Your Global Automation Partner

TURCK

ARGEE 3
Reference Manual

MA3000
0521B

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.

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

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**.

1.5 Technical support

For additional support, email inquiries to 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

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?

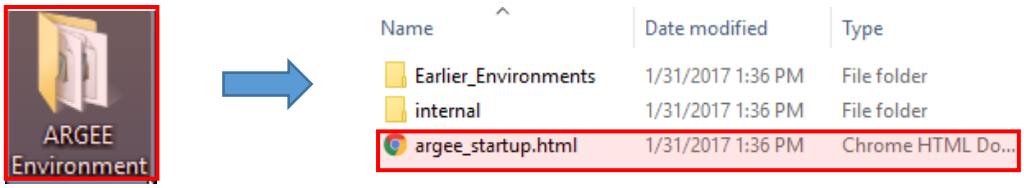
This manual is a reference guide for the ARGEE Programing Environment. This manual:

- 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.

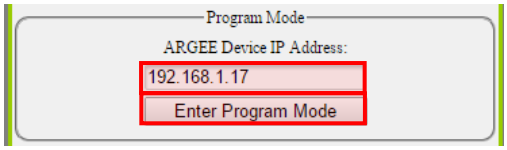


NOTE

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

3.2 Logging into the Program Mode

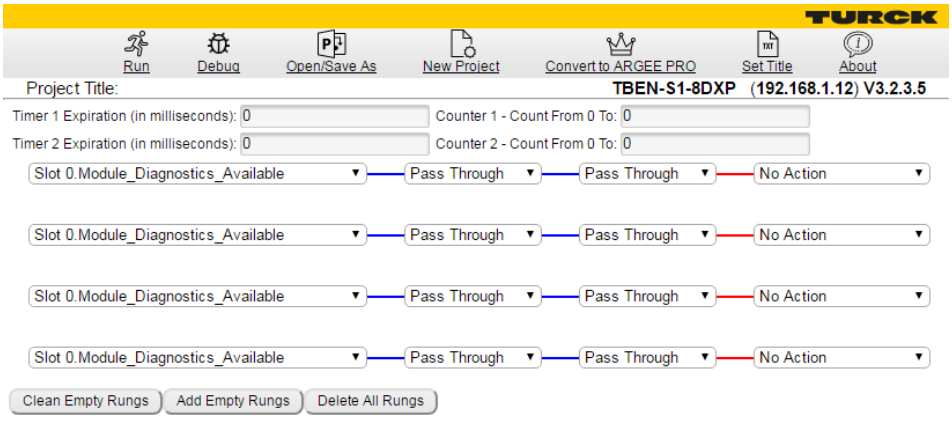
➢ Type **your** device's IP Address into the ARGEE Device IP Address text box, and then click Enter Program Mode.



NOTE

Simulation Mode is explained in chapter 8 ARGEE Simulation Mode.

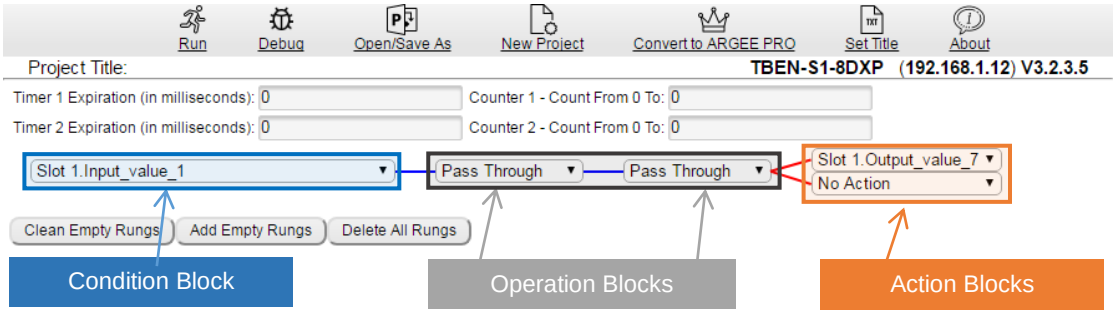
3.3 Welcome to Flow Chart



4 Flow Chart

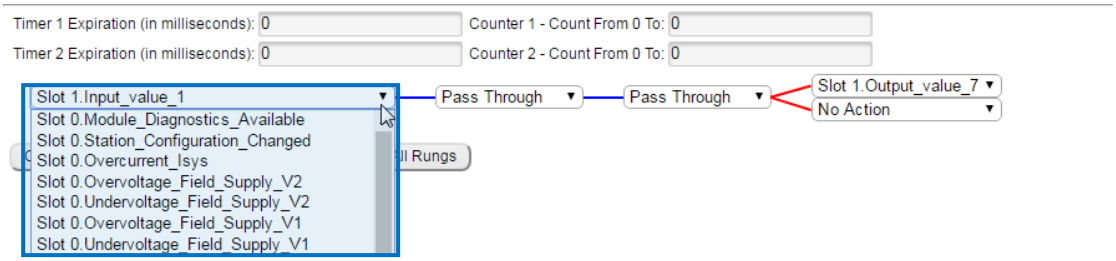
4.1 The Basics

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.

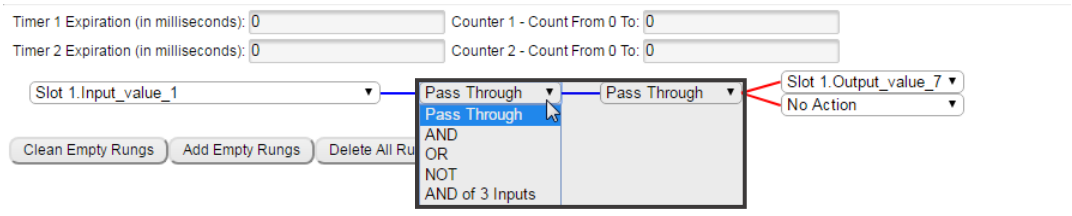


NOTE

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*.

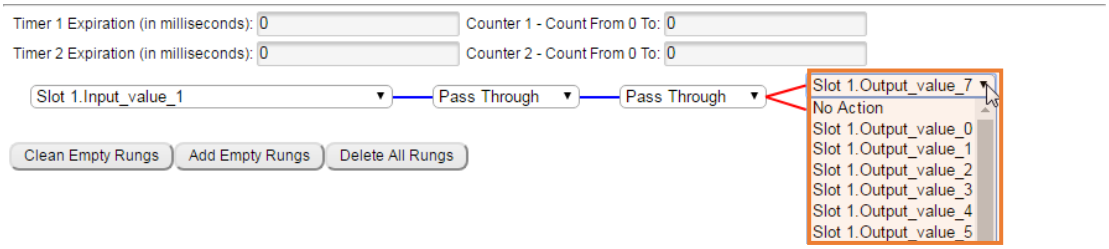


NOTE

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.

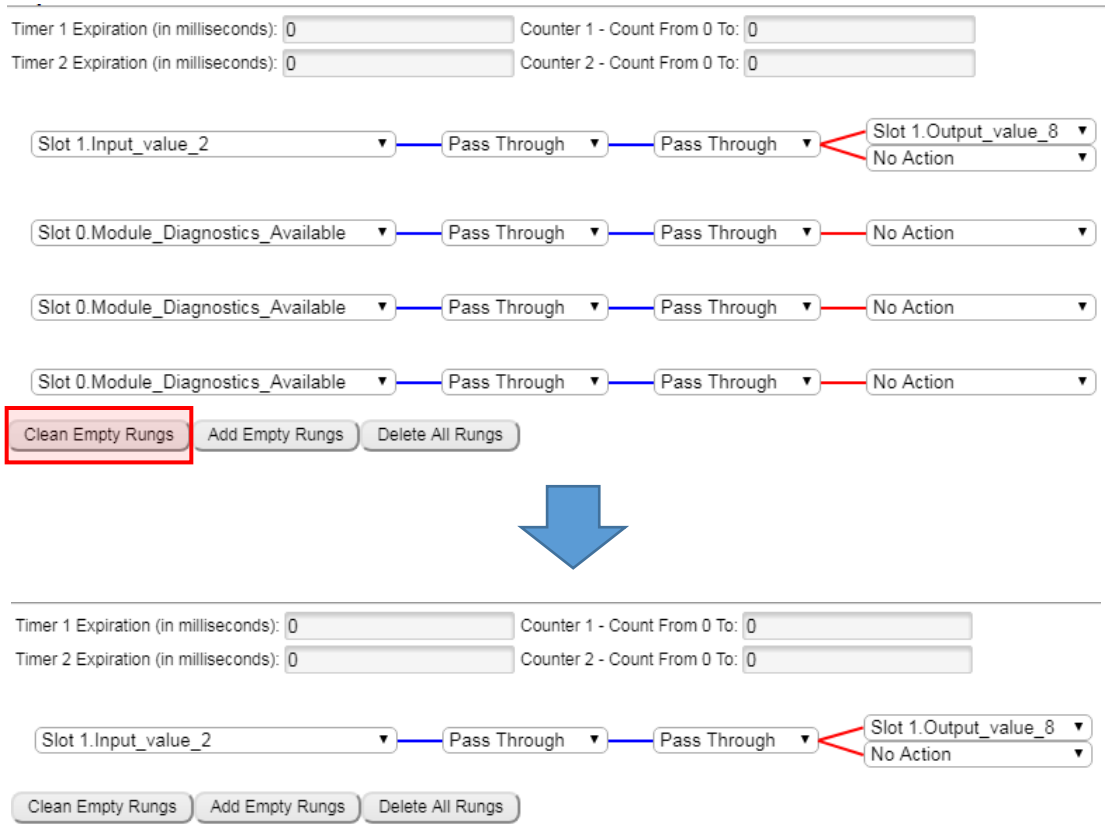


NOTE

TON Timer X (**T**imer **ON** Timer X), CTU Counter X (**C**oun**T** **U**p 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.

