to be successful in writing any code. In ARGEE 3 Pro and Pro Advanced, expressions are everything on the right side of the screen, and variables are on the left side of the screen.

The screenshot shows the ARGEE PRO interface with the "Variables and Definitions" section on the left and the "ARGEE Program" section on the right.

**Variables and Definitions:**

- Program Variables:** A table with columns "Name" and "Type". It lists three variables: State (State/Enum), Reg1 (Number), and Reg2 (Number). There is an "Add Variable" button.
- Alias Variables (hidden):** A section with a "Function Block" dropdown and an "Add" button.
- States:** A table with columns "Name" and "Type". It lists two states: Wait\_For\_Submit and Check\_Process\_Value.

**ARGEE Program:**

- Task - MainTask:** A section with a "Condition" dropdown and an "Add Condition" button.
- Condition:** A section with a "Condition" dropdown and an "Add Condition" button.
- Expressions:** A section with a "Destination" and an "Expression" field. It shows two expressions: "Reg1 + 1" and "Check\_Process\_Value".

Callouts from the right side of the image point to the following components:

- Expressions



**NOTE**

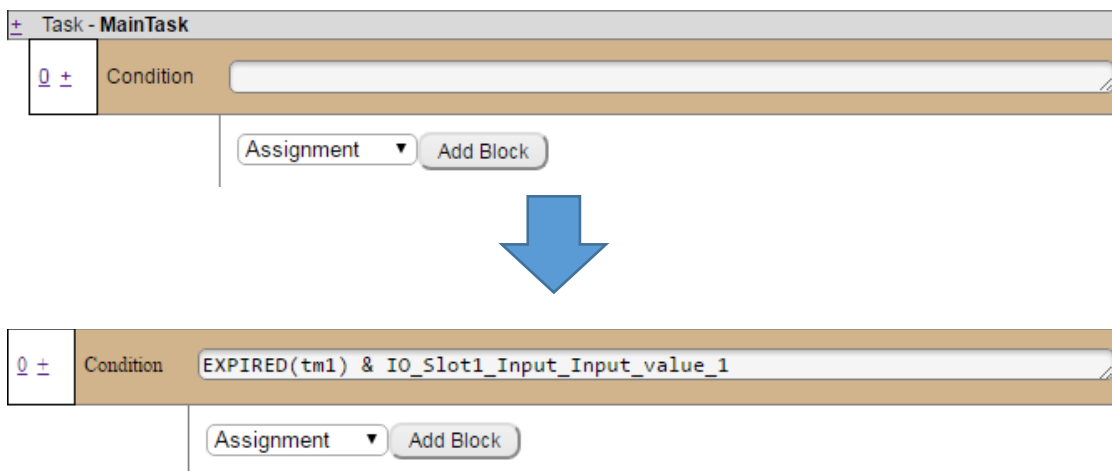
Information on the functions available in ARGEE can be found in the Appendix [11.2 Built-in Functions \(Ctrl-f\)](#), and information on variables can be found in chapter [5.7 Program Variables](#).

## 5.3 Condition

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

- A digital sensor going true (or false)
- An analog sensor getting into a specific range
- A specific RFID tag being presented to a transceiver
- A “start” command from the ARGEE HMI or any other PLC
- A timer or counter expiring
- A timer or counter reaching a specific value
- ...many other things can be used as an input condition

The Condition box also allows the user to combine several different inputs at once.

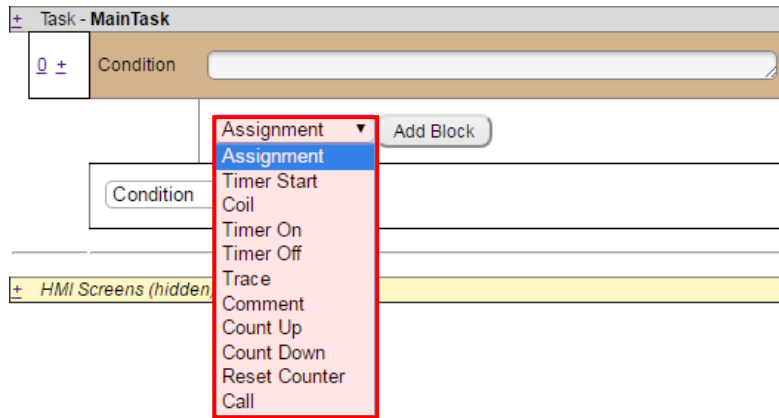


**Explaining the Example:** The above Condition will only become true when timer 1 expires and Input\_value\_1 goes true.

## 5.4 Actions

ARGEE allows the user to execute several Actions under a single Condition statement. There are 11 Actions available in ARGEE PRO. Please note that Actions are only available under a Condition statement. They are excluded from While, For, If, Else If, Else statements (see ARGEE Pro Advanced mode for more details)

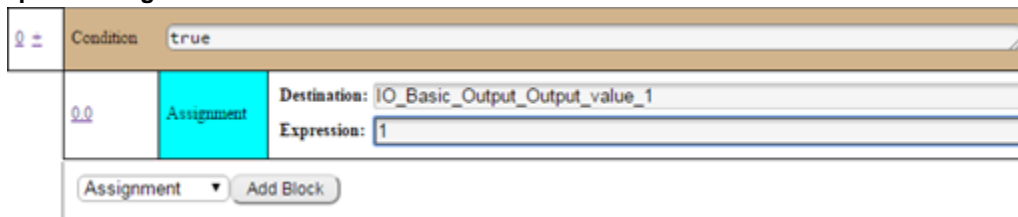
- Assignment
- Timer Start
- Coil
- Timer On
- Timer Off
- Trace
- Comment
- Count Up
- Count Down
- Reset Counter
- Call



### 5.4.1 Assignment

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

**Example of Assignment:**



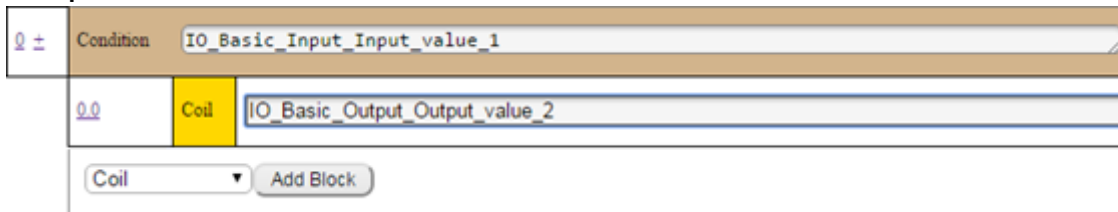
**Explaining the Example:** The Condition in the above statement is always “true.” The value “1” is loaded into register Output\_value\_1. In other words, this means that the user’s Output 1 will always be on.

### 5.4.2 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:



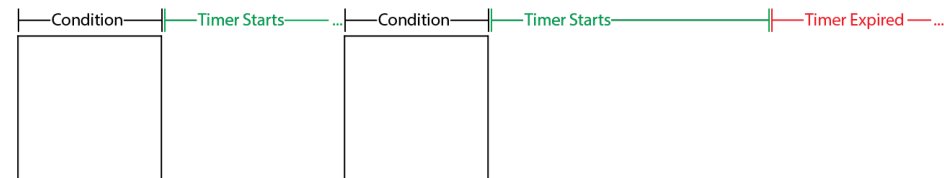
**Explaining the Example:** When Input\_value\_1 is true, Output\_value\_2 is true. When Input\_value\_1 is false, Output\_value\_2 is false.

## 5.4.3 Timer Start

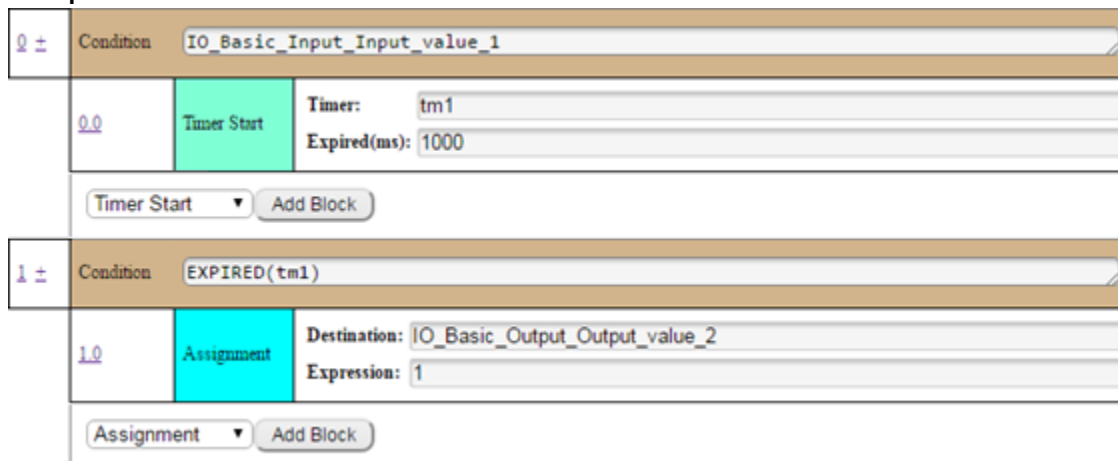
The user will use the *Timer Start* action if they want to start a timer after the Condition has occurred.



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



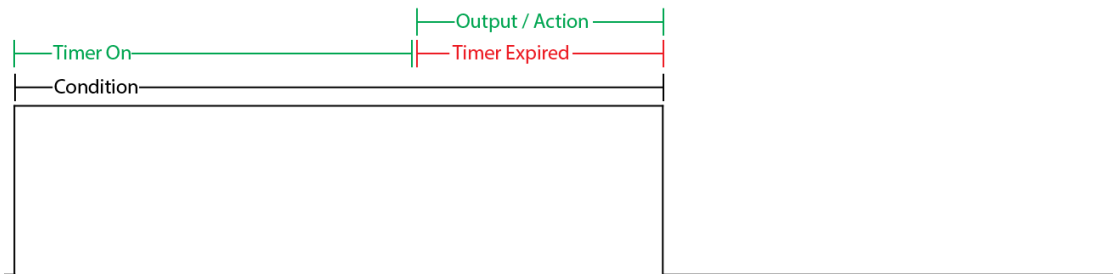
## Example of Timer Start:



**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).

### 5.4.4 Timer On

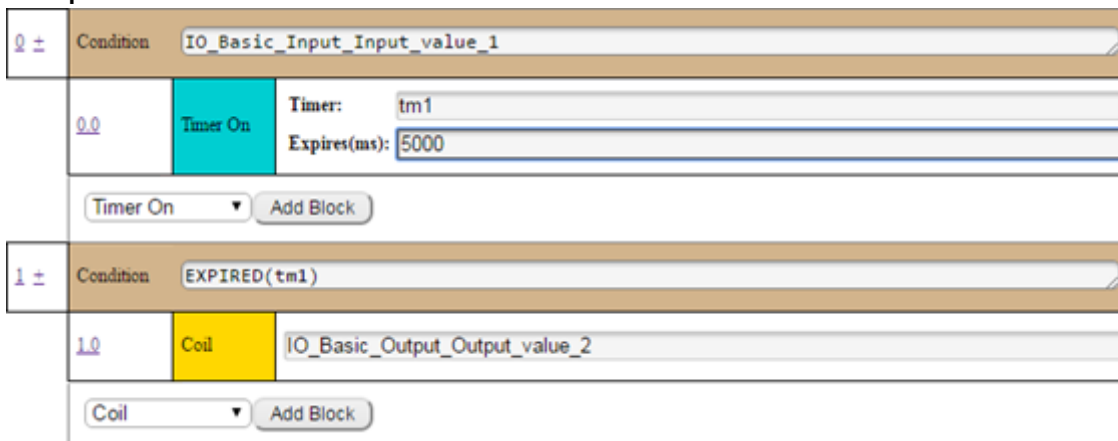
The user will use the Timer On action if they want a timer to run while a Condition is true. The user will normally tie an additional Action or Output to the timer expired Condition.



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



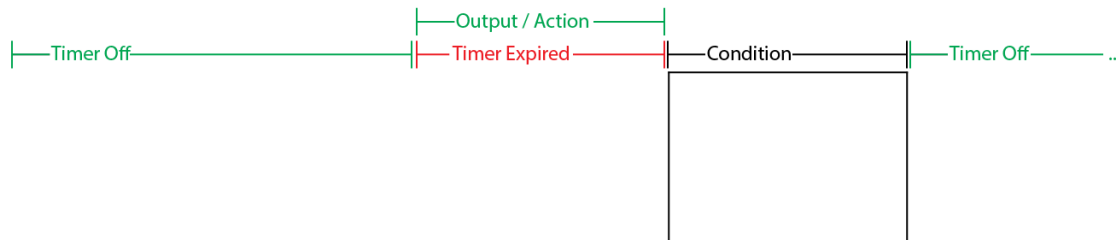
#### Example of Timer On:



**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.

### 5.4.5 Timer Off

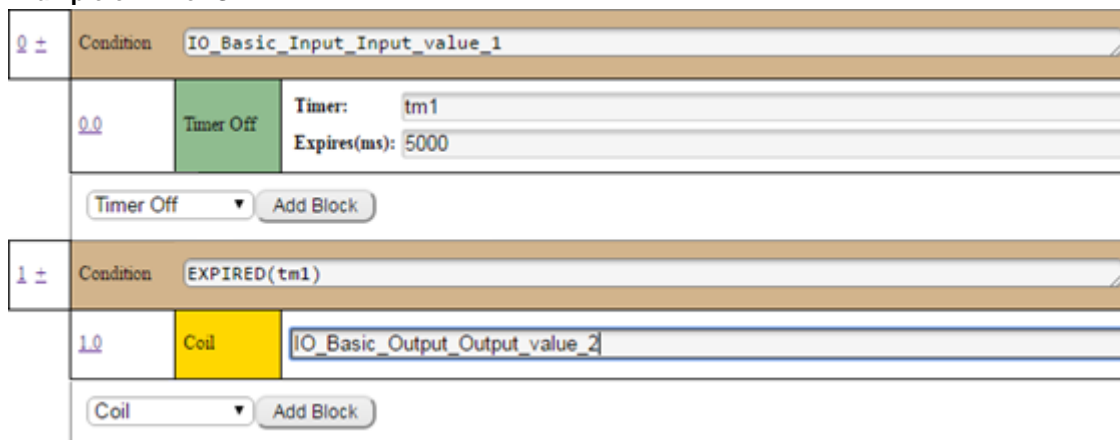
The user will use the Timer Off action if they want a timer to run while a Condition is false. The user will normally tie an additional Action or Output to the timer expired Condition.



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



#### Example of Timer Off:



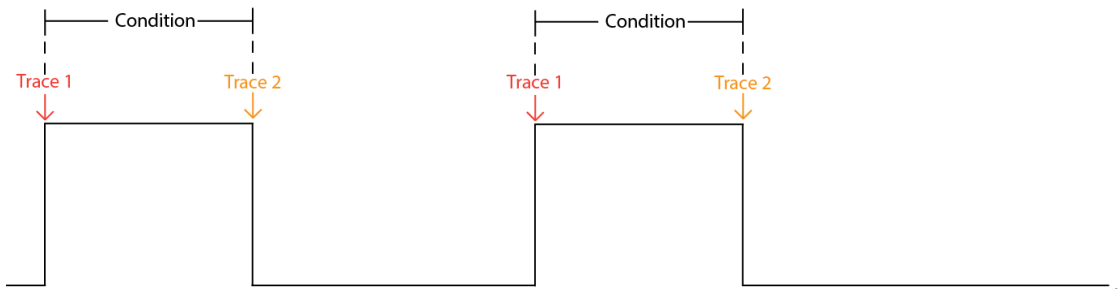
**Explaining the Example:** Timer 1 starts counting as soon as the program starts. When timer 1 expires, Output\_value\_2 is coiled on. When Input\_value\_1 is true, timer 1 is reset to zero and Output\_value\_2 goes false. When Input\_value\_1 is false, timer 1 starts counting again.

### 5.4.6 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 (F\_COS) trigger in the condition block. The Change of State trigger is discussed in the Appendix [11.2.9.1 Change of State \(F\\_COS\)](#).

0 ±	Condition	(F_COS(IO_Basic_Input_Input_value_0,Temp_1) & IO_Basic_Input_Input_value_0=1)	
0.0	Trace	Prefix String:	Trace_1
		Expression:	0
	Trace	Add Block	
1 ±	Condition	(F_COS(IO_Basic_Input_Input_value_0,Temp_2) & IO_Basic_Input_Input_value_0=0)	
1.0	Trace	Prefix String:	Trace_2
		Expression:	1
	Trace	Add Block	

**Explaining the Example:** When Input\_value\_0 is true, Trace\_1 time stamps that event. When Input\_value\_0 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.



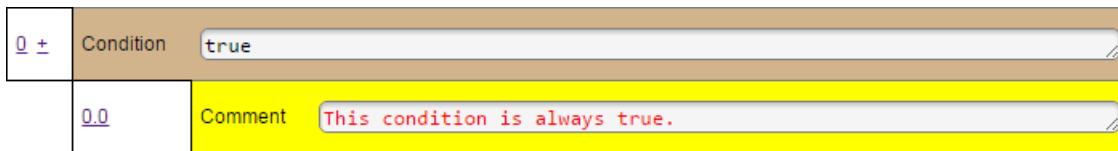
#### NOTE

An example of *Trace* can be found in the Appendix [12.2 Trace Example](#).



### 5.4.7 Comment

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

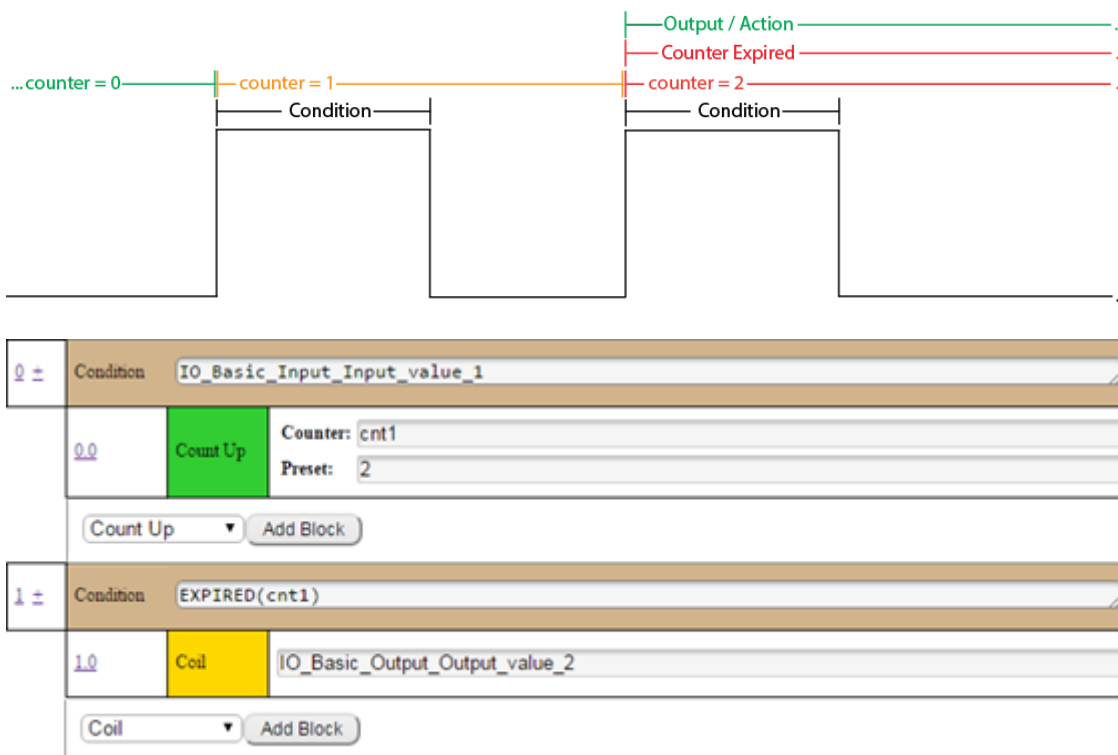


### 5.4.8 Count Up

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:

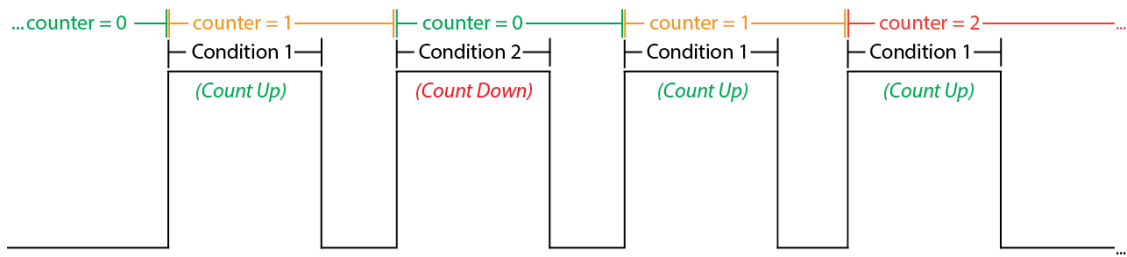
The user wants to do an Action after the same Condition has occurred two times.



**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.

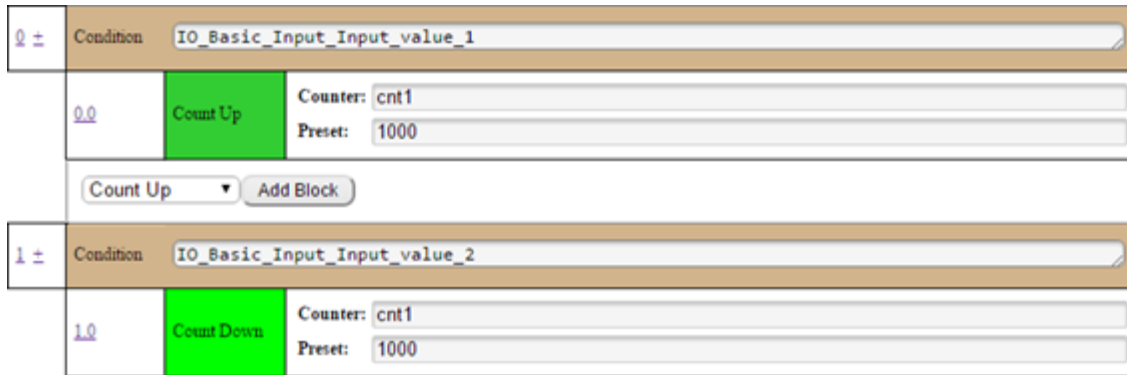
### 5.4.9 Count Down

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:

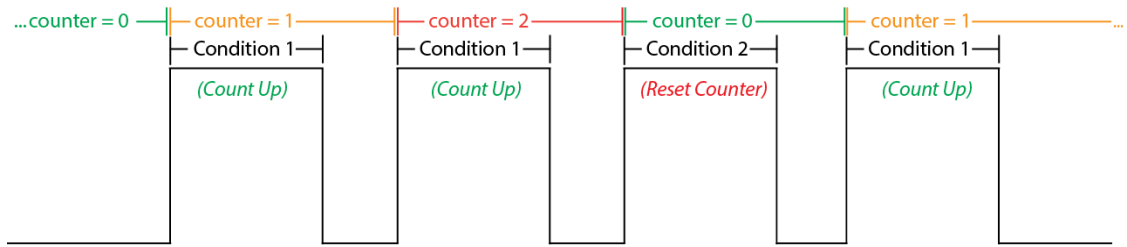
The user wants to keep track of the number of guests in the store. When a guest walks in the store the counter goes up, but when a guest walks out of the store the counter goes down.



**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.

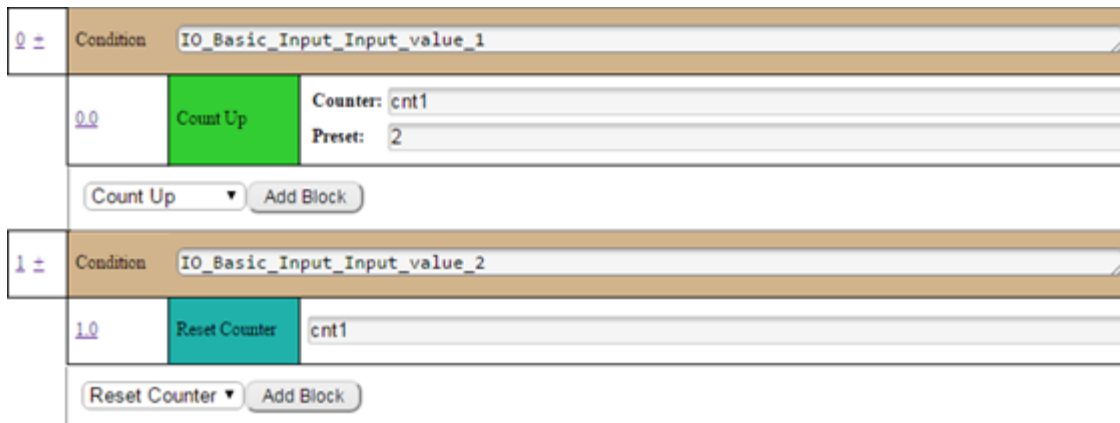
#### 5.4.10 Reset Counter

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.

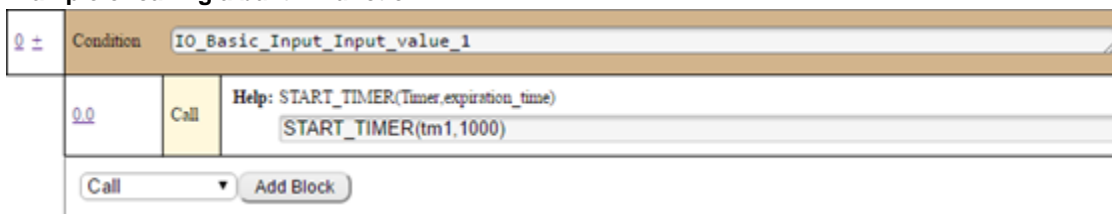


**Explaining the Example:** Each time Input\_value\_1 is true, counter 1 counts up one time. Each time Input\_value\_2 is true, counter 1 resets to zero.

#### 5.4.11 Call

The user will use the *Call* action if they want to call a built-in function or user-made Function Blocks. The *Call* action has a built-in help text that displays the arguments in the called function.

**Example of calling a built-in function:**



**Explaining the Example:** Each time Input\_value\_1 is true, the built-in function *Start\_Timer* will be called.



#### NOTE

The built-in function Start\_Timer (and all other built-in functions) are explained in chapter [11.2 Built-in Functions \(Ctrl-f\)](#). An example of calling a user made function block can be found [12.3 Call Example](#)

#### 5.4.12 How Actions respond to Conditions

<b>Action</b>	<b>Condition=FALSE</b>	<b>Condition=TRUE</b>
<b>Assignment</b>	No action	Assigns a destination variable to a result of expression evaluation.
<b>Timer start</b>	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 >= "Expired" preset value.
<b>Coil</b>	Resets a variable to 0	Sets the variable to 1
<b>Timer On</b>	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 >="Expired" preset value. In that case, the Done flag is raised.
<b>Timer Off</b>	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.
<b>Trace</b>	-	Record trace information into a trace buffer.
<b>Comment</b>	-	-
<b>Count up</b>	Increments the counter whenever the condition changes from false to true.	
<b>Count down</b>	Decrements the counter whenever the condition changes from false to true. (note - the Preset can be a negative value)	
<b>Reset Counter</b>	-	Restarts the counter to - 0
<b>Call</b>	-	Executes a function or a function block.

## 5.5 Program Variables

Program Variables can be added, deleted, and renamed. The user can also change the variable type by using the *Type* drop-down arrow. *Program Variables* are usable throughout the entire program.

### Variables and Definitions

	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter

Add Variable

1 Alias Variables (hidden)

Function Block Add

### ARGEE Program

**Program Variables**

Keyboard shortcuts:  
 Press Ctrl-q for list of program variables  
 Press Ctrl-l for list of function block variables  
 Press Ctrl-i for list of I/O variables  
 Press Ctrl-f for list of built-in functions  
 Press Ctrl-s for list of State Names

Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl -"left/right arrow" to adjust the size of variables panel

Block select program statements by clicking on the "number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

Task - MainTask

Add Condition

HMI Screens (hidden)

### 5.5.1 Variable Name

The variable name section is where the user identifies variables that are used in the program.

	Name	Type
1	reg1	Number

➡

	Name	Type
1	Variable_1	Number

### 5.5.2 Variable Types

The user can set his desired variable type by selecting the Type drop down arrow.

	Name	Type
1	reg1	Number
2	reg2	Number

Add Variable

Number

Number

Floating

String

Byte

WORD

Timer/Counter

State/Enum

Retain Number

Retain Float

- Number - Stores integers between -2,147,483,658 and 2,147,483,657 (4 byte signed integer).

- Floating - Stores an integer and its decimal in the register.

### Variables and Definitions

	Name	Type
1	variable_1	Floating
2	variable_2	Number
3		Number

Add Variable

1 Alias Variables (hidden)

Function Block Add

### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask

Q	Condition	Destination	Expression
0.0	Assignment	variable_1	1.1
0.1	Assignment	variable_2	1.1

Assignment Add Block

Add Condition



### Runtime Status

TRACE

PROG\_CYCLE\_TIME: 2

PLC\_CONNECTED: 0

VARIABLE\_1: 1.100000023841858

VARIABLE\_2: 1

MainTask

- Local IO: Slot0
- Local IO: Slot1 - Input
- Local IO: Slot1 - Output
- Local IO: Slot1 - Diagnostics
- PLC\_TO\_ARGEE
- ARGEE\_TO\_PLC

### ARGEE Program

Task - MainTask

Q	Condition	Destination	Expression
0.0	Assignment	variable_1	1.1
0.1	Assignment	variable_2	1.1



#### NOTE

variable\_1's type is set to Floating and variable\_2's type is set to Number. Both registers are loaded with the value 1.1. Notice the ".1000" is cutoff in VARIABLE\_2 (variable\_2) but not in VARIABLE\_1 (variable\_1).

- String - Stores integers and/or characters in an array.

### Variables and Definitions

	Name	Type
	# of Array Elements: 32 (Clear field to disable array)	
1	Variable_1	String

Add Variable

### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask

Q	Condition	Help
0.0	Call	STR_COPY(source str dest str) STR_COPY("Noah is playing with Strings", Variable_1)



### Runtime Status

- TRACE
- PROG CYCLE TIME : 5
- PLC CONNECTED : 0
- ALIAS VARIABLES
- VARIABLE 1 Noah is playing with Strings
- MainTask
- Local IO: Slot0
- Local IO: Slot1 - Input
- Local IO: Slot4 - Output

### ARGEE Program

Task - MainTask			
0 ±	Condition	true	
0.0	Call	STR_COPY("Noah is playing with Strings", Variable_1)	



#### NOTE

The Call action is discussed in chapter [5.4.11 Call](#). Strings are discussed in chapter [11.2.2 Strings/Arrays](#).

- Byte - One unsigned byte. Stores integers from 0 to 255, or hex values from 0x00 to 0xff.
- WORD - Two unsigned bytes. Stores integers from 0 to 65535, or hex values from 0x0000 to 0xffff.
- Timer/Counter - Timer/Counter registers can store a value from -2,147,483,658 and 2,147,483,657.



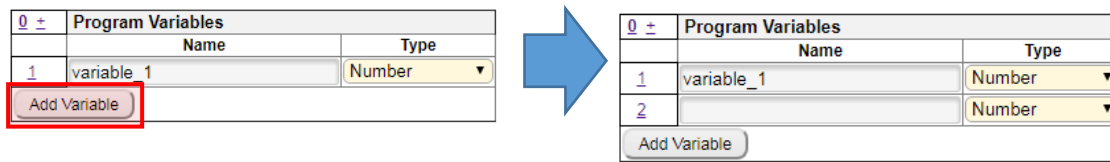
#### NOTE

2,147,486,657 milliseconds is about 23 days.

- State/Enum - The user would select *State/Enum (Enumeration)* if he wanted to create a state variable. State variables are used in state machines.
- Retain Number - Retains integers between -2,147,483,658 and 2,147,483,657 through a power cycle. It syncs about every 2 minutes.
- Retain Float - Retains an integer and its decimal through a power cycle. It syncs about every 2 minutes.

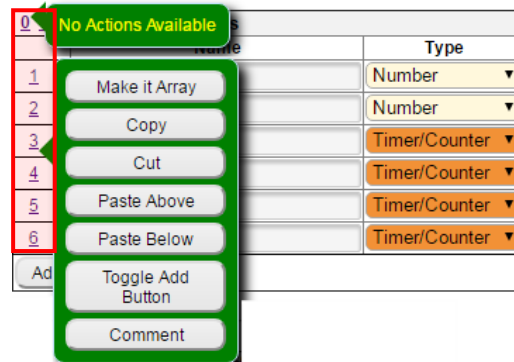
### 5.5.3 Add Variable

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

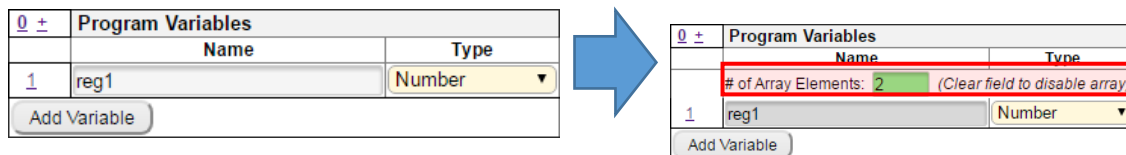


### 5.5.4 Program Variables Context Menu

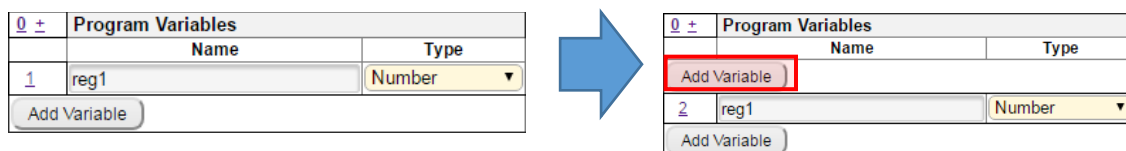
- To access this menu, click the number in front of the variable.



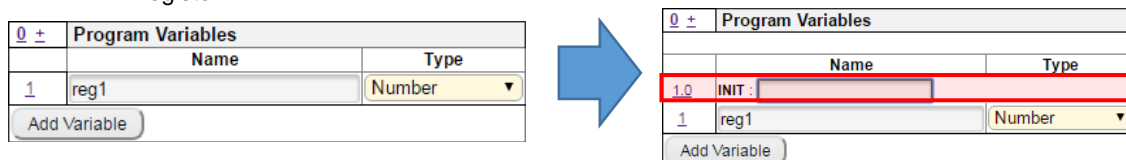
- Make it Array - Turns the variable into an array.



- Copy - Copies the variable so the user can paste it in another place.
- Cut - Cuts the variable out so the user can then paste it in another place.
- Paste Above - Paste a cut/copied variable above the selected position.
- Paste Below - Paste a cut/copied variable below the selected position.
- Toggle Add Button - Selecting this will place an "Add Variable" button above the variable



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





The user can also Initialize an array if they want to pre-set the value in a Program Variable's register.

The diagram illustrates the process of initializing an array. On the left, a 'Program Variables' table shows a single entry 'reg1' of type 'Number'. A blue arrow points to the right, where the same table is shown with an additional row. This new row, highlighted with a red border, is for an array: it has a '1.0' in the index column, 'INIT : [1]' in the Name column, and '=2' in the Type column. Below this, it says '# of Array Elements: 2' with a green box around the '2' and a note '(Clear field to disable array)'. The original 'reg1' entry remains below it.



## NOTE

The user can press Control-q while in the *Program Variable* name area to automatically prompt variable initialization.

- Comment - Selecting this will insert a comment line above the program variable.

The diagram illustrates the process of adding a comment to a variable. On the left, a 'Program Variables' table shows a single entry 'reg1' of type 'Number'. A blue arrow points to the right, where the same table is shown with an additional row. This new row, highlighted with a red border, is for a comment: it has a '1' in the index column, 'Comment' in the Name column, and an empty text field in the Type column. Below this, the original 'reg1' entry remains.

## 5.6 Alias Variables

Alias Variables give friendly names to I/O Points and PLC Variables. In many cases, it is much easier to understand the code when the user uses *Alias Variables*.

1 ±	Alias Variables	
	Name	IO Point
0	Friendly_IO_Point_Name	IO_Point
1	car_sensor	IO_Basic_Input_Input_value_4
2	greenlight	IO_Basic_Output_Output_value_6
3	PLC_in	IO_PLC_TO_ARGEE_Word4
Add Variable		

## 5.7 Main Task

When the user converts to ARGEE PRO, a *Main Task* is created. The user can only add *Condition* blocks in the *Main Task*. Other function blocks can be created, but they need to be called from a *Condition*.

### Variables and Definitions

0 ±	Program Variables	
	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter
Add Variable		

1 ± Alias Variables (hidden)

Function Block Add

### ARGEE Program

+ Keyboard shortcuts:  
Press Ctrl-q for list of program variables  
Press Ctrl-f for list of function block variables  
Press Ctrl-i for list of I/O variables  
Press Ctrl-f for list of built-in functions  
Press Ctrl-s for list of State Names  
  
Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl-"left/right arrow" to adjust the size of variables panel  
  
Block select program statements by clicking on the "Number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

Task - MainTask (hidden)

HMI Screens (hidden)

Main Task



### NOTE

Function blocks are explained later in this chapter, and in chapter [5.8 Function Block Type](#).

## 5.7.1 Adding Conditions to the Main Task

If the user clicks the *Add Condition* button, a blank condition will be added to the ARGEE project.

### Variables and Definitions

	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter

Add Variable

1 Alias Variables (hidden)

Function Block Add

### ARGEE Program

Keyboard shortcuts:  
 Press Ctrl-q for list of program variables  
 Press Ctrl-l for list of function block variables  
 Press Ctrl-i for list of I/O variables  
 Press Ctrl-f for list of built-in functions  
 Press Ctrl-s for list of State Names

Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl-"left/right arrow" to adjust the size of variables panel

Block select program statements by clicking on the "number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

Task - MainTask

Add Condition

HMI Screens (hidden)

### Variables and Definitions

	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter

Add Variable

1 Alias Variables (hidden)

Function Block Add

### ARGEE Program

Keyboard shortcuts:  
 Press Ctrl-q for list of program variables  
 Press Ctrl-l for list of function block variables  
 Press Ctrl-i for list of I/O variables  
 Press Ctrl-f for list of built-in functions  
 Press Ctrl-s for list of State Names

Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl-"left/right arrow" to adjust the size of variables panel

Block select program statements by clicking on the "number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

Task - MainTask

Condition

Assignment Add Block

Add Condition



### NOTE

The Condition/Action relationship is similar to the If/Then relationship. For Example: "**If** this **condition** goes true, **then** perform these **actions**."

## 5.7.2 Adding Actions to the Main Task

Actions are selected from the *Add Block* drop-down menu. When the desired action is selected, the user can click on the *Add Block* button to add the action to the condition.

0 ±

Program Variables

	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter

Add Variable

1 ±

Alias Variables (hidden)

Function Block

Add

ARGEE Program

+ Keyboard shortcuts:  
Press Ctrl-q for list of program variables  
Press Ctrl-l for list of function block variables  
Press Ctrl-i for list of I/O variables  
Press Ctrl-f for list of built-in functions  
Press Ctrl-s for list of State Names

Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl-"left/right arrow" to adjust the size of variables panel

Block select program statements by clicking on the "number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

+ Task - MainTask

0 ±

Condition

Assignment

Assignment

Timer Start

Timer On

Timer Off

Trace

Comment

Count Up

Count Down

Reset Counter

Call

Add Block

Add Condition

+ HMI Screens (hidden)

Variables and Definitions

0 ±

Program Variables

	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter

Add Variable

1 ±

Alias Variables (hidden)

Function Block

Add

ARGEE Program

+ Keyboard shortcuts:  
Press Ctrl-q for list of program variables  
Press Ctrl-l for list of function block variables  
Press Ctrl-i for list of I/O variables  
Press Ctrl-f for list of built-in functions  
Press Ctrl-s for list of State Names

Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl-"left/right arrow" to adjust the size of variables panel

Block select program statements by clicking on the "number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

+ Task - MainTask

0 ±

Condition

0.0

Assignment

Destination:

Expression:

Assignment

Add Block

Add Condition

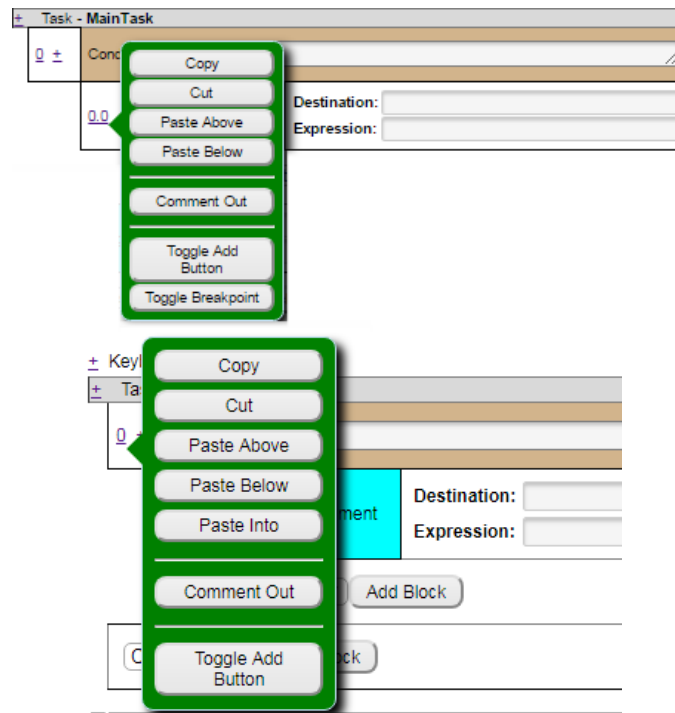
### NOTE

Actions are discussed more in this chapter [5.4 Actions](#).

40

Turck Inc. | 3000 Campus Drive, Minneapolis, MN 55441 | T +1 800 544 7769 | F +1 763 553 0708 | [www.turck.com](http://www.turck.com)

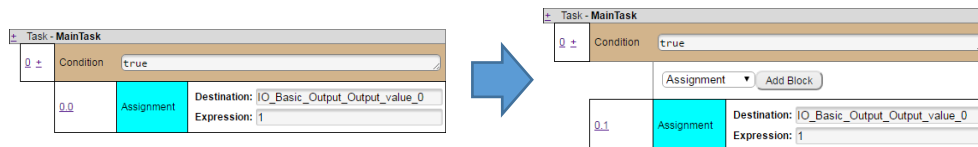
### 5.7.3 Main Task Context Menu



- **Copy** - Copies the variable so the user can paste it in another place.
- **Cut** - Cuts the variable out so the user can then paste it in another place.
- **Paste Above** – Paste a cut/copied variable above the selected position.
- **Paste Below** - Paste a cut/copied variable below the selected position.
- **Paste Into** – Paste a cut/copied variable into the position.
- **Comment Out** – Turns the statement into a comment.

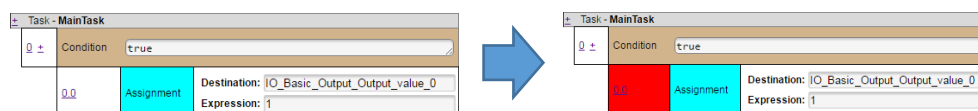


- **Toggle Add Button** - Selecting this will place an “Add Variable” button above the variable.

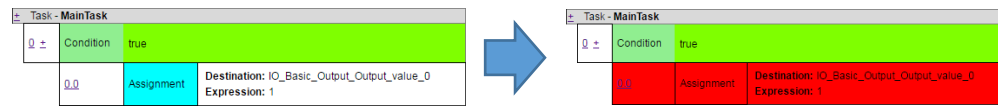


- **Toggle Breakpoint** – The statement becomes a breakpoint when the code is compiled and ran. The program will not progress farther than the selected statement. This can be done from the Edit Code screen or the Debug Code Screen.

(Edit Code Screen)



(Debug Screen)



#### NOTE

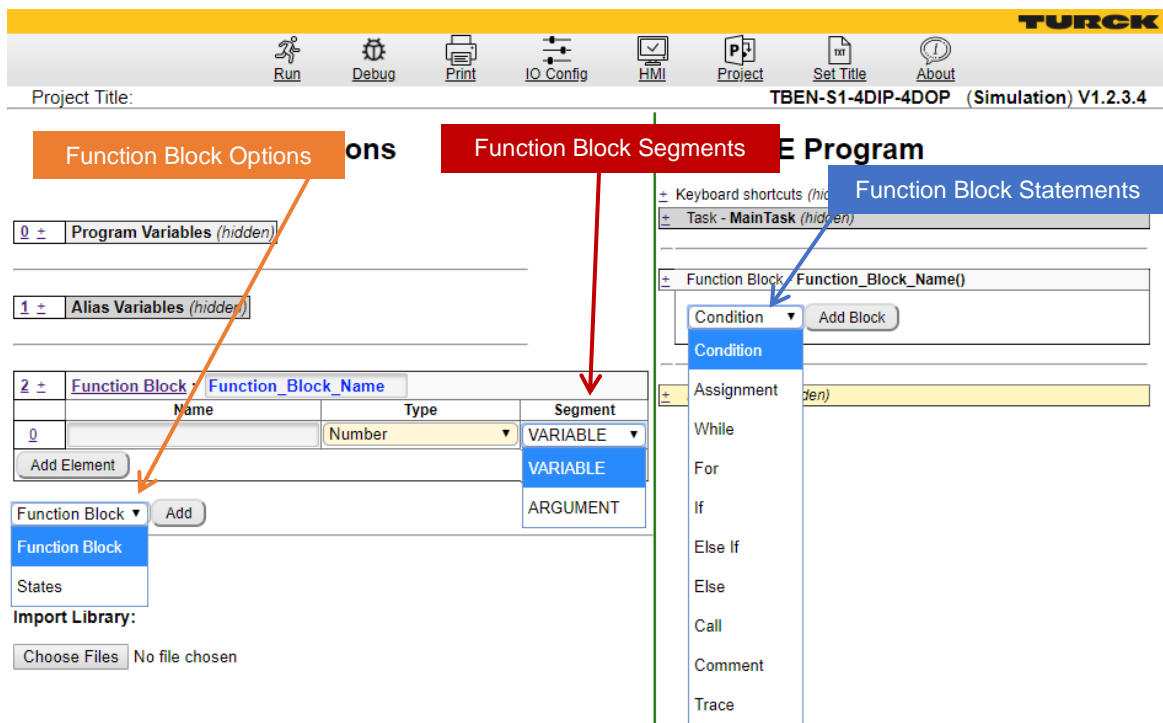
For more information about Debugging, check out [Chapter 7 Debugger](#) [Debugger 1](#)

## 5.8 Function Blocks

### 5.8.1 The Basics

The user will use a function block if the user wants to speed up their coding process, make their code easier to de-bug, or simply make their programs smaller.

The user will use a function block to make their code more reusable, make their code easier to de-bug, or make the code more readable.



#### NOTE

For more information about how to call a function block, check out [12.3 How to Call a Function Block](#)

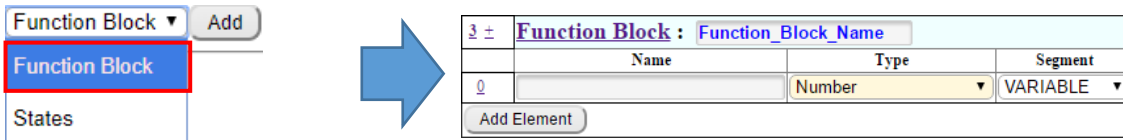
### 5.8.2 Function Block Options

In this section, the user can create function blocks or state names.

#### Function Block

- To add a function block, select Function Block from the drop-down and then click the Add button.

In ARGEE PRO, function blocks are called from the Main Task.



States

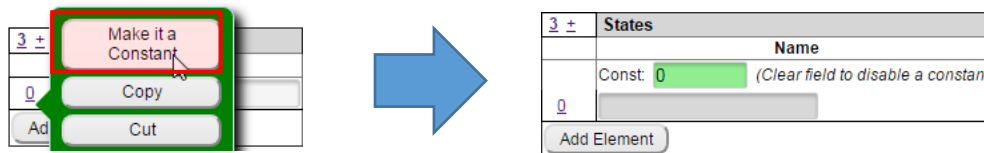
- To add a state name, select States from the dropdown and then click the Add button.

State names are used to make it easier to identify which state the program is in at any moment.



- Make it a Constant

Additionally, the States context menu has an additional option called “make it a constant.” **Make it a Constant** – Loads a constant value into the state name's register.



### 5.8.3 Function Block Segments

Variable

The user will select *Variable* under the segment type dropdown menu if the users wants to define an internal variable of the function block.

Argument

The user will select *Argument* under the segment type dropdown menu if the user wants to pass arguments to the functions block when the function block is called by the *Call* action in the *Main task* or from another function block. An *Argument* can be a number, a string, a variable or another function block. All the *Argument* elements should be defined as the first elements of the function block and their order determines the order of passing arguments.

### 5.8.4 Function Block Statements

If the user wants to use *Statements* in the *Main task*, the user needs to convert their program to [ARGE PRO Advanced Mode](#).



### 5.8.4.1 While

The *While* statement is one way to express a loop.

#### Example of *While*:

0	Assignment	Destination: Iteration Expression: 1
1 ±	While	Iteration <= 100
1.0	Assignment	Destination: IO_Basic_Output_Output_value_0 Expression: 1
1.1	Call	Help: START_TIMER(Timer,expiration_time) START_TIMER(tm1,Iteration*10)
1.2	Wait Until	EXPIRED(tm1)
1.3	Assignment	Destination: IO_Basic_Output_Output_value_0 Expression: 0
1.4	Call	Help: START_TIMER(Timer,expiration_time) START_TIMER(tm1,Iteration*10)
1.5	Wait Until	EXPIRED(tm1)
1.6	Assignment	Destination: Iteration Expression: Iteration + 1

**Explaining the Example:** This code is used to cycle an output at 10ms increments. During the first iteration, the output stays on for 10ms. During the second iteration, the output stays on for 20ms. This loop continues for 100 iterations.



#### NOTE

The Wait Until statement is discussed later in section [6.4 Wait Until](#).

### 5.8.4.2 For

The *For* statement is one way to express a loop.

#### Example of *For*:

0 ±	For	Iterator Variable:	Iteration
		Start Value:	1
		To Value:	100
0.0	Assignment	Destination:	IO_Basic_Output_Output_value_0
		Expression:	1
0.1	Call	Help:	START_TIMER(Timer,expiration_time)
			START_TIMER(tm1,Iteration*10)
0.2	Wait Until	EXPIRED(tm1)	
0.3	Assignment	Destination:	IO_Basic_Output_Output_value_0
		Expression:	0
0.4	Call	Help:	START_TIMER(Timer,expiration_time)
			START_TIMER(tm1,Iteration*10)
0.5	Wait Until	EXPIRED(tm1)	

**Explaining the Example:** This code is used to cycle an output at 10 ms (millisecond) increments. During the first iteration, the output stays on for 10ms. During the second iteration, the output stays on for 20ms. This loop continues for 100 iterations.



#### NOTE

The Wait Until statement is discussed later in section [6.4 Wait Until](#).

---

### 5.8.4.3 If

The *If* statement is similar to a *Condition*. If a condition is true, certain actions will be executed.

#### Example of If:

0 ±	If	Door_Open	
0.0	Assignment	Destination: Light	Expression: 1

**Explaining the Example:** If the door is opened, turn on a light.

### 5.8.4.4 Else If

The *Else If* statement has to follow an *If* statement.

#### Example of Else If:

0 ±	If	Door_Open	
0.0	Assignment	Destination: Light	Expression: 1
Assignment ▾ Add Block			
1 ±	Else If	!Door_Open	
1.0	Assignment	Destination: Light	Expression: 0

**Explaining the Example:** If the door is opened, turn on a light. If the door is not opened, turn off the light.



#### NOTE

“!” is the Boolean symbol for NOT. Boolean Logic is discussed in Chapter [11.2.7 Boolean Logic](#).

### 5.8.4.5 Else

The *Else* statement has to follow either an *If* or an *Else If* statement.

**Example of Else:**

The screenshot shows a ladder logic configuration. The first block is an 'If' statement (labeled 0) with the condition 'Door\_Open'. Below it is an 'Assignment' block (labeled 0.0) with 'Destination: Light' and 'Expression: 1'. The second block is an 'Else' statement (labeled 1) with an 'Assignment' block (labeled 1.0) with 'Destination: Light' and 'Expression: 0'. Buttons for 'Assignment' and 'Add Block' are visible between the two main blocks.

**Explaining the Example:** If the door is opened, turn on a light. Otherwise, turn the light off.

## 5.9 Libraries

### 5.9.1 What is a Library?

A library is an ARGEE element containing only State Names and Function Blocks (no Program variables nor Alias variables), and is designated by the ".st" (Structure Text) file extension. Libraries are useful for users who create many ARGEE programs that would require similar Function Blocks, i.e. RFID reading/writing, IO-Link programs, a timer-based halting function, etc.

In addition to creating libraries, the user can download official ARGEE libraries from [www.turck.com](http://www.turck.com).

### 5.9.2 Creating a Library

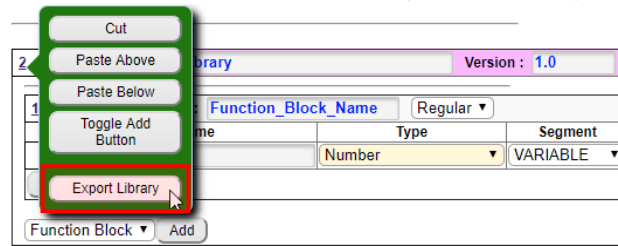
The user can create their own library by first clicking on the *Add Library* button.

The image shows a blue arrow pointing from an 'Add Library' button to a library configuration window. The window has a header with 'Library: New\_Library' and 'Version: 1.0'. Below the header is a 'Function Block' dropdown menu and an 'Add' button.

The user then creates their desired function blocks.

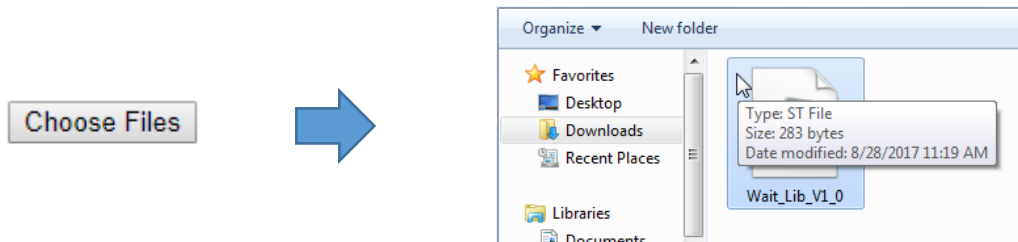
The screenshot shows a function block configuration window. The header displays 'Library: New\_Library' and 'Version: 1.0'. The main area shows 'Function Block: Function\_Block\_Name' with a 'Regular' type and 'VARIABLE' segment. Below this is a table with columns 'Name', 'Type', and 'Segment'. The 'Name' field is empty, 'Type' is 'Number', and 'Segment' is 'VARIABLE'. There is an 'Add Element' button and a 'Function Block' dropdown with an 'Add' button.

Once the library is complete, the user will select *Export Library* from the library context menu.



### 5.9.3 Importing a Library

The user can import an already pre-built library by clicking on the *Choose Files* button.



#### NOTE

If the user try's to import a library with the same name as an already installed library, ARGEE will ask the user remove the first library before importing the second.



#### NOTE

More information about Turck supported libraries can be found in [Appendix III - Libraries](#)

# 5.10 HMI Screens

The HMI editor is integrated into the code editor page. The user can only view their HMI after they have built an HMI.

Variables and Definitions

0 ±

Program Variables

	Name	Type
1	reg1	Number
2	reg2	Number
3	tm1	Timer/Counter
4	tm2	Timer/Counter
5	cnt1	Timer/Counter
6	cnt2	Timer/Counter

Add Variable

1 ±

Alias Variables (hidden)

Function Block

Add

ARGEE Program

Keyboard shortcuts:  
Press Ctrl-q for list of program variables  
Press Ctrl-l for list of function block variables  
Press Ctrl-i for list of I/O variables  
Press Ctrl-f for list of built-in functions  
Press Ctrl-s for list of State Names

Press Ctrl-"down arrow" collapse all elements which are collapsed by default, Ctrl-"left/right arrow" to adjust the size of variables panel

Block select program statements by clicking on the "number area" and dragging mouse down and selecting 2 or more statements. Once the block is selected, Ctrl-x can be used to cut statements, Ctrl-c to copy statements, Ctrl-d to comment out statements, Ctrl-Shift-d to uncomment statements.

Task - MainTask

Add Condition

HMI Screens

HMI Screen


HMI Screen

HMI Gnd Screen

HMI Image Group

Comment

Add Screen

 NOTE

Information on the HMI is available in chapter [9 ARGEE HMI](#).

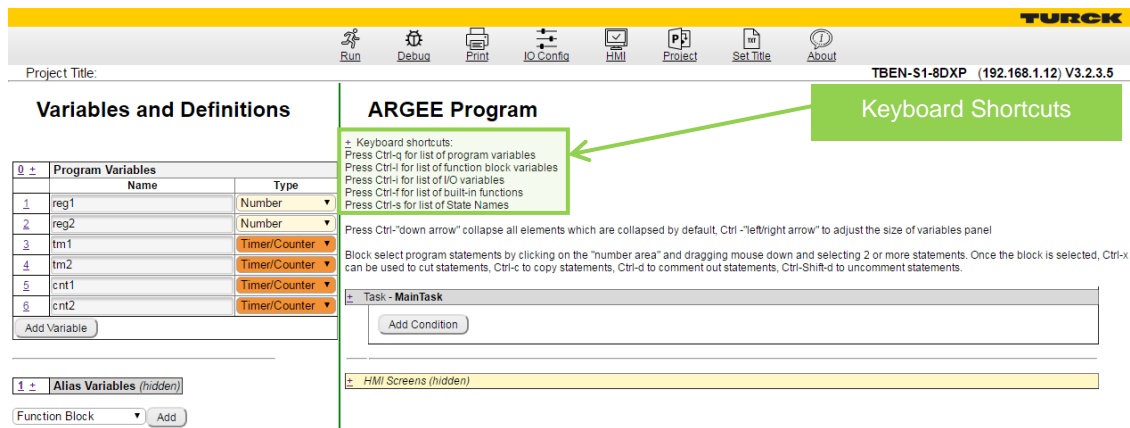
50

Turck Inc. | 3000 Campus Drive, Minneapolis, MN 55441 | T +1 800 544 7769 | F +1 763 553 0708 | [www.turck.com](http://www.turck.com)

## 5.11 Keyboard Shortcuts

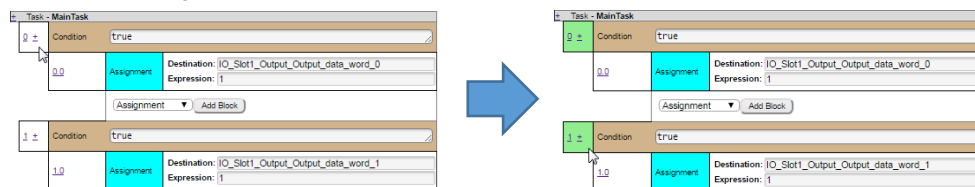
ARGEE 3 has many keyboard shortcuts to help make the user experience much easier. By default, the keyboard shortcuts are collapsed.

- Click on the  $\pm$  to expand the keyboard shortcuts.



### 5.11.1 List of Keyboard Shortcuts:

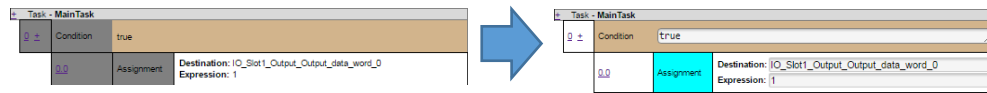
- **Ctrl - q** Brings up a list of Program Variables
- **Ctrl - L** Brings up a list of Function Block Variables (can only be initiated from inside of a Function Block)
- **Ctrl - i** Brings up a list of I/O Variables
- **Ctrl - f** Brings up a list of Built-In Functions available at current location
- **Ctrl - s** Brings up a list of State Names
- **Ctrl - “Down Arrow”** Collapses all elements which are collapsed by default
- **Ctrl - “Left/Right Arrow”** Adjusts the size of the Variable and Definitions panel
- **Selecting Multiple Statements** – Click the white space, hold and drag down until the statements turn green.



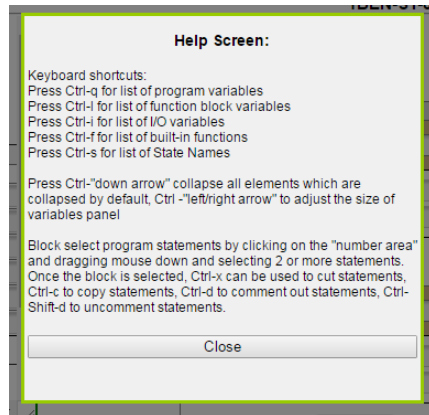
- **Ctrl - x** Cuts the selected statement(s)
- **Ctrl - c** Copies the selected statement(s)
- **Ctrl - z** Undoes the previous action (ARGEE 3 remembers 32 actions)
- **Ctrl - y** Redoes the previous action (ARGEE 3 remembers 32 actions)
- **Ctrl - d** Comment out selected statement(s). This turns the selected statements into comments, and will not be compiled when the code is run.



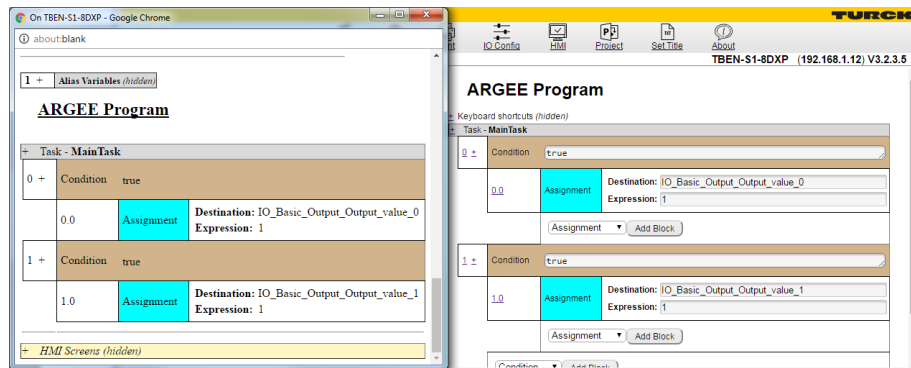
- **Ctrl – Shift – d** Uncomment out selected statement(s). This turns the comments back into statements, and will be compiled when the code is run.



- The user can press **F1** at any time to bring up a list of the keyboard shortcuts.



- The user can press **F2** to display a read-only view of the project. This is useful when doing side-by-side editing.



#### NOTE

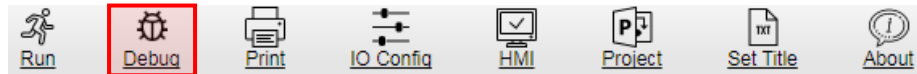
To bring up a read-only window that is scrolled to a specific function block, double-click the help text in its Call block, then press **F2**.



## 5.12 ARGEE PRO Menu Bar

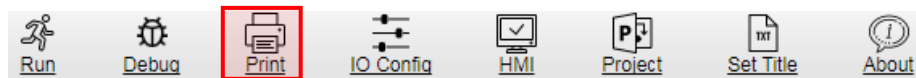
### 5.12.1 Debug (ARGEE PRO)

When the user clicks *Debug* while in ARGEE PRO, they get a brand-new menu bar with many more options. The ARGEE PRO debugger is discussed in Chapter 7 [Debugger](#).



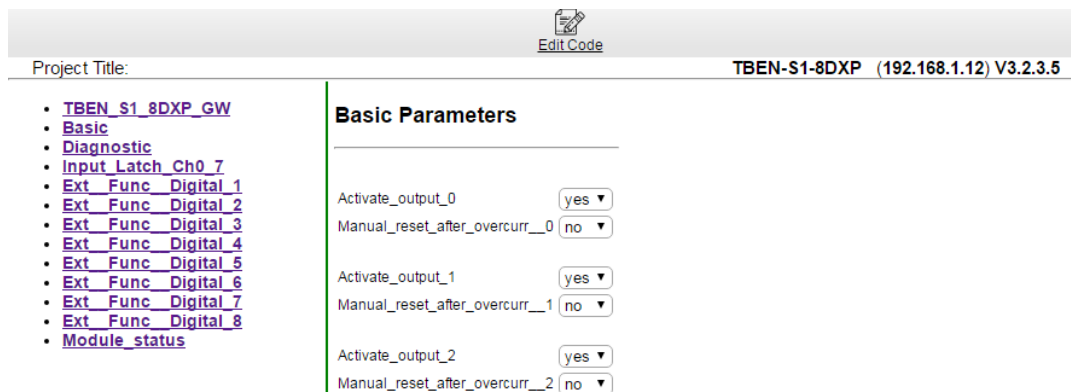
### 5.12.2 Print

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



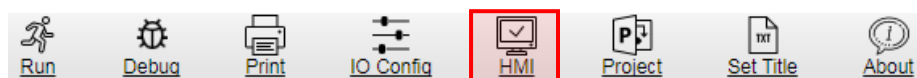
### 5.12.3 IO Config (I/O Configuration)

The user can configure all the device parameters by clicking on *IO Config*. This is extremely useful for IO-Link, RFID and Analog devices.



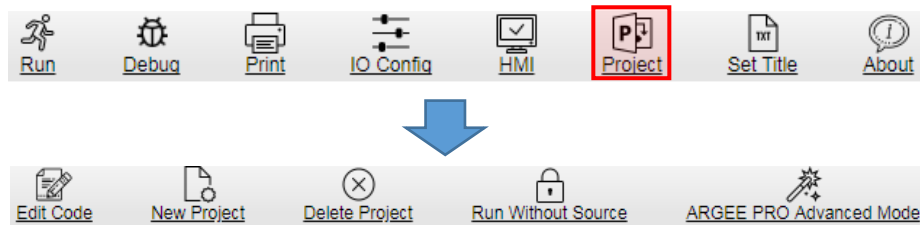
### 5.12.4 HMI

The *HMI* tab allows the user to view their HMI screen. This tab becomes active after the user has already built an HMI. The ARGEE HMI is discussed in Chapter 9 [ARGEE HMI](#).



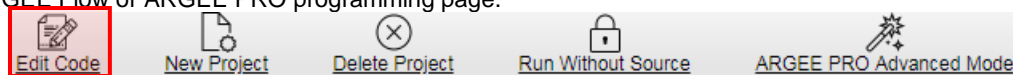
### 5.12.5 Project

When the user clicks on the *Project* tab, they will have access to a completely new ARGEE menu bar.



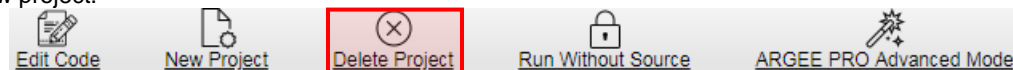
### 5.12.6 Edit Code

The user can find the *Edit Code* tab on many screens in the ARGEE 3 Flow and the ARGEE 3 PRO environment. The user will click *Edit Code* when he wants to leave his current location and return to the ARGEE Flow or ARGEE PRO programming page.



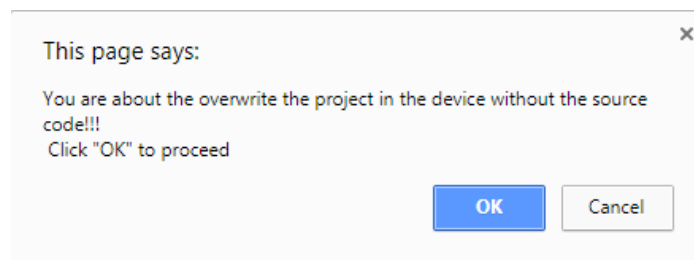
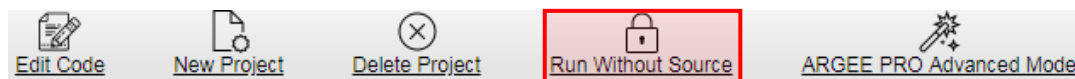
### 5.12.7 Delete Project

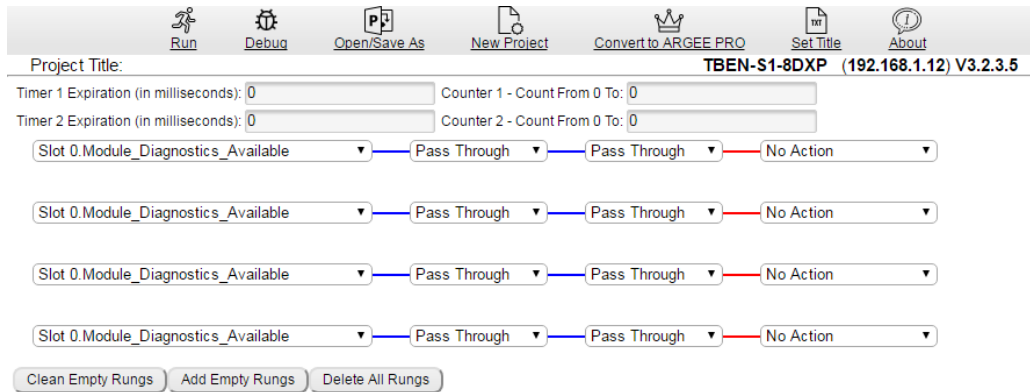
*Delete Project* is different from *New Project* because it erases the project from the device then starts a new project.



### 5.12.8 Run Without Source

Selecting *Run Without Source* will allow the device to run without displaying the actual code. This feature blocks the “end user” from viewing the program that the user wrote. *Run Without Source* is one of ARGEE's security protocols.





If the “end user” tries to log into this device, they will receive the following error message:

**Project without the source code is loaded into the device  
Erase it via the web server to be able to load new ARGEE programs!!!!**

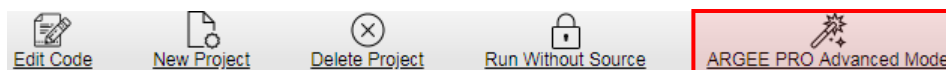


#### NOTE

The user needs to save a master copy of the program before clicking Run Without Source. If the user fails to do this, he will be unable to edit or even view the code in the future.

### 5.12.9 ARGEE PRO Advanced Mode

Clicking on the *ARGEE PRO Advanced Mode* button will expose several new features to the user. In ARGEE PRO Advanced Mode, the user will be able to use the While, For, If, Else If, Else, and Wait Until statements in the Main Task. They will also be able to use multitasking.



#### NOTE

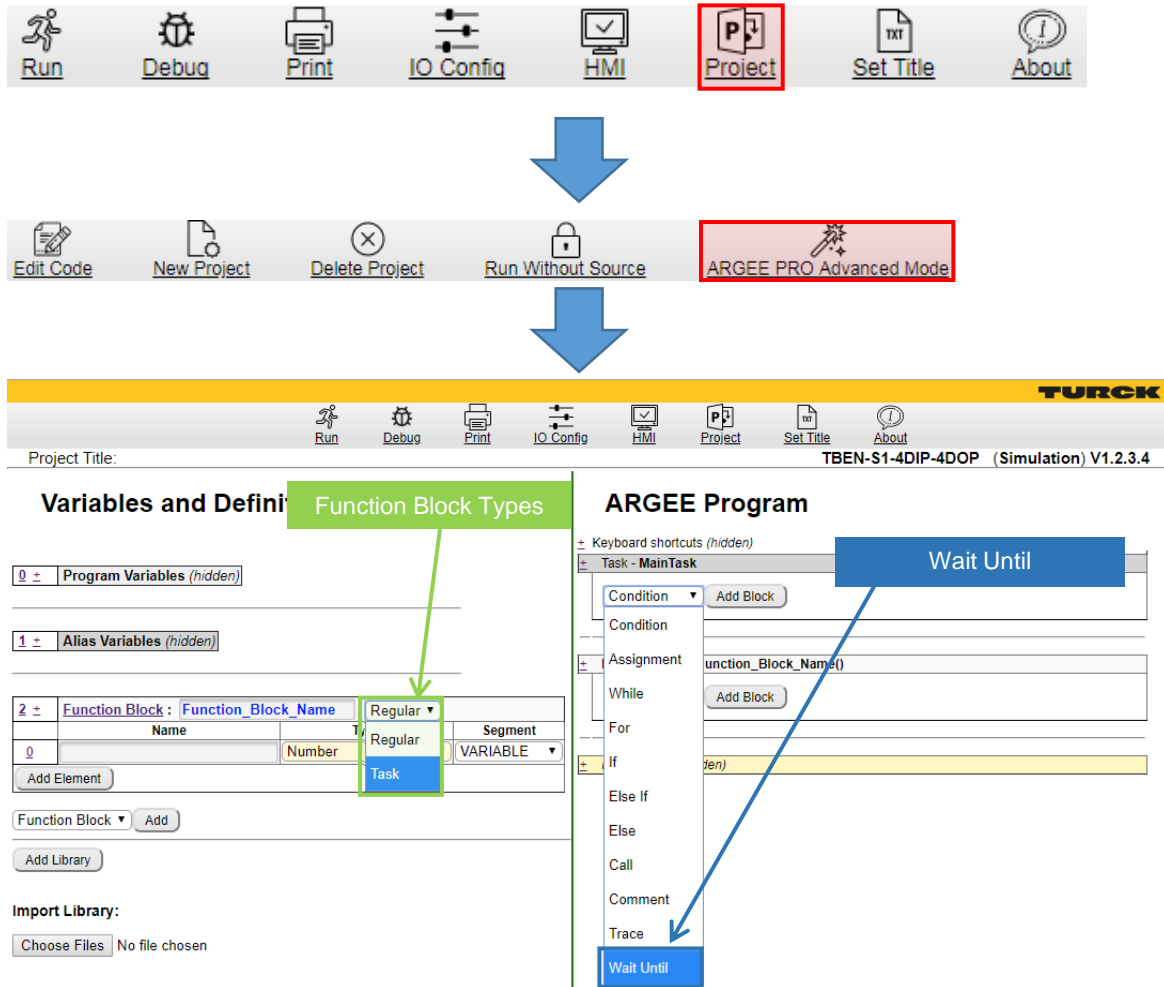
ARGEE PRO Advanced Mode is covered in greater detail in chapter [6 ARGEE PRO Advanced Mode](#).

## 6 ARGEE PRO Advanced Mode

### 6.1 The Basics

ARGEE PRO Advance Mode allows the user to use the While, For, If, Else If, Else and Wait Until statements in the Main Task. It also allows function blocks to be made into their own task. This feature is called multitasking.

- To get from ARGEE PRO to ARGEE PRO Advanced Mode, the user must click on Project, and then ARGEE PRO Advanced Mode.



#### NOTE

Multitasking is explained in chapter [6.2 Function Block Type](#).

## 6.2 Function Block Types

### 6.2.1 Regular

The user will use function block type *Regular* when the user wants the function block to run only when the function block is called from the main task or from another Function Block.

### 6.2.2 Task (Multitasking)

The user will use function block type *Task* when the user wants the function block to run in parallel with the main task. This concept is called multitasking.

## 6.3 Wait Until

*Wait Until* is a very powerful statement that halts the execution of a task until a certain condition is met. .

**Example of *Wait Until*:**

0	Wait Until	Door_Open
1	Assignment	Destination: Light Expression: 1

**Explaining the Example:** Wait until the door is opened, then turn on a light.

Example Multi-tasking using “Wait Until 1”: The task will stop executing for one cycle to allow other tasks to be executed.

2 ±	Function Block : Function_Block_Name		Task ▼
	Name	Type	Segment
0	Multitask	Number ▼	VARIABLE ▼
Add Element			

Function Block ▼
Add

Task - Function_Block_Name			
0 ±	While	1	
0.0	Assignment	Destination: Output_1 Expression: Input_1	
0.1	Wait Until	1	

## 7 Debugger

### 7.1 Debugger Information

If a user is using loops and function blocks in their code, it can become very complicated to follow the next instruction. To help with this, the ARGEE3 environment assists the user by inserting a “breakpoint” in every executable statement in the program. Due to this implementation, the user just needs to use *Halt* and *Step* command to stop and step through their code. In addition to *Halt* and *Step*, toggling *Break Points* and using the *Trace* feature are also useful tools for debugging.

#### 7.1.1 Single Task

If the user created a single-task ARGEE program (i.e. ARGEE PRO Advanced mode has not been enabled), the debugger starts at the top of Main Task and executes it block-by-block down the page until it gets to the end. Then, it starts over. This cycle continues until the user halts ARGEE, or the device is powered off.

#### 7.1.2 Multiple Tasks

If the user created an ARGEE program that uses multiple tasks, the Main Task will execute to completion, then the next task will execute, this process continues until all tasks have been executed. No two tasks may be executed at the same time. However, the user may switch between tasks. This is accomplished by using the Wait Until statement.



#### NOTE

The *Wait Until* statement is discussed in [6.3 Wait Until](#).

---

#### 7.1.3 Break Points

The user can add break points to their code from both the *Edit* menu and the *Debug* menu. The *Toggle Break Point* command is located in the *Main Task* and function block context menus.



#### NOTE

More about *Break Points* is discussed in [5.7.3 Toggle Breakpoint](#).

---

#### 7.1.4 Trace

Trace is a very powerful Debug tool. 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.



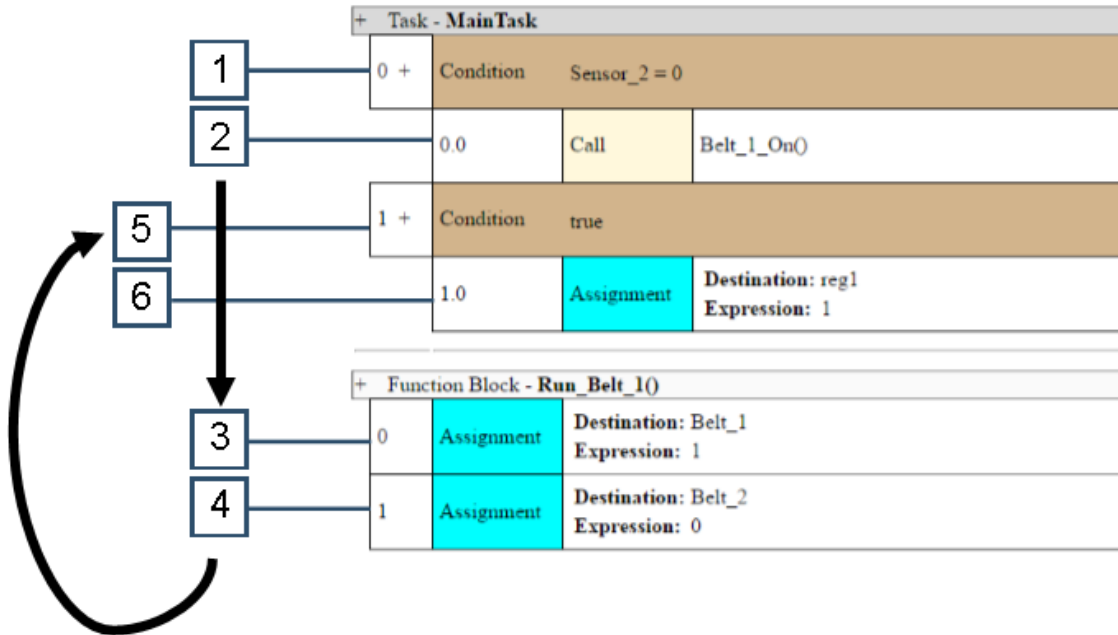
#### NOTE

More about *Trace* is discussed in [5.4.6 Trace](#) and [12.2 Trace Example](#)

---

### 7.1.5 Order of Operation – Calls & Function Blocks

If the user is debugging an ARGEE program that contains function blocks, and is advancing the program one step at a time, they will find that the debugger appears to skip around the program when a Call block is reached. This is because when ARGEE executes a Call block, it jumps down to its function block's definition, and executes that function block-by-block. As soon as the function block has been executed, ARGEE will return to the Call block's location, and continue down the program.



**Explaining the Example:** This is the sequence of calls of an ARGEE program containing a Function Block.

## 7.2 Debug Menu Bar (ARGEE PRO)

When the user clicks *Debug* while in ARGEE PRO, they get a brand new menu bar with many more options. The user can also see the status of every variable, input, output, timer and counter in their program.

The screenshot shows the ARGEE PRO interface. At the top, a menu bar contains icons for Run, Debug (highlighted with a red box), Print, IO Config, HMI, Project, Set Title, and About. A large blue arrow points down to the main interface. The main interface has a yellow header with the TURCK logo. Below the header is a sub-menu bar with icons for Edit Code, HMI, Halt, Step, Continue, and Modify Vars. Below this, a status bar shows 'Loadable code size 1210 bytes (out of 43008 bytes) Project size: 2228 bytes (out of 262144 bytes)'. The main area is split into two panes. The left pane, titled 'Runtime Status', shows a tree view with 'TRACE' expanded, displaying 'PROG\_CYCLE\_TIME: 2', 'PLC\_CONNECTED: 0', 'VARIABLE\_1: 0', and 'VARIABLE\_2: 0'. Below this is 'MainTask' with a list of local IOs: 'Local IO: TBEN\_S1\_8DXP\_GW', 'Local IO: Basic - Input', 'Local IO: Basic - Output', 'Local IO: Basic - Diagnostics', 'Local IO: Diagnostic - Input', 'Local IO: Input\_Latch\_Ch0\_7 - Input', and 'Local IO: Input\_Latch\_Ch0\_7 - Output'. The right pane, titled 'ARGEE Program', shows a table for 'Task - MainTask'.

Condition	IO_Slot1_Input_Input_value_0
0.0	Assignment Destination: Variable_1 Expression: 0
0.1	Assignment Destination: Variable_2 Expression: 0

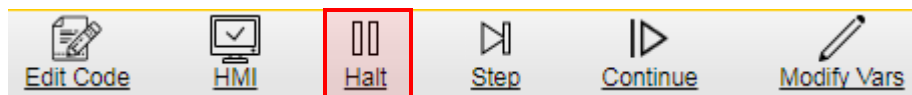


### NOTE

The blue links in the left hand column are clickable and will center the window on that specific area of the code. Active conditional statements (while, for, ifs/conditions) show up as green. Inactive conditions show up as gray. Wait\_until statements that are actively waiting will show with yellow background.

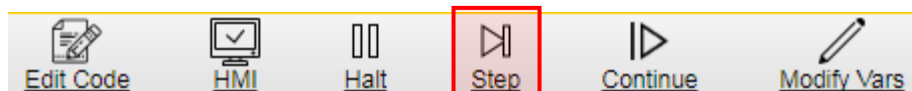
### 7.2.1 Halt

The user will use *Halt* to pause execution of the ARGEE program.



### 7.2.2 Step

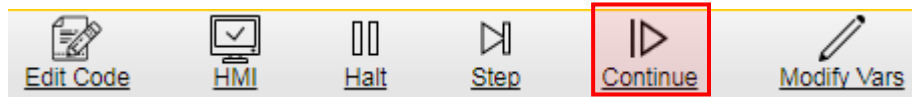
If the ARGEE program is halted, *Step* allows the user to step through the code, one line at a time.





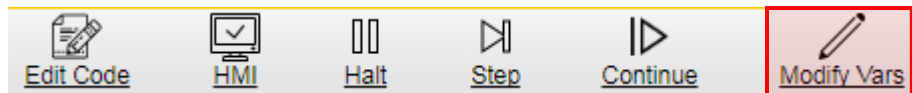
### 7.2.3 Continue

If the ARGEE program is halted, *Continue* allows the program to resume normal execution.



### 7.2.4 Modify Vars (Modify Variables)

Clicking the *Modify Vars* button will allow the user to manually change variables in the Runtime Status window. Recently modified variable values show up with “yellow” backgrounds.



[Finish Modifications](#)

---

Project Title:
TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

**Runtime Status**

TRACE

PROG\_CYCLE\_TIME: 2

PLC\_CONNECTED: 0

VARIABLE\_1: 25

VARIABLE\_2: 100

MainTask

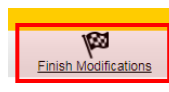
- Local IO: TBEN\_S1\_8DXP\_GW
- Local IO: Basic - Input
- Local IO: Basic - Output
- Local IO: Basic - Diagnostics
- Local IO: Diagnostic - Input

**ARGEE Program**

Task - MainTask		
Q ±	Condition	IO_Slot1_Input_Input_value_0
Q.0	Assignment	Destination: Variable_1 Expression: 0
Q.1	Assignment	Destination: Variable_2 Expression: 0

### 7.2.5 Finish Modifications

When the user is done modifying variables in the Runtime Status window, he can click on *Finish Modifications* to apply those changes.



[Finish Modifications](#)

---

Project Title:
TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

**Runtime Status**

TRACE

PROG\_CYCLE\_TIME: 2

PLC\_CONNECTED: 0

VARIABLE\_1: 25

VARIABLE\_2: 100

MainTask

- Local IO: TBEN\_S1\_8DXP\_GW
- Local IO: Basic - Input
- Local IO: Basic - Output
- Local IO: Basic - Diagnostics
- Local IO: Diagnostic - Input
- Local IO: Input\_Latch\_Ch0\_7 - Input

**ARGEE Program**

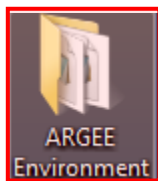
Task - MainTask		
Q ±	Condition	IO_Slot1_Input_Input_value_0
Q.0	Assignment	Destination: Variable_1 Expression: 0
Q.1	Assignment	Destination: Variable_2 Expression: 0

## 8 ARGEE Simulation Mode

For individuals new to programming, or unfamiliar with ladder logic, ARGEE offers a simulation mode. The simulation mode enables users to write and test their program without investing in any hardware. Below are the steps needed to implement the Simulation Mode.

### 8.1 Opening the Environment

- Open the ARGEE Environment and double click on `argee_startup.html`.



Name	Date modified	Type
Earlier_Environments	1/31/2017 1:36 PM	File folder
internal	1/31/2017 1:36 PM	File folder
argee_startup.html	1/31/2017 1:36 PM	Chrome HTML Do...

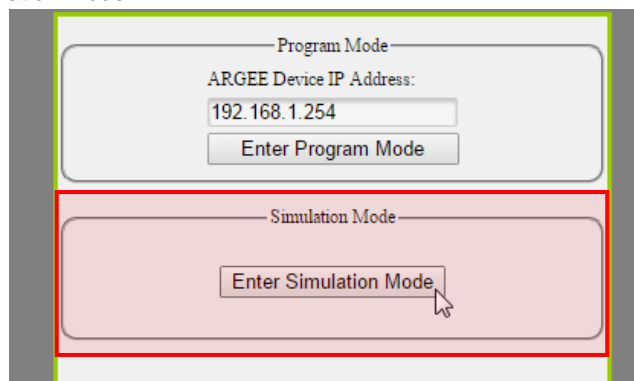


#### NOTE

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

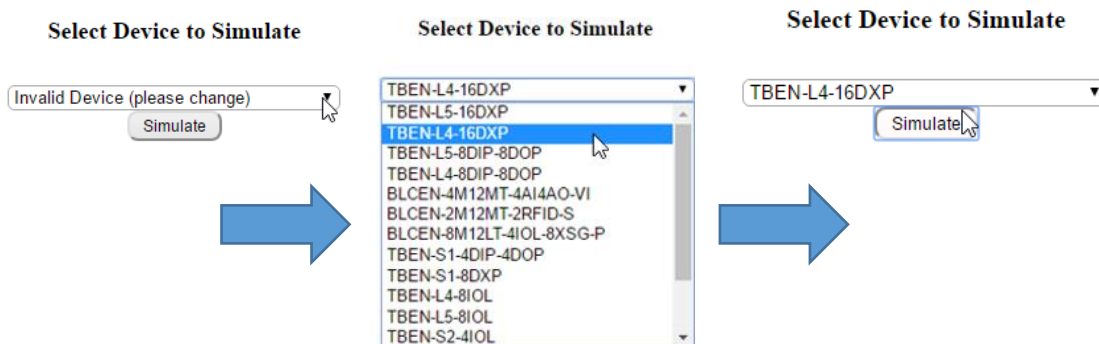
### 8.2 Logging into Simulation Mode

- Click Enter Simulation Mode.