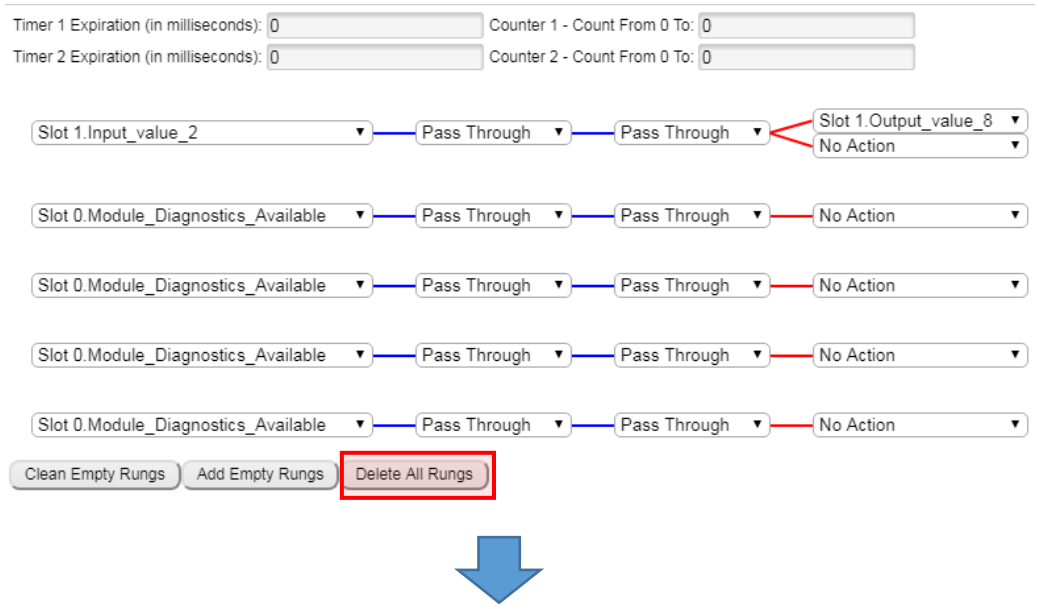


4.6 Add Empty Rungs

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.

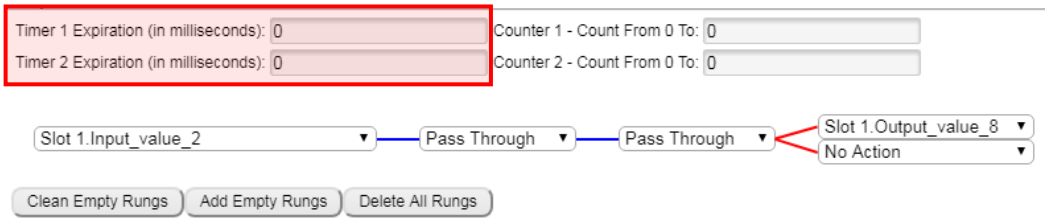


NOTE

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).

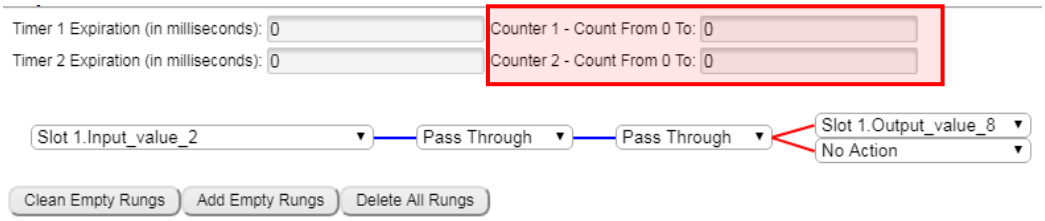


NOTE

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.

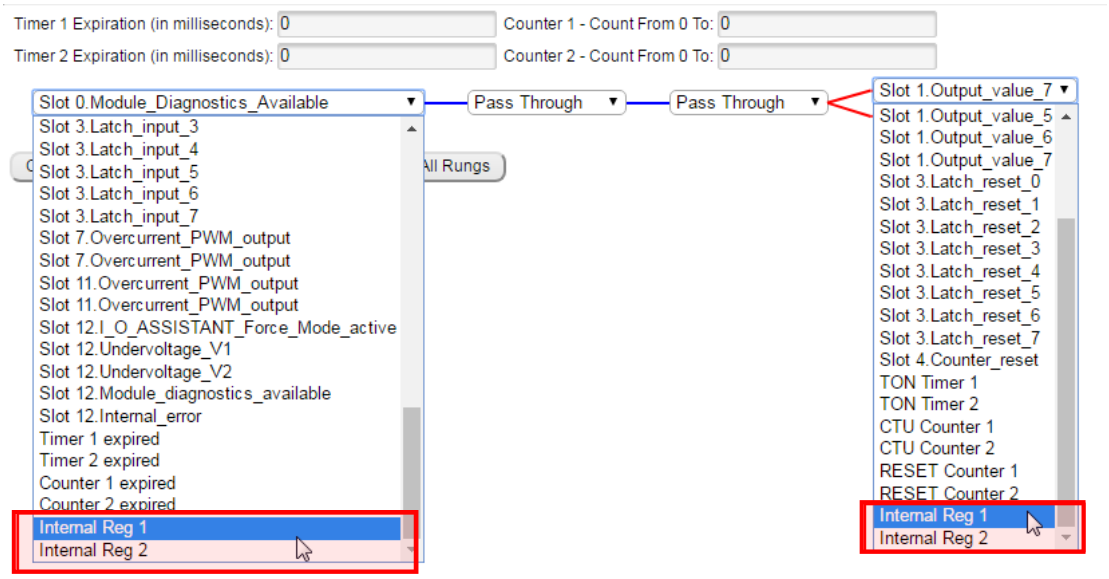


NOTE

Counters are discussed further in chapter 5.4 Actions.

4.10 Internal Reg

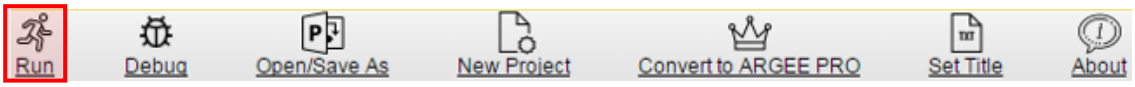
Flow Chart Editor contains two *Internal Regs (Reg = register)*. The user can use an internal register as a condition to trigger an action or as an action to trigger a condition.



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.

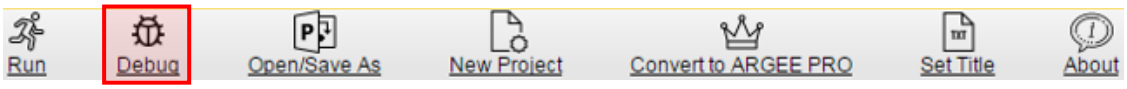


NOTE

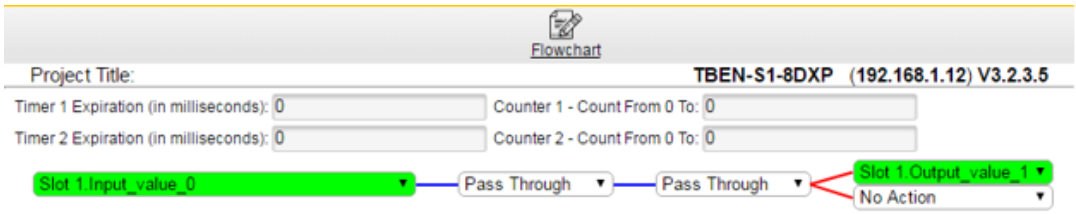
After the run button is pressed, the environment transitions to the Debug screen. More information about Debug can be found in chapter 7 Debugger.

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.