### 8.3 Selecting Device to Simulate

Select a device to simulate from the drop down menu.



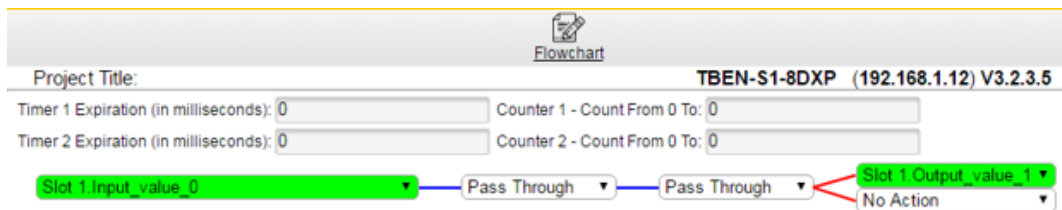


#### NOTE

- Not all ARGEE 3 supported devices are available in simulation mode.
- The default Simulation Mode environment is Flow Chart

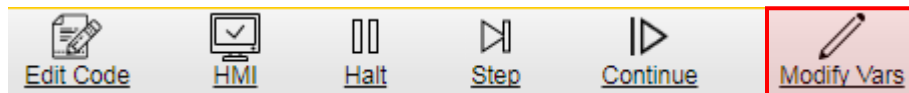
### 8.3.1 Flow Chart Simulation Mode

- To force an input value, double-click the input



### 8.3.2 Pro Simulation Mode

- To force an input value, Click the *Modify Vars* button.



- Enter the input value, and select *Finish Modifications*

**Runtime Status**

TRACE  
 PROG\_CYCLE\_TIME: 2  
 PLC\_CONNECTED: 0  
 VARIABLE\_1: 25  
 VARIABLE\_2: 100

**ARGEE Program**

Task - MainTask	Condition	IO_Slot1_Input_Input_value_0
0.0	Assignment	Destination: Variable_1 Expression: 0
0.1	Assignment	Destination: Variable_2 Expression: 0



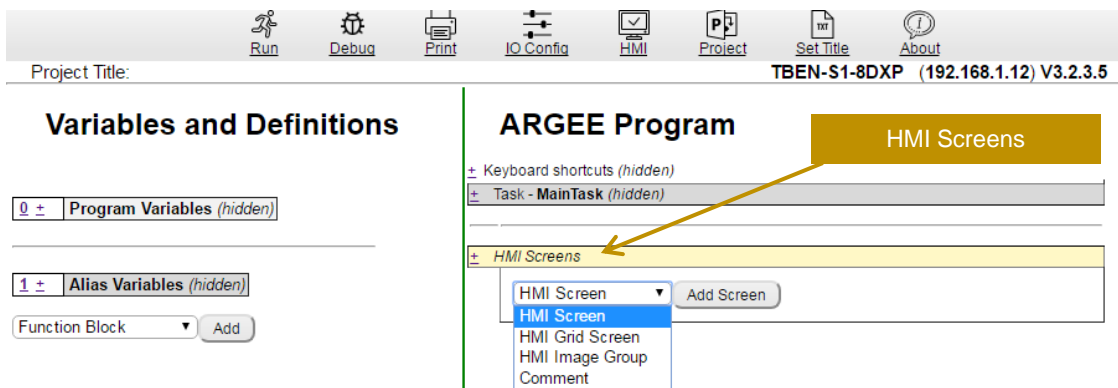
#### NOTE

Timers in Simulation Mode may not be as accurate as using a real ARGEE device.

## 9 ARGEE HMI

### 9.1 The Basics

The user will use the HMI screens if the user wants to create an HMI. The ARGEE HMI is composed of screens, sections and sections elements. The ARGEE HMI can also be viewed on any device that is on the network by going to [http://\(Device IP Address\)/hmi.html](http://(Device IP Address)/hmi.html) in a Google Chrome or Firefox web browser.



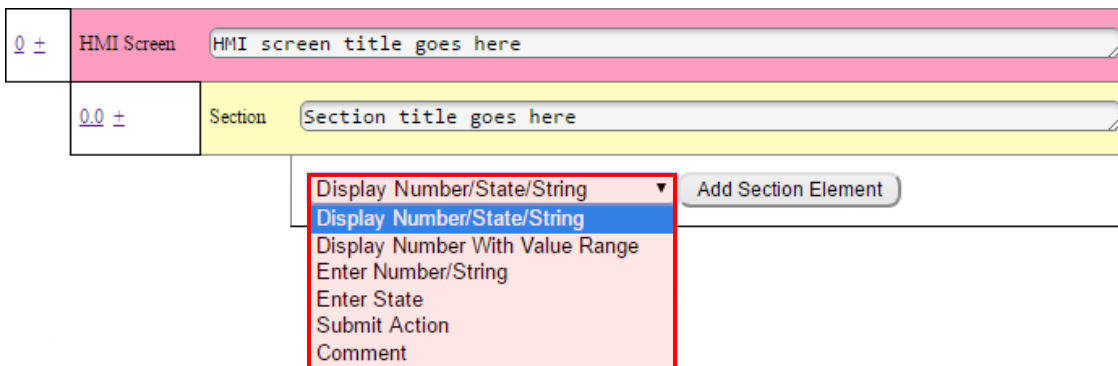
### 9.2 HMI Screen

When the user selects *HMI Screen* from the HMI drop-down list and clicks *Add Screen*, the users will see a new rung of logic pop-up. The user can enter that specific HMI screens title in this box. The user will also have the ability to add a new section to the HMI by highlighting *Section* and clicking *Add Section*.



#### 9.2.1 Sections

After the user adds a new section, the user will be able to add elements to the HMI screen by highlighting the desired element and clicking the *Add Section Element* button.



### 9.2.1.1 Display Number/State/String

The user will use the *Display Number/State/String* element if the user wants to display a number, state, or string in the HMI.

**Example of *Display Number/State/String*:**

The screenshot shows the ARGEE Program editor interface. At the top, there is a toolbar with icons for Run, Debug, Print, IO Config, HMI, Project, Set Title, and About. Below the toolbar, the Project Title is "TBEN-S1-8DXP (192.168.1.12) V3.2.3.5".

The left pane is titled "Variables and Definitions". It contains two sections: "Program Variables" and "Alias Variables (hidden)". The "Program Variables" section has a table with the following data:

	Name	Type
1	Register_1	Number

Below the table is an "Add Variable" button. The "Alias Variables (hidden)" section has a "Function Block" dropdown and an "Add" button.

The right pane is titled "ARGEE Program". It shows a task named "MainTask" with a condition set to "true". Below the condition is an "Assignment" block with the following properties:

- Destination: Register\_1
- Expression: 1

Below the assignment block is a "Condition" dropdown and an "Add Block" button.

Below the task is a section titled "HMI Screens". It contains a table with the following data:

	HMI Screen	Section	Display Number/State/String	Title	Variable	Units
0	Inventory	Cups		Total	Register_1	Cup(s)

**(HMI View):**

The screenshot shows the HMI View interface. At the top, there is a toolbar with icons for Edit Code and Debug. Below the toolbar, there is a link that says "To test the page on the device (click here)".

The main area is divided into two sections: "Screens" and "Inventory". The "Screens" section has a button labeled "Inventory". The "Inventory" section has a display element with the following properties:

- Section: Cups
- Title: Total
- Variable: 1
- Units: Cup(s)

**Explaining the Example:** The user created an inventory HMI screen that shows how many cups they currently have. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar.

### 9.2.1.2 Display Number with Valid Range

The user will use the *Display Number with Valid Range* element if the user wants to make sure a number displayed on the HMI stays within a certain range. If the number is within a certain range, the associated HMI section will be green. If the number is outside the specified range, the associated HMI section will be red.

**Example of *Display Number with Valid Range*:**

The screenshot shows the software interface with the following components:

- Project Title:** TBEN-S1-8DXP (192.168.1.12) V3.2.3.5
- Variables and Definitions:**
  - Program Variables:** A table with columns Name and Type. It contains one variable: Register\_1 (Type: Number).
  - Alias Variables (hidden):** A section with a Function Block dropdown and an Add button.
- Task - MainTask:** A task configuration area with a Condition (true) and an Assignment block. The Assignment block has Destination: Register\_1 and Expression: 2.
- HMI Screens:** A section with an HMI Screen named Inventory. It contains a Section named Cups, which includes a 'Display Number With Valid Range' element. The element's configuration is shown in a red box: Title: Total, Variable: Register\_1, Units: Cup(s), Min Valid Value: 2, Max Valid Value: 10.

(HMI View)

The HMI View shows two states of the 'Inventory' screen:

- Top Screenshot:** The 'Cups' section is green, indicating the value is within the valid range. The 'Total' value is 2 Cup(s).
- Bottom Screenshot:** The 'Cups' section is red, indicating the value is outside the valid range. The 'Total' value is 1 Cup(s).

**Explaining the Example:** The user created an inventory HMI screen that shows how many cups the user currently has in inventory. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar. When there is only one cup left in the inventory, the HMI turns red, letting the user know they need to order more cups.

### 9.2.1.3 Enter Number/String

The user will use the *Enter Number/String* element if the user wants to create an editable field on the HMI.

**Example of *Enter Number/String*:**

Project Title: **TBEN-S1-8DXP (192.168.1.12) V3.2.3.5**

#### Variables and Definitions

	Name	Type
1	Register_1	Number
2	Register_2	Number
3	Add_Cups_To_Inventory	Number
4	Remove_Cups_From_Inventory	Number

**ARGEE Program**

+ Keyboard shortcuts (hidden)

**Task - MainTask**

Condition	Assignment	Destination	Expression
0.0	Assignment	Add_Cups_To_Inventory	0
	Assignment	Register_2	(Register_2 + Register_1)
1.0	Assignment	Remove_Cups_From_Inventory	0
	Assignment	Register_2	(Register_2 - Register_1)

**HMI Screens**

**HMI Screen Inventory**

Section	Element	Title	Variable	Units
Cups	0.0.0	Display Number/State/String	Register_2	Cup(s)
	0.0.1	Enter Number/String	Register_1	Cup(s)
	0.0.2	Submit Action	Add Cups to Inventory	
	0.0.3	Submit Action	Remove Cups from Inventory	



### (HMI View)

**Inventory**

Cups	
Total	0 Cup(s)
Add/Remove Inventory	<input type="text" value="0"/> Cup(s)
<input type="button" value="Add Cups to Inventory"/>	
<input type="button" value="Remove Cups from Inventory"/>	

**Inventory**

Cups	
Total	0 Cup(s)
Add/Remove Inventory	<input type="text" value="4"/> Cup(s)
<input type="button" value="Add Cups to Inventory"/>	
<input type="button" value="Remove Cups from Inventory"/>	

**Inventory**

Cups	
Total	4 Cup(s)
Add/Remove Inventory	<input type="text" value="1"/> Cup(s)
<input type="button" value="Add Cups to Inventory"/>	
<input type="button" value="Remove Cups from Inventory"/>	

**Inventory**

Cups	
Total	3 Cup(s)
Add/Remove Inventory	<input type="text" value="1"/> Cup(s)
<input type="button" value="Add Cups to Inventory"/>	
<input type="button" value="Remove Cups from Inventory"/>	

**Explaining the Example:** The user wanted to create an HMI screen that shows how many cups he currently has in inventory. Additionally, the user wanted the ability to easily add and remove cups from his inventory while keeping his total inventory up-to-date. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar. The user used a *Display Number* element to display the total cups in his inventory. The user used an *Enter Number* element to create an editable field on his HMI. Lastly, the user created two *Submit Action* elements which both perform some math and update the total inventory with the new value.



#### NOTE

The Submit Action element will be talked about later in this chapter in section [9.4.3 Action](#).

---



### 9.2.1.4 Enter State

The user will use the *Enter State* element if the user wants to change program state through the HMI.

**Example of *Enter State*:**



(HMI View)



(HMI View)

**Cook Book**

**Recipes**

Currently Making:	CHILI
Change Recipe To:	<div>CHILI TOMATO_SOUP CHICKEN_SOUP CHILI</div>

**Cook Book**

**Recipes**

Currently Making:	CHILI
Change Recipe To:	CHICKEN_SOUP ▼
	<div>Submit</div>

**Cook Book**

**Recipes**

Currently Making:	CHICKEN_SOUP
Change Recipe To:	CHICKEN_SOUP ▼
	<div>Submit</div>

**Explaining the Example:** The user wanted to be able to change the soup recipes from the HMI. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar. The user used the *Display State* element to display the machines current state. The user used the *Enter State* element to give him the ability to change between different recipes. Lastly, the user created a *Submit Action* element submitted the changes to the machine.



NOTE

The Submit Action element is explained in detail in section [9.2.1.5 Action](#).

---

### 9.2.1.5 Submit Action

The user will use the *Submit Action* element when the user wants to create a button on their HMI which either confirms changes in editable HMI fields or acts as a start button to some other chain of events.

#### Example of *Submit Action*:

Project Title: **TBEN-S1-8DXP (192.168.1.12) V3.2.3.5**

#### Variables and Definitions

Program Variables		
	Name	Type
1	Submit	Number
2	Register_1	Number

Alias Variables (hidden): 1

Function Block: Add

#### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask

Condition			Submit	
0.0	Assignment	Destination:	Submit	Expression: 0
0.1	Assignment	Destination:	Register_1	Expression: Register_1 + 1

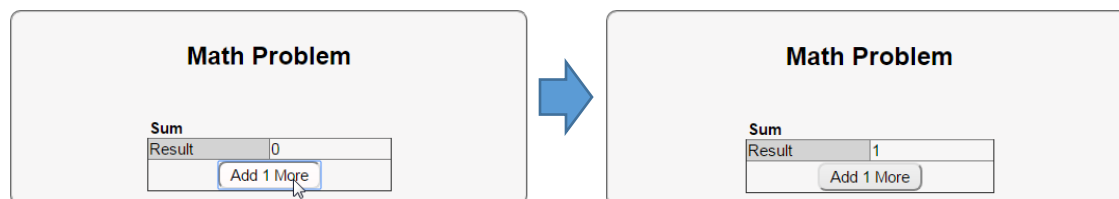
Assignment Add Block

Condition Add Block

HMI Screens

HMI Screen			Math Problem	
0.0.0	Section	Sum		
0.0.0	Display Number/State/String	Title:	Result	Variable: Register_1
0.0.1	Submit Action	Title:	Add 1 More	Variable: Submit

#### (HMI View)



**Explaining the Example:** The user created a simple HMI which increase the current value in Register\_1 by one every time the *Add 1 More* button is pressed. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar. The user used the *Display State* element to display the current value in Register\_1. The user used the *Submit Action* element to increment the value in Register\_1.

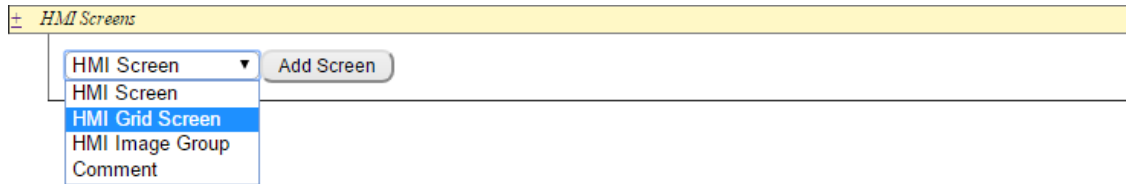


#### NOTE

The reason the user always sets the Submit Action back to "0" in the Main Task is because the user only wants the action to happen one time. If the user did not load a "0" into the Submit Action variable, the action would continue to happen every scan cycle.

## 9.3 HMI Grid Screen

The user will use *HMI Grid Screen* to create an HMI with custom graphics and colors. The *HMI Grid Screen* consists of a single table which has a user specified number of rows and cells and elements.

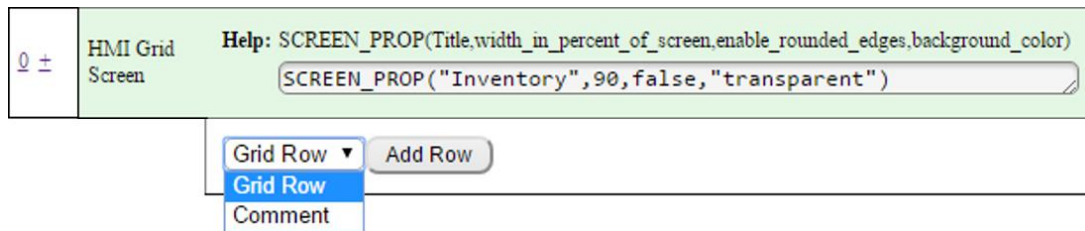


### 9.3.1 HMI Grid Screen

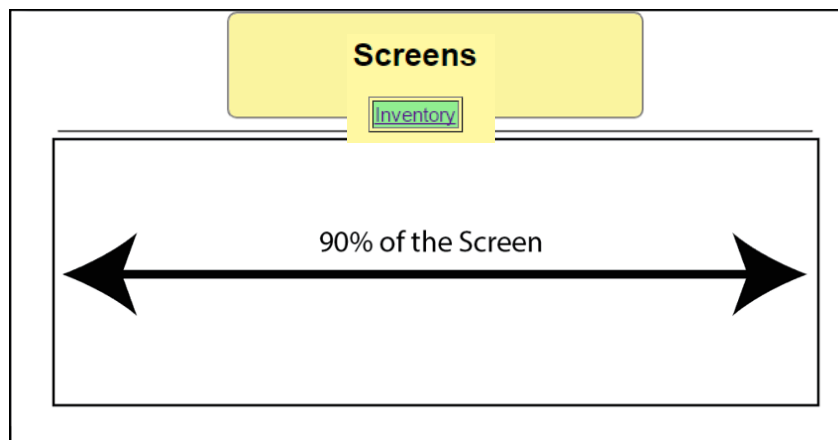
The HMI Grid Screen element has four arguments:

- Screen Title
- Screen Width
- Rounded Edges (True / False)
- Background Color

**Example of HMI Grid Screen:**



(HMI View)



**Explaining the example:** The user created a new HMI grid screen titled "Inventory" that stretches 90% of the screen. The user also set his rounded edges to false and his background color to transparent.



#### NOTE

ARGEЕ supports all colors that your web browser supports. This user can either type in the Hex value or the X11 color name. For example: the user could type "whitesmoke" or "#F5F5F5."

### 9.3.2 Grid Row

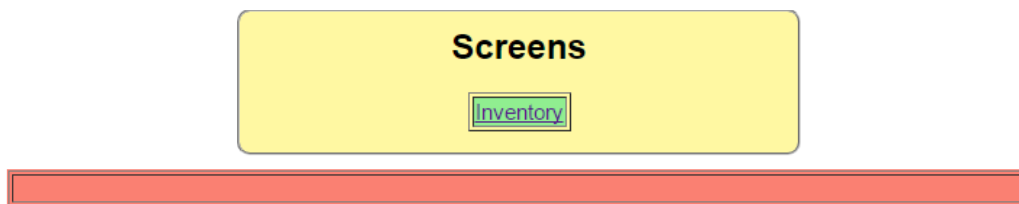
The *Grid Row* element has one argument:

- Background Color

**Example of *Grid Row*:**

0.0 ±	Grid Row	<p><b>Help:</b> ROW_PROP(background_color)</p> <input type="text" value="ROW_PROP('salmon')"/>
	<div>Grid Cell ▾</div> <div>Grid Cell</div> <div>Comment</div>	<div>Add Section</div>

(HMI View)



**Explaining the example:** The user added a row to his “Inventory” screen, and set the background color to salmon.

### 9.3.3 Grid Cell

The *Grid Cell* element has two arguments:

- Column Span
- Border Style
- 0 = No border
- 1 = Border around every element in the cell
- 2 = Single border around the entire cell

**Example of *Grid Cell*:**

0.0.0 ±	Grid Cell	<p><b>Help:</b> CELL_PROP(column_span,border_style)</p> <input type="text" value="CELL_PROP(2,1)"/>
0.0.0.0	Grid Element	<p><b>Help:</b> STATIC_TEXT(Text,color,size,background_color)</p> <input type="text" value="STATIC_TEXT('Column Span of 2','black',3,'lavender')"/>
0.0.1 ±	Grid Cell	<p><b>Help:</b> CELL_PROP(column_span,border_style)</p> <input type="text" value="CELL_PROP(1,1)"/>
0.0.1.0	Grid Element	<p><b>Help:</b> STATIC_TEXT(Text,color,size,background_color)</p> <input type="text" value="STATIC_TEXT('Column Span of 1','black',3,'yellow')"/>

(HMI View)



**Explaining the example:** The user separated the row into two columns. The lavender colored *Grid Cell* has a column span of two and the yellow colored *Grid Cell* has a column span of one.



NOTE

The Static Text element will be discussed later in this chapter in section [9.6.4 Static Text](#).

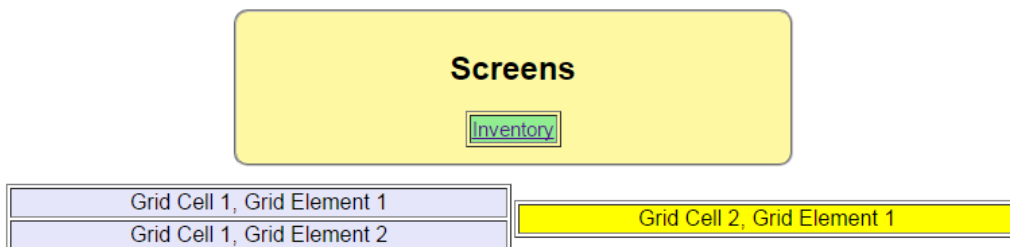
### 9.3.4 Grid Element

The *Grid Element* has several built-in functions. The user can access them by clicking inside the *Grid Element* and pressing Ctrl-f. Additionally, the height of the column is controlled by the number of *Grid Elements* in a *Grid Cell*.

**Example of *Grid Element*:**

<u>0.0.0</u> ±	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(2,2)
<u>0.0.0.0</u>	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color) STATIC_TEXT("Grid Cell 1, Grid Element 1","black",3,"lavender")
<u>0.0.0.1</u>	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color) STATIC_TEXT("Grid Cell 1, Grid Element 2","black",3,"lavender")
<u>0.0.1</u> ±	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,1)
<u>0.0.1.0</u>	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color) STATIC_TEXT("Grid Cell 2, Grid Element 1","black",3,"yellow")

(HMI View)



**Explaining the example:** The user separated the row into two columns. The lavender colored *Grid Cell* has a column span of two and the yellow colored *Grid Cell* has a column span of one. The height of the *Grid Row* automatically expanded to accommodate the second element in the lavender colored *Grid Cell*.



## NOTE

The Static Text element will be discussed later in this chapter in section [9.3.4.4 Static Text](#).

### 9.3.4.1 Display Value

The *Display Value* element has six arguments:

- Title
- Variable Name
- Units
- Font Color
- Font Size
- Background Color

**Example of *Display Value*:**

<a href="#">0 ±</a>	Program Variables	
	Name	Type
1	Thermometer	Number ▼

<a href="#">0.0.0.0</a>	Grid Element	Help: <code>DISPLAY_VALUE(Title,var,units_string,color,size,background_color)</code> <code>DISPLAY_VALUE("Temperature",Thermometer,"C°","black",3,"transparent")</code>
-------------------------	--------------	--

(HMI View)

Temperature	42 °C
-------------	-------

**Explaining the example:** The user wants to display the value of *Thermometer* in degrees Celsius.

### 9.3.4.2 Enter Value

The *Enter Value* element has six arguments:

- Title
- Variable Name
- Units
- Font Color
- Font Size
- Background Color

### Example of *Enter Value*:

0 ±	Program Variables	
	Name	Type
1.0	INIT : 0	
1	Total	Number ▼
2	Inventory	Number ▼
3	Submit	Number ▼

0.0.0.0	Grid Element	<b>Help:</b> DISPLAY_VALUE(Title,var,units_string,color,size,background_color) <code>DISPLAY_VALUE("Total Inventory",Total,"units","black",3,"transparent")</code>
0.0.0.1	Grid Element	<b>Help:</b> ENTER_VALUE(Title,var,units_string,color,size,background_color) <code>ENTER_VALUE("Inventory Added", Inventory,"unit","black",3,"transparent")</code>
0.0.0.2	Grid Element	<b>Help:</b> BUTTON(Title,var,color,size,background_color) <code>BUTTON("Add to Inventory",Submit,"black",3,"transparent")</code>

### (HMI View)

Total Inventory	0 units
Inventory Added	<input type="text" value="42"/> unit
<input type="button" value="Add to Inventory"/>	

Total Inventory	42 units
Inventory Added	<input type="text" value="42"/> unit
<input type="button" value="Add to Inventory"/>	

**Explaining the example:** The user wants to keep track of his inventory. As units come in, he types in the quantity and clicks *Add to Inventory*. The total inventory increments on the input.



#### NOTE

The Button element will be discussed later in this chapter in section [9.3.4.3 Button](#).



### 9.3.4.3 Button

The *Button* element has five arguments:

- Title
- Variable Name
- Color
- Size
- Background Color

**Example of *Button*:**

0 ±	Program Variables	
	Name	Type
1.0	INIT : 0	
1	Total	Number
2	Submit	Number

0.0.0.0	Grid Element	<b>Help:</b> DISPLAY_VALUE(Title,var,units_string,color,size,background_color) <code>DISPLAY_VALUE("Total",Total,"unit","black",3,"transparent")</code>
0.0.0.1	Grid Element	<b>Help:</b> BUTTON(Title,var,color,size,background_color) <code>BUTTON("Add 1",Total,"black",3,"transparent")</code>

(HMI View)

Total0 unit

Add 1

Total1 unit

Add 1

**Explaining the example:** The user added a button to increment the total unit count (increment code not displayed). When the button is pressed, the total increments by one.

### 9.3.4.4 Static Text

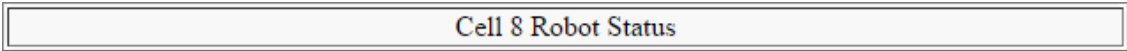
The *Static Text* element has four arguments:

- Text
- Font Color
- Font Size
- Background Color

**Example of *Static Text*:**

0.0.0.0	Grid Element	<b>Help:</b> STATIC_TEXT(Text,color,size,background_color) <code>STATIC_TEXT("Cell 8 Robot Status","black",3,"transparent")</code>
---------	--------------	---

(HMI View)



**Explaining the example:** The user wants to label certain information in the HMI “Cell 8 Robot Status.”

9.3.4.5 Screen List

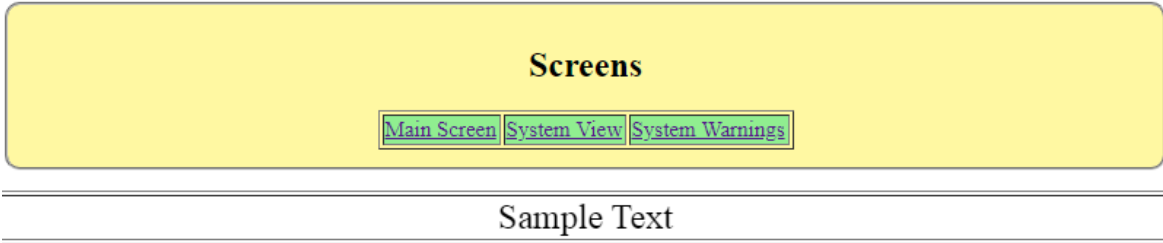
The developer can use the *Screen List* element to have more control over the HMI Screens. They can move the location of where it is displayed, alter the background color, and change the text size.

- Title
- Title Font Size
- Title Color

**Example of *Screen List*:**

1.3 ±	Grid Row	Help: ROW_PROP(background_color) ROW_PROP("salmon")
1.3.0 ±	Grid Cell (hidden)	CELL_PROP(1,1)
1.3.1 ±	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(2,0)
1.3.1.0	Grid Element	Help: SCREEN_LIST(Title,title_font_size,title_color) SCREEN_LIST("Screen Links","1.0","black")
1.3.2 ±	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,1)
1.3.2.0	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Sample Text","black","1.5","transparent","center")
1.3.3 ±	Grid Cell (hidden)	CELL_PROP(1,1)

(HMI View without the *Screen List* element)



(HMI View with the *Screen List* element)



0.0.0.0	Grid Element	<b>Help:</b> SCREEN_LIST(Title,title_font_size,title_color) SCREEN_LIST("Screens",5,"black")
---------	--------------	---

(HMI View without the *Screen List* element)

### Screens

Main Screen
System View
System Warnings

Total Inventory	0 units
Inventory Added	<input type="text" value="0"/> unit
<input type="button" value="Add to Inventory"/>	

(HMI View with the *Screen List* element)

### Screens

Main Screen
System View
System Warnings

Total Inventory	0 units
Inventory Added	<input type="text" value="0"/> unit
<input type="button" value="Add to Inventory"/>	

**Explaining the Example:** The user has three HMI screens: “Main Screen,” “System View,” and “System Warnings.” By using the *Screen List* element, the user is able to move his screen list anywhere on his HMI, alter the text size, and change the background color.



**NOTE**

The screen list can be placed anywhere on the HMI by modifying the Grid Row and Grid Cell properties.

### 9.3.4.6 Static Graphics

The *Static Graphics* element has three arguments:

- Image File Variable
- Background Color
- Zoom Percentage

**Example of *Static Graphics*:**

0.0.0.0	Grid Element	<b>Help:</b> STATIC_GRAPHICS(image_file_variable,background_color,default_zoom) STATIC_GRAPHICS("Turck Logo","transparent",100)
---------	--------------	--

(HMI View)



**Explaining the example:** The user imported a static image to display on the HMI.



**NOTE**

Importing images will be discussed later in this chapter in section [9.3 HMI Screen](#).

**9.3.4.7 Multi-State Display String**

The user will use the *Multi-State Display String* element when the users wants to show different strings when a change of state occurs. The *Multi-State Display String* element has at least 11 arguments, more maybe used depending on how many strings the developer is using.

- Title
- Variable
- Font Size
- Title Color
- Background Color
- Value 1
- Image 1
- Background 1
- Value 2
- Image 2
- Background 2
- ...

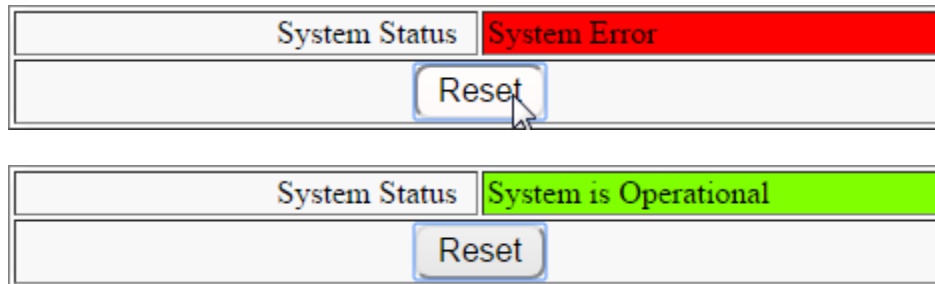
**Example of *Multi-State Display String*:**

<u>0</u> ±	Program Variables	
	Name	Type
<u>1.0</u>	INIT : System_ERROR	
<u>1</u>	System_State	Number ▼
<u>2</u>	Submit	Number ▼

<u>3</u> ±	States
	Name
<u>0</u>	System_OK
<u>1</u>	System_ERROR

<a href="#">0.0.0.0</a>	Grid Element	<b>Help:</b> MULTI_STATE_DISPLAY_STRING(Title,var,size,title_color,title_background_color,value1,color1,background1,.....) <pre>MULTI_STATE_DISPLAY_STRING("System Status",System_State,3,"black","transparent", 0, "System is Operational","black","chartreuse", 1, "System Error", "black", "red")</pre>
<a href="#">0.0.0.1</a>	Grid Element	<b>Help:</b> BUTTON(Title,var,color,size,background_color) <pre>BUTTON("Reset",Submit,"black",3,"transparent")</pre>

(HMI VIEW)



**Explaining the Example:** The user wrote some code to monitor the System\_State (not displayed). When the System\_State changes from System\_OK to System\_ERROR, ARGEE will display the user's specified strings.

### 9.3.4.8 Multi-State Display Graphics

The user will use the *Multi-State Display Graphics* element when the users wants to show different graphics when a change of state occurs. The *Multi-State Display Graphics* element has at least 12 arguments, maybe more depending on how many images the user needs.

- Title
- Variable
- Font Size
- Title Color
- Background Color
- Image Zoom Percentage
- Value 1
- Image 1
- Background 1
- Value 2
- Image 2
- Background 2
- ...

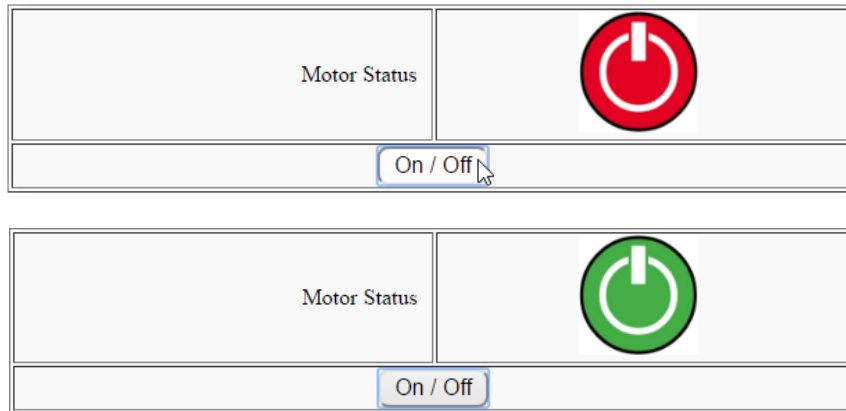
**Example of Multi-State Display Graphics:**

<a href="#">0 ±</a>	Program Variables	
	Name	Type
<a href="#">1</a>	Current_Motor_State	State/Enum ▼
<a href="#">2</a>	Submit	Number ▼

<a href="#">3 ±</a>	States
	Name
<a href="#">0</a>	Motor_OFF
<a href="#">1</a>	Motor_ON

<a href="#">0.0.0.0</a>	Grid Element	<b>Help:</b> MULTI_STATE_DISPLAY_GRAPHICS(Title,var,title_size,title_color,title_background_color,image_zoom_level,value1,image1,background1...) <div>MULTI_STATE_DISPLAY_GRAPHICS("Motor Status",Current_Motor_State,3,"black","transparent",30,0,"Red Button","transparent",1,"Green Button","transparent")</div>
<a href="#">0.0.0.1</a>	Grid Element	<b>Help:</b> BUTTON(Title,var,color,size,background_color) <div>BUTTON("Submit",Submit,"black",3,"transparent")</div>

### (HMI View)



**Explaining the Example:** The user wrote some On / Off code to control the Current\_Motor\_State (not displayed). The user also imported red and green power images to represent motor state. When the user clicks the On / Off button, it changes the Current\_Motor\_State which then changes the image displayed in the HMI.



#### NOTE

Importing images is discussed later in this chapter in section [9.3 HMI Screen](#).

### 9.3.4.9 Dropdown List

Dropdown List is used to give the user a list of options to change a variable. The *Dropdown List* element has at least 9 arguments, more maybe used depending on how many options the developer needs:

- Title
- Var
- Size
- Title Color
- Background Color
- Value 1
- Text 1
- Value 2
- Text 2
- ...

### Example of Multi-State Display Graphics:

0 ±	Program Variables	
	Name	Type
1	Total	Number ▼
2	Inventory	Number ▼
3	Submit	Number ▼

0.0.1.0	Grid Element	Help: <code>DISPLAY_VALUE(Title,var,units_string,color,size,background_color)</code> <code>DISPLAY_VALUE("Total Inventory",Total,"unit","black","1.5","transparent")</code>
0.0.1.1	Grid Element	Help: <code>DROPDOWN_LIST(Title,var,size,title_color,background_color,value1,text1,value2,text2.....)</code> <code>DROPDOWN_LIST("",Inventory,"1.5","black","transparent",-2,"Subtract</code>
0.0.1.2	Grid Element	Help: <code>BUTTON(Title,var,color,size,background_color)</code> <code>BUTTON("Change Inventory Total",Submit,"black","1.5","transparent")</code>

Total Inventory	7unit
Add One	▼
Change Inventory Total	

**Explaining the Example:** The developer wrote some code and created a simple Inventory HMI. The user can add or subtract one or two units from the Total Inventory using the dropdown list.

### 9.3.4.10 Display Value with Health

The *Display Value with Health* element has six arguments:

- Title
- Title color
- Font Size
- Variable Name
- Units
- Health Variable Name
  - 0 = Green
  - 1 = Yellow
  - 2 = Red

### Example of *Display Value with Health*:

0 ±	Program Variables	
	Name	Type
1.0	INIT : 4	
1	Total	Number ▼
2.0	INIT : 0	
2	Health_Variable	Number ▼

0.0.0.0	Grid Element	Help: <code>DISPLAY_VALUE_WITH_HEALTH(Title,title_color,size,var,units_string,health_var)</code> <code>DISPLAY_VALUE_WITH_HEALTH("Inventory","black",5,Total,"Cups",Health_Variable)</code>
---------	--------------	--

### (HMI View)

Inventory	4 Cups
-----------	--------

Inventory	2 Cups
-----------	--------

**Explaining the Example:** The user wrote some code and created a simple Inventory HMI. When there are 4 cups in the inventory, the HMI turns green. When there are only 2 cups left in the inventory the HMI turns yellow.

#### 9.3.4.11 Link

The *Link* element allows the user to create buttons that link different HMI screens. The difference between *Link* and *Screen List* is that the *Link* buttons can change color, change text, or be hidden completely.

The *Link* element has four arguments:

- Title Variable (String)
- Value Variable (Number, 0 or 1)
- Background Color Variable (String)
- Size

#### Example of *Link*:

<a href="#">0.0.0.0</a>	Grid Element	Help: LINK(Title_var,value_var,background_color_var,size) LINK(Fridge_Title_Variable,Fridge_Link_Variable,Fridge_Color_Variable,3)
<a href="#">0.2.1.0</a>	Grid Element	Help: LINK(Title_var,value_var,background_color_var,size) LINK(Store_Title_Variable,Store_Link_Variable,Store_Color_Variable,3)

0 ±	Program Variables	
	Name	Type
	# of Array Elements: 32 (Clear field to disable array)	
1	ACTIVE_HMI_SCREEN	String
	# of Array Elements: 32 (Clear field to disable array)	
2	Fridge_Title_Variable	String
	# of Array Elements: 32 (Clear field to disable array)	
3	Fridge_Color_Variable	String
4	Fridge_Link_Variable	Number
	# of Array Elements: 32 (Clear field to disable array)	
5	Store_Title_Variable	String
	# of Array Elements: 32 (Clear field to disable array)	
6	Store_Color_Variable	String
7	Store_Link_Variable	Number

### (HMI View)

#### “Fridge” HMI Screen:

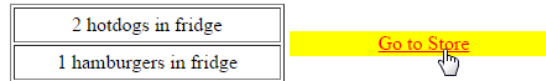
7248 hotdogs in fridge
3624 hamburgers in fridge

84



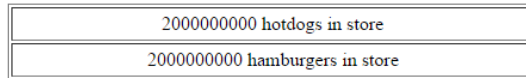
Eating food...

Get low enough, and the Store screen link appears.  
Click the link to go to that screen.



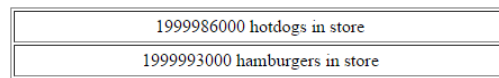
Buying food...

**"Store" HMI Screen:**



Buy More Food

Buy enough, and the Fridge screen link appears.  
Click the link to go to that screen.



Buy More Food

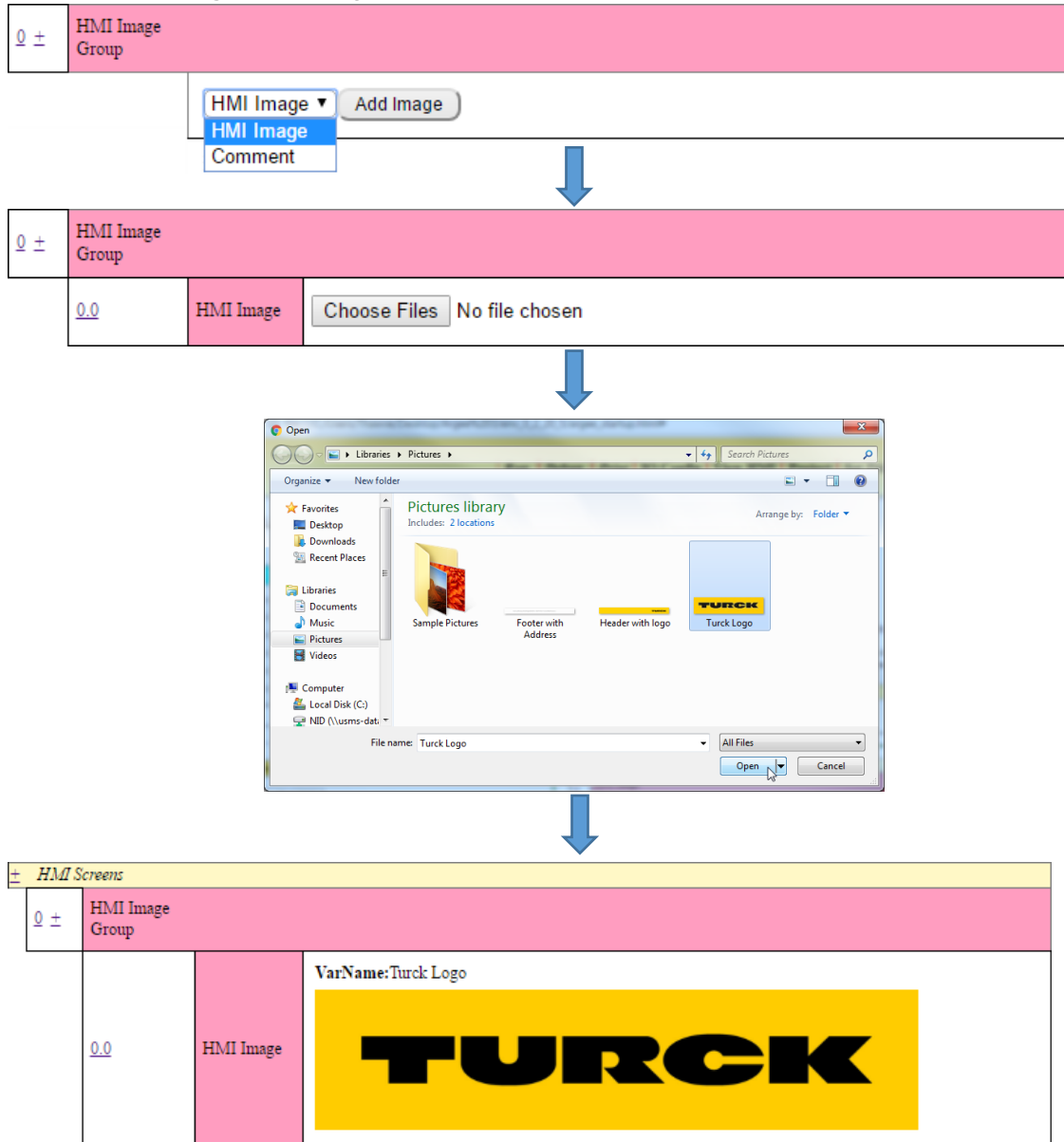


**Explaining the example:** The user wrote code that decreases the number of hotdogs and hamburgers in their fridge over time. If the count gets too low, the yellow link for the "Store" HMI screen is made visible so more can be bought. After enough food is bought at the store, the green link for the "Fridge" HMI screen appears so he can go back.

## 9.4 HMI Image Group

The user will use the *HMI Image Group* if the user want to upload an image to be used in their HMI design. This can be used to present company logos or certain types of dynamic graphics such as tank levels. Each individual image should be kept below 20kb file size to save space on the IO block.