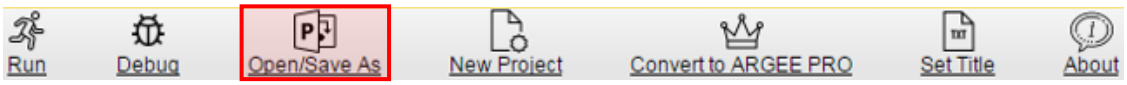


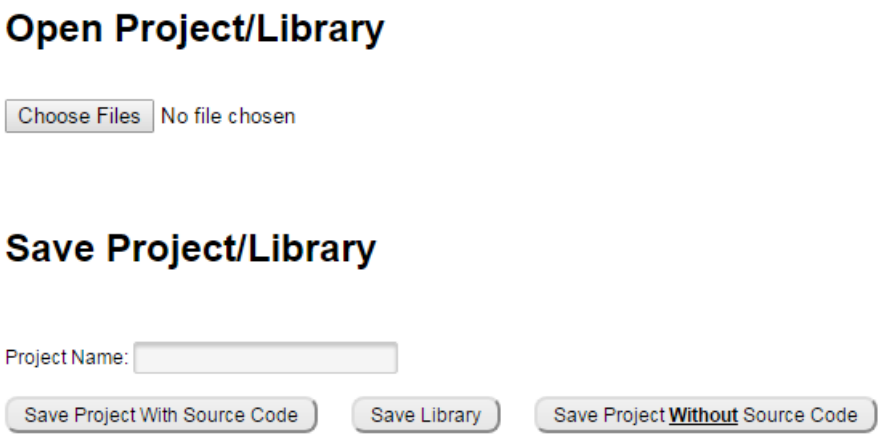
If the user clicks *Debug* while in Flow Chart, the first thing the user will notice is that the Flow Chart will enter *Debug* mode. As conditions become true, the user can visually observe code progression.



4.11.3 Open/Save As

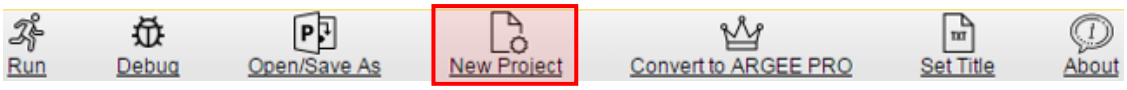
The Open/Save feature allows the user to save a current project or load a previous project. The user can access the *Open/Save As* screen from different places depending on if they are in Flow Chart or ARGEE PRO. From Flow Chart, the *Open/Save As* tab is available in the ARGEE Menu Bar. While in *ARGEE PRO*, the user can access the Open/Save As screen by clicking on the *Project* tab.





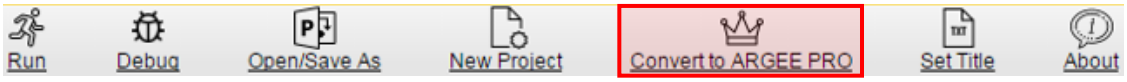
4.11.4 New Project

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.

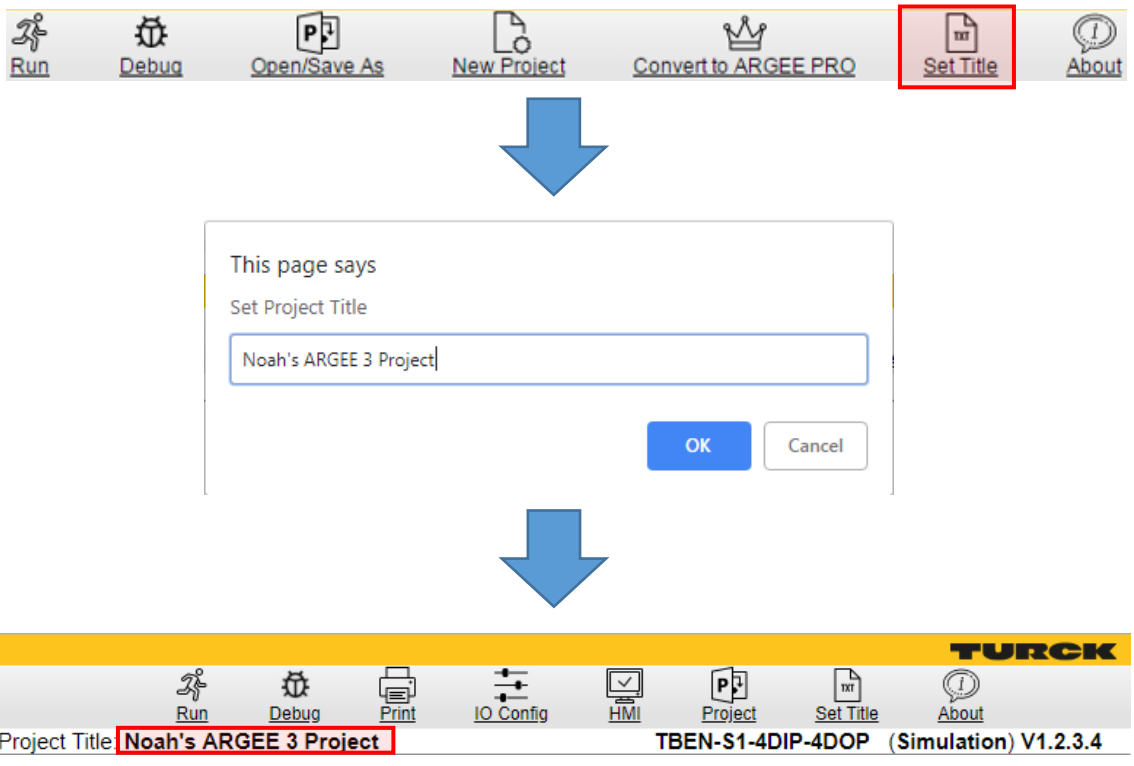


NOTE

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

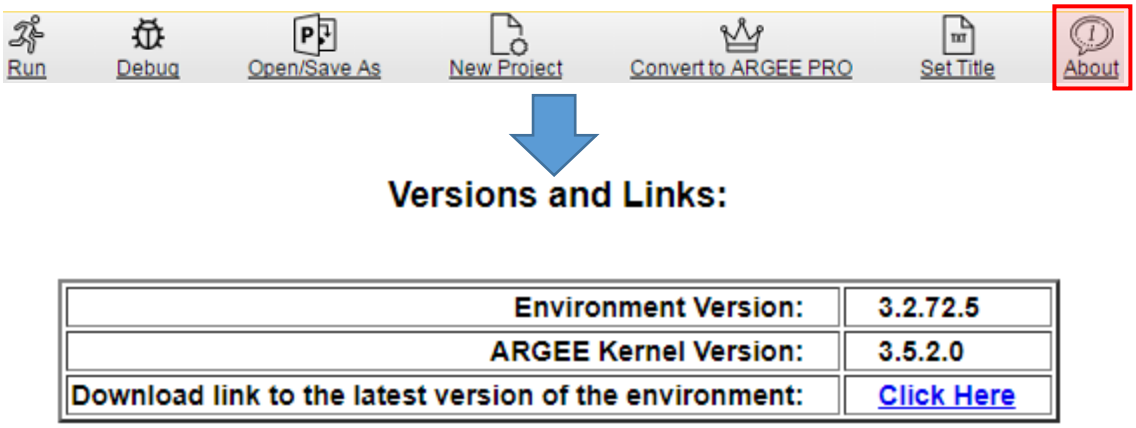
4.11.6 Set Title

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



4.11.7 About

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

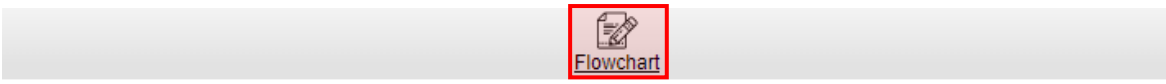


NOTE

The user can use the "**Click Here"** hyperlink to download the latest ARGEE environment.

4.11.8 Flowchart

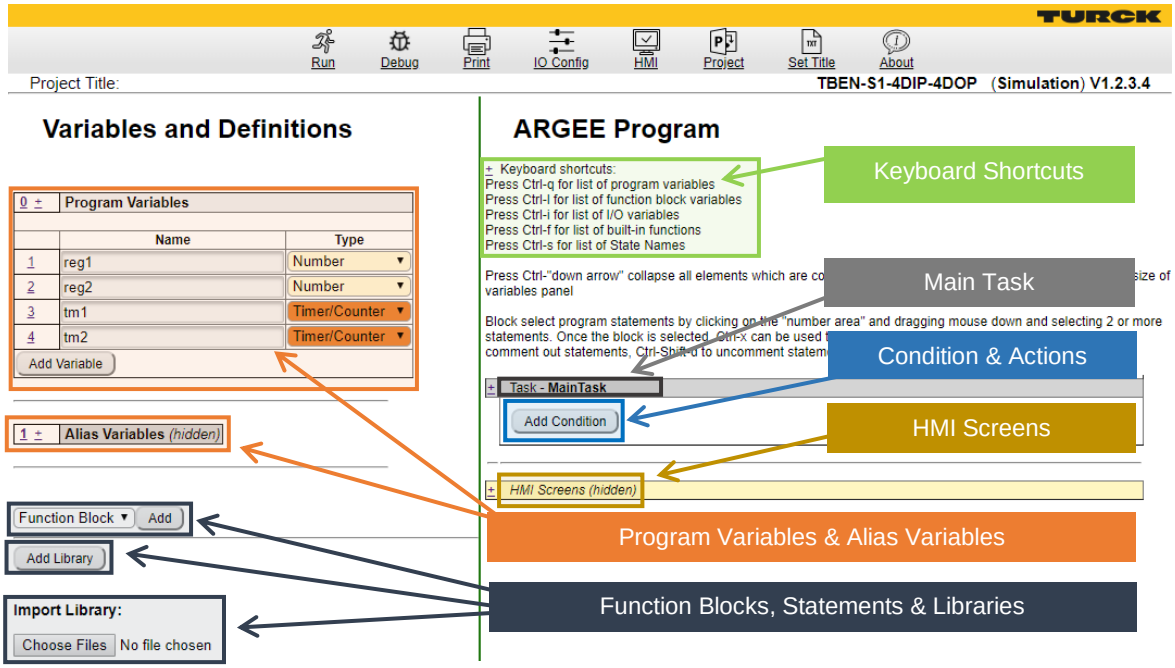
The user will click *flowchart* when they want to leave the debug page and return to the ARGEE Flow Chart screen.