**Example of creating an *HMI Image Group*:**



**Explaining the Example:** The user added an image of the Turck logo to their ARGEE project. The user will now create an *HMI Grid Screen* and place the logo in their HMI.



### NOTE

To access the list of the HMI variable names, the user can press Ctrl-I from anywhere inside the HMI screens section.

## 9.5 HMI Formatting Tips

### 9.5.1 Cell Spacing in a HMI

Spacing is important to make a HMI look good. A HMI cell will naturally take up all of the space allocated to it in a row. So adding empty cells with different cell span sizes can make cells a more reasonable size. The following code demonstrates this.

This Inventory table's width is the whole screen because it is the only cell in its row, this leaves a lot of empty space.

<a href="#">1.1</a> ±	Grid Row	Help: ROW_PROP(background_color) ROW_PROP("transparent")
<a href="#">1.1.0</a> ±	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,1)
<a href="#">1.1.0.0</a>	Grid Element	Help: DISPLAY_VALUE(Title,var,units_string,color,size,background_color) DISPLAY_VALUE("Total Inventory",Total," unit","black","1.5","transparent")
<a href="#">1.1.0.1</a>	Grid Element	Help: ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("Inventory Added",Inventory,"
<a href="#">1.1.0.2</a>	Grid Element	Help: BUTTON(Title,var,color,size,background_color) BUTTON("Submit",Submit,"black","1.5","transparent")

Total Inventory	0unit
Inventory Added	0 unit
Submit	

Now because the cell with the Inventory table in it has a column span of 1 and there are 3 columns in the row the width will be 1/3 of the screen.

<a href="#">1.0</a> ±	Grid Row	Help: ROW_PROP(background_color) ROW_PROP("transparent")
<a href="#">1.0.0</a> ±	Grid Cell (hidden)	CELL_PROP(1,1)
<a href="#">1.0.1</a> ±	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,1)
<a href="#">1.0.1.0</a>	Grid Element	Help: DISPLAY_VALUE(Title,var,units_string,color,size,background_color) DISPLAY_VALUE("Total Inventory",Total," unit","black","1.5","transparent")
<a href="#">1.0.1.1</a>	Grid Element	Help: ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("Inventory Added",Inventory,"
<a href="#">1.0.1.2</a>	Grid Element	Help: BUTTON(Title,var,color,size,background_color) BUTTON("Submit",Submit,"black","1.5","transparent")
	Grid Element ▾	Add Element
<a href="#">1.0.2</a> ±	Grid Cell (hidden)	CELL_PROP(1,1)

Total Inventory	0unit
Inventory Added	0 unit
Submit	

\*The spacer cells only have borders so they are easier to view for this documentation, the boarder would usually be set to 0 no border.

## 9.5.2 Row Spacing in a HMI

HMI Grid Elements within a cell will not naturally align themselves with another cells elements in the same row if they are different sizes. Each cell will vertically center its elements to the center of the largest cell in the row. So in some cases it is easier for the developer to align elements in separate rows.

The code bellow shows that the Enter Value element is larger than the static text labels they have next to them. This makes the labels not clearly match the element they are describing.

<a href="#">1.1 ±</a>	Grid Row	<b>Help:</b> ROW_PROP(background_color) ROW_PROP("transparent")	
<a href="#">1.1.0 ±</a>	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)	
<a href="#">1.1.0.0</a>	Grid Element	<b>Help:</b> STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Enter Value 1:","black","1.5","transparent","center")	
<a href="#">1.1.0.1</a>	Grid Element	<b>Help:</b> STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Enter Value 2:","black","1.5","transparent","center")	
<a href="#">1.1.0.2</a>	Grid Element	<b>Help:</b> STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Enter Value 3:","black","1.5","transparent","center")	
<a href="#">1.1.1 ±</a>	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)	
<a href="#">1.1.1.0</a>	Grid Element	<b>Help:</b> ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("",Test,"","black","1.5","transparent")	
<a href="#">1.1.1.1</a>	Grid Element	<b>Help:</b> ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("",Test,"","black","1.5","transparent")	
<a href="#">1.1.1.2</a>	Grid Element	<b>Help:</b> ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("",Test,"","black","1.5","transparent")	
<a href="#">1.1.2 ±</a>	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(8,0)	

Enter Value 1:

Enter Value 2:

Enter Value 3:

With the static text and enter value elements separated into individual rows they are now aligned.

<a href="#">1.2 ±</a>	Grid Row	Help: ROW_PROP(background_color) ROW_PROP("transparent")
<a href="#">1.2.0 ±</a>	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,0)
<a href="#">1.2.0.0</a>	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Enter Value 1:","black","1.5","transparent","center")
<a href="#">1.2.1 ±</a>	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,0)
<a href="#">1.2.1.0</a>	Grid Element	Help: ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("",Test,"","black","1.5","transparent")
<a href="#">1.2.2 ±</a>	Grid Cell (hidden)	CELL_PROP(8,0)
<a href="#">1.3 ±</a>	Grid Row	Help: ROW_PROP(background_color) ROW_PROP("transparent")
<a href="#">1.3.0 ±</a>	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,0)
<a href="#">1.3.0.0</a>	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Enter Value 2:","black","1.5","transparent","center")
<a href="#">1.3.1 ±</a>	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,0)
<a href="#">1.3.1.0</a>	Grid Element	Help: ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("",Test,"","black","1.5","transparent")
<a href="#">1.3.2 ±</a>	Grid Cell (hidden)	CELL_PROP(8,0)
<a href="#">1.4 ±</a>	Grid Row	Help: ROW_PROP(background_color) ROW_PROP("transparent")
<a href="#">1.4.0 ±</a>	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,0)
<a href="#">1.4.0.0</a>	Grid Element	Help: STATIC_TEXT(Text,color,size,background_color,alignment) STATIC_TEXT("Enter Value 3:","black","1.5","transparent","center")
<a href="#">1.4.1 ±</a>	Grid Cell	Help: CELL_PROP(column_span,border_style) CELL_PROP(1,0)
<a href="#">1.4.1.0</a>	Grid Element	Help: ENTER_VALUE(Title,var,units_string,color,size,background_color) ENTER_VALUE("",Test,"","black","1.5","transparent")
<a href="#">1.4.2 ±</a>	Grid Cell (hidden)	CELL_PROP(8,0)

Enter Value 1:  0

Enter Value 2:  0

Enter Value 3:  0

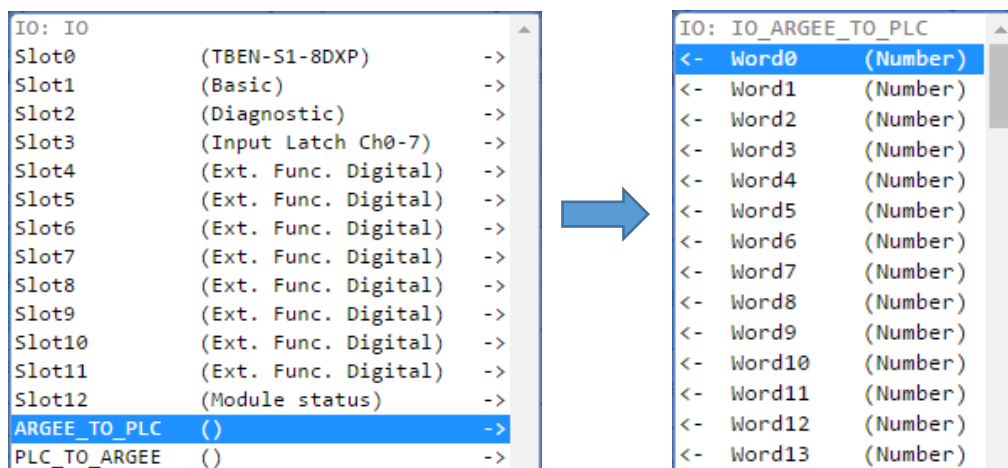
## 10 PLC Connectivity

### 10.1 Communicating with EtherNet/IP Master – RSLogix5000 / Studio5000

ARGEE blocks have the ability to communicate with an EtherNet/IP Master. The E/IP Master can establish communication via connection points 101 & 110 when running ARGEE with up to 240 Words of input and 240 words of output data.

**Example of Communicating with an EtherNet/IP Master:**

(ARGEE Setup)



0	Condition	true
0.0	Assignment	Destination: IO_ARGEE_TO_PLC_Word0 Expression: 1

(RSLogix 5000 Setup)

Module Properties Report: Local (ETHERNET-MODULE1.1)

General Connection Module Info

Type: ETHERNET-MODULE Generic Ethernet Module

Vendor: Allen-Bradley

Parent: Local

Name: TBEN\_S1\_8DXP

Description:

Comm Format: Data - INT

Address / Host Name

☒ IP Address: 192 . 168 . 1 . 87

☐ Host Name:

Connection Parameters

Input	Assembly Instance	Size
101	10	(16-bit)
110	6	(16-bit)
Configuration: 106	1	(8-bit)

Status Input:

Status Output:

Status: Offline

OK Cancel Apply Help

**Explaining the Example so far:** The user wants to pass data from an ARGEE block to the E/IP master. The user's code will write the value "1" into word 0 bit 0 of the *ARGEE\_TO\_PLC* register. The user then created a generic Ethernet device in RSLogix 5000 and set the connection points to be 101 & 110.

The screenshot displays the TURCK software interface. On the left, a variable declaration table lists `TBEN_S1_8DXP.I.Data` and `TBEN_S1_8DXP.O.Data` arrays. The main window shows the **ARGEE Program** with a task `MainTask` containing a condition `True` and an assignment `Destination: IO_ARGEE_TO_PLC_Word0 Expression: 1`. The **Runtime Status** pane shows `PLC TO ARGEE` and `ARGEE TO PLC` with a value of `0x0001`. A red box highlights the `ARGEE TO PLC` value.

This screenshot is similar to the one above, showing the same TURCK software interface. It highlights the `ARGEE TO PLC` value in the runtime status, which is `0x0001`. A red box highlights the `ARGEE TO PLC` value.

**Explaining the example:** The above image is showing that the data has been successfully passed back and forth between ARGEE and the RSLogix 5000.



#### NOTE

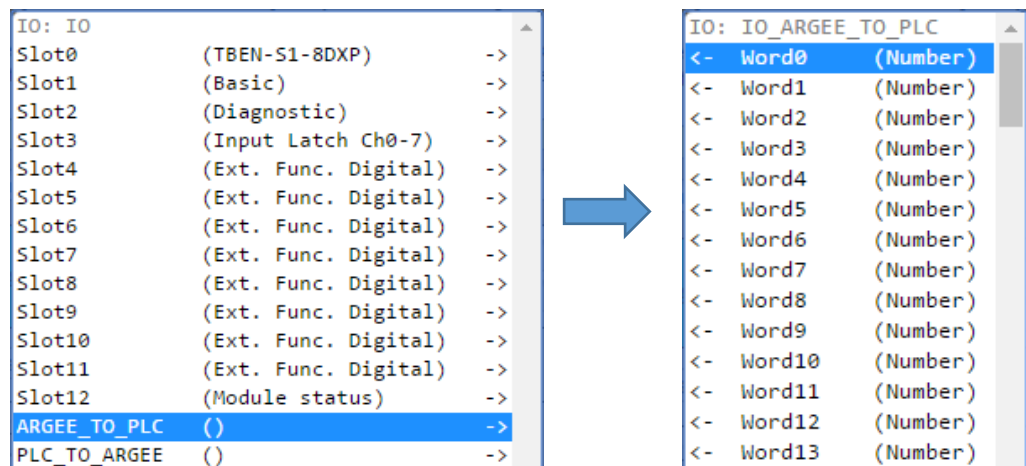
If the user wants to accomplish "bit offsetting," they need to manually adjust the *IO Variable Format* (discussed in [11.4.2 IO Variable Formats](#)). For example, the user wants to force word 0 bit 5 true, the destination variable would be `IO_ARGEE_TO_PLC_Word0.5`

# 10.2 Communicating with a PROFINET Master – SIMATIC STEP 7

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

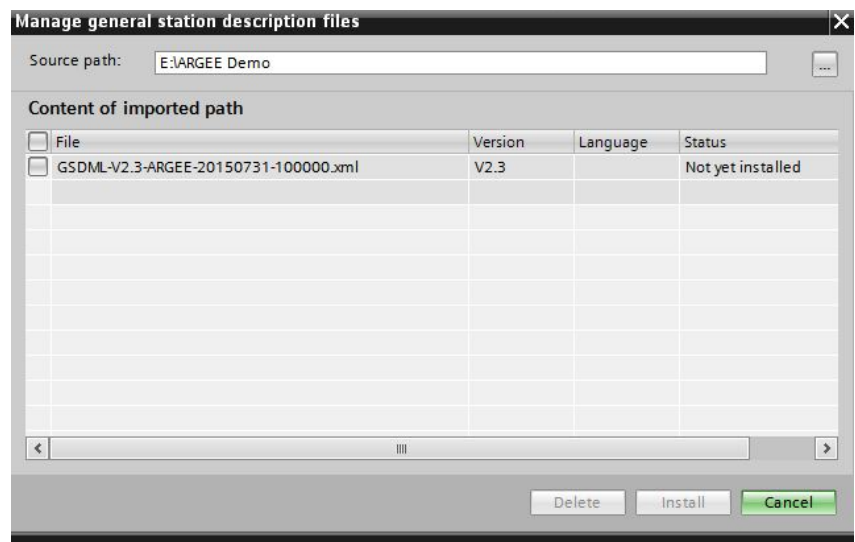
Example of Communicating with a PROFINET Master:

(ARGEE Setup)

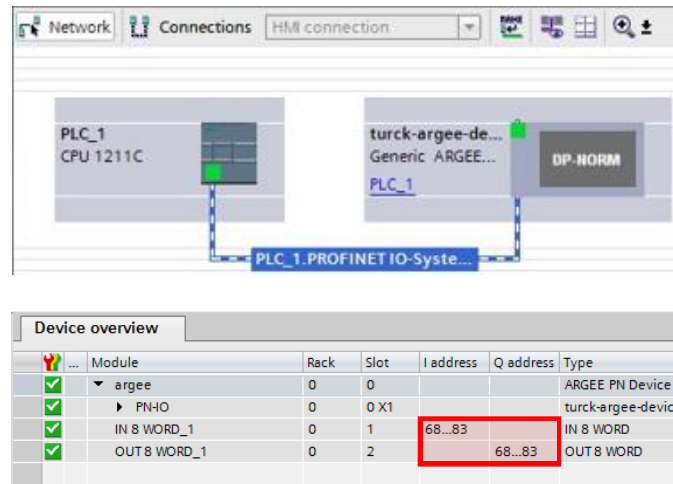


0 ±	Condition	True
0.0	Assignment	Destination: IO_ARGEE_TO_PLC_Word0 Expression: 2

(SIMATIC STEP 7 Setup)







**Explaining the Setup:** The user wants to pass data from an ARGEE block to the PROFINET master. The user's code will write the value "1" into word 0 bit 0 of the ARGEE\_TO\_PLG register. The user then defines the "I address" and "Q address" from the Step 7 Device overview screen.

**ARGEE Program**

Task - MainTask

Condition: True

Assignment: Destination: IO\_ARGEE\_TO\_PLG\_Word0 Expression: 1

Project1 > PLC\_1 [CPU 1211C DC/DC/DC] > Watch and force tables > Watch table\_1

Name	Address	Display format	Monitor value	Modify value	Comment
%QW68	16#0003	Hex	16#0003		
%IW68	16#0002	Hex			

**ARGEE Program**

Task - MainTask

Condition: True

Assignment: Destination: IO\_ARGEE\_TO\_PLG\_Word0 Expression: 1

**Explaining the example:** The above image is showing that the data has been successfully passed back and forth between ARGEE and the SIMATIC STEP 7 engineering software.



#### NOTE

If the user wants to accomplish "bit offsetting," they need to manually adjust the *IO Variable Format* (discussed in [11.4.2 IO Variable Formats](#)). For example, the user wants to force word 0 bit 5 true, the destination variable would be IO\_ARGEE\_TO\_PLG\_Word0.5

### 10.3 Communicating with a Modbus TCP/IP Master – Crimson 3

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*:

##### (ARGEE Setup)

The diagram illustrates the setup for communicating with a Modbus TCP/IP Master using ARGEE blocks. It shows two panels: the left panel lists IO slots (Slot0 to Slot12) and their functions (e.g., TBEN-S1-8DXP, Basic, Diagnostic, Input Latch Ch0-7, Ext. Func. Digital, Module status). The right panel shows the corresponding PLC configuration, where each slot is mapped to a specific Word (Word0 to Word13) in the IO\_ARGEE\_TO\_PLC block. A blue arrow indicates the flow from the IO slots to the PLC configuration.

IO: IO	Function	Direction
Slot0	(TBEN-S1-8DXP)	->
Slot1	(Basic)	->
Slot2	(Diagnostic)	->
Slot3	(Input Latch Ch0-7)	->
Slot4	(Ext. Func. Digital)	->
Slot5	(Ext. Func. Digital)	->
Slot6	(Ext. Func. Digital)	->
Slot7	(Ext. Func. Digital)	->
Slot8	(Ext. Func. Digital)	->
Slot9	(Ext. Func. Digital)	->
Slot10	(Ext. Func. Digital)	->
Slot11	(Ext. Func. Digital)	->
Slot12	(Module status)	->
ARGEE_TO_PLC	( )	->
PLC_TO_ARGEE	( )	->

IO: IO_ARGEE_TO_PLC	Function	Direction
Word0	(Number)	<-
Word1	(Number)	<-
Word2	(Number)	<-
Word3	(Number)	<-
Word4	(Number)	<-
Word5	(Number)	<-
Word6	(Number)	<-
Word7	(Number)	<-
Word8	(Number)	<-
Word9	(Number)	<-
Word10	(Number)	<-
Word11	(Number)	<-
Word12	(Number)	<-
Word13	(Number)	<-

Condition	Assignment	Destination	Expression
0	Condition	true	
0.0	Assignment	IO_ARGEE_TO_PLC_Word0	1

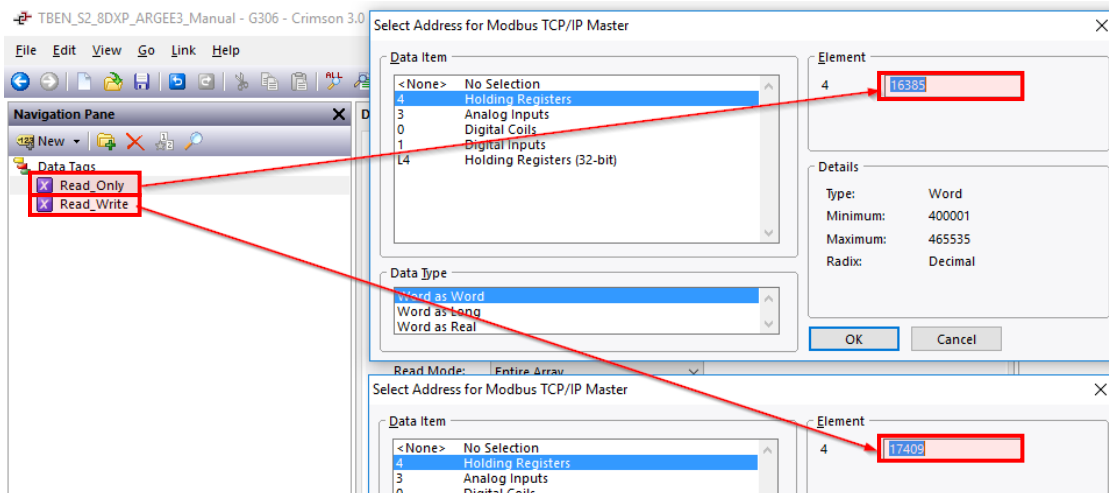
##### (Crimson 3 Setup)

The screenshot shows the configuration window for the Communications - Network - Protocol 1 - TBEN\_S1\_8DXP. The window is divided into several sections: Device Settings, Device Identification, and Protocol Options. The Device Settings section includes an 'Enable Device' checkbox set to 'Yes'. The Device Identification section includes fields for 'Primary IP Address' (192.168.1.87), 'Fallback IP Address' (0.0.0.0), 'TCP Port' (502), and 'Unit Number' (1). The Protocol Options section includes a 'Ping Holding Register' field set to 0, 'Ignore Read Exceptions' set to 'No', 'Link Type' set to 'Use Dedicated Socket', and 'ICMP Ping' set to 'Enable'.



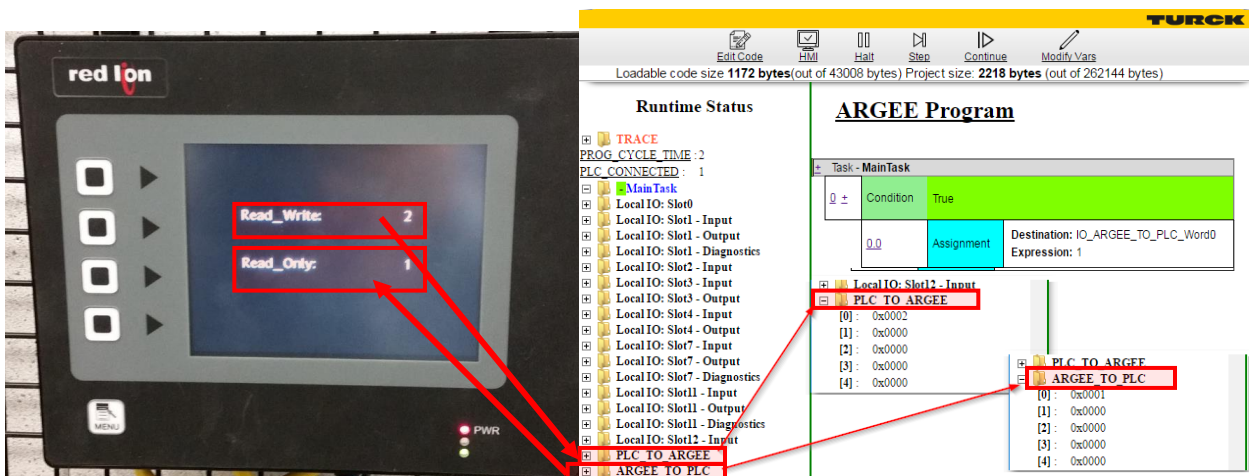
#### NOTE

If using a Red Lion HMI, set the Ping Holding Register to zero.



#### NOTE

Red Lion Modbus master register addressing = Original address + 1. If the original address 0x400(hex) = 16384 the Red Lion address would be (16384 + 1) 16385.



**Explaining the example:** The above image is showing that the data has been successfully passed back and forth between ARGEE and Crimson 3.



#### NOTE

If the user wants to accomplish "bit offsetting," they need to manually adjust the IO Variable Format (discussed in [11.4.2 IO Variable Formats](#)). For example, if the user wants to force word 0 bit 5 true, the destination variable would be IO\_ARGEE\_TO\_PLC\_Word0.5.

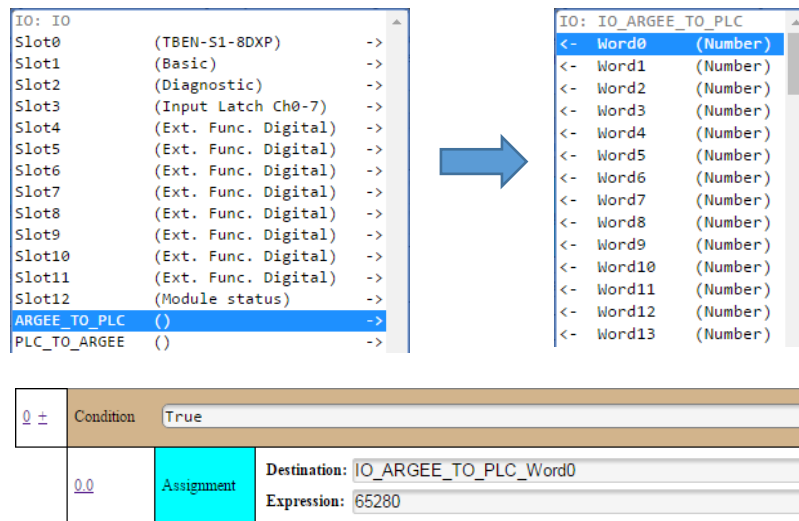
## 10.4 Communicating with a Turck PLC or TX500 Series HMI – CODESYS 3

### 10.4.1 EtherNet IP

ARGEE blocks have the ability to communicate with an EtherNet IP Scanner communication via tags. The input assembly instance is 101 (0x65), the output assembly instance is 110 (0x6E), and the configuration assembly instance is 01 (0x01). The size of the input and output assemblies (in bytes) is defined by the number of input and output words in your ARGEE program. The configuration size is always 0.

**Example of Communicating with a Turck PLC or TX500 Series HMI:**

#### (ARGEE Setup)



#### (CODESYS 3 Setup)

**Connection Path Settings**

- ☒ generate path automatically
  - ☒ Configuration Assembly
    - Class ID 16# 4
    - Instance ID 16# 1
    - Attribute ID 16# 3
  - ☒ Consuming Assembly (O-->T)
    - Class ID 16# 4
    - Instance ID 16# 6E
    - Attribute ID 16# 3
  - ☒ Producing Assembly (T-->O)
    - Class ID 16# 4
    - Instance ID 16# 65
    - Attribute ID 16# 3
- ☐ user-defined path

**Generic Parameters**

Connection Path: 20 04 24 01 2C 6E 2C 65

Trigger Type: **Cyclic** RPI (ms): 10

Transport Type: **Exclusive Owner** Timeout Multiplier: 4

**Scanner to Target (Output)**

O-->T Size (Bytes): 4

Config#1 Size (Bytes): 0

Config#2 Size (Bytes): 0

Connection Type: **Point to Point**

Fixed/Variable: **Fixed**

Transfer Format: **32 Bit Run/Idle**

Inhibit Time (ms): 0

**Target to Scanner (Input)**

T-->O Size (Bytes): 2

Connection Type: **Point to Point**

Fixed/Variable: **Fixed**

Transfer Format: **pure Data**

Inhibit Time (ms): 0

The screenshot displays the configuration interface for a Generic Ethernet/IP device. The left sidebar shows the configuration tree with 'EtherNet/IP I/O Mapping' selected. The main window is divided into two panes.

**EtherNet/IP I/O Mapping Pane:**

Variable	Channel	Address	Type	Current Value
Generic AssemblyParam0		%IB7	BYTE	0
Bit0		%IX7.0	BOOL	FALSE
Bit1		%IX7.1	BOOL	FALSE
Bit2		%IX7.2	BOOL	FALSE
Bit3		%IX7.3	BOOL	FALSE
Bit4		%IX7.4	BOOL	FALSE
Bit5		%IX7.5	BOOL	FALSE
Bit6		%IX7.6	BOOL	FALSE
Bit7		%IX7.7	BOOL	FALSE
Generic AssemblyParam1		%IB8	BYTE	255
Bit0		%IX8.0	BOOL	TRUE
Bit1		%IX8.1	BOOL	TRUE
Bit2		%IX8.2	BOOL	TRUE
Bit3		%IX8.3	BOOL	TRUE
Bit4		%IX8.4	BOOL	TRUE
Bit5		%IX8.5	BOOL	TRUE
Bit6		%IX8.6	BOOL	TRUE
Bit7		%IX8.7	BOOL	TRUE
Generic AssemblyParam2		%QB4	BYTE	1
Bit0		%QX4.0	BOOL	TRUE
Bit1		%QX4.1	BOOL	FALSE
Bit2		%QX4.2	BOOL	FALSE

**Task - MainTask Pane:**

Condition	Assignment	Destination	Expression
0 ±	True	IO_ARGEE_TO_PLC_Word0	65280

**Local IO: Module\_status - Input**

- PLC\_TO\_ARGEE
- [0] : 0x0000
- [1] : 0x0001
- [2] : 0x0000
- [3] : 0x0000
- [4] : 0x0000

**Explaining the example:** The code loads 65280 into Word 0. This turns the high byte true (255) and the low byte false. The PLC loads a 1 into word 1 output and sets PLC\_Input to 1.



#### NOTE

If the user wants to accomplish “bit offsetting,” they need to manually adjust the IO Variable Format (discussed in [11.4.2 IO Variable Formats](#)). For example, if the user wants to force word 0 bit 5 true, the destination variable would be IO\_ARGEE\_TO\_PLC\_Word0.5.

10.4.2 PROFINET

ARGEE blocks have the ability to communicate with a PROFINET Controller via tags.

Example of Communicating with a *Turck PLC or TX500 Series HMI*:

(ARGEE Setup)

IO: IO

Slot0 (TBEN-S1-8DXP) ->

Slot1 (Basic) ->

Slot2 (Diagnostic) ->

Slot3 (Input Latch Ch0-7) ->

Slot4 (Ext. Func. Digital) ->

Slot5 (Ext. Func. Digital) ->

Slot6 (Ext. Func. Digital) ->

Slot7 (Ext. Func. Digital) ->

Slot8 (Ext. Func. Digital) ->

Slot9 (Ext. Func. Digital) ->

Slot10 (Ext. Func. Digital) ->

Slot11 (Ext. Func. Digital) ->

Slot12 (Module status) ->

ARGEE\_TO\_PLC () ->

PLC\_TO\_ARGEE () ->

IO: IO\_ARGEE\_TO\_PLC

<- Word0 (Number)

<- Word1 (Number)

<- Word2 (Number)

<- Word3 (Number)

<- Word4 (Number)

<- Word5 (Number)

<- Word6 (Number)

<- Word7 (Number)

<- Word8 (Number)

<- Word9 (Number)

<- Word10 (Number)

<- Word11 (Number)

<- Word12 (Number)

<- Word13 (Number)

0 ±

Condition True

0.0

Assignment

Destination: IO\_ARGEE\_TO\_PLC\_Word0

Expression: 1

**NOTE** Use the ARGEE GSDML File to add the device to the project. It can be found in the ARGEE Environment folder at [www.Turck.com](http://www.Turck.com)

(CODESYS 3 Setup)

Ethernet (Ethernet)

PN\_Controller (PN-Controller)

ARGEE\_PN\_Device (ARGEE PN Device)

IN\_1\_WORD (IN 1 WORD)

OUT\_1\_WORD (OUT 1 WORD)

Channels					
Variable	Mapping	Channel	Address	Type	Current Value
		ARGEE input	%IW4	UINT	1
Variable	Mapping	Channel	Address	Type	Current Value
		ARGEE output	%QW1	UINT	1

LocalIO: Module_status - Input		Task - MainTask	
PLC TO ARGEE			
[0] : 0x0001		0 +	Condition True
[1] : 0x0000			
[2] : 0x0000			
[3] : 0x0000			
[4] : 0x0000		0.0	Assignment Destination: IO_ARGEE_TO_PLC_Word0 Expression: 1

**Explaining the example:** The code loads a 1 into *IO\_ARGEE\_TO\_PLC\_WORD0*. The PLC loads a 1 into word 1 output.



#### NOTE

If the user wants to accomplish “bit offsetting,” they need to manually adjust the IO Variable Format (discussed in [11.4.2 IO Variable Formats](#)). For example, if the user wants to force word 0 bit 5 true, the destination variable would be *IO\_ARGEE\_TO\_PLC\_Word0.5*.

### 10.4.3 Modbus TCP/IP

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.

**Example of Communicating with a *Turck PLC or TX500 Series HMI*:**

**(ARGEE Setup)**

The diagram illustrates the setup for communicating with a Turck PLC or TX500 Series HMI using ARGEE blocks. It shows two panels: the left panel lists IO slots (Slot0 to Slot12) and their functions, with 'ARGEE\_TO\_PLC' selected; the right panel shows the 'IO: IO\_ARGEE\_TO\_PLC' block with 14 words (Word0 to Word13) configured as 'Number'. A blue arrow points from the left panel to the right panel. Below these panels is a table with two rows: the first row has 'Condition' set to 'true', and the second row has 'Assignment' set to 'Destination: IO\_ARGEE\_TO\_PLC\_Word0' and 'Expression: 1'.

IO: IO	Function	Direction
Slot0	(TBEN-S1-8DXP)	->
Slot1	(Basic)	->
Slot2	(Diagnostic)	->
Slot3	(Input Latch Ch0-7)	->
Slot4	(Ext. Func. Digital)	->
Slot5	(Ext. Func. Digital)	->
Slot6	(Ext. Func. Digital)	->
Slot7	(Ext. Func. Digital)	->
Slot8	(Ext. Func. Digital)	->
Slot9	(Ext. Func. Digital)	->
Slot10	(Ext. Func. Digital)	->
Slot11	(Ext. Func. Digital)	->
Slot12	(Module status)	->
ARGEE_TO_PLC	( )	->
PLC_TO_ARGEE	( )	->

IO: IO_ARGEE_TO_PLC	Word	Type
<-	Word0	(Number)
<-	Word1	(Number)
<-	Word2	(Number)
<-	Word3	(Number)
<-	Word4	(Number)
<-	Word5	(Number)
<-	Word6	(Number)
<-	Word7	(Number)
<-	Word8	(Number)
<-	Word9	(Number)
<-	Word10	(Number)
<-	Word11	(Number)
<-	Word12	(Number)
<-	Word13	(Number)

Condition	Assignment
0 Condition true	Destination: IO_ARGEE_TO_PLC_Word0 Expression: 1

**(CODESYS 3 Setup)**

The screenshot shows the 'ModbusChannel' configuration dialog box. The 'Channel' section has 'Name' set to 'Channel 0'. The 'Access Type' is set to 'Read/Write Multiple Registers (Function Code 23)'. The 'Trigger' is set to 'Cyclic' with a 'Cycle Time (ms)' of 100. The 'READ Register' section has 'Offset' set to '0x4000' and 'Length' set to '2'. The 'WRITE Register' section has 'Offset' set to '0x4400' and 'Length' set to '2'. The 'Error Handling' is set to 'Keep last Value'. The 'OK' and 'Cancel' buttons are at the bottom.

Channel
Name: Channel 0
Access Type: Read/Write Multiple Registers (Function Code 23)
Trigger: Cyclic, Cycle Time (ms): 100
Comment:
READ Register: Offset: 0x4000, Length: 2
WRITE Register: Offset: 0x4400, Length: 2
Error Handling: Keep last Value
Buttons: OK, Cancel



**Modbus\_TCP\_Slave** x **Modbus\_TCP\_Master** **Ethernet** **Device**

**General**

Modbus Slave Channel

Modbus Slave Init

ModbusTCPSlave Parameters

ModbusTCPSlave I/O Mapping

Status

**Channels**

Variable	Channel	Address	Type	Current Value
Channel 0	Channel 0	%IW4	ARRA...	
Channel 0[0]	Channel 0	%IW4	WORD	1
Bit0	Channel 0	%IX8.0	BOOL	TRUE
Bit1	Channel 0	%IX8.1	BOOL	FALSE
Channel 0	Channel 0	%QW1	ARRA...	
Channel 0[0]	Channel 0	%QW1	WORD	0
Channel 0[1]	Channel 0	%QW2	WORD	1
Bit0	Channel 0	%QX4.0	BOOL	TRUE

**Local I/O: Module\_status - Input**

**PLC TO ARGEE**

[0] : 0x0001

[1] : 0x0000

[2] : 0x0000

[3] : 0x0000

**Task - MainTask**

Condition	Assignment
True	Destination: IO_ARGEE_TO_PLC_Word0 Expression: 1

**Explaining the example:** The above image is showing that the data has been successfully passed back and forth between ARGEE and CODESYS 3.



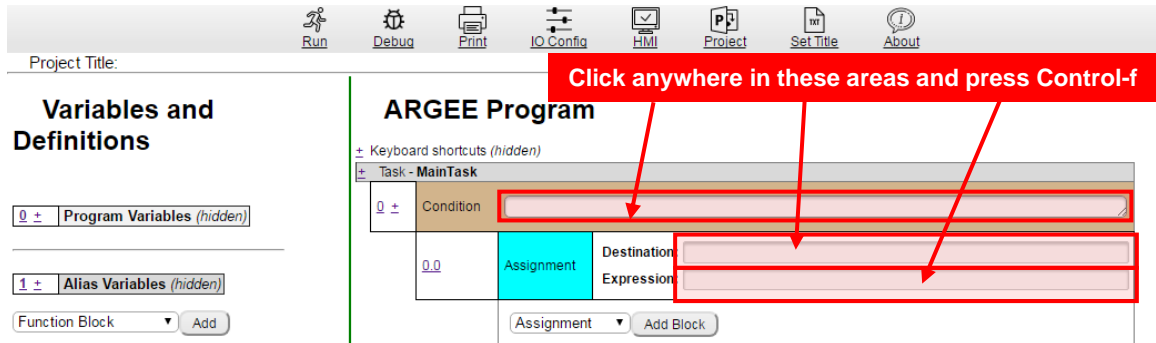
#### NOTE

If the user wants to accomplish "bit offsetting," they need to manually adjust the IO Variable Format (discussed in [11.4.2 IO Variable Formats](#)). For example, if the user wants to force word 0 bit 5 true, the destination variable would be IO\_ARGEE\_TO\_PLC\_Word0.5.

## 11 Appendix I - Definitions

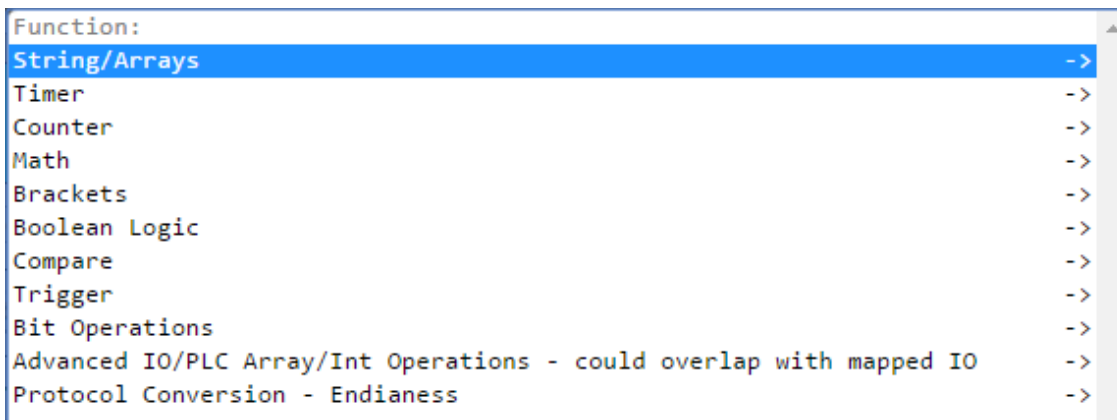
### 11.1 Built-in Functions (Ctrl-f)

To access *Built-in Functions*, the user can simply click anywhere in the code, and press Ctrl-f.



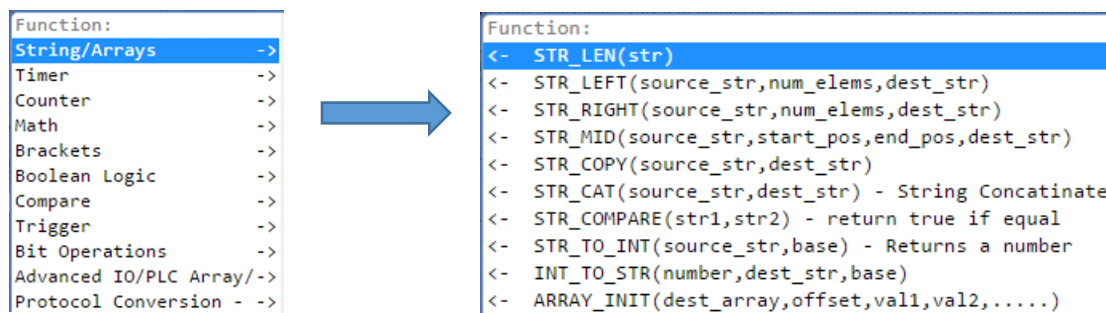
## 11.2 Built-in Functions Menu

The user can use their mouse or the arrow keys on their keyboard to navigate the built-in functions menu.



### 11.2.1 Strings/Arrays

To access the String/Arrays functions, highlight *String/Arrays*, and press “→” on the keyboard or click “->” with the mouse to advance to the next or previous level.



### 11.2.1.1 String Length

The user will use `STR_LEN` when the user wants to know the length of a string. The string length is returned as an integer.

#### Example of String Length:

The screenshot displays the TURCK IDE interface for a project titled "TBEN-S1-8DXP (192.168.1.12) V3.2.3.5".

**Variables and Definitions:**

	Name	Type
# of Array Elements: 32 (Clear field to disable array)		
1	Register_1	String
2	Register_2	Number

**ARGEE Program:**

Task - MainTask

0	Call	Help: STR_COPY(source_str,dest_str) STR_COPY("Intern is playing with Strings", Register_1)
1	Assignment	Destination: Register_2 Expression: STR_LEN(Register_1)

Below the program editor, a blue arrow points to the runtime status.

**Runtime Status:**

TRACE  
PROG CYCLE TIME : 2  
PLC CONNECTED : 0  
REGISTER\_1 : Intern is playing with Strings  
REGISTER\_2 : 30

**ARGEE Program (Runtime):**

Task - MainTask

0	Call	STR_COPY("Intern is playing with Strings", Register_1)
1	Assignment	Destination: Register_2 Expression: STR_LEN(Register_1)

**Explaining the Example:** `STR_COPY` copies the string "Intern is playing with Strings" into Register\_1. The string "Intern is playing with Strings" is 30 elements long. The length of that string is stored in Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations "".



#### NOTE

`STR_COPY` is discussed later in this chapter in section [11.2.1.5 String Copy](#).

### 11.2.1.2 String Left

The user will use `STR_LEFT` when the user wants to count from the left a certain amount source string elements and store them in a different destination string. All destination string elements will be overwritten.

#### Example of String Left:

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

#### Variables and Definitions

	Name	Type
	# of Array Elements: 32 (Clear field to disable array)	
1	Register_1	String
	# of Array Elements: 32 (Clear field to disable array)	
2	Register_2	String

Add Variable

#### ARGEE Program

+ Keyboard shortcuts (hidden)

Task - MainTask	
0	Call STR_COPY(source_str,dest_str) STR_COPY("Noah is playing with Strings", Register_1)
1	Call STR_LEFT(source_str,num_elems,dest_str) STR_LEFT(Register_1,7,Register_2)

Call Add Block

↓

#### Runtime Status

TRACE  
 PROG CYCLE TIME: 2  
 PLC CONNECTED: 0  
 REGISTER\_1: Noah is playing with Strings  
 REGISTER\_2: Noah is  
 MainTask  
 Local IO: TBEN\_S1\_8DXP\_GW  
 Local IO: Basic - Input  
 Local IO: Basic - Output

#### ARGEE Program

Task - MainTask	
0	Call STR_COPY("Noah is playing with Strings", Register_1)
1	Call STR_LEFT(Register_1,7,Register_2)

**Explaining the Example:** `STR_COPY` copies the string "Noah is playing with Strings" into Register\_1. `STR_LEFT` takes the first 7 elements in Register\_1 and places them in Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations " ".



#### NOTE

`STR_COPY` is discussed later in this chapter in section [11.2.1.5 String Copy](#).

### 11.2.1.3 String Right

The user will use *STR\_RIGHT* when the user wants to count from the right a certain amount source string elements and store them in a different destination string. All destination string elements will be overwritten.

#### Example of String Right:

Project Title: **TBEN-S1-8DXP (192.168.1.12) V3.2.3.5**

#### Variables and Definitions

Name	Type
# of Array Elements: 32 (Clear field to disable array)	
1 Register_1	String
# of Array Elements: 32 (Clear field to disable array)	
2 Register_2	String

#### ARGEE Program

Task - MainTask

Step	Call	Help
0	Call	STR_COPY(source_str,dest_str) STR_COPY("Noah is playing with Strings", Register_1)
1	Call	STR_RIGHT(source_str,num_elems,dest_str) STR_RIGHT(Register_1,7,Register_2)

Loadable code size **1272 bytes**(out of 43008 bytes) Project size: **2254 bytes** (out of 262144 bytes)

#### Runtime Status

TRACE  
PROG CYCLE TIME : 2  
PLC CONNECTED : 0  
REGISTER\_1 : Noah is playing with Strings  
REGISTER\_2 : Strings  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW  
Local IO: Basic - Input  
Local IO: Basic - Output

#### ARGEE Program

Task - MainTask

Step	Call	Help
0	Call	STR_COPY("Noah is playing with Strings", Register_1)
1	Call	STR_RIGHT(Register_1,7,Register_2)

**Explaining the Example:** STR\_COPY copies the string "Noah is playing with Strings" into Register\_1. STR\_RIGHT takes the last 7 elements in Register\_1 and places them in Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations " ".