5 ARGEE PRO

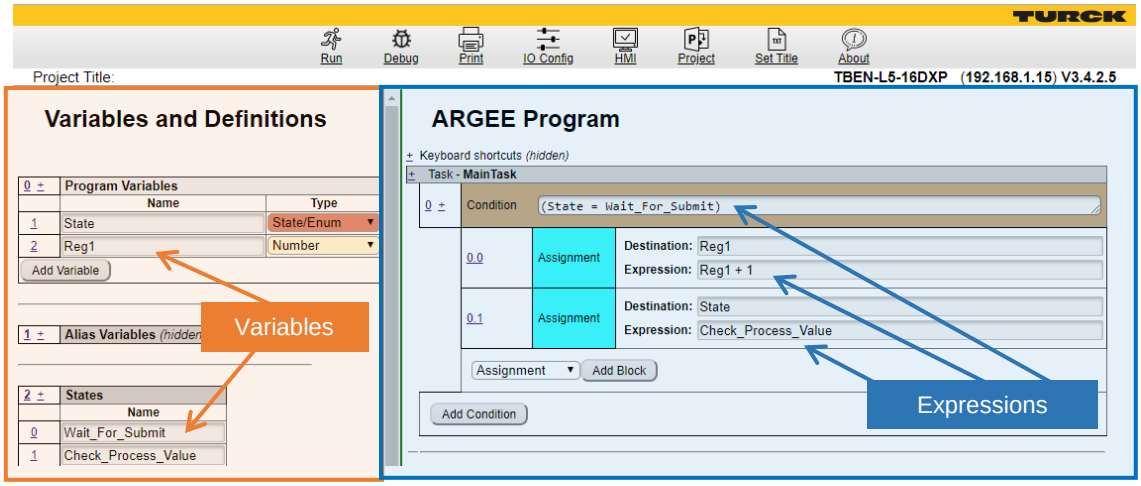
5.1 The Basics

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.



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.



NOTE

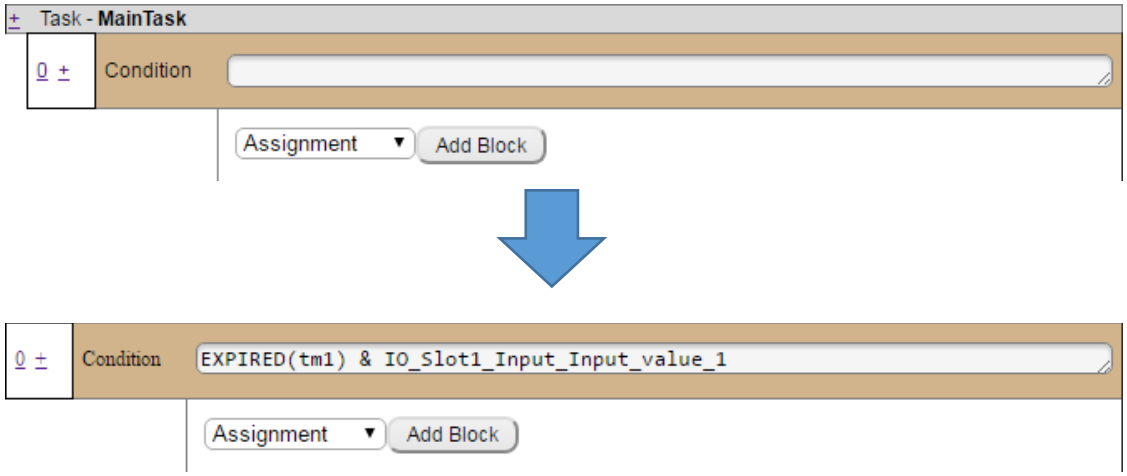
Information on the functions available in ARGEE can be found in the Appendix 11.2 Built-in Functions (Ctrl-f), and information on variables can be found in chapter 5.7 Program Variables.

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.

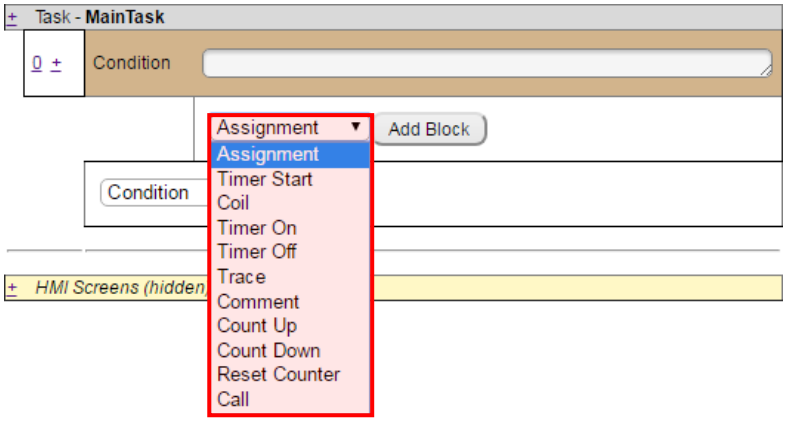


Explaining the Example: The above Condition will only become true when timer 1 expires and Input_value_1 goes true.

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)

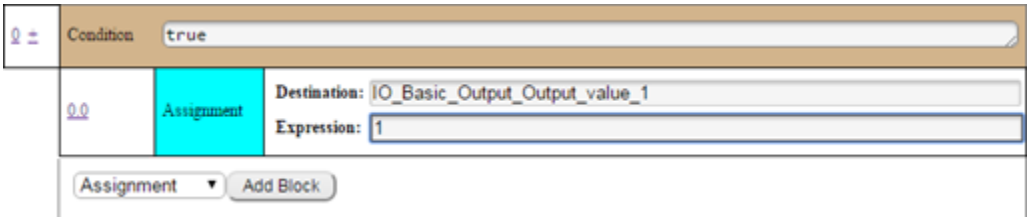
- Assignment
- Timer Start
- Coil
- Timer On
- Timer Off
- Trace
- Comment
- Count Up
- Count Down
- Reset Counter
- Call



5.4.1 Assignment

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

Example of Assignment:

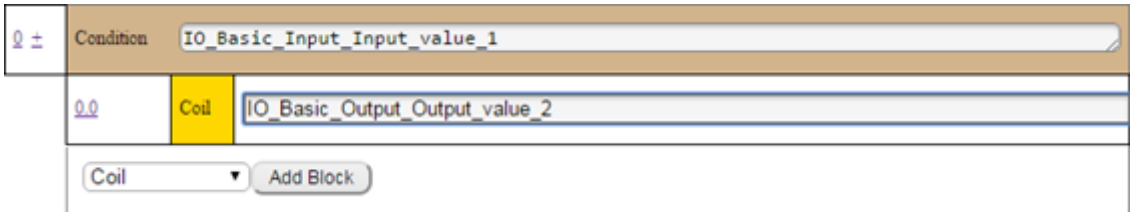


Explaining the Example: The Condition in the above statement is always “true.” The value “1” is loaded into register Output_value_1. In other words, this means that the user’s Output 1 will always be on.

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.

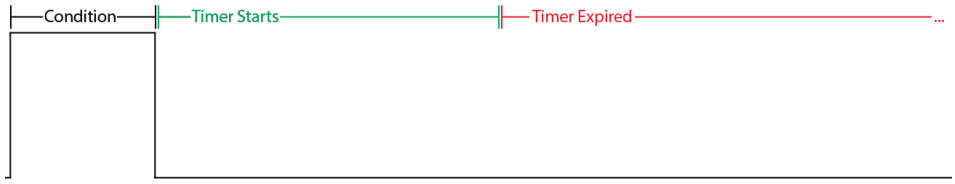
Example of Coil:



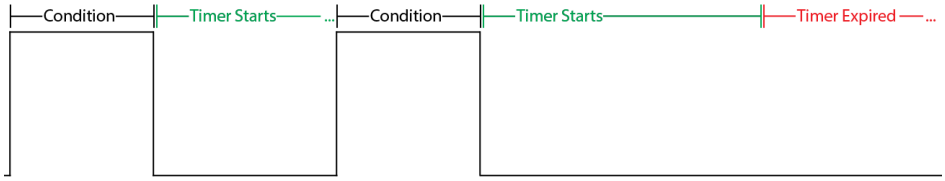
Explaining the Example: When Input_value_1 is true, Output_value_2 is true. When Input_value_1 is false, Output_value_2 is false.

5.4.3 Timer Start

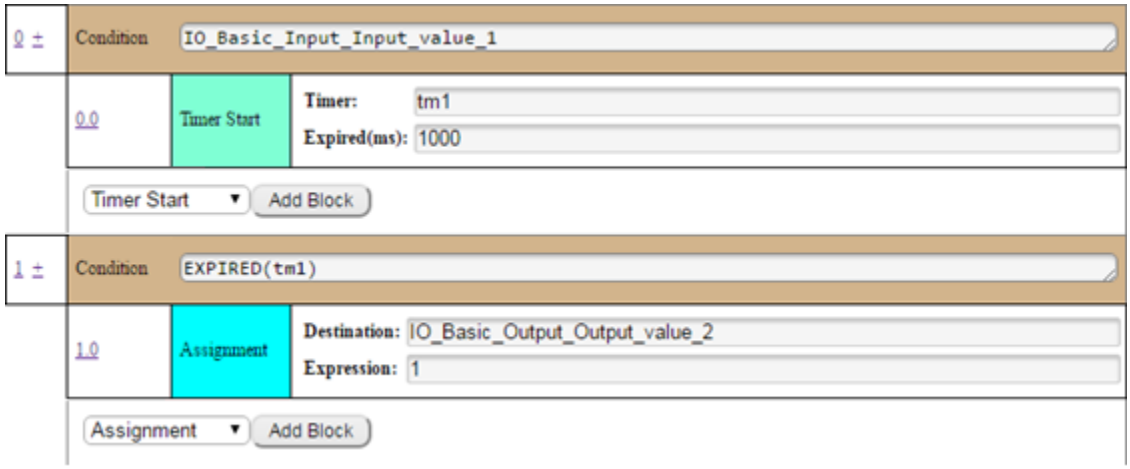
The user will use the *Timer Start* action if they want to start a timer after the Condition has occurred.



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



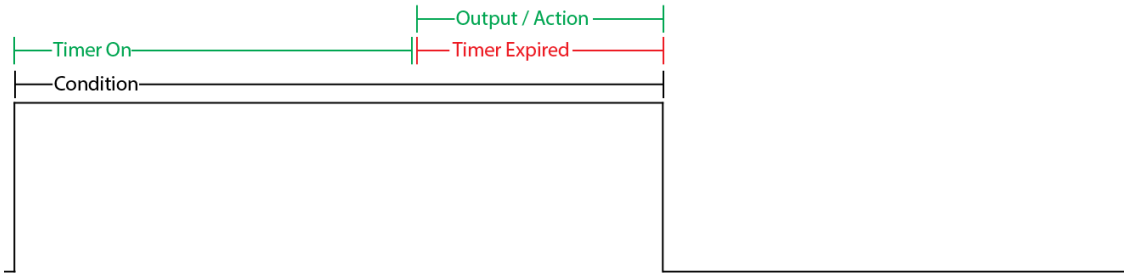
Example of Timer Start:



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).

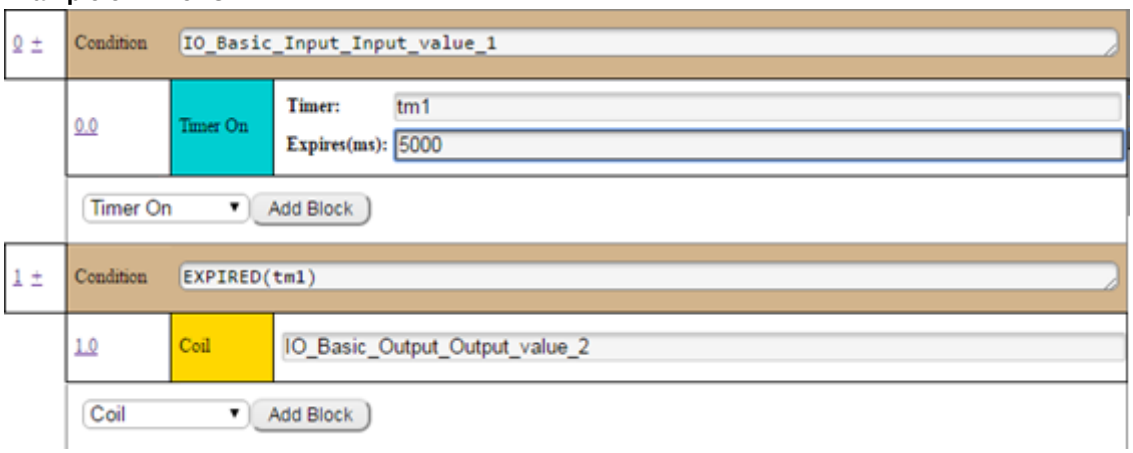
5.4.4 Timer On

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.

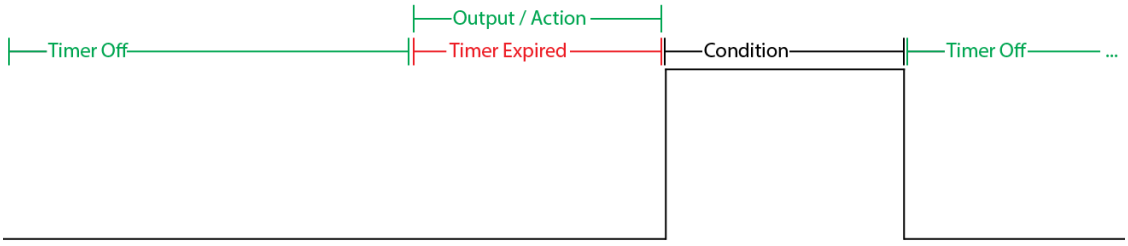
Example of Timer On:



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.

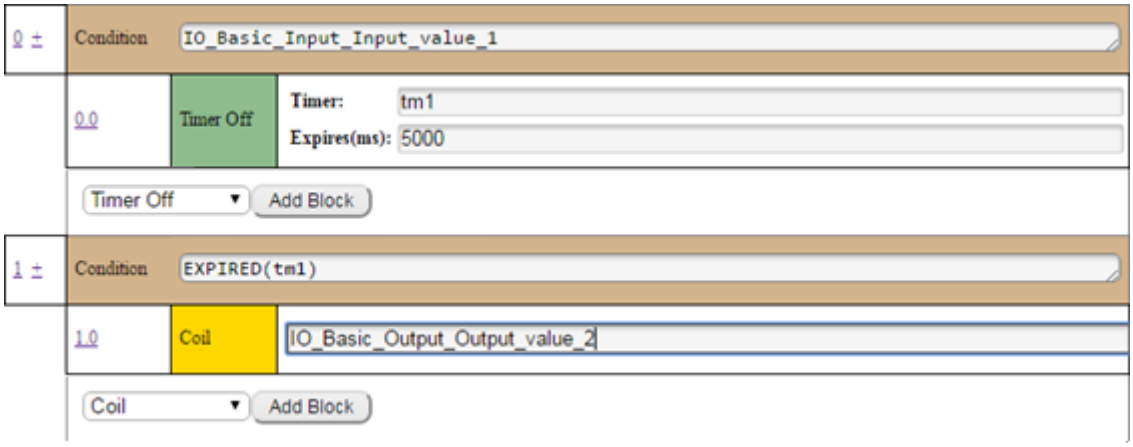
5.4.5 Timer Off

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:

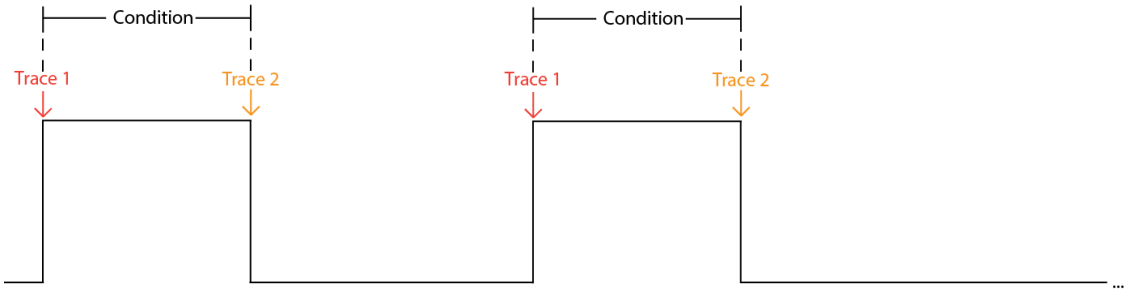


Explaining the Example: Timer 1 starts counting as soon as the program starts. When timer 1 expires, Output_value_2 is coiled on. When Input_value_1 is true, timer 1 is reset to zero and Output_value_2 goes false. When Input_value_1 is false, timer 1 starts counting again.