#### NOTE

STR\_COPY is discussed later in this chapter in section [11.2.2.5 String Copy](#).

### 11.2.1.4 String Middle

The user will use `STR_MID` when the user wants to pick out a certain amount of middle source string elements and store them in a different destination string. All destination string elements will be overwritten.

#### Example of String Middle:

Project Title: **TBEN-S1-8DXP (192.168.1.12) V3.2.3.5**

#### Variables and Definitions

#	Name	Type
1	Register_1	String
# of Array Elements: 32 (Clear field to disable array)		
2	Register_2	String
# of Array Elements: 32 (Clear field to disable array)		

#### ARGEE Program

Task - MainTask

#	Call	Help
0	Call	STR_COPY(source_str,dest_str) STR_COPY("Noah is playing with Strings", Register_1)
1	Call	STR_MID(source_str,start_pos,end_pos,dest_str) STR_MID(Register_1,8,21,Register_2)

Loadable code size: **1276 bytes** (out of 43008 bytes) Project size: **2259 bytes** (out of 262144 bytes)

#### Runtime Status

TRACE  
PROG\_CYCLE\_TIME: 2  
PLC\_CONNECTED: 0

REGISTER\_1: Noah is playing with Strings  
REGISTER\_2: playing with

#### ARGEE Program

Task - MainTask

#	Call	Help
0	Call	STR_COPY("Noah is playing with Strings", Register_1)
1	Call	STR_MID(Register_1,8,21,Register_2)

**Explaining the Example:** `STR_COPY` copies the string "Noah is playing with Strings" into Register\_1. `STR_MID` takes elements 8 through 21 in Register\_1 and places them in Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations " ".



#### NOTE

`STR_COPY` is discussed later in this chapter in section [11.2.1.5 String Copy](#).

### 11.2.1.5 String Copy

The user will use `STR_COPY` when the user wants to copy elements into a string. All destination string elements will be overwritten.

#### Example of String Copy:

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

**Variables and Definitions**

Name	Type
# of Array Elements: 32 (Clear field to disable array)	
1 Register_1	String

**ARGEE Program**

Task - MainTask

Call STR\_COPY(source\_str,dest\_str)  
STR\_COPY("Noah is playing with Strings", Register\_1)

Runtime Status

TRACE  
PROG\_CYCLE\_TIME: 2  
PLC\_CONNECTED: 0  
REGISTER\_1: Noah is playing with Strings  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW

**ARGEE Program**

Task - MainTask

Call STR\_COPY("Noah is playing with Strings", Register\_1)

**Explaining the Example:** `STR_COPY` copies the string "Noah is playing with Strings" into Register\_1.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations " ".

### 11.2.1.6 String Concatenate

The user will use `STR_CAT` when the user wants to combine two strings to make a single string.

#### Example of String Concatenate:

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

**Variables and Definitions**

Name	Type
# of Array Elements: 64 (Clear field to disable array)	
1 Register_1	String
# of Array Elements: 32 (Clear field to disable array)	
2 Register_2	String

**ARGEE Program**

Task - MainTask

Call STR\_COPY(source\_str,dest\_str)  
STR\_COPY("Noah is playing with Strings", Register\_1)

Call STR\_COPY(source\_str,dest\_str)  
STR\_COPY(" and Arrays", Register\_2)

Call STR\_CAT(source\_str,dest\_str) - String Concatenate  
STR\_CAT(Register\_2, Register\_1)



Loadable code size: 1338 bytes (out of 43008 bytes) Project size: 2342 bytes (out of 262144 bytes)

### Runtime Status

TRACE

PROG\_CYCLE\_TIME: 2

PLC\_CONNECTED: 0

REGISTER\_1: Noah is playing with Strings and Arrays

REGISTER\_2: and Arrays

MainTask

Local IO: TBEN\_S1\_8DXP\_GW

Local IO: Basic - Input

Local IO: Basic - Output

Local IO: Diagnostic - Input

Local IO: Input\_Latch\_Ch0\_7 - Input

### ARGEE Program

Task - MainTask		
0	Call	STR_COPY("Noah is playing with Strings", Register_1)
1	Call	STR_COPY(" and Arrays", Register_2)
2	Call	STR_CAT(Register_2, Register_1)

**Explaining the Example:** STR\_COPY copies the string "Noah is playing with Strings" into Register\_1. STR\_COPY copies the string " and Arrays" into Register\_2. STR\_CAT concatenates both strings together to make the new string "Noah is playing with Strings and Arrays."



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations " ".

### 11.2.1.7 String Compare

The user will use *STR\_COMPARE* when the user wants to check and see if two strings are equal.

#### Example of String Compare:

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

### Variables and Definitions

	Name	Type
0	Register_1	String
1	Register_2	String

Add Variable

### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask		
0	If	STR_COMPARE(Register_1, Register_2)
0.0	Assignment	Destination: IO_Basic_Output_Output_value_0 Expression: 1

Assignment Add Block

↓

### Runtime Status

TRACE

PROG\_CYCLE\_TIME: 2

PLC\_CONNECTED: 0

REGISTER\_1:

REGISTER\_2:

MainTask

Local IO: TBEN\_S1\_8DXP\_GW

Local IO: Basic - Input

Local IO: Basic - Output

Output value 0: 1

Output value 1: 0

### ARGEE Program

Task - MainTask		
0	If	STR_COMPARE(Register_1, Register_2)
0.0	Assignment	Destination: IO_Basic_Output_Output_value_0 Expression: 1

**Explaining the Example:** STR\_COMPARE is constantly comparing the string elements in Register\_1 to the string elements in Register\_2. When the two strings are equal, Output 0 turns on.



## NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

### 11.2.1.8 String to Integer

The user will use `STR_TO_INT` when the user wants to move a string into an integer register. The user can also convert a binary, octal, decimal, or hexadecimal base number into decimal as it moves into the new register.

#### 11.2.1.8.1 String to Integer - Base 2 – Binary

The diagram illustrates the configuration and execution of the `STR_TO_INT` function for binary conversion. It is divided into two parts: configuration and execution.

**Configuration:** The **Program Variables** table shows `Register_1` as a **String** and `Register_2` as a **Number**. The **Task - MainTask** block is configured with **Destination: Register\_2** and **Expression: STR\_TO\_INT(Register\_1,2)**. The **Assignment** block is highlighted in blue.

**Execution:** The **TRACE** window shows the state of the registers. `REGISTER_1` contains the binary string "1110" and `REGISTER_2` contains the decimal value 14. The **MainTask** block is highlighted in blue.

**Explaining the Example:** Register\_1 is initialized to value “1110.” `STR_TO_INT` takes the binary string in Register\_1, converts it into a decimal integer and puts it into Register\_2.



## NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

#### 11.2.1.8.2 String to Integer - Base 8 – Octal

The diagram illustrates the configuration and execution of the `STR_TO_INT` function for octal conversion. It is divided into two parts: configuration and execution.

**Configuration:** The **Program Variables** table shows `Register_1` as a **String** and `Register_2` as a **Number**. The **Task - MainTask** block is configured with **Destination: Register\_2** and **Expression: STR\_TO\_INT(Register\_1,8)**. The **Assignment** block is highlighted in blue.

**Execution:** The **TRACE** window shows the state of the registers. `REGISTER_1` contains the octal string "16" and `REGISTER_2` contains the decimal value 14. The **MainTask** block is highlighted in blue.

**Explaining the Example:** Register\_1 is initialized to value “16.” `STR_TO_INT` takes the octal string in Register\_1, converts it into a decimal integer and puts it into Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

#### 11.2.1.8.3 String to Integer – Base 10 – Decimal

Program Variables		
	Name	Type
1.0	INIT: "14"	
	# of Array Elements: 32	(Clear field to disable array)
1	Register_1	String
2	Register_2	Number

Add Variable

Task - MainTask

Assignment

Destination: Register\_2  
Expression: STR\_TO\_INT(Register\_1,10)

Assignment ▼ Add Block

↓

```

PROG_CYCLE_TIME:2
PLC_CONNECTED: 0
REGISTER_1: 14
REGISTER_2: 14
MainTask
    
```

Task - MainTask

Assignment

Destination: Register\_2  
Expression: STR\_TO\_INT(Register\_1,10)

**Explaining the Example:** Register\_1 is initialized to value “14.” STR\_TO\_INT takes the decimal string in Register\_1, converts it into a decimal integer and puts it into Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

#### 11.2.1.8.4 String to Integer – Base 16 – Hexadecimal

##### Example of String to Integer - Base 16 – Hexadecimal:

Program Variables		
	Name	Type
1.0	INIT: "e"	
	# of Array Elements: 32	(Clear field to disable array)
1	Register_1	String
2	Register_2	Number

Add Variable

Task - MainTask

Assignment

Destination: Register\_2  
Expression: STR\_TO\_INT(Register\_1,16)

Assignment ▼ Add Block

↓

```

PROG_CYCLE_TIME:2
PLC_CONNECTED: 0
REGISTER_1: e
REGISTER_2: 14
MainTask
    
```

Task - MainTask

Assignment

Destination: Register\_2  
Expression: STR\_TO\_INT(Register\_1,16)

**Explaining the Example:** Register\_1 is initialized to value “e.” STR\_TO\_INT takes the hexadecimal string in Register\_1, converts it into a decimal integer and puts it into Register\_2.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

### 11.2.1.9 Integer to String

The user will use *INT\_TO\_STR* when the user wants to move an integer into a string. The user can also convert the integer into a binary, octal, decimal or hexadecimal base.

#### 11.2.1.9.1 Integer to String – Base 2 – Binary

The screenshot shows the 'Program Variables' table and the 'Task - MainTask' configuration. In the 'Program Variables' table, the 'Name' column has 'Register\_1' and the 'Type' column has 'String'. The '# of Array Elements' is set to 32. In the 'Task - MainTask' configuration, the 'Call' block is set to 'INT\_TO\_STR(14, Register\_1, 2)'. A red box highlights the '2' in the function call, and a red box highlights the 'Help' text 'Help: INT\_TO\_STR(number,dest\_str,base)'. A blue arrow points down to the next screenshot.

Program Variables

Name	Type
Register_1	String

Task - MainTask

Call: INT\_TO\_STR(14, Register\_1, 2)

Help: INT\_TO\_STR(number,dest\_str,base)

PROG\_CYCLE\_TIME: 2  
PLC\_CONNECTED: 0  
REGISTER\_1: 1110  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW

**Explaining the Example:** INT\_TO\_STR converts the decimal integer 14 into binary and puts that value into Register\_1.

#### 11.2.1.9.2 Integer to String – Base 8 – Octal

The screenshot shows the 'Program Variables' table and the 'Task - MainTask' configuration. In the 'Program Variables' table, the 'Name' column has 'Register\_1' and the 'Type' column has 'String'. The '# of Array Elements' is set to 32. In the 'Task - MainTask' configuration, the 'Call' block is set to 'INT\_TO\_STR(14, Register\_1, 8)'. A red box highlights the '8' in the function call, and a red box highlights the 'Help' text 'Help: INT\_TO\_STR(number,dest\_str,base)'. A blue arrow points down to the next screenshot.

Program Variables

Name	Type
Register_1	String

Task - MainTask

Call: INT\_TO\_STR(14, Register\_1, 8)

Help: INT\_TO\_STR(number,dest\_str,base)

PROG\_CYCLE\_TIME: 2  
PLC\_CONNECTED: 0  
REGISTER\_1: 16  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW

**Explaining the Example:** INT\_TO\_STR converts the decimal integer 14 into octal and puts that value into Register\_1.

### 11.2.1.9.3 Integer to String – Base 10 – Decimal

Program Variables		
	Name	Type
	# of Array Elements: 32	(Clear field to disable array)
1	Register_1	String

Add Variable

Task - MainTask

Call Help: INT\_TO\_STR(number,dest\_str;base)  
INT\_TO\_STR(14, Register\_1,10)

Call Add Block

↓

```

PROG_CYCLE_TIME : 2
PLC_CONNECTED : 0
REGISTER_1 : 14
MainTask
Local IO: TBEN_S1_8DXP_GW
        
```

Task - MainTask

Call INT\_TO\_STR(14, Register\_1,10)

**Explaining the Example:** INT\_TO\_STR converts the decimal integer 14 into decimal and puts that value into Register\_1.

### 11.2.1.9.4 Integer to String – Base 16 – Hexadecimal

Program Variables		
	Name	Type
	# of Array Elements: 32	(Clear field to disable array)
1	Register_1	String

Add Variable

Task - MainTask

Call Help: INT\_TO\_STR(number,dest\_str;base)  
INT\_TO\_STR(14, Register\_1,16)

Call Add Block

↓

```

PROG_CYCLE_TIME : 2
PLC_CONNECTED : 0
REGISTER_1 : e
MainTask
Local IO: TBEN_S1_8DXP_GW
        
```

Task - MainTask

Call INT\_TO\_STR(14, Register\_1,16)

**Explaining the Example:** INT\_TO\_STR converts the decimal integer 14 into hexadecimal and puts that value into Register\_1.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

### 11.2.1.10 Array Initialize

The user will use `ARRAY_INIT` when the user wants to load certain array elements with pre-set values.

#### Example of Array Initialize:

PROG\_CYCLE\_TIME : 2  
PLC\_CONNECTED : 0  
REGISTER\_1  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW

Task - MainTask  
Call  
Help: ARRAY\_INIT(dest\_array,offset,val1,val2,...)  
ARRAY\_INIT(Register\_1,2,16,15,14,13)  
Call Add Block

**Explaining the Example:** `ARRAY_INIT` looks at `Register_1`, offsets the elements by two and then writes the integer values 16-13 into elements 2-5.

### 11.2.2 Timer

To access the Timer functions, highlight *Timer* and press “→” on the keyboard or click “->” with the mouse to advance to the next or previous level.

Function:  
String/Arrays ->  
Timer ->  
Counter ->  
Math ->  
Brackets ->  
Boolean Logic ->  
Compare ->  
Trigger ->  
Bit Operations ->  
Advanced IO/PLC ->  
Protocol Conver: ->

Function:  
<- START\_TIMER(Timer,expiration\_time)  
<- EXPIRED(Timer) - returns True if timer expired  
<- COUNT(Timer) - returns the number of ms since the

#### 11.2.2.1 Start Timer

The user will use `START_TIMER` when the user wants to start a timer. All values are in milliseconds.

#### Example of Start Timer:

Run Debug Print IO Config HMI Project Set Title About  
Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

Variables and Definitions  
Name Type  
1 Timer\_1 Timer/Counter  
2 Temp Number  
Add Variable

ARGEE Program  
Keyboard shortcuts (hidden)  
Task - MainTask  
If R\_TRIG(Door\_Open, Temp)  
0.0 Call Start\_Time(Timer\_1, 5000)  
Call Add Block

**Explaining the Example:** If the door opens, Timer\_1 starts counting. Timer\_1 expires after 5000ms (or 5 seconds).



## NOTE

R\_TRIG (Rising Edge Trigger) is discussed later in this chapter in section [11.2.8.2 Rising Edge Trigger \(R\\_TRIG\)](#).

### 11.2.2.2 Timer Expired

The user will use *Expired* when the user wants an action to occur after a timer has expired.

#### Example of *Timer Expired*:

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

#### Variables and Definitions

	Name	Type
1	Timer_1	Timer/Counter
2	Temp	Number

Add Variable

Alias Variables (hidden)

Function Block Add

#### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask

0 ± If R\_TRIG(Door\_Open, Temp)

0.0 Call Start\_Time(Timer\_1, 5000)

Call Add Block

1 ± If Door\_Open & Expired(Timer\_1)

1.0 Assignment Destination: Alarm Expression: 1

**Explaining the Example:** When the door opens, Timer\_1 starts. If the door is still open when Timer\_1 expires, the alarm turns on.



## NOTE

R\_TRIG (Rising Edge Trigger) is discussed later in this chapter in section [11.2.8.2 Rising Edge Trigger \(R\\_TRIG\)](#).

### 11.2.2.3 Timer Count

The user will use *Count* when the user wants an action to occur at a certain instant in time (before the timer has expired).

#### Example of *Timer Count*:

The screenshot shows the ARGEE Program interface. The top toolbar includes icons for Run, Debug, Print, IO Config, HMI, Project, Set Title, and About. The Project Title is 'TBEN-S1-8DXP (192.168.1.12) V3.2.3.5'.

**Variables and Definitions**

	Name	Type
1	Timer_1	Timer/Counter
2	Temp	Number

Below the table is an 'Add Variable' button. Underneath, there is a section for 'Alias Variables (hidden)' with a 'Function Block' dropdown and an 'Add' button.

**ARGEE Program**

The program structure is as follows:

- Task - MainTask
  - 0 If R\_TRIG(Door\_Open, Temp)
    - 0.0 Call Start\_Time(Timer\_1, 5000)
      - Call Add Block
  - 1 If Door\_Open & (Count(Timer\_1)=2500)
    - 1.0 Assignment
      - Destination: Alarm
      - Expression: 1

**Explaining the Example:** When the door opens, Timer\_1 starts. If the door is still open after 2500ms (2.5 seconds), a light will turn on.



#### NOTE

R\_TRIG (Rising Edge Trigger) is discussed later in this chapter in section [11.2.8.2 Rising Edge Trigger \(R\\_TRIG\)](#).

### 11.2.3 Counter

To access the Counter functions, highlight *Counter* and press “->” on the keyboard or click “->” with the mouse to advance to the next or previous level.

The screenshot shows two panels. The left panel is a 'Function:' list with categories: String/Arrays, Timer, Counter, Math, Brackets, Boolean Logic, Compare, Trigger, Bit Operations, Advanced IO/PLC Array, and Protocol Conversion. The 'Counter' category is highlighted with a blue arrow pointing to the right panel.

The right panel shows the 'Function:' list with two sub-functions highlighted in blue:

- <- EXPIRED(Counter) - returns True if Counter
- <- COUNT(Counter) - returns the current count



## 11.2.3.1 Counter Expired

The user will use *Expired* when the user wants an action to occur after a counter has expired.

### Example of Counter Expired:

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

### Variables and Definitions

Program Variables	
Name	Type
1 Timer_1	Timer/Counter
2 Temp	Number

Add Variable

1 Alias Variables (hidden)

Function Block Add

### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask

0.0 Condition R\_TRIG(Door\_Open, Temp)

0.0 Count Up Counter: Counter\_1 Preset: 10

Count Up Add Block

Condition Add Block

2.0 If EXPIRED(Counter\_1)

2.0 Assignment Destination: Alarm Expression: 1

**Explaining the Example:** When the door opens, Counter\_1 counts up one time. Counter\_1 expires after 10 counts. If Counter\_1 expires, an alarm turns on.



#### NOTE

R\_TRIG (Rising Edge Trigger) is discussed later in this chapter in section [11.2.8.2 Rising Edge Trigger \(R\\_TRIG\)](#).

### 11.2.3.2 Counter Count

The user will use *Count* when the user wants an action to occur at a certain count (before the counter has expired).

#### Example of Counter Count:

The screenshot shows the ARGEE Program interface. On the left, the 'Variables and Definitions' panel lists 'Timer\_1' as a 'Timer/Counter' and 'Temp' as a 'Number'. The main workspace shows a 'Task - MainTask' with a 'Condition' block 'R\_TRIG(Door\_Open, Temp)' leading to a 'Count Up' block for 'Counter\_1' with a 'Preset' of 10. Below this is an 'If' block with the condition 'COUNT(Counter\_1) = 2' (highlighted with a red box), leading to an 'Assignment' block where 'Alarm' is set to '1'.

**Explaining the Example:** When the door opens, Counter\_1 counts up one time. Counter\_1 expires at 10 counts. If the door is opened 2 times, a light turns on.



#### NOTE

R\_TRIG (Rising Edge Trigger) is discussed later in this chapter in section [11.2.8.2 Rising Edge Trigger \(R\\_TRIG\)](#).

### 11.2.4 Math

The user will use Math Operations if they want to monitor, compare, or combine data from different registers. To access the Math functions, highlight Math and press “→” on the keyboard or click “->” with the mouse to advance to the next or previous level.

The screenshot shows a transition from a list of categories to a list of math functions. On the left, the 'Math' category is highlighted with a blue bar. A large blue arrow points to the right, where a list of math functions is displayed, including '<- +', '<- -', '<- \*', '<- /', '% - Modulo', 'abs(number)', 'min(num1,num2)', and 'max(num1,num2)'.

### 11.2.4.1 Addition

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

#### Example of Add:

0 ±	Condition IO_Basic_Input_Input_value_0		
0.0	Assignment	Destination: Temporary_Register	Expression: Register_A + Register_B

**Explaining the Example:** When Input\_value\_0 goes true, the value in Register\_A will be added to the value in Register\_B. The result is placed in Temporary\_Register.

### 11.2.4.2 Subtraction

The user will use subtraction (-) to subtract one value from another value.

#### Example of Subtraction:

0 ±	Condition (Register_A - Register_B) > 1		
0.0	Coil	IO_Basic_Output_Output_value_1	

**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.

### 11.2.4.3 Multiplication

The user will use multiplication (\*) to multiply one value with another value.

#### Example of Multiplication:

0 ±	Condition (Register_A * Register_B) < 1000		
0.0	Coil	IO_Basic_Output_Output_value_1	

**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.

### 11.2.4.4 Division

The user will use division (/) to divide one value into another value.

#### Example of Division:

0 ±	Condition IO_Basic_Input_Input_value_1		
0.0	Assignment	Destination: Temporary_Register	Expression: (Register_A / Register_B)

**Explaining the Example:** When Input\_value\_1 goes true, the value in Register\_A will be divided by the value in Register\_B. The result is placed in Temporary\_Register.



#### NOTE

If the user is concerned about keeping the fractions, the user should set their program variable type to “Floating.”

0 ±	Program Variables	
	Name	Type
1	Temporary_Register	Floating ▼
2	Register_A	Floating ▼
3	Register_B	Floating ▼

If the registers are not set to floating, 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”

#### 11.2.4.5 Modulo

The user will use modulo (%) if the user wants to capture the “remainder” after a division (/) has occurred.

Example of *Modulo*:

0 ±	Condition	IO_Basic_Input_Input_value_1	
0.0	Assignment	Destination:	Temporary_Register
		Expression:	(Register_A % Register_B)

**Explaining the Example:** When Input\_value\_1 goes 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:

$36 / 6 = \text{“6”}$  with a remainder of “0” → ARGEE displays “0”

$34 / 6 = \text{“5”}$  with a remainder of “4” → ARGEE displays “4”

$6 / 36 = \text{“0”}$  with a remainder of “6” → ARGEE displays “6”

### 11.2.4.6 Absolute Value

The user will use absolute value (abs) to capture the magnitude of a real number without regard to its sign (+/-).

#### Example of Absolute Value:

0 ±	Condition IO_Basic_Input_Input_value_1		
0.0	Assignment	Destination: Temporary_Register	Expression: abs(Register_A)

**Explaining the Example:** When Input\_value\_1 goes true, ARGEE will take the absolute value of the integer in Register\_A, and place into Temporary\_Register.

### 11.2.4.7 Minimum Value

The user will use the minimum value (min) to compare multiple registers and place the smallest value in to the destination register. The user can also use the minimum value (min) to compare multiple registers and use the smallest value in a math operation.

#### Example of Minimum Value:

0 ±	Condition IO_Basic_Input_Input_value_1		
0.0	Assignment	Destination: Temporary_Register	Expression: min(Register_A, Register_B)

**Explaining the Example:** When Input\_value\_1 goes true, ARGEE will take the smallest value between Register\_A and Register\_B and place that value into Temporary\_Register

OR

0 ±	Condition IO_Basic_Input_Input_value_1		
0.0	Assignment	Destination: Temporary_Register	Expression: Register_C + min(Register_A, Register_B)

**Explaining the Example:** When Input\_value\_1 goes true, ARGEE will take the smallest value between Register\_A and Register\_B and place that value into the addition operation. The result will be put into Temporary\_Register.

### 11.2.4.8 Maximum Value

The user will use the maximum value (max) to compare multiple registers and place the largest value into the destination register. The user can also use the maximum value (max) to compare multiple registers, and use the largest value in a math operation.

#### Example of *Maximum Value*:

0 ±	Condition	IO_Basic_Input_Input_value_1	
0.0	Assignment	Destination:	Temporary_Register
		Expression	max(Register_A, Register_B)

**Explaining the Example:** When Input\_value\_1 goes true, ARGEE will take the largest value between Register\_A and Register\_B and place that value into Temporary\_Register.

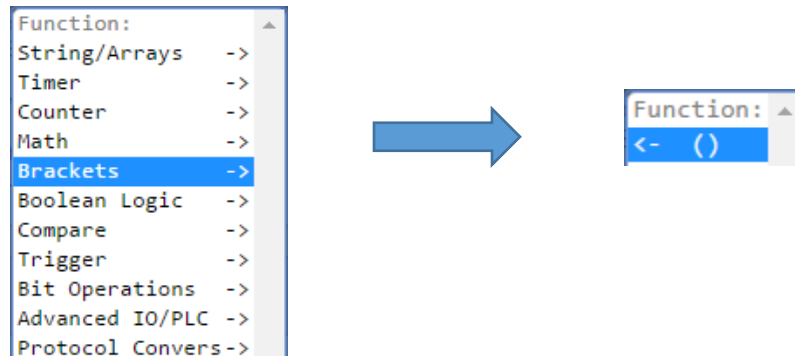
OR

0 ±	Condition	IO_Basic_Input_Input_value_1	
0.0	Assignment	Destination:	Temporary_Register
		Expression	Register_C + max(Register_A, Register_B)

**Explaining the Example:** When Input\_value\_1 goes 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 put into Temporary\_Register.

### 11.2.5 Brackets

To access the bracket function, highlight *Bracket* and press “→” on the keyboard or click “->” with the mouse to advance to the next or previous level.



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

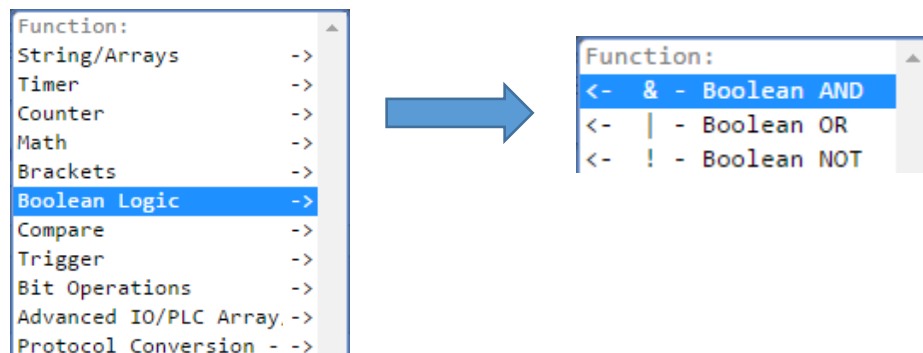
#### Example of Brackets:

0 ±	Condition	IO_Basic_Input_Input_value_1	
0.0	Assignment	Destination:	Temporary_Register
		Expression:	Register_A / (Register_B + Register_C)

**Explaining the Example:** When Input\_value\_1 goes true, ARGEE will examine the “(Register\_B + Register\_C)” operation first, and then divide that answer into the value in Register\_A. The result will be stored in Temporary\_Register.

### 11.2.6 Boolean Logic

Boolean logic consists of AND (&), OR (!) and NOT (!) statements. To access the Boolean Logic functions, highlight *Boolean Logic* and press “->” on the keyboard or click “->” with the mouse to advance to the next or previous level.



#### NOTE

For information on bitwise Boolean operations, see [12.7.5 Advanced Bitwise Operations – Bit Masking](#).

### 11.2.6.1 Boolean AND

The user will use the Boolean AND (&) operation if the user wants a specific Action to occur when more than one condition is met.”

#### Example of Boolean AND:

0 ±	Condition	IO_Basic_Input_Input_value_1 & IO_Basic_Input_Input_value_2	
0.0	Assignment	Destination: Register_A	Expression: 1

**Explaining the Example:** When both Input\_value\_1 AND input\_value\_2 are true, load the value “1” into Register\_A.

### 11.2.6.2 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:

0 ±	Condition	IO_Basic_Input_Input_value_1   IO_Basic_Input_Input_value_2	
0.0	Assignment	Destination: Register_A	Expression: 1

**Explaining the Example:** When either Input\_value\_1 OR input\_value\_2 are true, load the value “1” into Register\_A.

### 11.2.6.3 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:

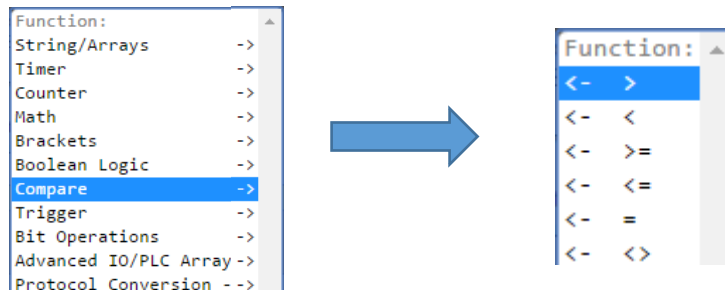
0 ±	Condition	IO_Basic_Input_Input_value_1	
0.0	Assignment	Destination: Register_A	Expression: 1
Assignment ▼ Add Block			
1 ±	Condition	! IO_Basic_Input_Input_value_1	
1.0	Assignment	Destination: Register_A	Expression: 0

**Explaining the Example:** When Input\_value\_1 is true, load the value “1” into Register\_A. When Input\_value\_1 is NOT true (or false), load the value “0” into Register\_A.



## 11.2.7 Compare

The user will select the Compare function if he needs to compare two numbers and find the smallest or largest, or see if they are equal or unequal. To access the Compare functions, highlight Compare and press “->” on the keyboard or click “->” with the mouse to advance to the next or previous level.



### 11.2.7.1 Greater Than

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

**Example of Greater Than:**

0 ±	Condition	Register_A > Register_B	
0.0	Assignment	Destination:	Register_C
		Expression:	1

**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.

### 11.2.7.2 Greater Than or Equal to

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

**Example of Greater Than or Equal to:**

0 ±	Condition	Register_A >= Register_B	
0.0	Assignment	Destination:	Register_C
		Expression:	1

**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.

### 11.2.7.3 Less Than

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

#### Example of Less Than:

0 ±	Condition	Register_A < Register_B	
0.0	Assignment	Destination:	Register_C
		Expression:	1

**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.

### 11.2.7.4 Less Than or Equal to

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

#### Example of Less Than or Equal To:

0 ±	Condition	Register_A <= Register_B	
0.0	Assignment	Destination:	Register_C
		Expression:	1

**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.

### 11.2.7.5 Equal

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

#### Example of Equal:

0 ±	Condition	Register_A = Register_B	
0.0	Assignment	Destination:	Register_C
		Expression:	1

**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.

### 11.2.7.6 Not Equal

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

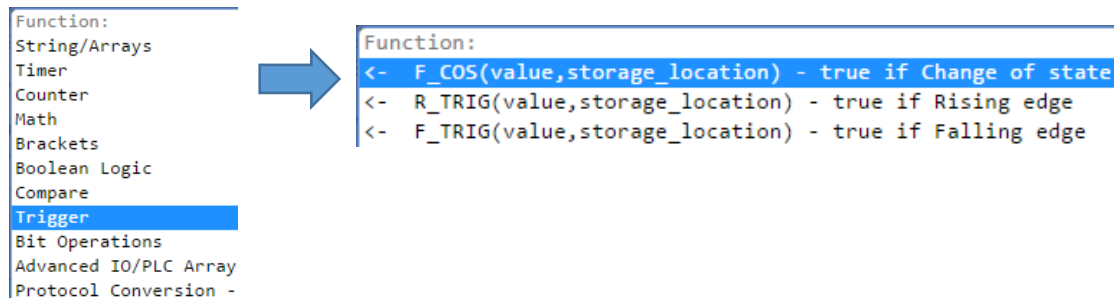
**Example of Not Equal:**

0 ±	Condition	Register_A = Register_B	
0.0	Assignment	Destination: Register_C	Expression: 1
<div>Assignment ▼ Add Block</div>			
1 ±	Condition	Register_A <> Register_B	
1.0	Assignment	Destination: Register_C	Expression: 0

**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.

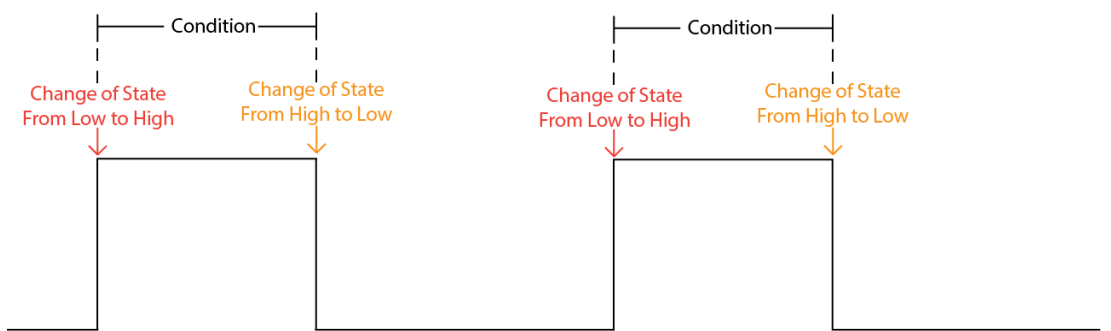
### 11.2.8 Trigger

To access the Trigger functions, highlight *Trigger* and press “->” on the keyboard or click “->” with the mouse to advance to the next or previous level.



#### 11.2.8.1 Change of State (F\_COS)

The user will use F\_COS if the user wants an action to occur only when a condition changes state.



Condition	
	F_COS(IO_Slot1.Input.Input_value_1, Temp_1) & IO_Slot1.Input.Input_value_1 = 1

- = 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 (F\_COS)*:

0 ±	Condition	F_COS(IO_Basic_Input_Input_value_1, Temp_1) & IO_Basic_Input_Input_value_1 = 1	
	0.0	Assignment	Destination: Register_A Expression: 1
1 ±	Condition	F_COS(IO_Basic_Input_Input_value_2, Temp_2) & IO_Basic_Input_Input_value_1 = 0	
	1.0	Assignment	Destination: Register_A Expression: 0

**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 temp register.

### 11.2.8.2 Rising Edge Trigger (R\_TRIG)

The user will use R\_TRIG if the user wants an action to occur only during the rising edge of a condition.

#### Example of *R\_TRIG*:

0 ±	Condition	R_TRIG(IO_Basic_Input_Input_value_1, Temp_1)	
	0.0	Assignment	Destination: Register_A Expression: 1

**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.



#### NOTE

Each monitored condition requires its own temp register.

### 11.2.8.3 Falling Edge Trigger (F\_TRIG)

The user will use F\_TRIG if the user wants an action to occur only during the falling edge of a condition.

**Example of F\_TRIG:**

0 ±	Condition	F_TRIG(IO_Basic_Input_Input_value_1, Temp_1)	
0.0	Assignment	Destination: Register_A	Expression: 1

**Explaining the Example:** When Input\_value\_1 does a Change of State from high (1) to Low (0), the value "1" is loaded into Register\_A



NOTE

Each monitored condition requires its own temp register.

### 11.2.9 Bit Operations

To access the Bit Operations functions, highlight *Bit Operations* and press "→" on the keyboard or click "->" with the mouse to advance to the next or previous level.

Function:

- String/Arrays ->
- Timer ->
- Counter ->
- Math ->
- Brackets ->
- Boolean Logic ->
- Compare ->
- Trigger ->
- Bit Operations ->**
- Advanced IO/PLC Array->
- Protocol Conversion ->

➔

Function:

```
<- GET_BITS(curr_val,offset,length) - return
<- SET_BITS(curr_val,offset,length,bitfield)
```

#### 11.2.9.1 Get Bits

The user will use GET\_BITS if the user wants to get bits from a certain register, and put them into another register.

**Example of GET\_BITS (Target Register, Bit Offset, Bit Length):**

Program Variables		
	Name	Type
1.0	INIT : 14	
1	Register_1	Number
2	Register_2	Number

Add Variable

Task - MainTask

0 ±	Condition	true	
0.0	Assignment	Destination: Register_2	Expression: GET_BITS(Register_1,2,2)

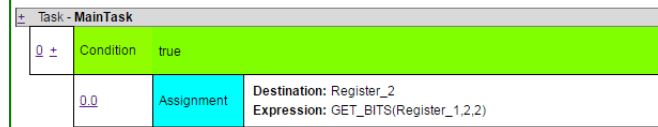
Assignment Add Block

↓

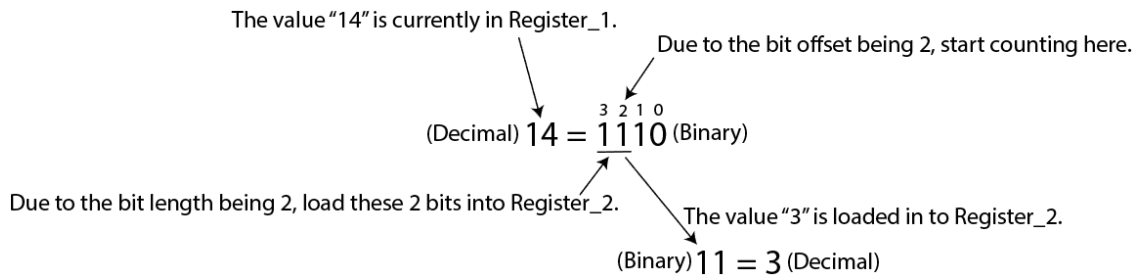
```

PROG_CYCLE_TIME : 2
PLC_CONNECTED : 0
REGISTER_1 : 14
REGISTER_2 : 3
MainTask
  Local IO: TBEN_S1_8DXP_GW
  Local IO: Basic - Input
  Local IO: Basic - Output

```



**Explaining the Example:** The value “14” is loaded into Register\_1. Due to the bit offset being 2, ARGEE starts counting at bit 2. Due to the bit length being 2, ARGEE takes the next 2 bits, converts them to decimal, and places the value “3” in Register\_2. (View the below example)



### 11.2.9.2 Set Bits

The user will use SET\_BITS if the user wants to get bits from a certain register and put them into another register.

**Example of SET\_BITS (Target Register, Bit Offset, Bit Length, Replacement Value):**

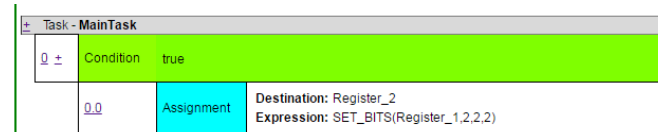
Program Variables		
	Name	Type
1.0	INIT - 14	
1	Register_1	Number
2	Register_2	Number



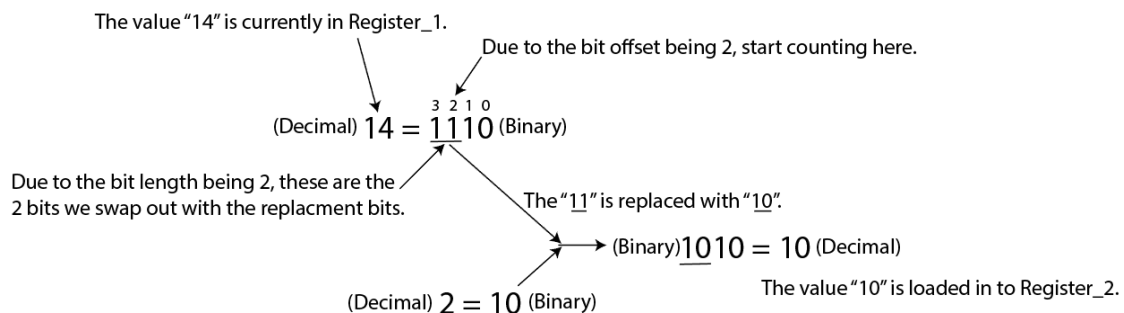
```

PROG_CYCLE_TIME : 2
PLC_CONNECTED : 0
REGISTER_1 : 14
REGISTER_2 : 10
MainTask
  Local IO: TBEN_S1_8DXP_GW
  Local IO: Basic - Input
  Local IO: Basic - Output

```

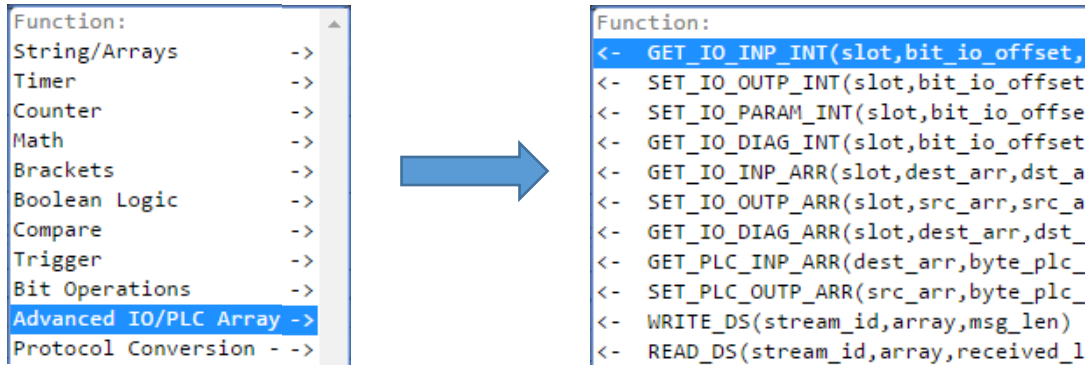


**Explaining the Example:** The value “14” is loaded into Register\_1. Due to the bit offset being 2, ARGEE starts counting at bit 2. Due to the bit length being 2, ARGEE takes the next 2 bits, replaces those bits with the replacement value (a binary “2”), converts the new number to decimal and place that value in Register\_2.



### 11.2.10 Advanced IO/PLC Array

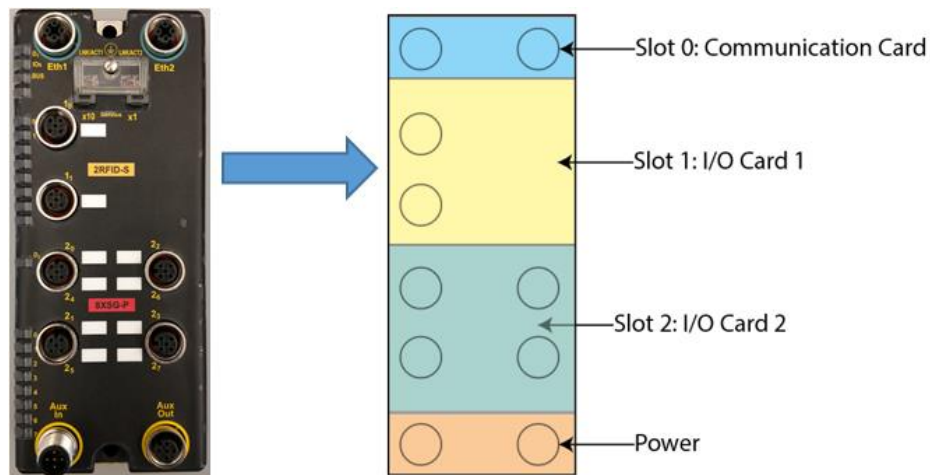
To access the Advanced IO/PLC Array functions, highlight *Advanced IO/PLC Array* and press “→” on the keyboard or click “→” with the mouse to advance to the next or previous level.



#### NOTE

The Advanced IO/PLC Array built-in function blocks are for advanced users.

For the next several examples, we will be using a Turck BLCEN-6M12LT-2RFID-S-8XSG-P. It is important for the user to know that BL Compacts are broken down into different sections (or slots). Slot 0 is the communication card, Slot 1 is the first I/O card and Slot 2 is the second I/O card.



### 11.2.10.1 Get IO Input Integer

The user will use GET\_IO\_INP\_INT if the user wants get input bits from a certain register and put them into another register.

**Example of GET\_IO\_INP\_INT (Target Slot, Bit Offset, Bit Length):**

8XSG-P Input Data Map

INPUT	BYTE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	24	DI 2 <sub>7</sub>	DI 2 <sub>6</sub>	DI 2 <sub>5</sub>	DI 2 <sub>4</sub>	DI 2 <sub>3</sub>	DI 2 <sub>2</sub>	DI 2 <sub>1</sub>	DI 2 <sub>0</sub>
	25	-	-	-	-	-	-	-	-

Program Variables		
	Name	Type
1	Register_1	Number
Add Variable		

Task - MainTask		
Q ±	Condition	true
Q.0	Assignment	Destination: Register_1 Expression: GET_IO_INP_INT(2,0,1)



```

PROG_CYCLE_TIME : 5
PLC_CONNECTED : 0
REGISTER_1 : 0
MainTask
  Local IO: Slot0
  Local IO: Slot1 - Input
  Local IO: Slot1 - Output
  Local IO: Slot1 - Diagnostics
  Local IO: Slot2 - Input
    Input_value_0 : 0
    Input_value_1 : 0

```

Task - MainTask		
Q ±	Condition	true
Q.0	Assignment	Destination: Register_1 Expression: GET_IO_INP_INT(2,0,1)



```

PROG_CYCLE_TIME : 5
PLC_CONNECTED : 0
REGISTER_1 : 1
MainTask
  Local IO: Slot0
  Local IO: Slot1 - Input
  Local IO: Slot1 - Output
  Local IO: Slot1 - Diagnostics
  Local IO: Slot2 - Input
    Input_value_0 : 1
    Input_value_1 : 0

```

Task - MainTask		
Q ±	Condition	true
Q.0	Assignment	Destination: Register_1 Expression: GET_IO_INP_INT(2,0,1)

**Explaining the Example:** The user is using a BLCEN-6M12LT-2RFID-S-8XSG-P. The user wants to monitor Input\_value\_0 on the 8XSG-P card and store that value in Register\_1. The user uses the GET\_IO\_INP\_INT command and targets slot 2 (the 8XSG card), Bit 0 and the user only wants to monitor 1 bit. As Input\_value\_0 goes true, so does Register\_1.



### 11.2.10.2 Set IO Output Integer

The user will use SET\_IO\_OUTP\_INT if the user wants set bits in an output register.

**Example of SET\_IO\_OUTP\_INT (Target Slot, Bit Offset, Bit Length, Replacement Value):**

8XSG-P Output Data Map

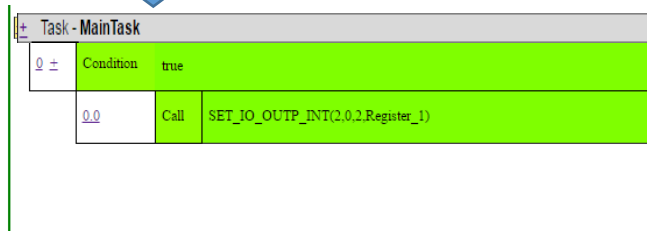
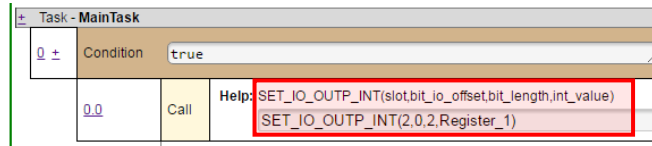
OUTPUT	BYTE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	24	DO 2 <sub>7</sub>	DO 2 <sub>6</sub>	DO 2 <sub>5</sub>	DO 2 <sub>4</sub>	DO 2 <sub>3</sub>	DO 2 <sub>2</sub>	DO 2 <sub>1</sub>	DO 2 <sub>0</sub>
	25	-	-	-	-	-	-	-	-

Program Variables		
	Name	Type
1.0	INIT : 3	
1	Register_1	Number

```

PROG_CYCLE_TIME : 5
PLC_CONNECTED : 0
REGISTER_1 : 3
MainTask
  Local IO: Slot0
  Local IO: Slot1 - Input
  Local IO: Slot1 - Output
  Local IO: Slot1 - Diagnostics
  Local IO: Slot2 - Input
  Local IO: Slot2 - Output
  Output_value_0 : 1
  Output_value_1 : 1

```



**Explaining the Example:** The user is using a BLCEN-6M12LT-2RFID-S-8XSG-P. The user wants to set outputs on the 8XSG-P card to correspond to the value that is in Register\_1 (Register\_1 is initialized to value "3" for this example). The user uses the SET\_IO\_OUTP\_INT command and targets slot 2 (the 8XSG card), Bit 0, sets his bit length to 2 and loads the value "3" (or 11 in binary) into the output register. As a result, Output\_value\_0 & Output\_value\_1 go true.

### 11.2.10.3 Set IO Parameters Integer

The user will use SET\_IO\_PARAM\_INT if the user wants set bits in a parameter register. Turck recommends that the user sets their device parameters via the *IO Config* tab or via the device webserver. If the user wants to use this feature, please contact Turck for more information.

## 11.2.10.4 Get IO Diagnostics Integer

The user will use GET\_IO\_DIAG\_INT if the user wants get diagnostic bits from a certain register and put them into another register.

**Example of GET\_IO\_DIAG\_INT (Target Slot, Bit Offset, Bit Length):**

BLCEN-8M12LT-4IOL-4AI4AO-VI Diagnostic Data Map

INPUT	BYTE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Diagnostics	34	Hardware Failure	-	-	-	AI 2 <sub>s</sub> Over-flow/Under-flow	-	Wire Break AI 2 <sub>s</sub> (4...20 mA range only)	Range Error AI 2 <sub>s</sub>

BLCEN-8M12LT-4IOL-4AI4AO-VI Web Server Diagnostic

! Gateway Diagnostics  
[Event Log](#)  
[Ethernet Statistics](#)  
[EtherNet/IP™ Memory Map](#)  
[Modbus TCP Memory Map](#)  
[Links](#)  
[Gateway Configuration](#)  
[Network Configuration](#)  
[Change Admin Password](#)  


---

[Slot 1 - 4IOL](#)  
! Slot 2 - 4AI4AO-V/I

### Diagnostics

Please use the refresh function (e.g. F5) of your browser to update the values

Slot	Source	Diagnostics
0	Gateway	Module Diagnostics Available INFO: ARGEE Project Running
2	4AI4AO-V/I	Analog In 0 - Measured value out of range active Analog In 0 - Wire break (4-20 mA only) active

0 + Program Variables		
	Name	Type
1.0	INIT : 3	
1	Register_1	Number ▼
Add Variable		

Task - MainTask	
0 ±	Condition true
0.0	Call
Help: GET_IO_DIAG_INT(slot.bit_io_offset.bit_length) GET_IO_DIAG_INT(2,0,8)	



```

PROG_CYCLE_TIME : 6
PLC_CONNECTED : 0
REGISTER_1 : 3
MainTask
├─ Local IO: Slot0
├─ Local IO: Slot1 - Input
├─ Local IO: Slot1 - Output
├─ Local IO: Slot1 - Diagnostics
├─ Local IO: Slot2 - Input
├─ Local IO: Slot2 - Output
├─ Local IO: Slot2 - Diagnostics
└─ Measured value out of range 0 : 1
   Measured value out of range 1 : 0
   Measured value out of range 2 : 0
   Measured value out of range 3 : 0
   Wire break 4-20 mA only 0 : 1

```

Task - MainTask	
0 ±	Condition true
0.0	Assignment
Destination: Register_1 Expression: GET_IO_DIAG_INT(2,0,8)	

**Explaining the Example:** The user is using a BLCEN-8M12LT-4IOL-4AI4AO-VI. The user wants to monitor diagnostic data on the 4AI 4AO card and store that value in Register\_1. The user uses the GET\_IO\_DIAG\_INT command and targets slot 2 (the 4AI4AO card), Bit 0 and the user wants to monitor 8 bits. When a wire break and an out of range error occur, the value “3” (or Binary 0000 0011) gets loaded into Register\_1.



### NOTE

To monitor port 2 diagnostics, the user should set their offset to 16.  
 To monitor port 3 diagnostics, the user should set their offset to 32.  
 To monitor port 4 diagnostics, the user should set their offset to 48.  
 The user should read the device data sheet to figure out additional information.

### 11.2.10.5 Get IO Input Array

The user will use GET\_IO\_INP\_ARR if the user wants to get an input array from a device. This command is primarily used with Turck RFID and IO-Link modules. Turck recommends that the user uses the ARGEE libraries when working with RFID and IO-Link. If the user wants to use this feature, please contact Turck for more information.

### 11.2.10.6 Set IO Output Array

The user will use SET\_IO\_OUTP\_ARR if the user wants to set an output array on a device. This command is primarily used with Turck RFID and IO-Link modules. Turck recommends that the user uses the ARGEE libraries when working with RFID and IO-Link. If the user wants to use this feature, please contact Turck for more information.

### 11.2.10.7 Get IO Diagnostics Array

The user will use GET\_IO\_DIAG\_ARR if the user wants to get a diagnostic array from a device. This command is primarily used with Turck RFID and IO-Link modules. Turck recommends that the user uses the ARGEE libraries when working with RFID and IO-Link. If the user wants to use this feature, please contact Turck for more information.

### 11.2.10.8 Get PLC Input Array

The user will use GET\_PLC\_INP\_ARR if the user wants to read an entire array from the PLC. This command is extremely helpful when transferring RFID write data to the device.

**Example of GET\_PLC\_INP\_ARR (Destination Array, Byte PLC Offset, Byte Length):**

The screenshot illustrates the configuration of the GET\_PLC\_INP\_ARR command in the software interface. It is divided into four main sections:

- Program Variables:** A table showing the variable 'RFID\_Write\_Data' of type 'Byte'. The array length is set to 4.
- Task - MainTask Configuration:** A configuration window for the 'MainTask' showing a 'Call' action set to 'GET\_PLC\_INP\_ARR(RFID\_Write\_Data,0,4)'. A help text box provides the syntax: 'Help: GET\_PLC\_INP\_ARR(dest\_arr,byte\_plc\_offset,byte\_length)'.
- RFID\_WRITE\_DATA Array:** A list of array elements with their values: [0]: 0x11, [1]: 0x22, [2]: 0x33, [3]: 0x44.
- Task - MainTask Configuration (Bottom):** A configuration window for the 'MainTask' showing a 'Call' action set to 'GET\_PLC\_INP\_ARR(RFID\_Write\_Data,0,4)'.

**Explaining the Example:** RFID write data is sent from the PLC and loaded into program variable RFID\_Write\_Data. The user uses the GET\_PLC\_INP\_ARR command, sets an offset of 0 and transfers 4 bytes (or two words) from the PLC to the devices.

### 11.2.10.9 Set PLC Output Array

The user will use SET\_PLC\_OUTP\_ARR if the user wants to transfer an entire array to the PLC. This command is extremely helpful when transferring RFID read data to a PLC.

**Example of SET\_PLC\_OUTP\_ARR (Source Array, Byte PLC Offset, Byte Length):**

The diagram illustrates the configuration of the SET\_PLC\_OUTP\_ARR command. It consists of three main parts:

- Program Variables Table:** A table with columns 'Name' and 'Type'. It lists several initialization values (INIT : [3]=4, INIT : [2]=3, INIT : [1]=2, INIT : [0]=1) and a variable 'RFID\_READ\_Data' of type 'Byte'. The number of array elements is set to 4.
- Task - MainTask Ladder Logic:** A snippet showing a 'Condition' block set to 'true' and a 'Call' block for 'SET\_PLC\_OUTP\_ARR(RFID\_Write\_Data,0,4)'. The 'Call' block is highlighted with a red box.
- Variable Declaration Block:** A block of code showing the declaration of 'RFID\_READ\_DATA' as an array of 4 elements, each of size 1. The array elements are listed as [0] : 0x01, [1] : 0x02, [2] : 0x03, and [3] : 0x04. The first four elements are highlighted with a red box.

A blue arrow points from the 'Task - MainTask' ladder logic snippet to the 'Variable Declaration Block', indicating the data flow from the array to the PLC output.

**Explaining the Example:** RFID read data is loaded into program variable RFID\_Read\_Data. The user uses the SET\_PLC\_OUTP\_ARR command, sets an offset of 0 and transfers 4 bytes (or two words) to the PLC.

### 11.2.10.10 Write Data Stream

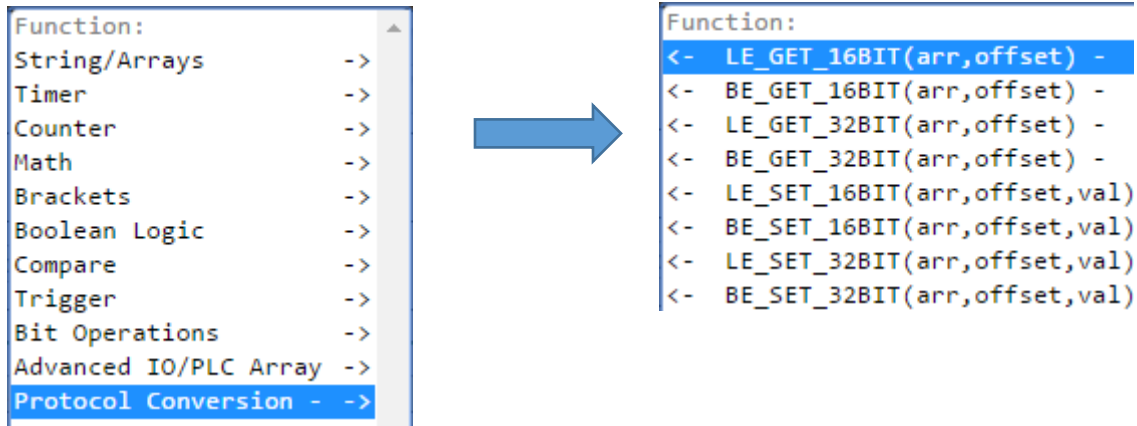
The user will use the WRITE\_DS command if the user is working with acyclic messaging. This command is primarily used with IO-Link modules. Turck recommends that the user uses the ARGEE libraries when working with IO-Link. If the user wants to use this feature, please contact Turck for more information.

### 11.2.10.11 Read Data Stream

The user will use the READ\_DS command if the user is working with acyclic messaging. This command is primarily used with IO-Link modules. Turck recommends that the user uses the ARGEE libraries when working with IO-Link. If the user wants to use this feature, please contact Turck for more information.

## 11.2.11 Protocol Conversion

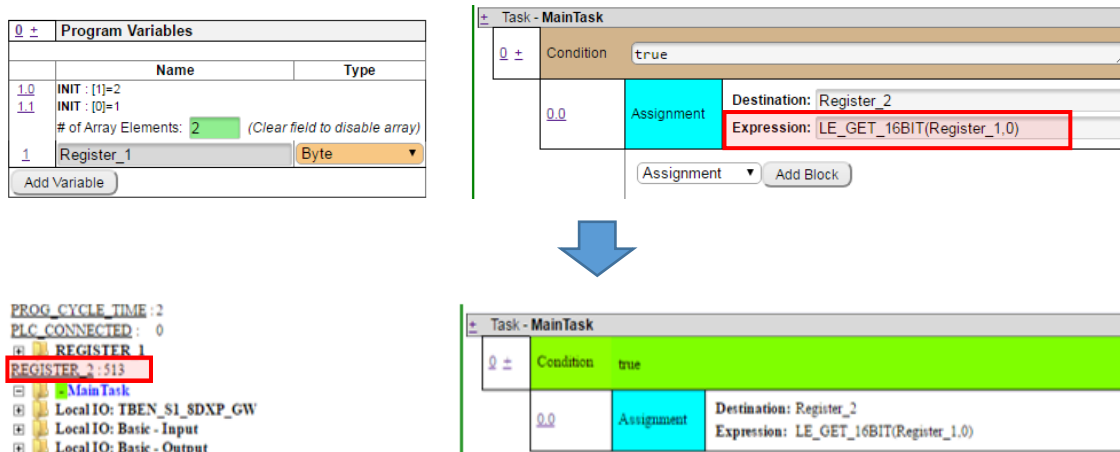
To access the Protocol Conversion functions, highlight *Protocol Conversion* and press “→” on the keyboard or click “→” with the mouse to advance to the next or previous level.



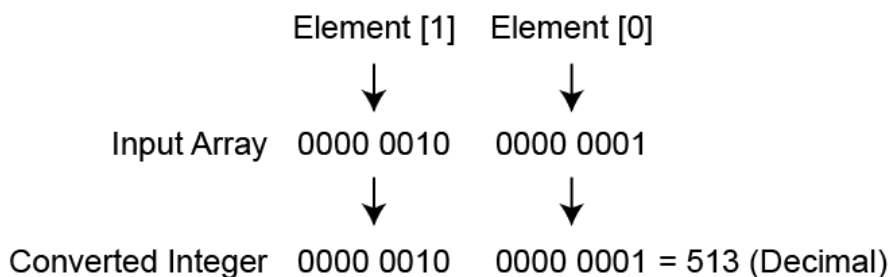
### 11.2.11.1 Little-endian, Get 16 Bits

The user will use LE\_GET\_16BIT if the user wants to do a protocol conversion from Big-endian to Little-endian. All registers in ARGEE are Little-endian.

**Example of LE\_GET\_16BIT (Target Array, Offset):**



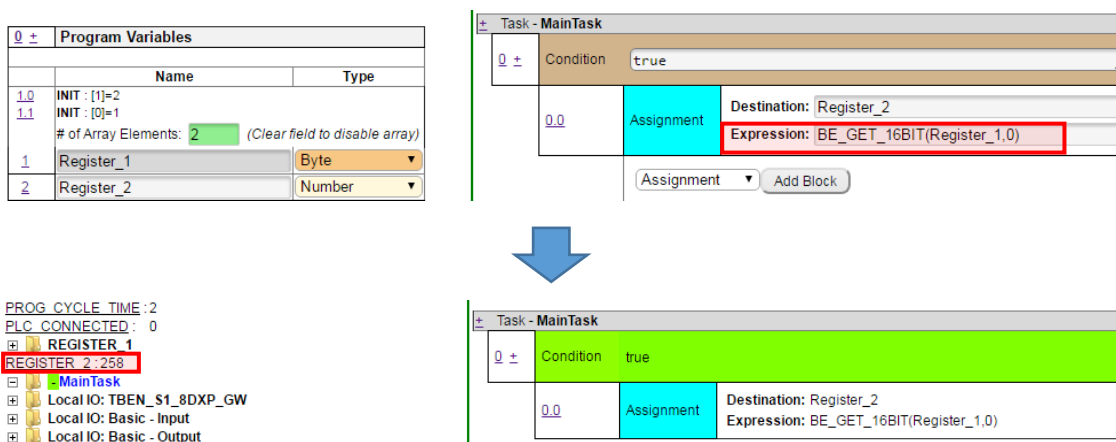
**Explaining the Example:** This example does not actually do anything special because the user is converting Little-endian to Little-endian. The value “1” is loaded into Register\_1 position zero and the value “2” is loaded into Register\_1 position one.



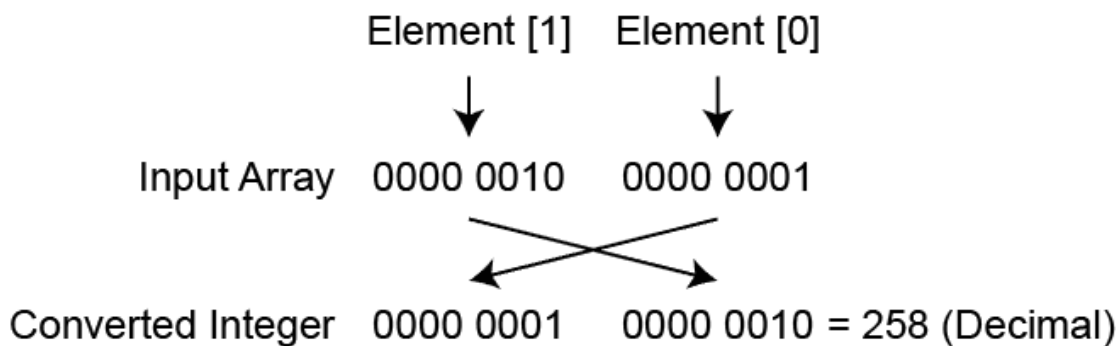
### 11.2.11.2 Big-endian, Get 16 Bits

The user will use BE\_GET\_16BIT if the user wants to do a protocol conversion from Little-endian to Big-endian. All registers in ARGEE are Little-endian.

**Example of BE\_GET\_16BIT (Target Array, Offset):**



**Explaining the Example:** The value “1” is loaded into Register\_1 position zero and the value “2” is loaded into Register\_1 position one. The BE\_GET\_16BIT command swaps byte 1 with byte 2 and loads the value “258” into Register\_2.



### 11.2.11.3 Little-endian, Get 32 Bits

The user will use LE\_GET\_32BIT if the user wants to do a protocol conversion from Big-endian to Little-endian. All registers in ARGEE are Little-endian.

**Example of LE\_GET\_32BIT (Target Array, Offset):**

**Program Variables**

	Name	Type
1.0	INIT : [3]=4	
1.1	INIT : [2]=3	
1.2	INIT : [1]=2	
1.3	INIT : [0]=1	
# of Array Elements: 4 (Clear field to disable array)		
1	Register_1	Byte
2	Register_2	Number

**Task - MainTask**

Condition	Assignment	Destination	Expression
0 ±	true	Register_2	LE_GET_32BIT(Register_1,0)

**REGISTER\_2** : 67305985

**Main task**

Condition	Assignment	Destination	Expression
0 ±	true	Register_2	LE_GET_32BIT(Register_1,0)

**Explaining the Example:** This example does not do anything special because the user is converting Little-endian to Little-endian. The value "1" is loaded into Register\_1 position zero, the value "2" is loaded into Register\_1 position one, the value "3" is loaded into Register\_1 position two and the value "4" is loaded into Register\_1 position three.

	Element [3]	Element [2]	Element [1]	Element [0]
	↓	↓	↓	↓
Input Array	0000 0100	0000 0011	0000 0010	0000 0001
	↓	↓	↓	↓
Converted Integer	0000 0100	0000 0011	0000 0010	0000 0001 = 67305985 (Decimal)

### 11.2.11.4 Big-endian, Get 32 Bits

The user will use BE\_GET\_16BIT if the user wants to do a protocol conversion from Little-endian to Big-endian. All registers in ARGEE are Little-endian.

**Example of BE\_GET\_32BIT (Target Array, Offset):**

Program Variables		
	Name	Type
1.0	INIT : [3]=4	
1.1	INIT : [2]=3	
1.2	INIT : [1]=2	
1.3	INIT : [0]=1	
# of Array Elements: 4 (Clear field to disable array)		
1	Register_1	Byte
2	Register_2	Number

Add Variable

Task - MainTask

Condition true

0.0

Assignment

Destination: Register\_2  
Expression: BE\_GET\_32BIT(Register\_1,0)

Assignment Add Block

Condition Add Block

```

PROG_CYCLE_TIME : 2
PLC_CONNECTED : 0
REGISTER_1
REGISTER_2 : 16909060
mainTask
Local IO: TBEN_S1_8DXP_GW
Local IO: Basic - Input
Local IO: Basic - Output
        
```

Task - MainTask

Condition true

0.0

Assignment

Destination: Register\_2  
Expression: BE\_GET\_32BIT(Register\_1,0)

**Explaining the Example:** The value “1” is loaded into Register\_1 position zero, the value “2” is loaded into Register\_1 position one, the value “3” is loaded into Register\_1 position two and the value “4” is loaded into Register\_1 position three. The BE\_GET\_32BIT command swaps all four bytes and loads the value “16909060” into Register\_2.

	Element [3]	Element [2]	Element [1]	Element [0]
Input Array	0000 0100	0000 0011	0000 0010	0000 0001
Converted Integer	0000 0001	0000 0010	0000 0011	0000 0100

= 16909060 (Decimal)

140

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



### 11.2.11.5 Little-endian, Set 16 Bits

The user will use LE\_SET\_16BIT if the user wants to set a value in an array that is in Little-endian format.

**Example of LE\_SET\_16BIT (Target Array, Offset, Replacement Value):**

Name	Type
# of Array Elements: 2	(Clear field to disable array)
1 Register_1	Byte

PROG\_CYCLE\_TIME: 2  
PLC\_CONNECTED: 0  
REGISTER\_1  
Array: length=2, elem\_size=1  
[0]: 0x0e  
[1]: 0x00  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW

Task - MainTask  
Condition: true  
Call: LE\_SET\_16BIT(Register\_1,0,14)  
Help: LE\_SET\_16BIT(arr,offset,val) - sets value in the array offset

**Explaining the Example:** The user loads the value “14” (or Hex “e”) into Register\_1. Little-endian counts bytes from right to left so the hex value “0x0e” (or decimal 14) is placed in Register\_1 position zero.

### 11.2.11.6 Big-endian, Set 16 Bits

The user will use BE\_SET\_16BIT if the user wants to set a value in an array that is in Big-endian format.

**Example of BE\_SET\_16BIT (Target Array, Offset, Replacement Value):**

Name	Type
# of Array Elements: 2	(Clear field to disable array)
1 Register_1	Byte

PROG\_CYCLE\_TIME: 2  
PLC\_CONNECTED: 0  
REGISTER\_1  
Array: length=2, elem\_size=1  
[0]: 0x00  
[1]: 0x0e  
MainTask  
Local IO: TBEN\_S1\_8DXP\_GW

Task - MainTask  
Condition: true  
Call: BE\_SET\_16BIT(Register\_1,0,14)  
Help: BE\_SET\_16BIT(arr,offset,val) - sets value in the array offset

**Explaining the Example:** The user loads the value “14” (or Hex “e”) into Register\_1. Big-endian counts bytes from left to right so the hex value “0x0e” (or decimal 14) is placed in Register\_1 position one.

### 11.2.11.7 Little-endian, Set 32 Bits

The user will use LE\_SET\_32BIT if the user wants to set a value in an array that is in Little-endian format.

**Example of LE\_SET\_32BIT (Target Array, Offset, Replacement Value):**

The screenshot illustrates the configuration and execution of the LE\_SET\_32BIT function. On the left, the 'Program Variables' table shows 'Register\_1' as a 'Byte' array with 4 elements. Below it, the 'REGISTER\_1' array is shown with values: [0]: 0x0e, [1]: 0x00, [2]: 0x00, [3]: 0x00. The '0x0e' value is highlighted in red. On the right, the 'Task - MainTask' is shown with a 'Condition' of 'true' and a 'Call' block containing 'LE\_SET\_32BIT(Register\_1,0,14)'. A red box highlights the help text: 'Help: LE\_SET\_32BIT(arr,offset,val) - sets value in the array offset'. A blue arrow points down to the next state.

**Program Variables**

Name	Type
Register_1	Byte

# of Array Elements: 4 (Clear field to disable array)

0x0e

0x00

0x00

0x00

**Task - MainTask**

Condition	Call
true	LE_SET_32BIT(Register_1,0,14)

Help: LE\_SET\_32BIT(arr,offset,val) - sets value in the array offset

LE\_SET\_32BIT(Register\_1,0,14)

**Explaining the Example:** The user loads the value “14” (or Hex “e”) into Register\_1. Little-endian counts bytes from right to left so the hex value “0x0e” (or decimal 14) is placed in Register\_1 position zero.

### 11.2.11.8 Big-endian, Set 32 Bits

The user will use BE\_SET\_32BIT if the user wants to set a value in an array that is in Big-endian format.

**Example of BE\_SET\_32BIT (Target Array, Offset, Replacement Value):**

The screenshot illustrates the configuration and execution of the BE\_SET\_32BIT function. On the left, the 'Program Variables' table shows 'Register\_1' as a 'Byte' array with 4 elements. Below it, the 'REGISTER\_1' array is shown with values: [0]: 0x00, [1]: 0x00, [2]: 0x00, [3]: 0x0e. The '0x0e' value is highlighted in red. On the right, the 'Task - MainTask' is shown with a 'Condition' of 'true' and a 'Call' block containing 'BE\_SET\_32BIT(Register\_1,0,14)'. A red box highlights the help text: 'Help: BE\_SET\_32BIT(arr,offset,val) - sets value in the array offset'. A blue arrow points down to the next state.

**Program Variables**

Name	Type
Register_1	Byte

# of Array Elements: 4 (Clear field to disable array)

0x00

0x00

0x00

0x0e

**Task - MainTask**

Condition	Call
true	BE_SET_32BIT(Register_1,0,14)

Help: BE\_SET\_32BIT(arr,offset,val) - sets value in the array offset

BE\_SET\_32BIT(Register\_1,0,14)

**Explaining the Example:** The user loads the value “14” (or Hex “e”) into Register\_1. Big-endian counts bytes from left to right so the hex value “0x0e” (or decimal 14) is placed in Register\_1 position three.

## 11.3 ARGEE Security Features

### 11.3.1 Visual Behavior

If there is an ARGEE program running on the block, 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.

### **11.3.2 Connection Behavior**

#### **11.3.2.1 EtherNet IP Master**

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

#### **11.3.2.2 Modbus TCP Master**

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"

#### **11.3.2.3 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. The 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 PLC Connectivity](#).

### 11.3.3 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 block's IP address into any HTML5-compatible web browser.

The screenshot shows a web browser window displaying the Turck webserver interface. The address bar shows the URL [192.168.1.54/info.html](http://192.168.1.54/info.html). The page has a yellow header with the TURCK logo and a navigation menu on the left. The main content area displays 'Station Information' for the device TBEN-L4-8IOL.

Station Information	
Type	TBEN-L4-8IOL
Identification Number	6814082
Firmware Revision	V3.2.3.0
Bootloader Revision	V1.0.0.0
EtherNet/IP™ Revision	V2.7.38.0
PROFINET Revision	V1.7.9.0
Modbus TCP Revision	V2.4.0.0
IO Framework Revision	V1.0.24.0
IO-Link Master Revision	V2.13.6.0
Digital IO Revision	V1.0.23.0
Build Number	327



#### NOTE

The default password to log into the block's webserver is "password".

To password-protect the user's ARGEE environment, the user must change the webserver password. To change the webserver password, select *Change Admin Password* link, follow the instructions, and click *Submit*. An example is shown below.



Change Admin Password

TURCK.COM For comments or questions, please email TURCK Support

TURCK

TBEN-L4-8IOL LOGOUT [ADMIN@192.168.1.99]

**STATION**

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

**BASIC**

- IO-LINK PORT 1

**Change Admin Password**

This form allows you to setup your own password for your station. If you alter the default password, there's no way to recover the password except sending it to the TURCK service.

Old password: \*\*\*\*\*

New password: \*\*\*\*

Retype new password: \*\*\*\*

Submit Reset

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

**Enter Password:**

submit



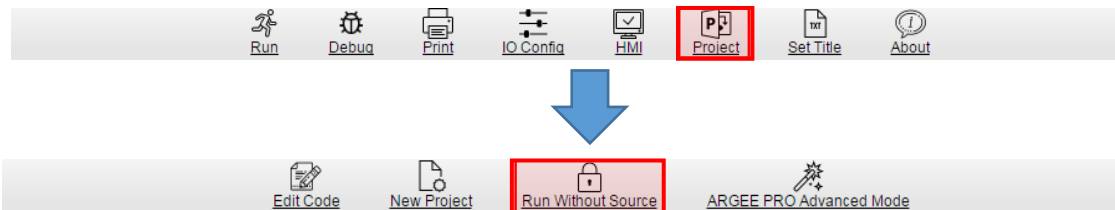
## NOTE

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

### 11.3.4 Source Code Protection – Run Without Source

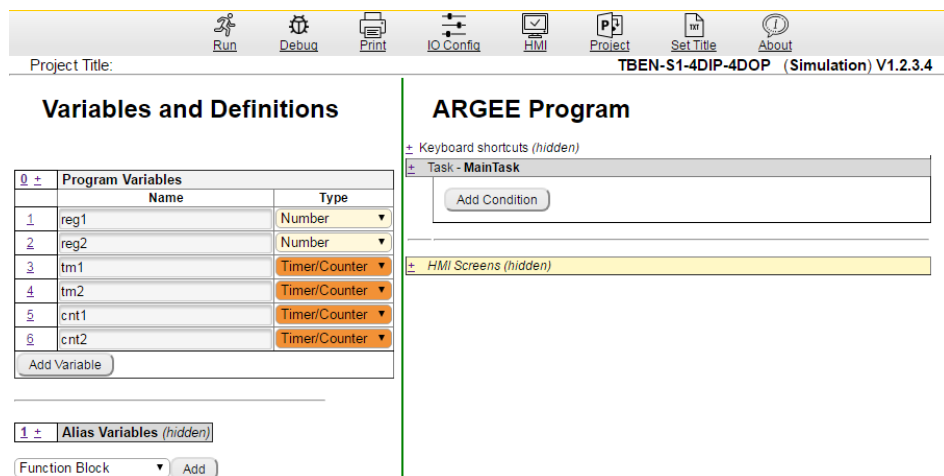
If a user wants to prevent “end users” from logging into the block and seeing or modifying code, the user will want to use the *Run Without Source* feature.

To access *Run Without Source*, the user must first click on the *Project* link in the ARGEE menu bar.



If the user clicks on *Run Without Source* and then logs out of the environment, the ARGEE program code will be hidden to anyone who tries to log into the block.

#### Logging in before clicking Run Without Source:



#### Logging in after the user click Run without Source:



#### NOTE

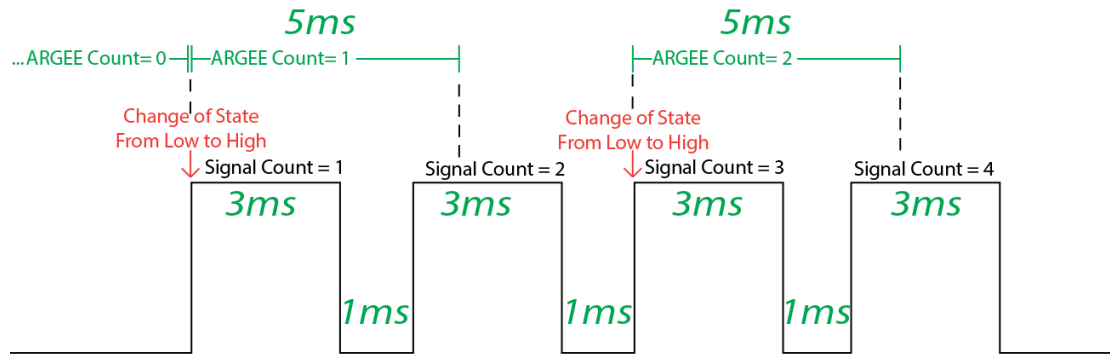
The user needs to save a Master Copy of the program before the user logs out of the environment if the user wants to view or edit the code in the future

## 11.4 System Performance

### 11.4.1 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 high speed motion applications.

### 11.4.2 IO Variable Formats

*IO Variable Formats* are normally used when working with IO-Link or transferring data with a PLC.

```

PROG_CYCLE_TIME: 5
PLC_CONNECTED: 0
MainTask
  Local IO: TBEN_L4_8IOL_GW
  Local IO: Basic - Input
  Local IO: Basic - Output
  Local IO: Basic - Diagnostics
  Local IO: IO_Link_Port_1 - Input
  Local IO: IO_Link_Port_1 - Output
  Output data word 0: 1
  Output data word 1: 0
  
```

Task - MainTask			
Q ±	Condition	True	
Q 0	Assignment	Destination:	IO_IO_Link_Port_1_Output_Output_data_word_0
		Expression:	1

**Explaining the Example:** The user set IO-Link Port 1 (bit 0) true.

```

PROG_CYCLE_TIME: 5
PLC_CONNECTED: 0
MainTask
  Local IO: TBEN_L4_8IOL_GW
  Local IO: Basic - Input
  Local IO: Basic - Output
  Local IO: Basic - Diagnostics
  Local IO: IO_Link_Port_1 - Input
  Local IO: IO_Link_Port_1 - Output
  Output data word 0: 4096
  Output data word 1: 0
  
```

Task - MainTask			
Q ±	Condition	True	
Q 0	Assignment	Destination:	IO_IO_Link_Port_1_Output_Output_data_word_0.12
		Expression:	1

**Explaining the Example:** The user set IO-Link Port 1 (bit 12) true.

The user can also target bits in a word by using Word.Offset.BitLength.

PROG\_CYCLE\_TIME : 2

PLC\_CONNECTED : 0

REG1 : 23219

REG2 : 5

[-] MainTask

Local IO: TBEN\_L5\_8IOL\_GW

Local IO: Basic - Input

Local IO: Basic - Output

Local IO: Basic - Diagnostics

Local IO: IO\_Link\_Port\_1 - Input

Input\_data\_word\_0 : 23219

Input\_data\_word\_1 : 0

Input\_data\_word\_2 : 0

Input\_data\_word\_3 : 0

Task - MainTask

0 ±	Condition	True
0.0	Assignment	Destination: reg1 Expression: IO_IO_Link_Port_1_Input_Input_data_word_0
1 ±	Condition	True
1.0	Assignment	Destination: reg2 Expression: IO_IO_Link_Port_1_Input_Input_data_word_0.12.3

**Explaining the Example:** The input value from IO-Link Port 1 is placed in REG1 and the value of word 0, offset 12, 3 bits is placed in REG2.

HEX 5AB3

DEC 23,219

OCT 55 263

BIN 0101101010110011



### 11.4.3 Defining Variable Types – (Advanced Definitions)

Type	Description	Type	Allowed arithmetic expressions	Specific actions	Size
<b>Number</b>	A 32-bit signed integer to be used for arithmetic	32-bit signed integer	All integer arithmetic	Assignment	4 bytes
<b>Floating</b>	Single precision floating point. Only available in TBEN and FEN20-4DIP-4DXP	32-bit signed integer	All integer arithmetic	Assignment	4 bytes
<b>String</b>	Null-terminated array of ASCII character values stored as bytes				X
<b>Byte</b>	One unsigned byte.		All integer arithmetic	Assignment	1 byte
<b>Word</b>	One unsigned word.		All integer arithmetic	Assignment	2 bytes
<b>Timer/Counter</b>	Used with appropriate functions, such as “expired,” “count,” and appropriate actions such as “Timer On”	32-bit signed integer	argument to functions “expired” and “count”	Specific actions: Timer on, Timer off, Start timer, Count up, Count down	4 bytes
<b>State/Enum</b>	Integer variable that is used to designate states in state machine. Behaves identically to a regular integer variable except for 2 things: 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 bytes
<b>Retain Number</b>	Integer which is automatically saved to flash. Syncs about every two minutes.	32 bit signed integer	All integer arithmetic	Assignment	8 bytes (4 bytes of data, 4 bytes of additional information)
<b>Retain Float</b>	Single-precision floating point variable which is automatically saved to flash. Syncs about every two minutes.	32 bit signed integer	All integer arithmetic	Assignment	8 bytes (4 bytes of data, 4 bytes of additional information)

Type	Description	Type	Allowed arithmetic expressions	Specific actions	Size
<b>PLC Variables</b>	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	8 bytes (4 bytes of data, 4 bytes of additional information)
<b>Local IO</b>	Input/Output/Diagnostic points	They are mapped to integer variables in the program	All integer arithmetic	Assignment	8 bytes (4 bytes of data, 4 bytes of additional information)
<b>System Variables – PLC Connected</b>	PLC Connected	32 bit integer		Only 1 bit is used to indicate PLC connected state	8 bytes (4 bytes of data, 4 bytes of additional information)
<b>System Variables – Program Cycle Time</b>	Max cycle time (since program start)	32 bit integer indicating time in milliseconds		Time from the previous cycle to the current cycle.	8 bytes (4 bytes of data, 4 bytes of additional information)

## 11.5 I/O Variable Definitions

### 11.5.1 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

### 11.5.2 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

### 11.5.3 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

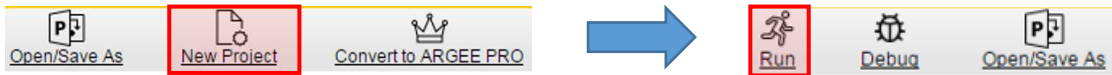
**Write\_data\_0\_0 - Write\_data\_7\_0** : Write Registers

## 12 Appendix II – Example Code

### 12.1 How to Erase a Project from a Device

#### 12.1.1 Running an empty Project

One way to erase the code on the device, is to first start a *New Project* and then click *Run*. This action will load an empty project to the device.



#### NOTE

Just starting a new project does not erase the code on your device. The user needs to run an empty project to erase the device.

#### 12.1.2 Using the Webserver Page

The user can also remove the ARGEE code by selecting *Erase ARGEE Program* from the device's webserver page.

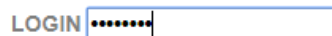
- On Google Chrome or Firefox, type the device's IP Address into the URL and hit Enter.



#### NOTE

The user can find their device's IP Address on the block itself, located in the hatch, set by rotary dials, or by using the Turck Service Tool application.

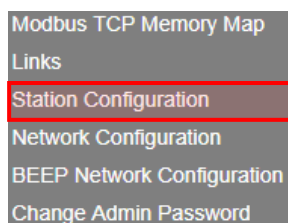
- On the webserver page, login to the device and you should see 4 new tabs show up on the left.



#### NOTE

The default password for logging in should be "password". If the user can't login and has obtained the device from another user, they may have changed the password.

- On the left side of webserver page, click the Station Configuration tab.



- At the bottom of the page click Erase ARGEE Program



### 12.1.3 Using the Turck Service Tool

The user can also remove the ARGEE program via the Turck Service Tool

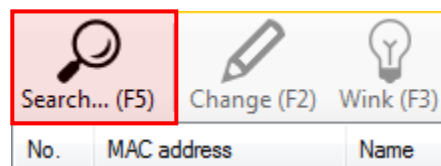
- Open the Turck Service Tool application



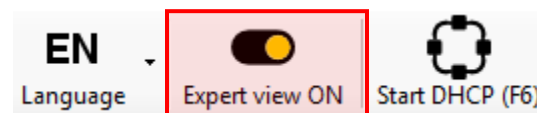
#### NOTE

The Turck Service Tool is available for download at [www.turck.us](http://www.turck.us)

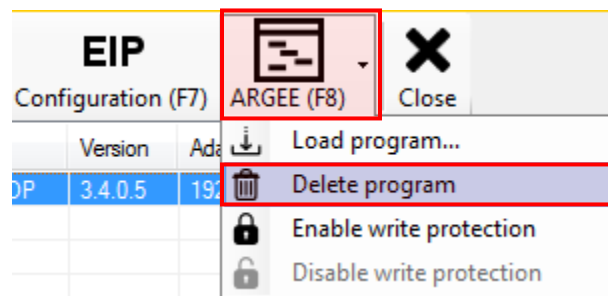
- Click the search tab to find your device.



- Enable Expert View.



- Select your device simply by clicking on it and then click ARGEE (F8) and under the tab you should see Delete program



## 12.2 Trace Example

0 ±	Condition	(F_COS(IO_Basic_Input_Input_value_0,Temp_1) & IO_Basic_Input_Input_value_0=1)	
0.0	Trace	Prefix String: Trace_1	Expression: 0
		Trace	Add Block
1 ±	Condition	(F_COS(IO_Basic_Input_Input_value_0,Temp_2) & IO_Basic_Input_Input_value_0=0)	
1.0	Trace	Prefix String: Trace_2	Expression: 1
		Trace	Add Block

**Explaining the Example:** When Input\_value\_0 is true, Trace\_1 time stamps that event. When Input\_value\_0 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.



### NOTE

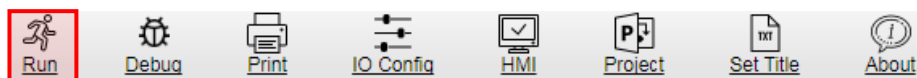
An example of *Trace* can be found in the Appendix [11.2.8.1 Change of State \(F\\_COS\)](#).

The Trace example is continued on the next page.