5.4.6 Trace

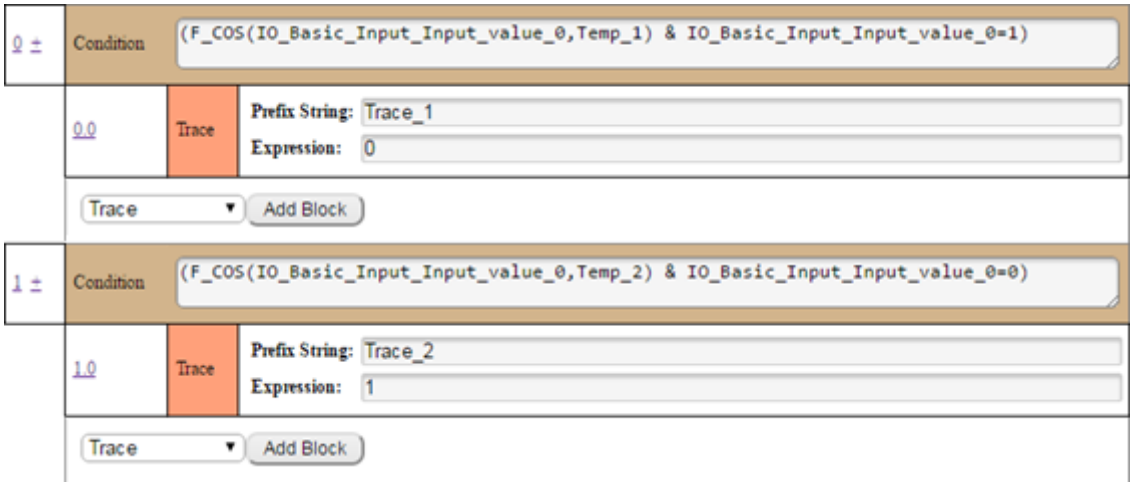
The user will use the Trace function if they want to time stamp exactly when an event occurred. Trace can be used to measure a programs run-time behavior, how long each state takes and even which states were visited in which order.

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).



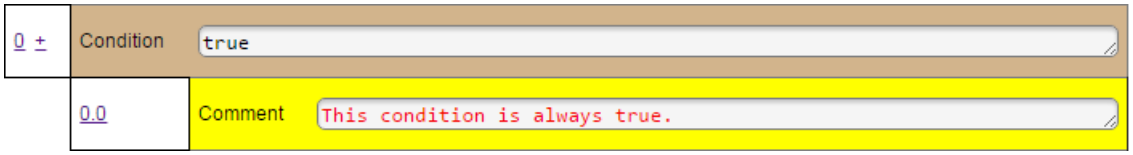
Explaining the Example: When Input_value_0 is true, Trace_1 time stamps that event. When Input_value_0 goes false, Trace_2 time stamps that event. The Prefix String is a name that makes sense to the user. The Expression can be any value or even another variable name that makes sense to the user.

NOTE

An example of *Trace* can be found in the Appendix 12.2 Trace Example.

5.4.7 Comment

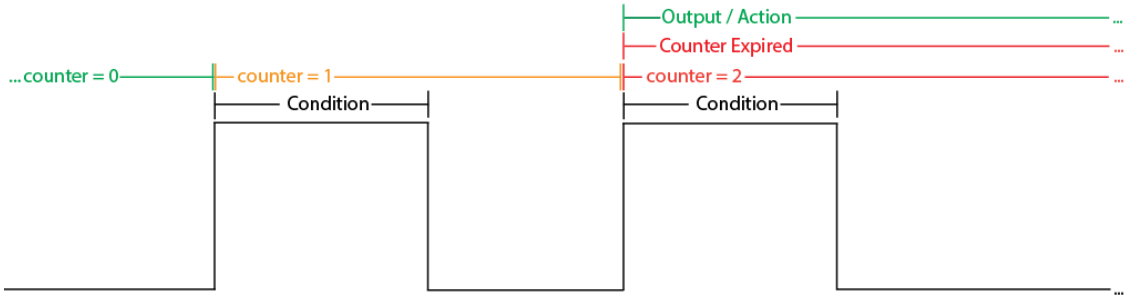
The user can use a *Comment* to explain the Condition and Action statements.

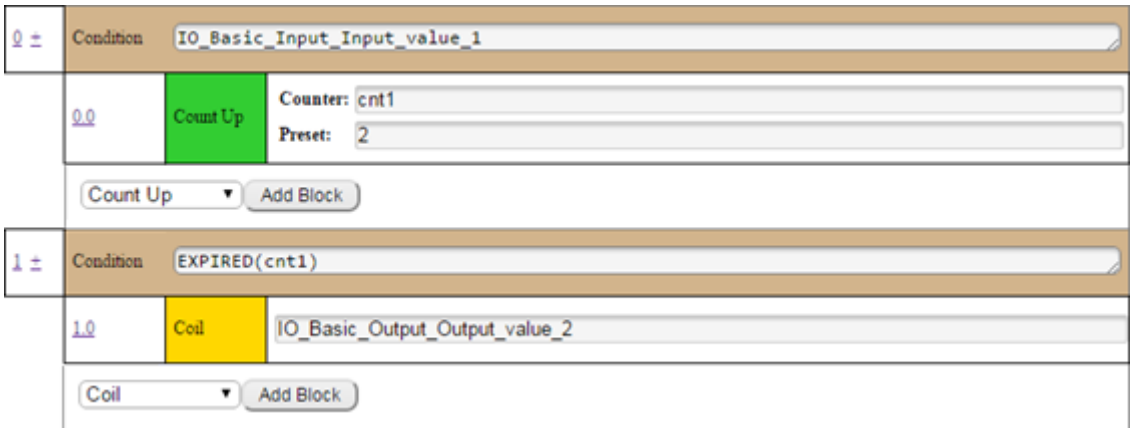


5.4.8 Count Up

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.

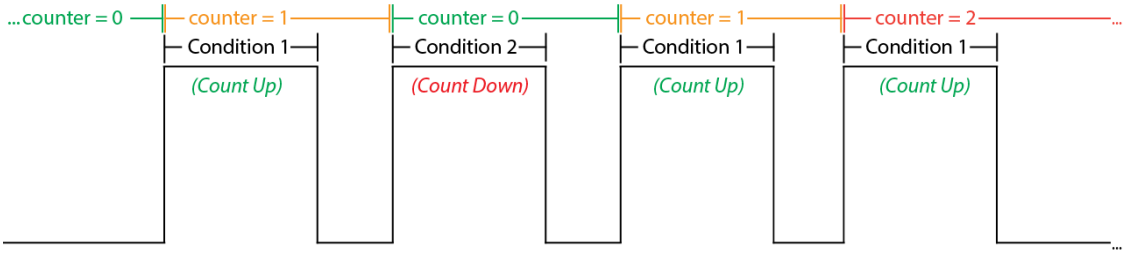




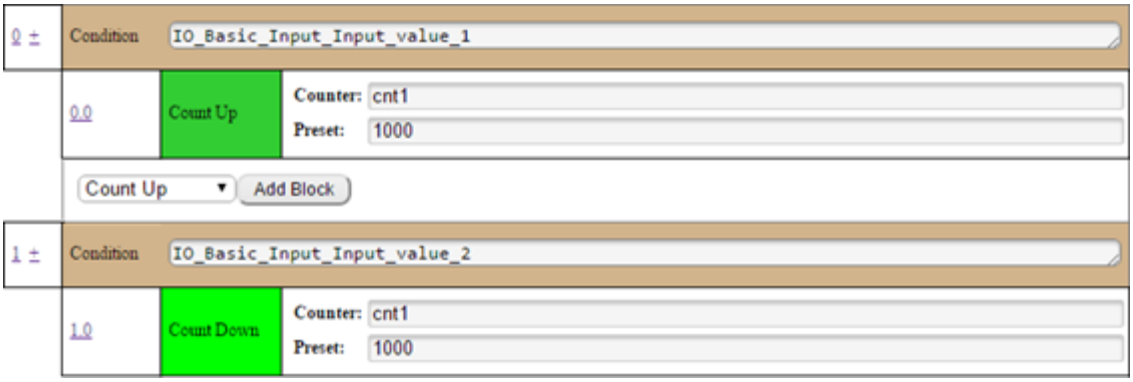
Explaining the Example: Each time Input_value_1 is true, counter 1 counts up one time. Counter 1 expires after two counts. When counter 1 expires, Output_value_2 is coiled on.

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.



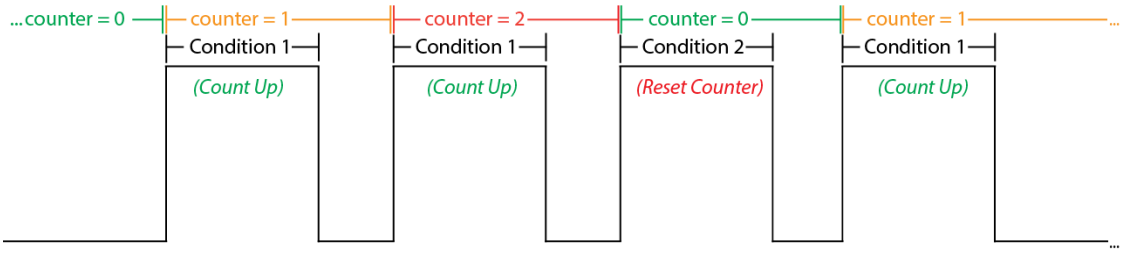
Example of Count Down:
The user wants to keep track of the number of guests in the store. When a guest walks in the store the counter goes up, but when a guest walks out of the store the counter goes down.



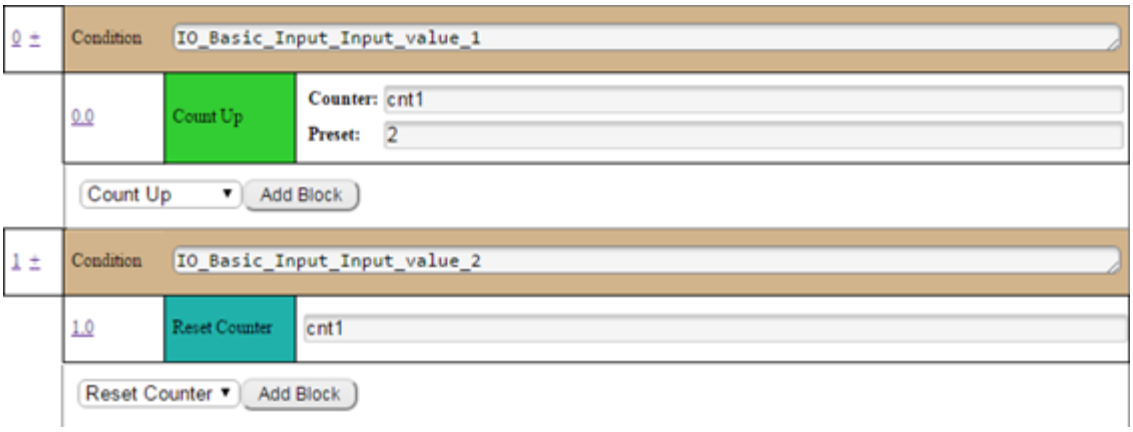
Explaining the Example: Each time Input_value_1 is true (or a guest walks in the store), counter 1 counts up one time. Each time Input_value_2 is true (or a guest walks out of the store), counter 1 counts down one time.

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.

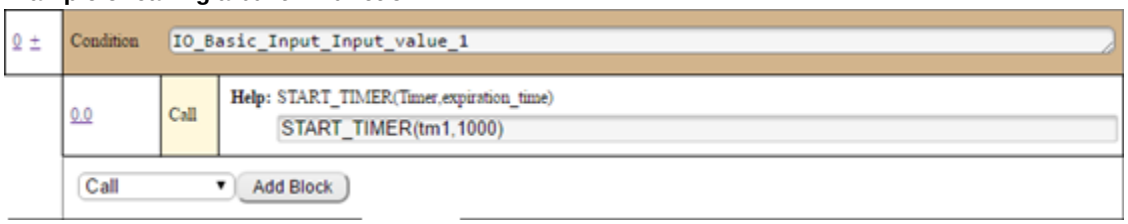


Explaining the Example: Each time Input_value_1 is true, counter 1 counts up one time. Each time Input_value_2 is true, counter 1 resets to zero.

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

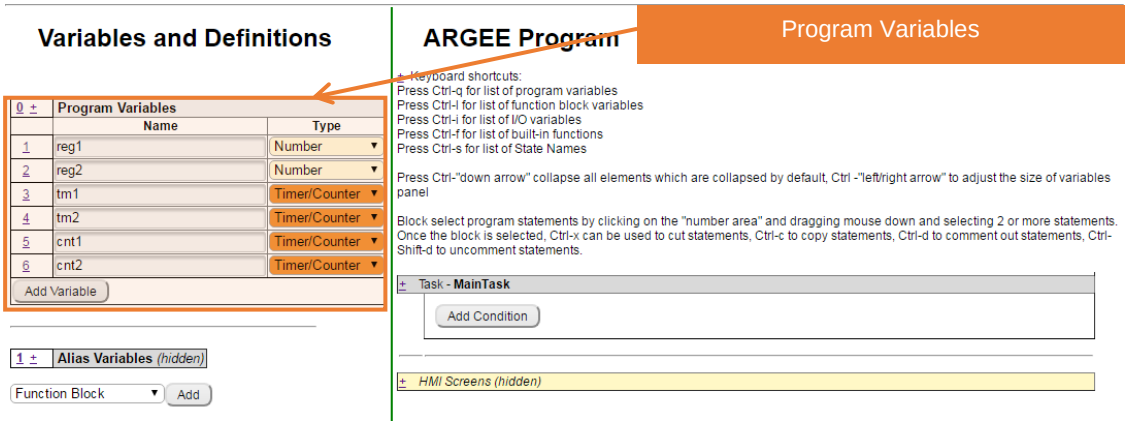
---

5.4.12 How Actions respond to Conditions

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

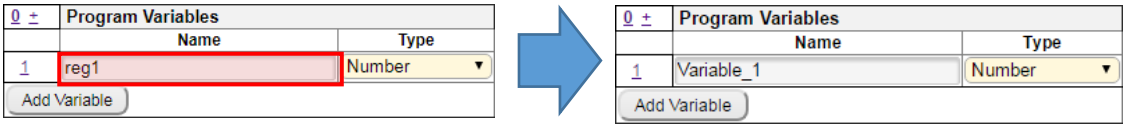
5.5 Program Variables

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.



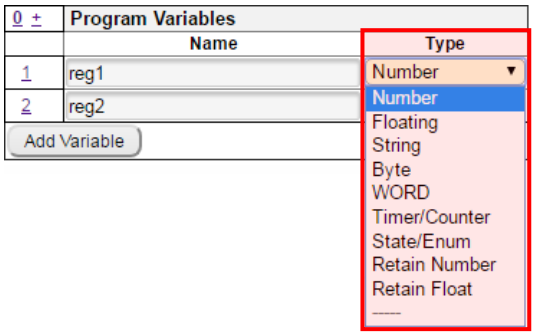
5.5.1 Variable Name

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



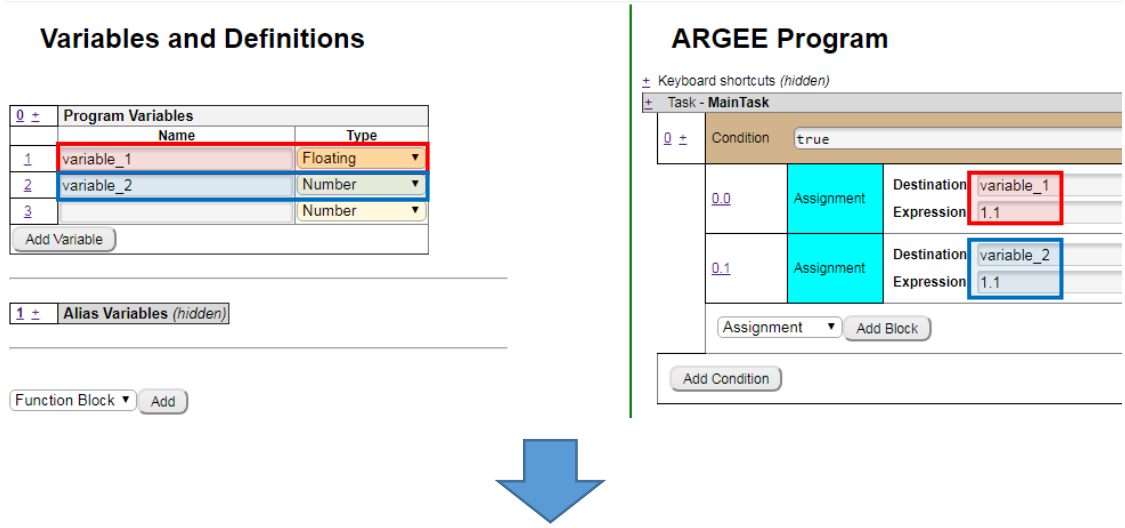
5.5.2 Variable Types

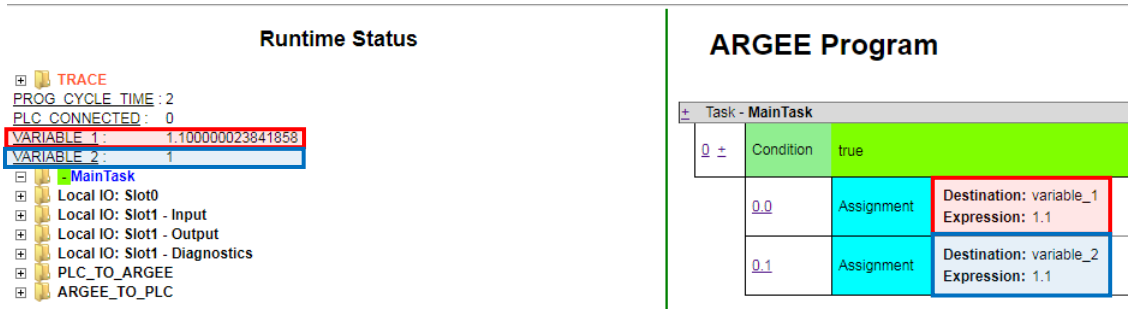
The user can set his desired variable type by selecting the Type drop down arrow.



- 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.