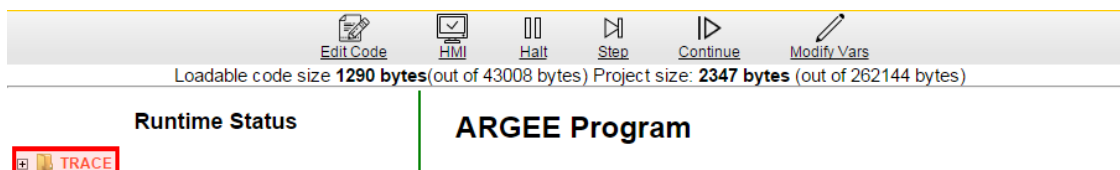


### Trace Example (Continued):

- Once the user has written the code, the user will click Run.



To view the Trace, the user will expand the *Trace* folder underneath the Runtime Status heading.





As the user triggers their condition true and false, the *Trace* data populates under the trace folder.

The screenshot shows the ARGEE Program interface. On the left, the 'Runtime Status' panel displays a table of trace data. On the right, the 'ARGEE Program' tree shows the 'Task - MainTask' with two conditions and two trace blocks.

Time	line	data
2758540	3	Trace_2:1
2758446	1	Trace_1:0
2758368	3	Trace_2:1
2758282	1	Trace_1:0
2758190	3	Trace_2:1
2758108	1	Trace_1:0
2758002	3	Trace_2:1
2757922	1	Trace_1:0
2757830	3	Trace_2:1

**ARGEE Program**

- Task - MainTask
  - Condition (F\_COS(IO\_Basic\_Input\_Input\_value\_0,Temp\_1) & IO\_Basic\_Input\_Input\_value\_0 = 1)
    - Trace Prefix String: Trace\_1 Expression: 0
  - Condition (F\_COS(IO\_Basic\_Input\_Input\_value\_0,Temp\_2) & IO\_Basic\_Input\_Input\_value\_0 = 0)
    - Trace Prefix String: Trace\_2 Expression: 1



#### NOTE

To calculate how long the user's condition is true, the user must subtract the two time stamps from one another:  $221196 - 221294 = 2$  ms.

## 12.3 How to Call a Function Block

### Example of calling a user-made Function Block:

The screenshot shows the ARGEE Program interface. On the left, the 'Variables and Definitions' panel displays a table of program variables and a table of function blocks. On the right, the 'ARGEE Program' tree shows the 'Task - MainTask' with a condition and a call block, and a 'Function Block - Unlock\_The\_Door()' with a condition and an assignment block.

Name	Type
Unlock	Unlock_The_Door

Name	Type	Segment
Unlock_Door_1	Number	VARIABLE
Unlock_Door_2	Number	VARIABLE
Unlock_Door_3	Number	VARIABLE

**ARGEE Program**

- Task - MainTask
  - Condition IO\_Basic\_Input\_Input\_value\_1
    - Call Help: Unlock\_The\_Door() Unlock()
- Function Block - Unlock\_The\_Door()
  - Condition true
    - Assignment Destination: Unlock\_Door\_1 Expression: 1

**Explaining the Example:** When Input\_value\_1 goes true, the function block *Unlock\_The\_Door* will be called.



#### NOTE

Function blocks are explained in Chapter [5.8 Function Blocks](#).



## 12.4 Creating and Importing Structure Text (ST View)

Structure Text (ST) is a common PLC programming language that is based on Pascal. ARGEE allows users to export their ARGEE project (Flowchart or Pro) in ST format, as well as convert imported ST into ARGEE PRO. Individual variables and function blocks can also be imported and exported.

### 12.4.1 Example of Exporting an ARGEE Project as Structure Text

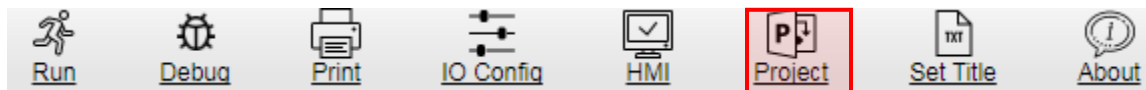
(ARGEE Setup)

0 ±	Program Variables	
	Name	Type
1	reg1	Number
2	my_fxn	my_function_block

2 ±	Function Block : my_function_block		
	Name	Type	Segment
0	var1	Number	VARIABLE

Task - MainTask		
0	Assignment	Destination: reg1 Expression: reg1+1
1	Call	Help: my_function_block() my_fxn()

Function Block - my_function_block()		
0	Assignment	Destination: var1 Expression: var1+1



ST View:

```

VAR
  default_task_1:Default_Task_1
;
reg1:INT
;
my_fxn:my_function_block
;
END_VAR
MODULE ("","");
TASK Default_Task_1()
VAR
END_VAR
VAR_INPUT
END_VAR
reg1:=reg1+1;
my_fxn();
END_TASK
FUNCTION_BLOCK my_function_block()
VAR
var1:INT;
END_VAR
VAR_INPUT
END_VAR
var1:=var1+1;
END_FUNCTION_BLOCK
HMI_BEGIN
END_HMI
  
```

ST View:

```

VAR
  default_task_1:Default_Task_1
;
reg1:INT
;
my_fxn:my_function_block
;
END_VAR
MODULE ("","");
TASK Default_Task_1()
VAR
END_VAR
VAR_INPUT
END_VAR
reg1:=reg1+1;
my_fxn();
END_TASK
FUNCTION_BLOCK my_function_block()
VAR
var1:INT;
END_VAR
VAR_INPUT
END_VAR
var1:=var1+1;
END_FUNCTION_BLOCK
HMI_BEGIN
END_HMI
  
```



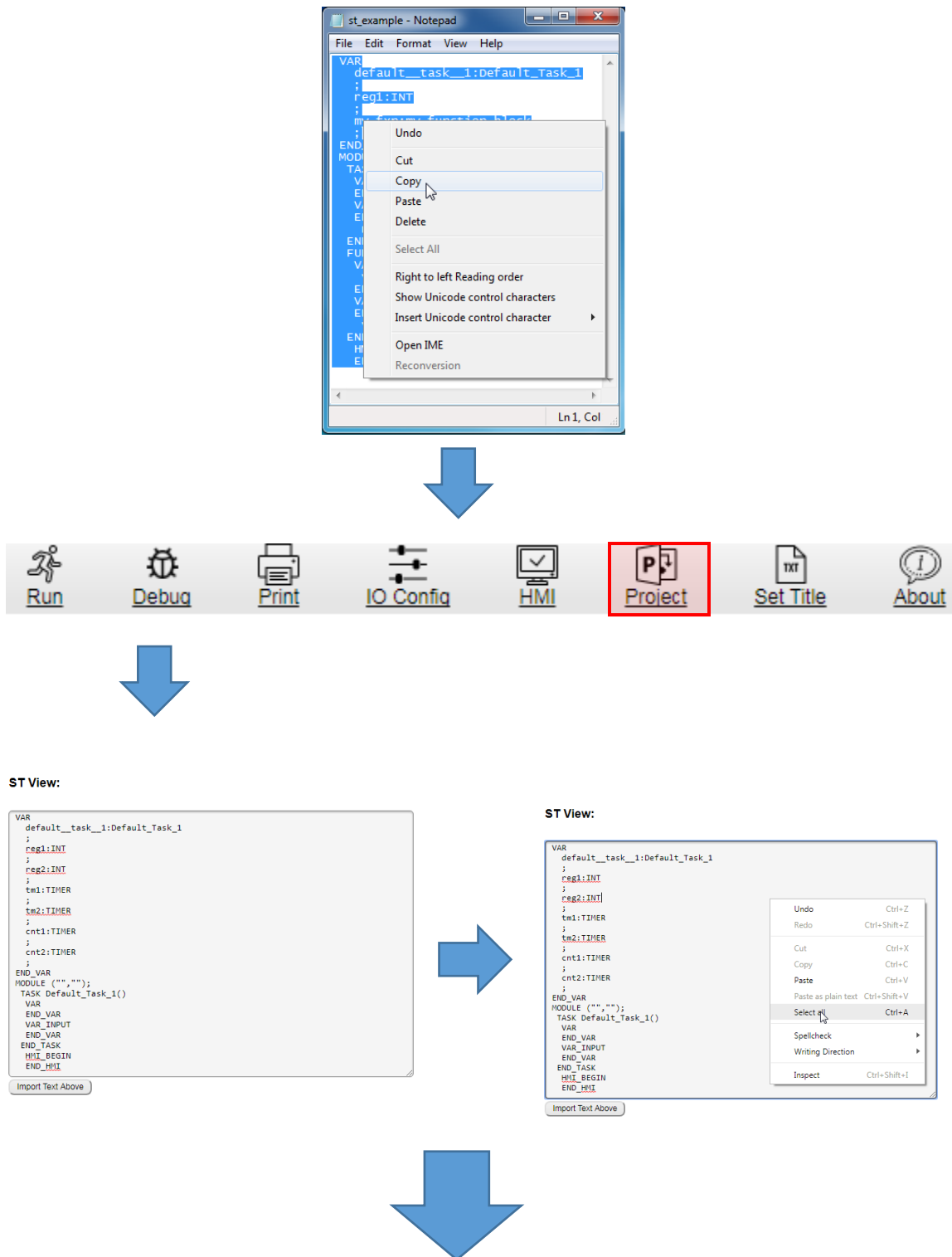
#### NOTE

From here, open the file where you want to store the text, and paste the text there. Turck recommends a blank .txt file created with Notepad.

**Explaining the Example:** An ARGEE project's Structure Text was copied, pasted, and saved into a .txt file.

## 12.4.2 Example of Importing Structure Text and Converting it into an ARGEE Project.

Open the file where the Structure Text is to be copied from. In this case, a .txt file created with Notepad was used to store the Structure Text from the previous example. Select it all, copy it all, and then switch to your open ARGEE environment:

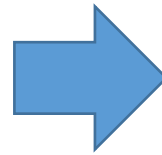


ST View:

```
VAR
default_task_1:Default_Task_1
;
reg1:INT
;
reg2:INT
;
tm1:TIMER
;
tm2:TIMER
;
cnt1:TIMER
;
cnt2:TIMER
;
END_VAR
MODULE ("", "");
TASK Default_Task_1()
VAR
END_VAR
VAR_INPUT
END_VAR
END_TASK
HMI_BEGIN
END_HMI

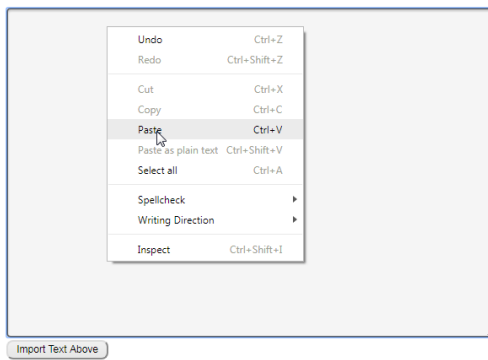
```

Import Text Above



Delete

ST View:



ST View:

```
VAR
default_task_1:Default_Task_1
;
reg1:INT
;
my_fxn:my_function_block
;
END_VAR
MODULE ("", "");
TASK Default_Task_1()
VAR
END_VAR
VAR_INPUT
END_VAR
reg1:=reg1+1;
END_TASK
FUNCTION_BLOCK my_function_block()
VAR
var1:INT;
END_VAR
VAR_INPUT
END_VAR
var1:=var1+1;
END_FUNCTION_BLOCK
HMI_BEGIN
END_HMI

```

Import Text Above

Project Title:

Run Debug Print IO Config HMI

### Variables and Definitions

Program Variables	
Name	Type
reg1	Number
my_fxn	my_function_block

Add Variable

---

1 Alias Variables (hidden)

---

2 Function Block: my\_function\_block Regular

Name	Type	Segment
var1	Number	VARIABLE

Add Element

### ARGEE Program

Keyboard shortcuts (hidden)

Task - MainTask

Assignment	Destination: reg1
	Expression: reg1+1

Assignment Add Block

---

Function Block - my\_function\_block()

Assignment	Destination: var1
	Expression: var1+1

Assignment Add Block

---

HMI Screens (hidden)

**Explaining the Example:** Preexisting Structure Text was copied, pasted, and converted into an ARGEE PRO project.



#### NOTE

To import individual function blocks, just copy the function block's definition in the preexisting structure text, paste it below the last function block definition in your open project's structure text, and click *Import Text Above*.

## 12.5 How to Export a CSV File

### 12.5.1 HMI export of arrays

The HMI can export a CSV file with the Submit Action. The CSV is saved to the Downloads folder on the connected computer. This action requires the following arguments.

- Program variable that holds the timer counter
- Update frequency of that timer in ms.
- Timestamp array containing the timer counter values.
- User's data array.

### 12.5.2 Example of Exporting a CSV

Program Variables		
	Name	Type
	# of Array Elements: 10 (Clear field to disable array)	
1	CSV_Transfer_Array	Number
	# of Array Elements: 10 (Clear field to disable array)	
2	Time_Stamp_Array	Number
3	Sample_Frequency	Timer/Counter
4	i	Number
5.0	INIT : 10	
5	Array_Full	Number
Add Variable		

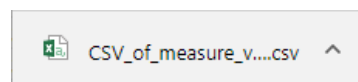
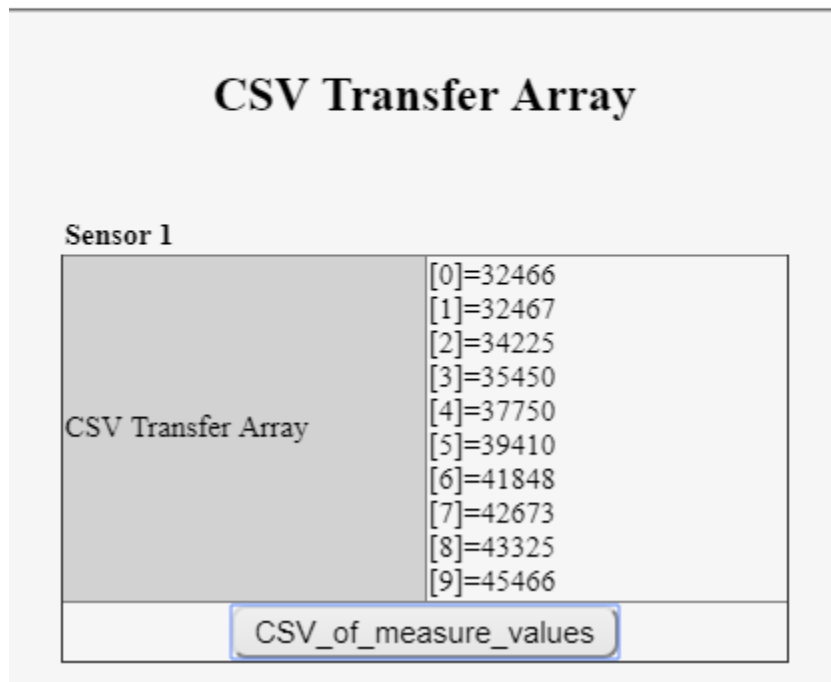
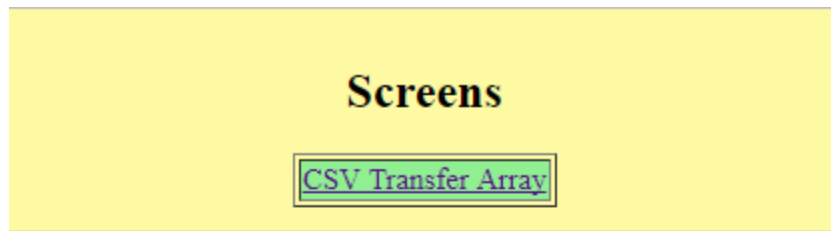
The user creates arrays for measurements and the timestamp.

Task - MainTask		
0	Call	Help: START_TIMER(Timer,expiration_time) START_TIMER(Sample_Frequency,2000)
1	Wait Until	EXPIRED(Sample_Frequency)
2	Assignment	Destination: CSV_Transfer_Array[i] Expression: IO_IO_Link_Port_1_Input_Input_data_word_0
3	Assignment	Destination: Time_Stamp_Array[i] Expression: i
4	Assignment	Destination: i Expression: i+1
5	Comment	Initialize the Array_Full variable to the same size as the CSV_Transfer_Array
6	If	i=Array_Full
6.0	Assignment	Destination: i Expression: 0

In this example when the timer expires the CSV\_TransferArray and Time\_Stamp\_Array are updated and the array pointer/variable is incremented by 1.

HMI Screens			
0	HMI Screen	CSV Transfer Array	
0.0	Section	Sensor 1	
0.0.0	Display Number/State/String	Title: CSV Transfer Array	Variable: CSV_Transfer_Array
		Units:	
0.0.1	Submit Action	Title: CSV_of_measure_values	Variable: CSV(i, 2000, Time_Stamp_Array, CSV_Transfer_Array)

The HMI will not only display but allow the export of measurements with the Submit Action by using the CSV( , , ) function.



	A	B
1	Thu Nov 30 2017 15:51:54 GMT-0600 (Central Standard Time)	32466
2	Thu Nov 30 2017 15:51:56 GMT-0600 (Central Standard Time)	32467
3	Thu Nov 30 2017 15:51:58 GMT-0600 (Central Standard Time)	34225
4	Thu Nov 30 2017 15:52:00 GMT-0600 (Central Standard Time)	35450
5	Thu Nov 30 2017 15:52:02 GMT-0600 (Central Standard Time)	37750
6	Thu Nov 30 2017 15:52:04 GMT-0600 (Central Standard Time)	39410
7	Thu Nov 30 2017 15:52:06 GMT-0600 (Central Standard Time)	41848
8	Thu Nov 30 2017 15:52:08 GMT-0600 (Central Standard Time)	42673
9	Thu Nov 30 2017 15:52:10 GMT-0600 (Central Standard Time)	43325
10	Thu Nov 30 2017 15:52:12 GMT-0600 (Central Standard Time)	45466

The measure data is displayed along with the time stamp. In this example, every 2 seconds.

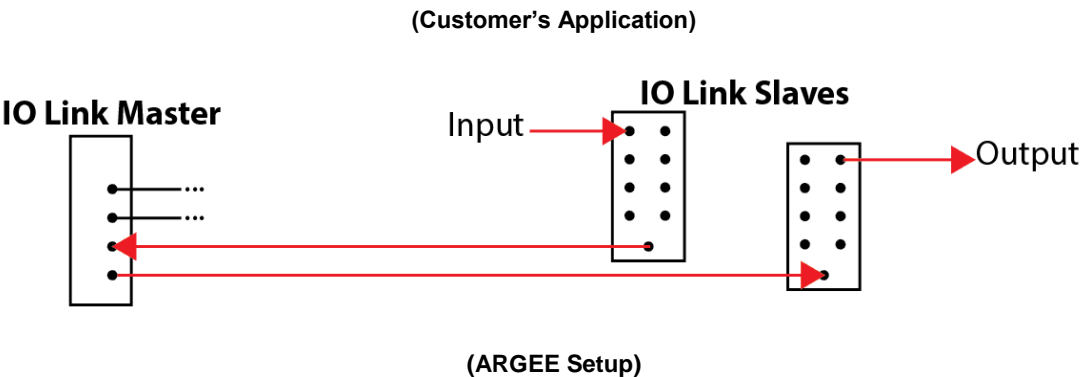
# 12.6 Advanced Application Examples

## 12.6.1 Working with IO-Link

When a user combines IO-Link technology with ARGEE, 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 *Turck IO-Link master manual* before attempting any IO-Link applications.

### 12.6.1.1 Working with IO-Link

Example of *IO-Link*:



<a href="#">0</a> ±	Condition	IO_IO_Link_Port_3_Input_Input_data_word_0.4
<a href="#">0.0</a>	Coil	IO_IO_Link_Port_4_Output_Output_data_word_0.7

**Explaining the example:** The user wanted an input on an IO-Link slave to turn on an output on a different IO-Link slave. The user modified the *IO Variable Formats* (Discussed in Chapter 12) to accomplish this task.



**NOTE**

Depending on the fieldbus used, it may be necessary to swap process data (Little-endian vs Big-endian). The process data can be changed from the IO Config tab. More information can be found in the Turck IO-Link master manual chapter 4, page 4-4.

### 12.6.1.2 Acyclic Communication – Read

When working with acyclic communication, the first thing the user needs to do is import the IO-Link libraries (Importing libraries is discussed in Chapter [5.9.3 Importing a Library](#)).

#### Example of Acyclic Communication – Read:

*(IODD file for an IO-Link Device)*

Index	Subindex	Name	Value Range	Default	Access Rights	Data Storage
67		Flashing Frequency				
67	1	Segment 1 Flashing Frequency (Hz)	0.5 through 20	1	rw	Yes

**(ARGE Setup)**

#### Variables and Definitions

Program Variables		
	Name	Type
1	IOL_Read	IOL_AsyncRead
	# of Array Elements: 100	(Clear field to disable array)
2	read_data_port1	Byte

#### ARGE Program

Keyboard shortcuts (hidden)

Task - MainTask

0 Call Help: IOL\_AsyncRead(port\_num,index,sub\_index,res\_data)  
IOL\_Read(1,67,1,read\_data\_port1)

Call Add Block

**(ARGE Debug Screen)**

```

READ_DATA_PORT1
Array: length=100,elem_size=1
[0]: 0x00
[1]: 0x0f
[2]: 0x00
[3]: 0x00

IOL_READ(IOL_ASYNCREAD)
DS_RX_ARR
DS_TX_ARR
INDEX: 67
PORT_NUM: 1
READ_RES: 18
RES: 8
RES_DATA_LEN: 4
SUB_INDEX: 1
TMP: 19
  
```

**Explaining the example:** The user input three arguments into the *IOL\_Read* function block: Port number, index and sub index. The IO-Link device is connected to port 1 and used his devices IODD file to figure out the correct index (67) and sub index (1). The returned value was put into the variable *READ\_DATA\_PORT1*. The returned value in this case was 0x0f (or 15 in decimal).

### 12.6.1.3 Acyclic Communication – Write

When working with acyclic communication, the first thing the user needs to do is import the IO-Link libraries (Importing libraries is discussed in chapter [5.9.3 Importing a Library](#)).

**Example of Acyclic Communication –Write:**

*(IODD file for an IO-Link Device)*

Index	Subindex	Name	Value Range	Default	Access Rights	Data Storage
67		Flashing Frequency				
67	1	Segment 1 Flashing Frequency (Hz)	0.5 through 20	1	rw	Yes

**(ARGEE Setup)**

#### Variables and Definitions

	Name	Type
1	IOL_Write	IOL_AsyncWrite ▼
2.0	INIT : [1]=0x0e	
	# of Array Elements: 100	(Clear field to disable array)
2	write_data_port1	Byte ▼

#### ARGEE Program

+ Keyboard shortcuts (hidden)

+ Task - MainTask

0 Call Help: IOL\_AsyncWrite(port\_num,index,sub\_index,wr\_data,wr\_data\_len)

IOL\_Write(1,67,1,write\_data\_port1,2)

Call ▼
Add Block

**(ARGEE Debug Screen)**

```

[-] WRITE_DATA_PORT1
  Array: length=100,elem_size=1
  [0]: 0x00
  [1]: 0x0e
  [2]: 0x00
  [3]: 0x00
[-] IOL_WRITE(IOL_ASYNCWRITE)
  [+ DS_RX_ARR
  [+ DS_TX_ARR
  CNT:      2
  IND1:     17
  INDEX:    67
  PORT_NUM: 1
  READ_RES: -8338944
  RES:      8
  SUB_INDEX: 1
  WR_DATA_LEN: 2
  
```

**Explaining the example:** The user input five arguments into the *IOL\_Write* function block: Port number, index, sub index, write data, and write data length. The user plugged their IO-Link device into port 1, and used his devices IODD file to figure out the correct index (67) and sub index (1). The user initialized *write\_data\_port1* with the value “e” in byte one. The user specified the data length to be 2 bytes. The value 0x0e (or 14 in decimal) was written to byte one.



## 12.6.2 Working with RFID

Many factors influence RFID Read/Write applications. The user can reference the RFID user manual for more information about RFID.

### 12.6.2.1.1 RFID Communication – Read

When working with RFID, the first thing the user needs to do is import the RFID libraries (Importing libraries is discussed in chapter [5.9.3 Importing a Library](#)).

#### Example of RFID Communication – Read:

##### (ARGE Setup)

0 ±	Program Variables	
	Name	Type
1	blden_rfid_s_read	BLCEN_RFIDS_Read ▼
	# of Array Elements: 64 (Clear field to disable array)	
2	read_data_port_1	Byte ▼
3	Tranciver_Power_On	Number ▼
4	temp	Number ▼
Add Variable		

1 ±	Alias Variables (hidden)	

3	Function Block Group : BLCEN_RFIDS_Routines	
4 ±	Function Block : BLCEN_RFIDS_Read	Regular ▼
(hidden)		

Task - MainTask		
0 ±	Condition Tranciver_Power_On=0	
0.0	Assignment	Destination: Tranciver_Power_On Expression: 1
0.1	Assignment	Destination: IO_Slot1_Output_XCVR_0 Expression: 1
Assignment ▼ Add Block		
1 ±	If R_TRIG(IO_Slot1_Input_TP_0,temp)	
1.0	Call	Help: BLCEN_RFIDS_Read(slot.channel.offset.res_data.num_bytes_to_read) blden_rfid_s_read(1,0,0,read_data_port_1,64)
Call ▼ Add Block		

##### (ARGE Debug Screen)

```

READ_DATA_PORT_1
Array: length=64,elem_size=1
[0]: 0x01
[1]: 0x00
[2]: 0x00
[3]: 0x00
BLCEN_RFID_S_READ(BLCEN_RFIDS_READ)
  BE_ADDR_ARR
  CHANNEL : 0
  CURR_POS : 64
  NUM_BYTES TO READ : 64
  OFFSET : 0
  RESULT : 1
  SLOT : 1
  TO_COPY : 8
  
```

**Explaining the example:** The user input five arguments into the *BLCEN\_RFIDS\_Read* function block: Slot number, channel number, bit offset, result data location, and number of bytes to read. *Condition statement 0* in the code is used to power up the transceiver. *If statement 1* says whenever the tag present bit goes true, perform one read command and store that value in *read\_data\_port\_1*.

### 12.6.2.1.2 RFID Communication – Write

When working with RFID, the first thing the user needs to do is import the RFID libraries (Importing libraries is discussed in chapter [5.9.3 Importing a Library](#)).

#### Example of *RFID Communication – Write*:

##### (ARGEE Setup)

The screenshot shows the ARGEE Setup interface. On the left, the 'Program Variables' table lists variables: `blcen_rfid_s_write` (Type: `BLCEN_RFIDS_Write`), `write_data_port_1` (Type: `Byte`), `Tranciver_Power_On` (Type: `Number`), and `temp` (Type: `Number`). Below this is the 'Alias Variables' section, which is hidden. The 'Function Block Group' is `BLCEN_RFIDS_Routines`. The 'Function Block' is `BLCEN_RFIDS_Write` (Type: `Regular`).

On the right, the 'Task - MainTask' configuration shows a 'Condition' block with the expression `Tranciver_Power_On=0`. Below this are two 'Assignment' blocks: the first assigns `Tranciver_Power_On` to `1`, and the second assigns `IO_Slot1_Output_XCVR_0` to `1`. Below these is an 'If' block with the condition `R_TRIG(IO_Slot1_Input_TP_0,temp)`. Inside the 'If' block is a 'Call' block that calls the `BLCEN_RFIDS_Write` function with arguments `(1,0,0,write_data_port_1,64)`.

##### (ARGEE Debug Screen)

The screenshot shows the ARGEE Debug Screen. It displays the state of variables and function calls. The `WRITE_DATA_PORT_1` variable is shown as an array with length 64 and element size 1. The first four elements are `[0]: 0x0e`, `[1]: 0x00`, `[2]: 0x00`, and `[3]: 0x00`. The `BLCEN_RFID_S_WRITE(BLCEN_RFIDS_WRITE)` function call is shown with the following parameters: `CHANNEL: 0`, `CURR_POS: 64`, `NUM_BYTES_TO_WRITE: 64`, `OFFSET: 0`, `RESULT: 1`, `SLOT: 1`, and `TO_COPY: 8`.

**Explaining the example:** The user input five arguments into the `BLCEN_RFIDS_Write` function block: Slot number, channel number, bit offset, output data location, and number of bytes to write. *Condition statement 0* in the code is used to power up the transceiver. *If statement 1* says, whenever the tag present bit goes true, perform one write command, and write the value in `write_data_port_1` to the tag. The value `0x0e` (or 14 in decimal) was write to byte one the tag.

### 12.6.2.1.3 RFID Communication – Strings

When working with RFID, the first thing the user needs to do is import the RFID libraries (Importing libraries is discussed in [Chapter 5.9](#)). Strings cannot be written or read from RFID tags directly. If writing, the user's string must be converted to a byte array, then written to the tag. If reading, the incoming byte array from the tag must be converted into a string by the user. These processes are shown below.

#### 12.6.2.1.3.1 Example of RFID Communication – Writing Strings

##### (ARGEE Setup)

0 ±	Program Variables	
	Name	Type
	# of Array Elements: 5 (Clear field to disable array)	
1	my_string	String
	# of Array Elements: 5 (Clear field to disable array)	
2	bytes_sent_to_tag	Byte
3	iterator	Number

##### (ARGEE Code)

0	Call	Help: STR_COPY(source_str,dest_str) str_copy("FULL", my_string)
1	Assignment	Destination: iterator Expression: 0
2 ±	While	iterator < str_len(my_string)
2.0	Assignment	Destination: bytes_sent_to_tag[iterator] Expression: my_string[iterator]
2.1	Assignment	Destination: iterator Expression: iterator + 1

**Explaining the example:** The user wants to write the string "FULL" to an RFID tag. The characters "FULL" are stored in my\_string, then my\_string is copied element-by-element to the byte array called bytes\_sent\_to\_tag. The data in bytes\_sent\_to\_tag is now ready to be written to the tag, using the appropriate Write function from the Turck RFID library (not shown).

#### 12.6.2.1.3.2 Example of RFID Communication – Reading Strings

##### (ARGEE Setup)

0 ±	Program Variables	
	Name	Type
	# of Array Elements: 5 (Clear field to disable array)	
1	my_string	String
	# of Array Elements: 5 (Clear field to disable array)	
2	bytes_read_from_tag	Byte
3	iterator	Number

##### (ARGEE Code)

0	Assignment	Destination: iterator Expression: 0
1 ±	While	iterator < 4
1.0	Assignment	Destination: my_string[iterator] Expression: bytes_read_from_tag[iterator]
1.1	Assignment	Destination: iterator Expression: iterator + 1
	Assignment	Add Block
2	Assignment	Destination: my_string[4] Expression: 0

**Explaining the example:** The user wants to store four characters read from a tag as a string. Data was read from the tag and stored in bytes\_read\_from\_tag by using the appropriate Read function from the Turck RFID library (not shown). Bytes\_read\_from\_tag is then copied element-by-element to my\_string. A zero is required at the end of my\_string, because strings are null-terminated.



#### NOTE

Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations " "

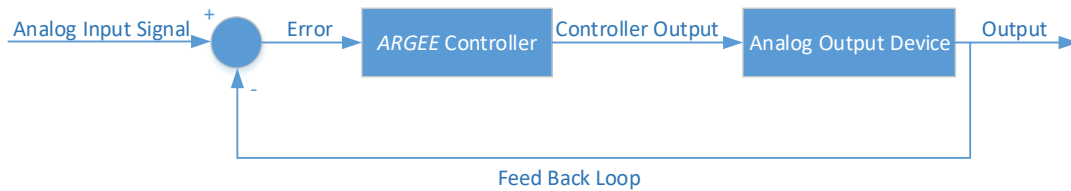
### 12.6.3 Working with Analog

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.

**Example of *Working with Analog*:**

(Customer Application)

#### Proportional Controller Example



(ARGEE Setup)

<a href="#">0</a>	Condition	<input type="text" value="true"/>
<a href="#">0.0</a>	Assignment	<div>Destination: <input type="text" value="IO_Slot2_Output_Output_value_4"/></div> <div>Expression: <input type="text" value="32767 - IO_Slot2_Input_Input_value_0"/></div>

**Explaining the example:** The user wants to make 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. Analog sensors use 16-bit signed integers. Therefore the range of the analog input signal is from -32767 to +32767. The user want's an inversely proportional controller, so they are taking  $32767 - \text{Input\_value\_0}$  and loading that value into `Output_value_4`.

## 12.7 Advanced Analog Example – Inclinometer

In this example, the user wants to use an inclinometer to track the angle of a boom, and display the angle on an HMI. If it is in a safe operation range in the X Axis, it will show a green light and display the safe operation angle on an HMI. If it is in a hazard operation angle in the Y Axis, it will sound an alarm and show the hazard angle on an HMI.

### (ARGE Setup)

0 ±	Program Variables	
	Name	Type
1	Conversion	Convert ▼
2	Status	Status_Check ▼

1 ±	Alias Variables	
	Name	IO Point
0	X_Degree_Value	IO_Slot1_Input_Input_value_2
1	Y_Degree_Value	IO_Slot1_Input_Input_value_0
2	Light	IO_Slot2_Output_Output_value_0
3	Alarm	IO_Slot2_Output_Output_value_4

3 ±	Function Block : Convert		Regular ▼
	Name	Type	Segment
0	X_Angle	Number ▼	VARIABLE ▼
1	Y_Angle	Number ▼	VARIABLE ▼
Add Element			
4 ±	Function Block : Status_Check		Regular ▼ (hidden)

+ Task - MainTask		
0	Call	Help: Convert() Conversion()
1	Call	Help: Status_Check() Status()

+ Function Block - Convert()		
0	Assignment	Destination: X_Angle Expression: ((16300- X_Degree_Value)/ 181)
1	Assignment	Destination: Y_Angle Expression: ((16300- Y_Degree_Value)/ 181)



+ Function Block - Status_Check()			
0 ±	If	(X_Degree_Value = 0)	
0.0	Assignment	Destination: Light	Expression: 1
1 ±	If	(X_Degree_Value > 0)	
1.0	Assignment	Destination: Light	Expression: 0
2 ±	If	(Y_Degree_Value < 8200)	
2.0	Assignment	Destination: Alarm	Expression: 1
3 ±	If	(Y_Degree_Value > 8200)	
3.0	Assignment	Destination: Alarm	Expression: 0

+ HMI Screens			
0 ±	HMI Screen	Inclinometer Readout	
0.0 ±	Section	Inclinometer Data	
0.0.0	Display Number With Valid Range	Title: Alarm	Variable: valid_range(Conversion.Y_Angle)
		Units: Degrees	Min Valid Value: -5
			Max Valid Value: 45
0.0.1	Display Number With Valid Range	Title: Light	Variable: valid_range(Conversion.X_Angle)
		Units: Degrees	Min Valid Value: 90
			Max Valid Value: 90

(ARGE HMI)

## Screens

Inclinometer Readout

### Inclinometer Readout Inclinometer

#### Inclinometer Data

AI Inclinometer	90 Degrees
Light	90 Degrees

**Explaining the example:** The user wrote the code to monitor the angle of the boom in both the X and Y axis. The angle of the boom in the Y axis is sounding an Alarm while the angle in the X axis is appropriate for operation.

### 12.7.1 Working with Encoders

If a user wants to use an encoder to monitor rotary positioning, and display the angle on an HMI, they no longer need a PLC. ARGEE has the ability to apply logic and math to the digital signals of an encoder.

**Example of Working with Encoders:**

(ARGEE Setup)

Program Variables		
	Name	Type
1	Position	Number
2	Degrees	Number
3	Position_fxn	position_Calc

Alias Variables		
	Name	IO Point
0	Position_Value	IO_Slot1_Input_REG_RD_DATA
1	Gate_Function	IO_Slot1_Output_Gate

Function Block : position_Calc			
	Name	Type	Segment

Add Element



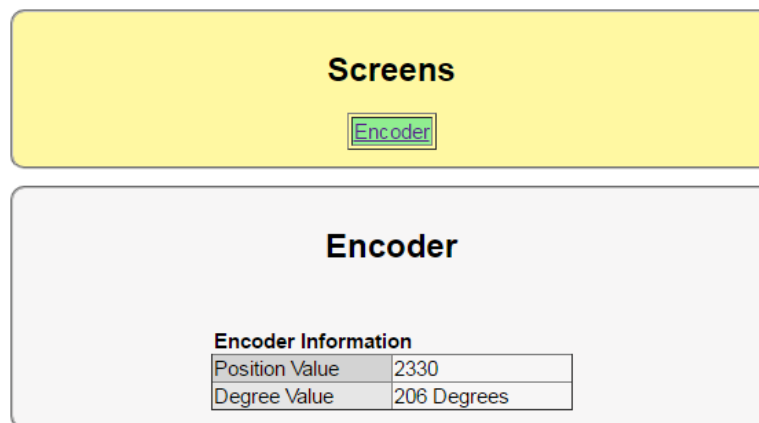
#### NOTE

The user will have to be in ARGEE PRO Advanced Mode to unlock multitasking. The position\_Calc function block will be running as a separate task.

Task - MainTask			
0	Assignment	Destination: Gate_Function	Expression: 1
1	Assignment	Destination: Position	Expression: (Position_Value - (4065*(Position_value/4065)))
2	Assignment	Destination: Degrees	Expression: (((1000*Position)/4065)*360)/10000)

+ HMI Screens			
0 ±	HMI Screen	Encoder	
0.0 ±	Section	Encoder Information	
0.0.0	Display Number/ State/String	Title: Position Value	Variable: Position
		Units:	
0.0.1	Display Number/ State/String	Title: Degree Value	Variable: Degrees
		Units: Degrees	

(ARGE HMI)



**Explaining the example:** The user is trying to get an input from a conventional incremental encoder. By normalizing the output signal, the user can display the process data and the associated angle of the encoder on an HMI.



#### NOTE

Download the device user manual at [www.turck.com](http://www.turck.com) to learn more about encoder settings.

## 12.7.2 Working with 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.

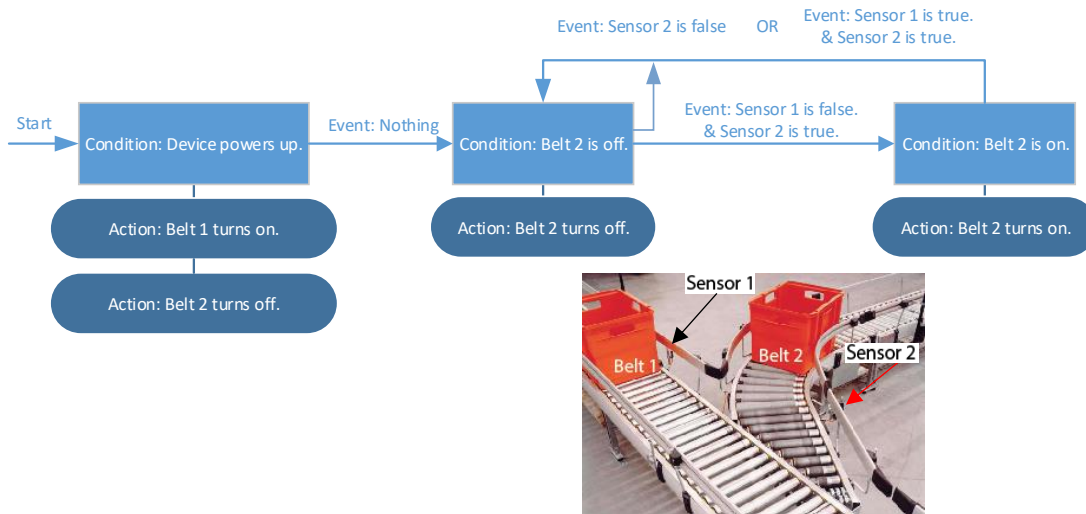
### 12.7.2.1 State Machine

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.

### 12.7.2.2 State Variables

#### Example of a State Variables:

(ARGEE Setup)

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

0 ±	Program Variables	
	Name	Type
1.0	INIT : Start_Up	
1	State	State/Enum ▼

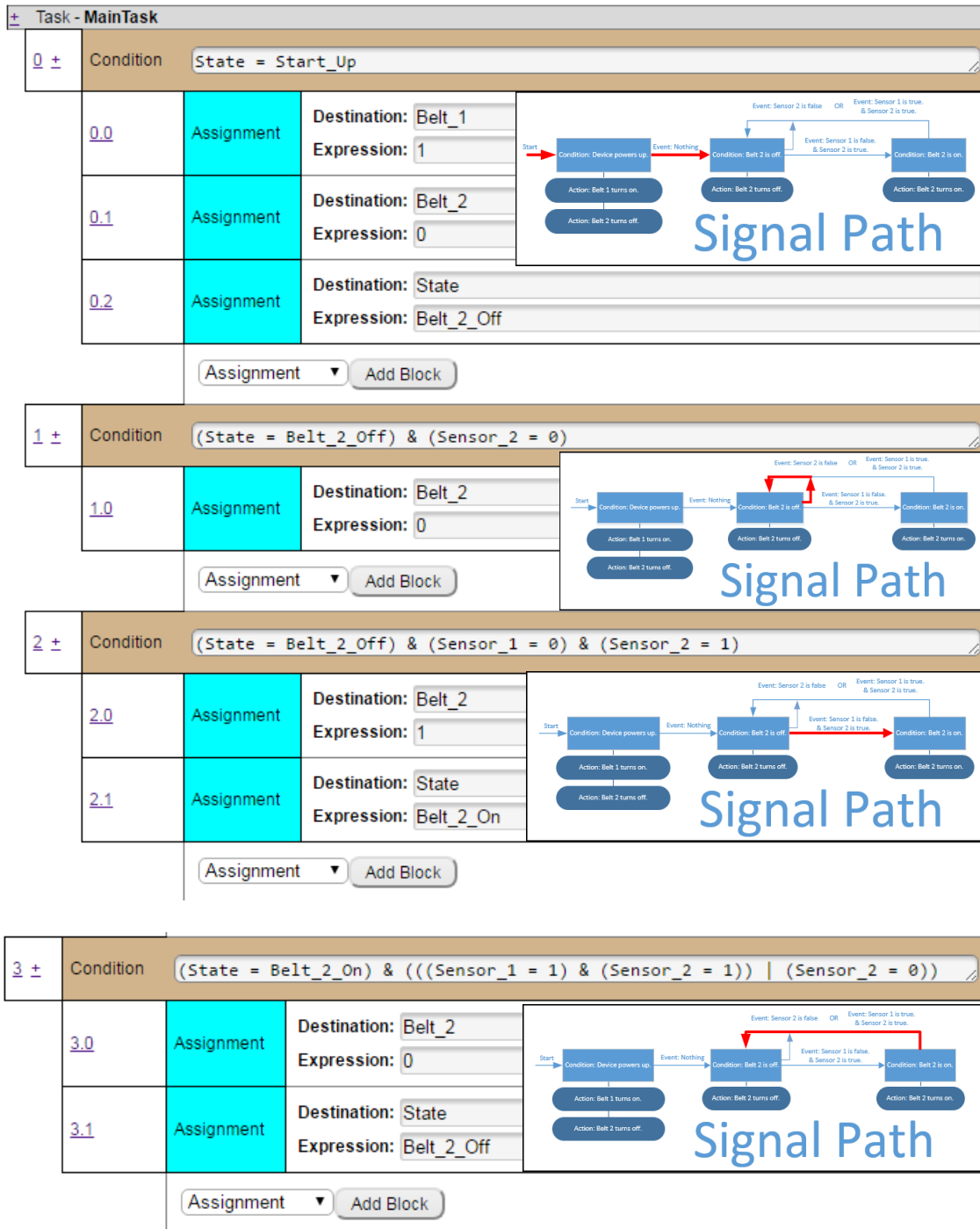
3 ±	States	
	Name	
0	Start_Up	
1	Belt_2_Off	
2	Belt_2_On	

1 ±	Alias Variables	
	Name	IO Point
0	Sensor_1	IO_Basic_Input_Input_value_0
1	Sensor_2	IO_Basic_Input_Input_value_1
2	Belt_1	IO_Basic_Output_Output_value_2
3	Belt_2	IO_Basic_Output_Output_value_3



#### NOTE

Program Variable "State" is initialized to Start-up.



**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.

**This same state machine can be written with Function Blocks and If statements:**

(ARGEE Setup)

<u>0</u> ±	Program Variables	
	Name	Type
<u>1</u>	Belt_1_On	Run_Belt_1 ▼
<u>2</u>	Belt_2_On	Run_Belt_2 ▼

<u>3</u> ±	Function Block: Run_Belt_1	Regular ▼	(hidden)
<u>4</u> ±	Function Block: Run_Belt_2	Regular ▼	(hidden)

<u>1</u> ±	Alias Variables	
	Name	IO Point
<u>0</u>	Sensor_1	IO_Basic_Input_Input_value_0
<u>1</u>	Sensor_2	IO_Basic_Input_Input_value_1
<u>2</u>	Belt_1	IO_Basic_Output_Output_value_2
<u>3</u>	Belt_2	IO_Basic_Output_Output_value_3

+ Task - MainTask			
<u>0</u> ±	Condition Sensor_2 = 0		
<u>0.0</u>	Call	Help: Run_Belt_1() Belt_1_On()	
	Call ▼	Add Block	
<u>1</u> ±	Condition (Sensor_1 = 1) & (Sensor_2 = 1)		
<u>1.0</u>	Call	Help: Run_Belt_1() Belt_1_On()	
	Call ▼	Add Block	
<u>2</u> ±	Condition (Sensor_1 = 0) & (Sensor_2 = 1)		
<u>2.0</u>	Call	Help: Run_Belt_2() Belt_2_On()	
	Call ▼	Add Block	

+ Function Block - Run_Belt_1()		
0	Assignment	Destination: <input type="text" value="Belt_1"/> Expression: <input type="text" value="1"/>
1	Assignment	Destination: <input type="text" value="Belt_2"/> Expression: <input type="text" value="0"/>
Assignment ▼ Add Block		

+ Function Block - Run_Belt_2()		
0	Assignment	Destination: <input type="text" value="Belt_1"/> Expression: <input type="text" value="0"/>
1	Assignment	Destination: <input type="text" value="Belt_2"/> Expression: <input type="text" value="1"/>
Assignment ▼ Add Block		

**Explaining the example:** If Sensor\_1 is true or false and Sensor\_2 is false, turn on Belt\_1 and turn off Belt\_2. If both sensors are true, turn on Belt\_1 and turn off Belt\_2. If Sensor\_1 is false and Sensor\_2 is true, Turn off Belt\_1 and turn on Belt\_2.

### 12.7.3 Working with User-Defined Data Types

A User-Defined Data Type (UDT) is a function block which contains variables but no code. A user would create a UDT if they were dealing with multiple objects with multiple properties.

#### Example of User-Defined Data Types:

Suppose the user has 2 cells, and each cell has 2 properties: temperature and flow. This is best illustrated as a matrix:

		COLUMNS	
		TEMPERATURE	FLOW
ROWS	CELL 1 (Element 0)	Temperature of Cell 1	Flow of Cell 1
	CELL 2 (Element 1)	Temperature of Cell 2	Flow of Cell 2

To express this in ARGEE, the user will create a Function Block with variables (the columns), and then create an array of this Function Block (the rows). No code goes into the Function Block; its only purpose is to contain variables.

#### (ARGEE Setup)

3 ±	Function Block : <u>Cell_Definition</u>		
	Name	Type	Segment
0	Temperature	Number ▼	VARIABLE ▼
1	Flow	Number ▼	VARIABLE ▼

0 ±	Program Variables	
	Name	Type
	# of Array Elements: 2	(Clear field to disable array)
1	Cell	Cell_Definition ▼

+ Function Block - Cell\_Definition()

Condition ▼ Add Block



#### NOTE

If the user wanted to add more rows to this matrix, he would increase the size of the array. If the user wanted to add more columns, he would create more Function Block variables.

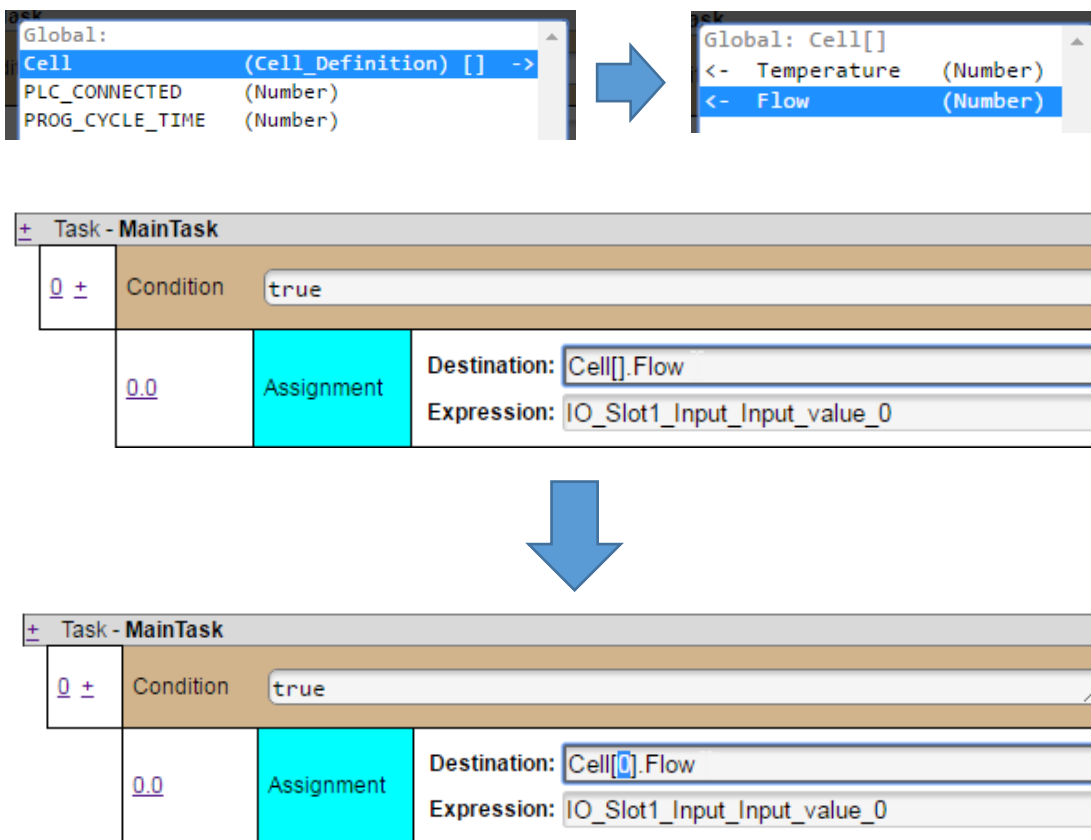
### 12.7.3.1 Referencing Internal Function Block Variables

Press Ctrl-q, select the desired variable, and then fill in the array element number (between the brackets).



#### NOTE

Cell number = element number+1, this is because array numbering starts at element 0.



**Explaining the example:** Input 0 is stored in the variable `Cell[0].Flow` (Row 1, Column 2).

### 12.7.3.2 User-Defined Data Types as Arguments to other Function Blocks

(ARGE Setup)

0 ±	Program Variables	
	Name	Type
1	convert_temp	IOL_Temp_Conversion ▼
2	instance_of_Float_UDT	Float_UDT ▼

3 ±	Function Block : Float_UDT Regular ▼		
	Name	Type	Segment
0	temperature_in_celsius	Floating ▼	VARIABLE ▼

4 ±	Function Block : IOL_Temp_Conversion Regular ▼		
	Name	Type	Segment
0	IOL_Raw_Value	Floating ▼	ARGUMENT ▼
1	storage_location	Float_UDT ▼	ARGUMENT ▼

+	Task - MainTask	
0	Call	Help: IOL_Temp_Conversion(IOL_Raw_Value,storage_location) convert_temp(2147,instance_of_Float_UDT)

+	Function Block - IOL_Temp_Conversion(IOL_Raw_Value,storage_location)	
0	Assignment	Destination: storage_location.temperature_in_celsius Expression: (((IOL_Raw_Value - 5120.0)*550.0/(60415-5120))-50)

(ARGE Debug Screen)

## Runtime Status

```

+ TRACE
PROG CYCLE TIME : 5
PLC CONNECTED : 0
- CONVERT_TEMP(IOL_TEMP_CONVERSION)
  IOL_RAW_VALUE : 2147
- MainTask
- INSTANCE_OF_FLOAT_UDT(FLOAT_UDT)
  TEMPERATURE_IN_CELSIUS : -79.5713882446289
  
```

**Explaining the example:** The user wants to convert raw data from their IO-Link temperature sensor to degrees Celsius, and store it as a variable inside a user-defined data type (UDT). They pass a raw value and the name of their UDT into their temperature conversion Function Block, which converts the value and stores the result in the *temperature\_in\_celsius* variable of the specified UDT. If the raw value is 2147, the temperature in Celsius is -79.57 degrees.

## 12.7.4 Working with Hex Values

ARGEE can easily convert any value to hex. For example: if the user types “Hex(12),” then the value “c” will be returned.

**Example of *Working with Hex Values*:**

### (ARGEE Setup)

The screenshot displays the ARGEE Setup interface. On the left, the 'Program Variables' table lists two variables: 'Submit' (Type: Number) and 'Decimal\_Value' (Type: Number). Below it is an 'Add Variable' button. The 'Alias Variables' section shows a 'Function Block' dropdown and an 'Add' button. The main area shows the 'Task - MainTask' configuration. It includes a 'Condition' block set to 'Submit' and an 'Assignment' block with 'Destination: Submit' and 'Expression: 0'. Below these are 'Add Condition' and 'Add Block' buttons. The 'HMI Screens' section shows a screen titled 'Convert Decimal to Hex' with a 'Section' named 'Converter'. The 'Converter' section contains three blocks: '0.0.0 Enter Number/String' (Title: Enter Decimal Value, Variable: Decimal\_Value), '0.0.1 Display Number/State/String' (Title: Hex Value, Variable: Hex(Decimal\_Value)), and '0.0.2 Submit Action' (Title: Submit, Variable: Submit).

### (HMI View)

The screenshot shows the HMI View of the 'Convert Decimal to Hex' screen. The screen has a title 'Convert Decimal to Hex' and a section 'Converter'. Inside the 'Converter' section, there are two input fields: 'Enter Decimal Value' with the value '2452345' and 'Hex Value' with the value '00 25 6b 79'. Below these fields is a 'Submit' button, which is highlighted by a mouse cursor.

**Explaining the example:** The user created a decimal to hex converter. If the user enters a decimal value into the *Enter Decimal Value* text box and clicks *Submit*, the hex value will show in the *Hex Value* display field.



## 12.7.5 Advanced Bitwise Operations – Bit Masking

### 12.7.5.1 What are Bitwise Operations?

A *bitwise operation* is a Boolean operation that compares variables' bits against each other, instead of comparing the variables' values. ARGEE has bitwise OR, AND, NOR, and NAND, though AND is the only operation with a practical use, which is bit masking.

### 12.7.5.2 What is Bit Masking?

Suppose you have an IO-Link laser distance sensor that outputs one word of process data; the first 15 bits are dedicated to distance data, and the last 3 bits are status bits. To use the distance as a number in your ARGEE code, you want to ignore the status bits, and just work with the distance data, represented as a 15-bit integer. That act of “covering up” unwanted bits is called bit masking.

### 12.7.5.3 Example of Bit Masking

(ARGEE Setup)

0 ±	Program Variables	
	Name	Type
1.0	INIT : 65535	
1	IOL_word_0	Number
2	distance	Number

0	Assignment	Destination: distance
		Expression: IOL_word_0 & 0x1FFF

(ARGEE Debug Screen)

### Runtime Status

+ TRACE		
PROG CYCLE TIME : 2		
PLC CONNECTED : 0		
DISTANCE :	8191	
IOL WORD 0 :	65535	

Decimal	Hex	Binary
65535	0xFFFF	1111 1111 1111 1111
8191	0x1FFF	0001 1111 1111 1111

**Explaining the example:** *IOL\_word\_0* is compared bit-by-bit against 0x1FFF. Whenever both bits of the numbers are TRUE, a 1 is assigned to that bit position in *distance*. If either bit is FALSE, a 0 is assigned to that bit position in *distance*. The result is that the last 3 bits of *IOL\_word\_0* are ignored.

## 12.7.6 Nesting Function Blocks

ARGEE 3 has the capability to nest Function Blocks. The user will nest Function Blocks if the user wants a function block to call another function block.

Run
 Debug
 Print
 IO Config
 HMI
 Project
 Set Title
 About

Project Title: TBEN-S1-8DXP (192.168.1.12) V3.2.3.5

### Variables and Definitions

0 ±

Program Variables		
	Name	Type
1	Main_Function	Function_Block_1 ▾

Add Variable

1 ±

Alias Variables (hidden)

3 ±

Function Block : Function\_Block\_1

Regular ▾

	Name	Type	Segment
0	Function_2	Function_Block_2 ▾	VARIABLE ▾

Add Element

4 ±

Function Block : Function\_Block\_2

Regular ▾

	Name	Type	Segment
--	------	------	---------

Add Element

Function Block ▾

Add

### ARGEE Program

±

Keyboard shortcuts (hidden)

±

Task - MainTask

0

Call

Help: Function\_Block\_1()

Main\_Function()

Call ▾

Add Block

±

Function Block - Function\_Block\_1()

0

Call

Help: Function\_Block\_2()

Function\_2()

Call ▾

Add Block

±

Function Block - Function\_Block\_2()

0

Condition

true

0.0

Assignment

Destination: IO\_Basic\_Input\_Input\_value\_0

Expression: IO\_Basic\_Input\_Input\_value\_0

**Explaining the example:** The MainTask calls *Main\_Function* which is *Function\_Block\_1*. *Function\_Block\_1* then calls *Function\_Block\_2*.



#### NOTE

To get a list of Local Variables for the Function Block, press Ctrl-L.

### 12.7.7 Advanced HMI Example – Tank monitoring with graphics

ARGEE 3 allows the user to code an HMI with static images and multi state graphics that respond to your code. The user is trying to monitor a tank with an ultrasonic sensor. The user then wants to display the status of the tank level on an HMI with representative pictures and a status color of each level.

#### (ARGEE Setup)

0 ±	Program Variables	
	Name	Type
0	Tank_Monitoring	Tank_Status ▾

1 ±	Alias Variables	
	Name	IO Point
0	Tank_Sensor	IO_Slot1_Input_Input_data_word_0

4 ±	States
	<b>Name</b>
0	High
1	Crit_High
2	OK
3	Low
4	Crit_Low

<u>3</u> ±		Function Block : Tank_Status		Regular ▾
	Name	Type	Segment	
<u>0</u>	Tank_State	State/Enum ▾	VARIABLE ▾	

+ Task - MainTask		
0	Call	Help: Tank_Status() Tank_Monitoring()



+ Function Block - Tank_Status()			
0 ±	If	Tank_Sensor < 1900 & (1401 < Tank_Sensor)	
	0.0	Assignment	Destination: Tank_State Expression: Crit_Low
1 ±	If	Tank_Sensor < 1400 & (1001 < Tank_Sensor)	
	1.0	Assignment	Destination: Tank_State Expression: Low
2 ±	If	Tank_Sensor < 1000 & (801 < Tank_Sensor)	
	2.0	Assignment	Destination: Tank_State Expression: OK
3 ±	If	Tank_Sensor < 800 & (401 < Tank_Sensor)	
	3.0	Assignment	Destination: Tank_State Expression: High
4 ±	If	Tank_Sensor < 400 & (0 < Tank_Sensor)	
	4.0	Assignment	Destination: Tank_State Expression: Crit_High









## NOTE

Your sensor range might be different, or need to be “taught” its range. Look at the user manual for your sensors on [www.turck.com](http://www.turck.com)

### Now let’s configure the HMI:

The plan is to place a logo in the top right corner and then have a central column with images that display tank level with a color based status background. Below this image we will display the tank state with the same color based status background. It’s a good idea to sketch out what you’re trying to accomplish so that you can code against a design. See Chapter 9: *ARGEER HMI* for more details.

First, we’ll add an HMI Image Group and upload our images.

HMI Screens		
0 ±	HMI Image Group	
0.0	HMI Image	VarName:Turck Logo 
0.1	HMI Image	VarName:Tank Crit_High 
0.2	HMI Image	VarName:Tank Crit_Low 
0.3	HMI Image	VarName:Tank High 
0.4	HMI Image	VarName:Tank Low 
0.5	HMI Image	VarName:Tank OK 



#### NOTE

The user can upload any image. Keep file size below 20kb.

Now let's add a grid screen.

HMI Grid Screen		<b>Help:</b> SCREEN_PROP(Title,width_in_percent_of_screen,enable_rounded_edges,background_color) SCREEN_PROP("Tank Status",90,true,"#F8F8F8")
2.0 ±	Grid Row	<b>Help:</b> ROW_PROP(background_color) ROW_PROP("transparent")
2.0.0 ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)
2.0.0.0	Grid Element	<b>Help:</b> STATIC_GRAPHICS(image_file_variable,background_color,default_zoom) STATIC_GRAPHICS("Turck Logo","transparent",100)
<input type="button" value="Add Element"/>		
2.0.1 ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(4,0)

Our first row is just to display the logo in the top left of the screen. We added two *Grid Cells* one that spans 1/5 of the screen and the other 4/5 screen. We then added a *Grid Element* to the first *Grid Cell* and used the STATIC\_GRAPHICS function to place our logo, using its variable name.



#### NOTE

HMI functions are available by hitting Ctrl-f, and image file variables are available by hitting Ctrl-i.

Our second row is empty; it will be used as a spacer.

2.1 ±	Grid Row	<b>Help:</b> ROW_PROP(background_color) ROW_PROP("transparent")
2.1.0 ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)

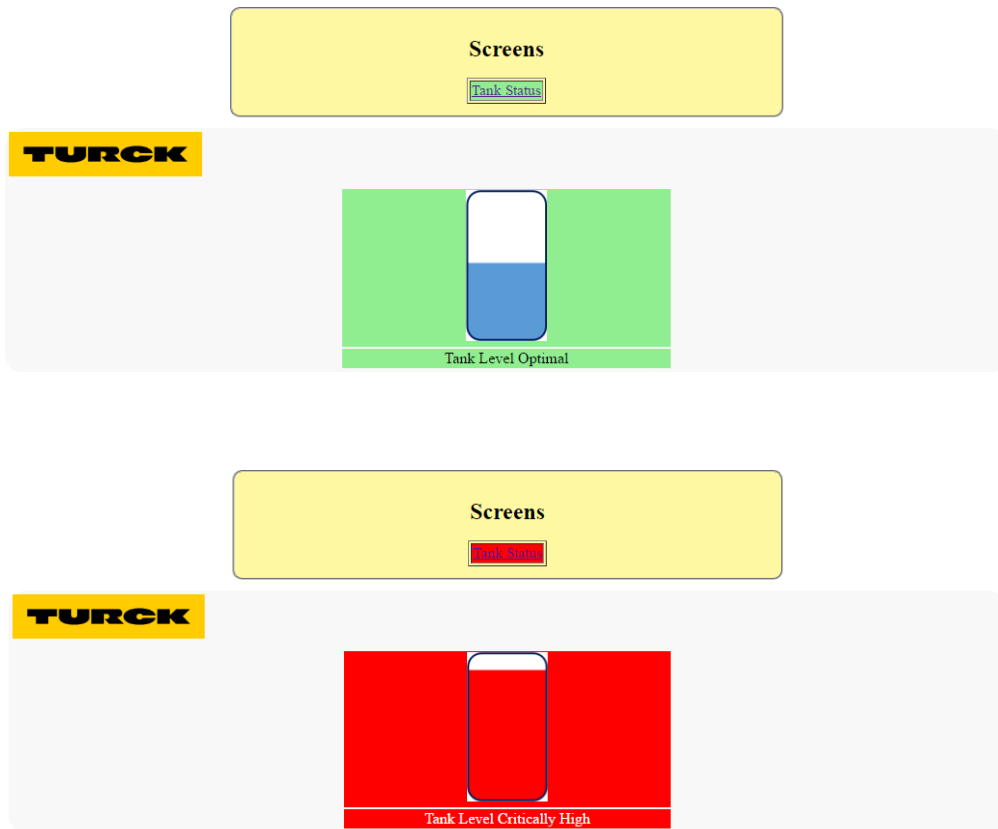
Our third row will have three cells, and the middle cell will have the *Grid Elements* Multi State Display Graphics and Multi State Display Strings. Both of these functions will use different background colors for each state.

<a href="#">2.2</a> ±	Grid Row	<b>Help:</b> ROW_PROP(background_color) ROW_PROP("transparent")	
<a href="#">2.2.0</a> ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)	
		Add Element	
<a href="#">2.2.1</a> ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)	
<a href="#">2.2.1.0</a>	Grid Element	<b>Help:</b> MULTI_STATE_DISPLAY_GRAPHICS(Title,var,title_size,title_color,title_background_color,image_zoom_level,value1,...) MULTI_STATE_DISPLAY_GRAPHICS("",Tank_Monitoring.Tank_State,3,"black","transparent",25, Crit_High,"Tank Crit_High", "red", High, "Tank High", "orange", OK, "Tank OK", "lightgreen", Low, "Tank Low", "orange", Crit_Low, "Tank Crit_Low", "red")	
<a href="#">2.2.1.1</a>	Grid Element	<b>Help:</b> MULTI_STATE_DISPLAY_STRING(Title,var,size,title_color,title_background_color,value1,...) MULTI_STATE_DISPLAY_STRING("",Tank_Monitoring.Tank_State,3,"black","transparent", Crit_High, "Tank Level Critically High", "white", "red", High, "Tank Level High", "black", "orange", OK, "Tank Level Optimal", "black", "lightgreen", Low, "Tank Level Low", "black", "orange", Crit_Low, "Tank Level Critically Low", "white", "red")	
		Add Element	
<a href="#">2.2.2</a> ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)	

The last row will be empty and used as a spacer.

<a href="#">2.3</a> ±	Grid Row	<b>Help:</b> ROW_PROP(background_color) ROW_PROP("transparent")	
<a href="#">2.3.0</a> ±	Grid Cell	<b>Help:</b> CELL_PROP(column_span,border_style) CELL_PROP(1,0)	

(HMI View)



**Explaining the example:** The user wrote some code to monitor the tank level, and then configured an HMI using custom graphics to display the state of the tank level. As the tank level changes, the HMI changes in response to the tank state.

## 13 Appendix III – Libraries

### 13.1 MISC

Import the MISC library.

#### 13.1.1 MISC\_wait\_ms

**Function:** When MISC\_wait\_ms is called it halts the task execution for the designated amount of time.

The imported MISC\_wait\_ms function block should look like the image below.

1 ±	Function Block : MISC_wait_ms Regular ▼		
	Name	Type	Segment
0	wait_time_in_ms	Number ▼	ARGUMENT ▼
1	timer	Timer/Counter ▼	VARIABLE ▼

**Program Variables:** A MISC\_wait\_ms program variable is needed to call the function.

0 ±	Program Variables	
	Name	Type
1	Wait	MISC_wait_ms ▼
2	Time	Number ▼

**How to Call:** The Call needs a wait time in ms argument. This can be a static number or a number program variable.

±	Task - MainTask	
0	Call	Help: MISC_wait_ms(wait_time_in_ms) Wait(Time)

#### 13.1.2 MISC\_array\_to\_string

**Function:** When MISC\_array\_to\_string is called the input array will be written into the output string for as many bytes that have been designated in the argument of the call.

The imported MISC\_array\_to\_string has Byte, String, and Number arguments.

2 ±	Function Block : MISC_array_to_string Regular ▼		
	Name	Type	Segment
	# of Array Elements: arr_arg (Clear field to disable array)		
0	array_input	Byte ▼	ARGUMENT ▼
	# of Array Elements: arr_arg (Clear field to disable array)		
1	string_output	String ▼	ARGUMENT ▼
2	number_of_bytes	Number ▼	ARGUMENT ▼
3	i	Number ▼	VARIABLE ▼



**Program Variables:** To call MISC\_array\_to\_string a MISC\_array\_to\_string variable, a string array variable, a byte variable, and a number variable are needed.

0 ±	Program Variables	
	Name	Type
1	array_to_string	MISC_array_to_string ▼
	# of Array Elements: 32 (Clear field to disable array)	
2	Input_array	Byte ▼
	# of Array Elements: 32 (Clear field to disable array)	
3	output_string	String ▼

**How to Call:** The call needs a number variable that is the array being input, a string variable that will hold the outputted string, and then a number that is the amount of bytes the array is long.

+ Task - MainTask		
0	Call	Help: MISC_array_to_string(array_input,string_output,number_of_bytes) array_to_string(Input_array,output_string,32)

### 13.1.3 MISC\_sort

**Function:** When MISC\_sort is called the output number array is filled with the data of the input array in order of increasing value, with length denoted by number\_of\_elements.

The sort function block has 2 number arrays (one for the input and one for the output), and a number that represents the length of the input array as arguments.

3 ±	Function Block : MISC_sort		Regular ▼
	Name	Type	Segment
	# of Array Elements: arr_arg (Clear field to disable array)		
0	array_input	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg (Clear field to disable array)		
1	array_output	Number ▼	ARGUMENT ▼
2	number_of_elements	Number ▼	ARGUMENT ▼
3	min_pos	Number ▼	VARIABLE ▼
4	i	Number ▼	VARIABLE ▼
5	j	Number ▼	VARIABLE ▼
6	tmp	Number ▼	VARIABLE ▼

**Program Variables:** To call sort a MISC\_sort variable, an input number array that holds the values that are being sorted, an output number array that will hold the sorted array, and the length of the input array are needed.

0 ±	Program Variables	
	Name	Type
1	Sort_	MISC_sort ▼
	# of Array Elements: 32 (Clear field to disable array)	
2	Input_array	Number ▼
	# of Array Elements: 32 (Clear field to disable array)	
3	output_array	Number ▼

**How to Call:** The call needs a number array for the input, a number array to hold the output, and number to represent the length of the input array.

+ Task - MainTask		
0	Call	Help: MISC_sort(array_input,array_output,number_of_elements) Sort_(Input_array,output_array,32)

#### 13.1.4 MISC\_filter\_sample\_into\_array

**Function:** When MISC\_filter\_sample\_into\_array is called it puts the current input sample value into the sequential\_array and the filtered\_array. The sequential\_array is an array that holds the input sample values in the order they were input, and the filtered\_array is an array that holds the input sample values in order of increasing size. filtered\_array and sequential\_array have lengths of 5, and all data input after the 5<sup>th</sup> will overwrite the first data values stored.

*\*To change the array length the user will need to alter the code in the function block change the 5s highlighted below to the desired length, and change the # of Array Elements of filtered\_arr and seq\_arr in the Function Blocks variables to the desired length.*

1	Assignment	Destination: curr_elem Expression: (curr_elem+1)%5
2	If	max_elem 5

The MISC\_filter\_sample\_into\_array function block only has a single number argument.

4	Function Block: MISC_filter_sample_into_ Regular		
	Name	Type	Segment
0	sample	Number	ARGUMENT
1	sort	MISC_sort	VARIABLE
	# of Array Elements: 5	(Clear field to disable array)	
2	filtered_array	Number	VARIABLE
	# of Array Elements: 5	(Clear field to disable array)	
3	sequential_array	Number	VARIABLE
4	max_elem	Number	VARIABLE
5	curr_elem	Number	VARIABLE
6	ind	Number	VARIABLE
7	filtered_sample	Number	VARIABLE

**Program Variables:** The needed variables to filter sample are a MISC\_filter\_sample\_into\_array variable to call, and a number variable to represent the sample values being input.

0	Program Variables	
	Name	Type
1	Filter	filter_sample
2	Samp	Number

**How to Call:** For MISC\_filter\_sample\_into\_array to be called successfully it cannot be continuously called, so it needs to be in a condition block. If it is called continuously the filtered array values will be filled with repeats of the current sample value, and not populated with 5 unique sample values.

+ Task - MainTask			
0 ±	Condition		
0.0	Call	Help: filter_sample(sample) Filter(Samp)	

### 13.1.5 MISC\_reset\_filter

**Function:** The MISC\_reset\_filter when called resets the MISC\_filter\_sample\_into\_array so that the next sample value is put into the arrays first data slot. It does not clear the MISC\_filter\_sample\_into\_array arrays, just resets where the next sample data goes in the array to the beginning.

The function block should look as it does bellow.

5 ±	Function Block : MISC_reset_filter		Regular ▼
	Name	Type	Segment
0	filter_being_reset	MISC_filter_sample_into_array ▼	ARGUMENT ▼

**Program Variables:** To call MISC\_reset\_filter a MISC\_reset\_filter variable is needed, and a MISC\_filter\_sample\_into\_array variable is needed for the argument.

0 ±	Program Variables	
	Name	Type
1	Reset	MISC_reset_filter ▼
2	Filter	MISC_filter_sample_into_array ▼

**How to Call:** When calling MISC\_reset\_filter the argument needs to be a MISC\_filter\_sample\_into\_array variable.

+ Task - MainTask			
0	Call	Help: MISC_reset_filter(filter_being_reset) Reset(Filter)	

### 13.1.6 MISC\_NUMBER\_st

**Function:** The function of MISC\_NUMBER\_st is to pass a number to a function block.

The function block should look as it does bellow, with no arguments.

7 ±	Function Block : MISC_NUMBER_st		Regular ▼
	Name	Type	Segment
0	number	Number ▼	VARIABLE ▼

**Program Variables:** The only variable needed is a MISC\_NUMBER\_st variable.

0 ±	Program Variables	
	Name	Type
1	My_number	MISC_NUMBER_st ▼

**How to Call:** This function block is not really called instead a number is assigned to the variable in the function block as shown below.

+ Task - MainTask		
0	Assignment	Destination: My_number.number
		Expression: 100

### 13.1.7 MISC\_copy\_byte\_to\_array

**Function:** MISC\_copy\_byte\_to\_array copies the data from a source byte array to a destination byte array, and the data from the source and to the destination can both be offset.

The Function Block has source and destination Byte array arguments, source and destination offset number arguments, and an array length number argument.

7 ± Function Block: MISC_copy_byte_to_array (Regular ▼)			
	Name	Type	Segment
0	# of Array Elements: arr_arg (Clear field to disable array)		
	source	Byte ▼	ARGUMENT ▼
1	# of Array Elements: arr_arg (Clear field to disable array)		
	destination	Byte ▼	ARGUMENT ▼
2	offset_source	Number ▼	ARGUMENT ▼
3	offset_destination	Number ▼	ARGUMENT ▼
4	length	Number ▼	ARGUMENT ▼
5	ind_src	Number ▼	VARIABLE ▼
6	ind_dst	Number ▼	VARIABLE ▼

**Program Variables:** The program variables needed are the MISC\_copy\_byte\_to\_array to call, the length of the arrays (in this case 32), and source and destination arrays. Offset values are also used but they do not need to be variables.

0 ± Program Variables		
	Name	Type
1	Copy	MISC_copy_byte_to_array ▼
2	# of Array Elements: 32 (Clear field to disable array)	
	src	Byte ▼
3	# of Array Elements: 32 (Clear field to disable array)	
	dst	Byte ▼

**How to Call:** To call MISC\_copy\_byte\_to\_array the user needs the source array, the destination array, a source offset number, a destination offset number, and number to represent the length of the arrays.

+ Task - MainTask		
0	Call	Help: MISC_copy_byte_to_array(sorce,destination,offset_sorce,offset_destination,length) Copy(src,dst,0,0,32)

### 13.1.8 Float\_to\_String

**Function:** Float\_to\_String takes an input float value and puts it into a string.

The Function Block should look as it does bellow.

1 ±	Function Block : float_to_string		
	Name	Type	Segment
0	Input_float	Floating ▼	ARGUMENT ▼
1	Number_of_Decimal_Positions	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg (Clear field to disable array)		
2	Output_string	String ▼	ARGUMENT ▼
3	flt1	Floating ▼	VARIABLE ▼
4	int1	Number ▼	VARIABLE ▼
5	flt2	Floating ▼	VARIABLE ▼
6	int2	Number ▼	VARIABLE ▼
7	pwr	Number ▼	VARIABLE ▼
8	i	Number ▼	VARIABLE ▼
	# of Array Elements: 10 (Clear field to disable array)		
9	temp_string	String ▼	VARIABLE ▼
10	negative_num	Number ▼	VARIABLE ▼
11	conv_arg_float	Floating ▼	VARIABLE ▼
Add Element			

**Program Variables:** To Call Float\_to\_String a Float\_to\_String variable is needed, a float variable, a string, and a number variable.

0 ±	Program Variables	
	Name	Type
1	Float_String	float_to_string ▼
2	Float_	Floating ▼
	# of Array Elements: 32 (Clear field to disable array)	
3	String_	String ▼
4	Decimal_Positions	Number ▼

**How to Call:** To call Float\_to\_String the following arguments must be satisfied; a float variable that holds the float being input, the number of decimal places the float variable has, and the string that the is being output with the value of the float variable.

Task - MainTask		
0	Call	Help: float_to_string(Input_float,Number_of_Decimal_Positions,Output_string) Float_String(Float_ ,Decimal_Positions,String_)

## 13.2 Technology

### 13.2.1 BLCEN\_RFIDS\_Routines

For BLCEN-RFIDS devices to read or write the transceiver needs to be turned on. This is done as shown below.

Task - MainTask			
0	Assignment	Destination:	IO_Slot1_Output_XCVR_0
		Expression:	1

### 13.2.2 BLCEN\_RFIDS\_Read

**Function:** BLCEN\_RFIDS\_Read when called waits for the next tag to be presented to read, and that data is held in the input read data.

The Function Block should look as it does bellow.

1 ±	Function Block : BLCEN_RFIDS_Read Regular ▼		
	Name	Type	Segment
0	slot	Number ▼	ARGUMENT ▼
1	channel	Number ▼	ARGUMENT ▼
2	offset	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg (Clear field to disable array)		
3	res_data	Byte ▼	ARGUMENT ▼
4	num_bytes_to_read	Number ▼	ARGUMENT ▼
5	curr_pos	Number ▼	VARIABLE ▼
	# of Array Elements: 2 (Clear field to disable array)		
6	be_addr_arr	Byte ▼	VARIABLE ▼
7	to_copy	Number ▼	VARIABLE ▼
8	result	Number ▼	VARIABLE ▼

**Program Variables:** To call BLCEN\_RFIDS\_Read a BLCEN\_RFIDS\_Read variable, and a byte array are needed.

0 ±	Program Variables	
	Name	Type
1	Read	BLCEN_RFIDS_Read ▼
	# of Array Elements: 8 (Clear field to disable array)	
2	Reset_data	Byte ▼

**How to Call:** When calling BLCEN\_RFIDS\_Read the following arguments need to be fulfilled; what slot of the BLCEN has the 2RFID channels, which channel is being used, how much the data being read should be offset, the reset data byte array, and the number of bytes that are being read from the tag.

Task - MainTask			
0	Call	Help:	BLCEN_RFIDS_Read(slot,channel,offset,res_data,num_bytes_to_read)
			Read(1,0,0,Reset_data,8)

### 13.2.3 BLCEN\_RFIDS\_Write

**Function:** When BLCEN\_RFIDS\_Write is called the data from an outp\_data is written onto the next tag that is put into the transvers field.

The Function Block should look as it does bellow.

2 ±	Function Block : BLCEN_RFIDS_Write Regular ▼		
	Name	Type	Segment
0	slot	Number ▼	ARGUMENT ▼
1	channel	Number ▼	ARGUMENT ▼
2	offset	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg (Clear field to disable array)		
3	outp_data	Byte ▼	ARGUMENT ▼
4	num_bytes_to_write	Number ▼	ARGUMENT ▼
5	curr_pos	Number ▼	VARIABLE ▼
	# of Array Elements: 2 (Clear field to disable array)		
6	be_addr_arr	Byte ▼	VARIABLE ▼
7	to_copy	Number ▼	VARIABLE ▼
8	result	Number ▼	VARIABLE ▼

**Program Variables:** To call BLCEN\_RFIDS\_Write a BLCEN\_RFIDS\_Write variable is needed, and a Byte array that holds the data that is being written is needed.

0 ±	Program Variables	
	Name	Type
1	Write	BLCEN_RFIDS_Write ▼
	# of Array Elements: 8 (Clear field to disable array)	
2	Write_Data	Byte ▼

**How to Call:** The arguments needed to call BLCEN\_RFIDS\_Write are, what slot of the BLCEN has the 2RFID channels, which channel is being used, how much the data being written should be offset onto the tag, the data array that is being written onto the tag, and the number of bytes that are being written onto the tag.

± Task - MainTask		
0	Call	<b>Help:</b> BLCEN_RFIDS_Write(slot,channel,offset,outp_data,num_bytes_to_write) Write(1,0,0,Write_Data,8)

### 13.2.4 TBEN\_S2\_RFID\_READ

**Function:** TBEN\_S2\_RFIDS\_READ when called waits for the next tag to be presented and reads it, and that data is held in the input read data.

The Function Block should look as it does bellow.

1 ±	Function Block : <b>TBEN_S2_RFID_READ</b> Regular ▼		
	Name	Type	Segment
0	channel	Number ▼	ARGUMENT ▼
1	offset	Number ▼	ARGUMENT ▼
2	length	Number ▼	ARGUMENT ▼
	# of Array Elements: <b>arr_arg</b> (Clear field to disable array)		
3	output_array	Byte ▼	ARGUMENT ▼
4	array_offset	Number ▼	ARGUMENT ▼
5	array_offR	Number ▼	VARIABLE ▼
6	offR	Number ▼	VARIABLE ▼
7	lenR	Number ▼	VARIABLE ▼
8	ctrl_slot	Number ▼	VARIABLE ▼
9	input_slot	Number ▼	VARIABLE ▼
10	lenI	Number ▼	VARIABLE ▼

**Program Variables:** To call TBEN\_S2\_RFIDS\_READ a TBEN\_S2\_RFIDS\_READ variable, and a byte array are needed.

0 ±	Program Variables	
	Name	Type
1	Read	TBEN_S2_RFID_READ ▼
	# of Array Elements: <b>32</b> (Clear field to disable array)	
2	Reset	Byte ▼

**How to Call:** When calling TBEN\_S2\_RFIDS\_READ the following arguments need to be fulfilled; which channel is being used, how much the data being read should be offset, the number of bytes that are being read from the tag, the reset data byte array, and how much the array data should be offset.

+ Task - MainTask		
0	Call	Help: TBEN_S2_RFID_READ(channel,offset,length,output_array,array_offset) Read(1,0,8,Reset,0)



### 13.2.5 TBEN\_S2\_RFID\_WRITE

**Function:** The function of the TBEN\_S2\_RFID\_WRITE when called writes the data from a byte array is written onto the next tag that is presented into the transceiver's field.

The Function Block should look as it does bellow.

2 ±	Function Block : TBEN_S2_RFID_WRITE (Regular ▼)		
	Name	Type	Segment
0	channel	Number ▼	ARGUMENT ▼
1	offset	Number ▼	ARGUMENT ▼
2	length	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg (Clear field to disable array)		
3	source_array	Byte ▼	ARGUMENT ▼
4	array_offset	Number ▼	ARGUMENT ▼
5	array_offW	Number ▼	VARIABLE ▼
6	offW	Number ▼	VARIABLE ▼
7	lenW	Number ▼	VARIABLE ▼
8	ctrl_slot	Number ▼	VARIABLE ▼
9	output_slot	Number ▼	VARIABLE ▼
10	lenI	Number ▼	VARIABLE ▼

**Program Variables:** To call TBEN\_S2\_RFID\_WRITE a TBEN\_S2\_RFID\_WRITE variable is needed, and a Byte array that holds the data that is being written is needed.

0 ±	Program Variables	
	Name	Type
1	Write	TBEN_S2_RFID_WRITE ▼
	# of Array Elements: 32 (Clear field to disable array)	
2	Write_Data	Byte ▼

**How to Call:** The arguments needed to call TBEN\_S2\_RFID\_WRITE are, which channel is being used, how much the data being written should be offset onto the tag, the length of the array being written onto the tag, the data array that is being written onto the tag, and how much the array data being written should be offset.

±	Task - MainTask	
0	Call	Help: TBEN_S2_RFID_WRITE(channel,offset,length,source_array,array_offset) Write(1,0,8,Write_Data,0)

### 13.2.6 TBEN\_IOL\_AsyncRead

**Function:** When TBEN\_IOL\_AsyncRead is called the parameter data from a chosen index and sub index is read into the ds\_tx\_array and ds\_rx\_array.

The function block should look as it does bellow.

2 ±	Function Block : <b>TBEN_IOL_AsyncRead</b> Regular ▼		
	Name	Type	Segment
0	port_num	Number ▼	ARGUMENT ▼
1	index	Number ▼	ARGUMENT ▼
2	sub_index	Number ▼	ARGUMENT ▼
	# of Array Elements: <b>arr_arg</b> (Clear field to disable array)		
3	reset_data	Byte ▼	ARGUMENT ▼
4	reset_data_length	Number ▼	VARIABLE ▼
	# of Array Elements: <b>32</b> (Clear field to disable array)		
5	ds_tx_array	Byte ▼	VARIABLE ▼
	# of Array Elements: <b>32</b> (Clear field to disable array)		
6	ds_rx_array	Byte ▼	VARIABLE ▼
7	reset	Number ▼	VARIABLE ▼
8	read_reset	Number ▼	VARIABLE ▼
9	tmp	Number ▼	VARIABLE ▼

**Program Variables:** The variables needed to call TBEN\_IOL\_AsyncRead are a TBEN\_IOL\_AsyncRead variable, and a byte array variable.

0 ±	Program Variables	
	Name	Type
1	Read	TBEN_IOL_AsyncRead ▼
	# of Array Elements: <b>8</b> (Clear field to disable array)	
2	Reset_	Byte ▼

**How to Call:** To call TBEN\_IOL\_AsyncRead the following arguments need to be filled; the port that is being used, the parameter index that the user is trying to read, the sub index that the user is trying to read, and a reset byte array.

+ Task - MainTask		
0	Call	Help: TBEN_IOL_AsyncRead(port_num,index,sub_index,reset_data) Read(1,20,0,Reset_)

### 13.2.7 TBEN\_IOL\_AsyncWrite

**Function:** When TBEN\_IOL\_AsyncWrite is called the data from a byte array is written into a chosen index and sub index.

The function block should look as it does bellow.

3 ±	Function Block : <b>TBEN_IOL_AsyncWrite</b> Regular ▼		
	Name	Type	Segment
0	port_num	Number ▼	ARGUMENT ▼
1	index	Number ▼	ARGUMENT ▼
2	sub_index	Number ▼	ARGUMENT ▼
	# of Array Elements: <b>arr_arg</b> (Clear field to disable array)		
3	write_data	Byte ▼	ARGUMENT ▼
4	write_data_length	Number ▼	ARGUMENT ▼
5	index1	Number ▼	VARIABLE ▼
	# of Array Elements: <b>32</b> (Clear field to disable array)		
6	ds_tx_array	Byte ▼	VARIABLE ▼
	# of Array Elements: <b>32</b> (Clear field to disable array)		
7	ds_rx_array	Byte ▼	VARIABLE ▼
8	reset	Number ▼	VARIABLE ▼
9	read_reset	Number ▼	VARIABLE ▼
10	cnt	Number ▼	VARIABLE ▼

**Program Variables:** The only program variables needed are a TBEN\_IOL\_AsyncWrite variable, and a byte array variable.

0 ±	Program Variables	
	Name	Type
1	Write	TBEN_IOL_AsyncWrite ▼
	# of Array Elements: <b>8</b> (Clear field to disable array)	
2	Write_	Byte ▼

**How to Call:** To call TBEN\_IOL\_AsyncWrite the following arguments need to be satisfied, the port that is being used, the parameter index that the user is trying to write into, the sub index that the user is trying to write into, the byte array that is being written, and the length of the array being written.

+ Task - MainTask		
0	Call	<b>Help:</b> TBEN_IOL_AsyncWrite(port_num,index,sub_index,write_data,write_data_length) Write(1,20,0,Write_,8)

TURCK sells its products through Authorized Distributors. These distributors provide our customers with technical support, service and local stock. TURCK distributors are located nationwide – Including all major metropolitan marketing areas  
For Application Assistance or for the location of your nearest TURCK distributor, call:  
**1-800-544-7769**

Specifications in this manual are subject to change without notice. TURCK also reserves the right to make modifications and makes no guarantee of the accuracy of the information contained herein.