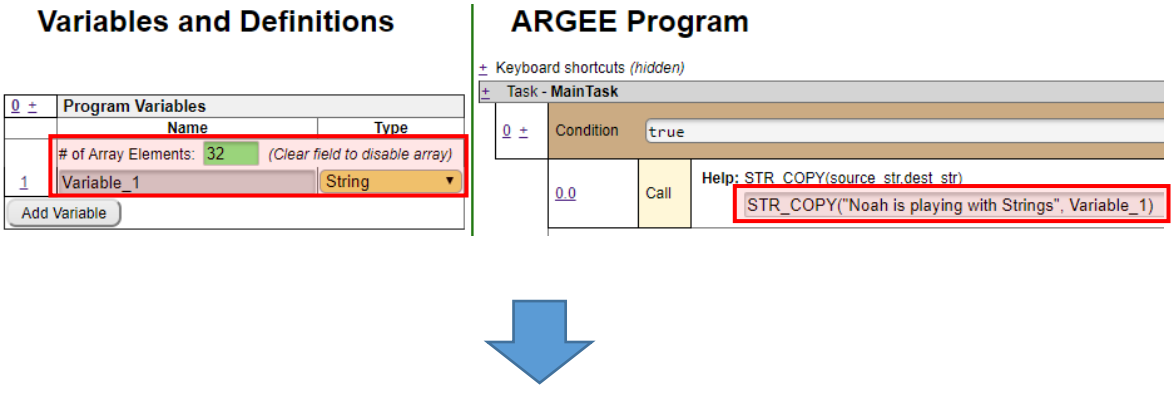


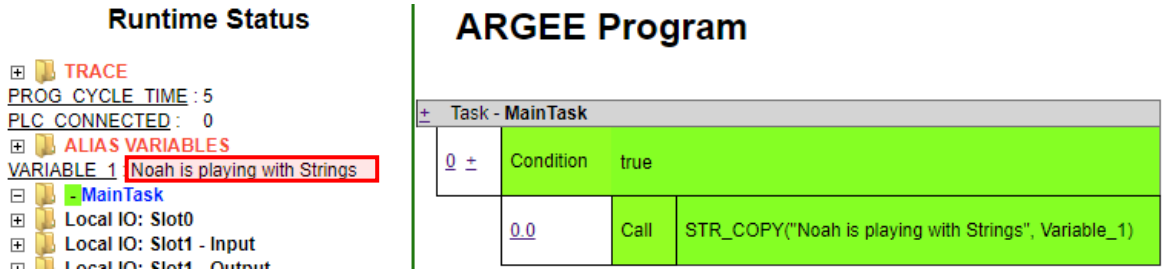


NOTE

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.





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.

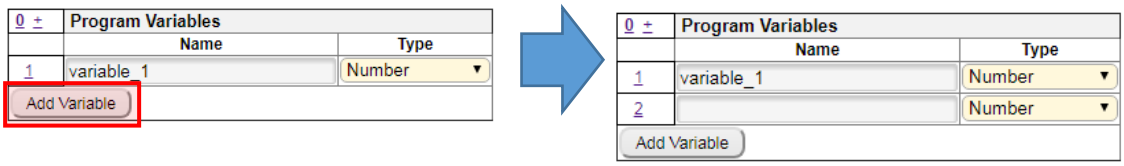
NOTE

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.

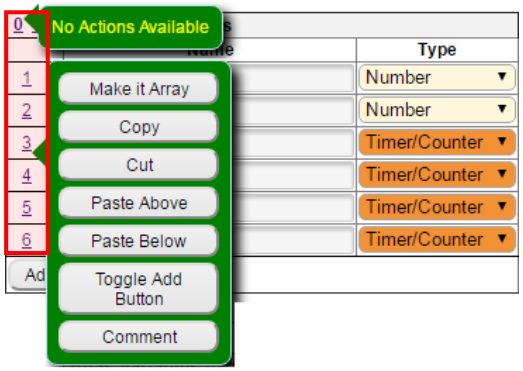
5.5.3 Add Variable

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

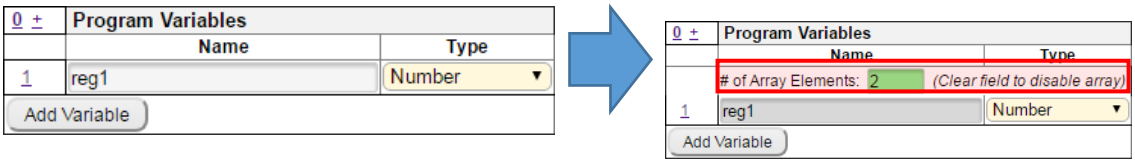


5.5.4 Program Variables Context Menu

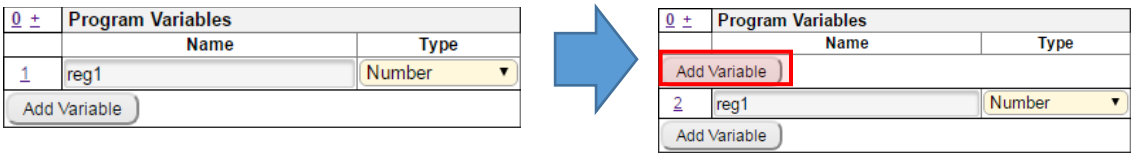
➢ To access this menu, click the number in front of the variable.



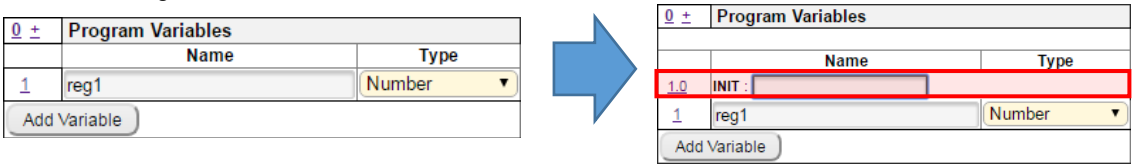
- Make it Array - Turns the variable into an array.



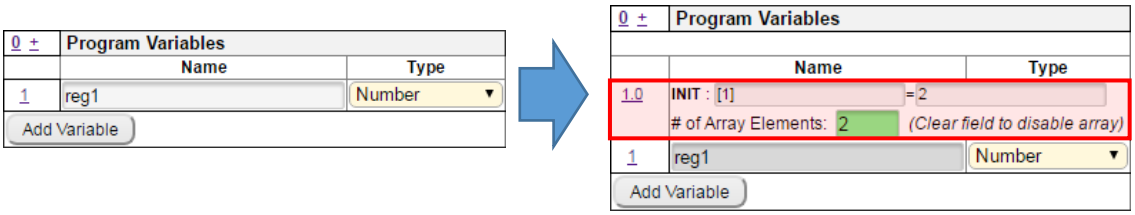
- Copy - Copies the variable so the user can paste it in another place.
- Cut - Cuts the variable out so the user can then paste it in another place.
- Paste Above - Paste a cut/copied variable above the selected position.
- Paste Below - Paste a cut/copied variable below the selected position.
- Toggle 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.



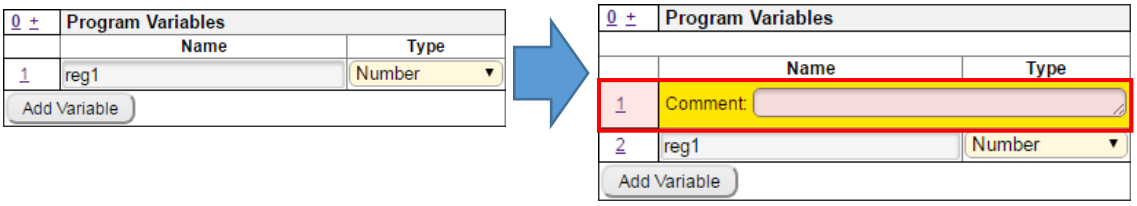
The user can also Initialize an array if they want to pre-set the value in a Program Variable's register.



NOTE

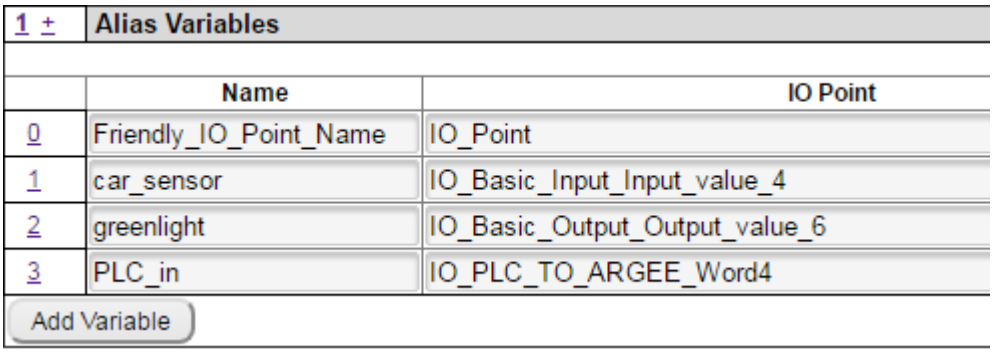
The user can press Control-q while in the *Program Variable* name area to automatically prompt variable initialization.

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



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*.

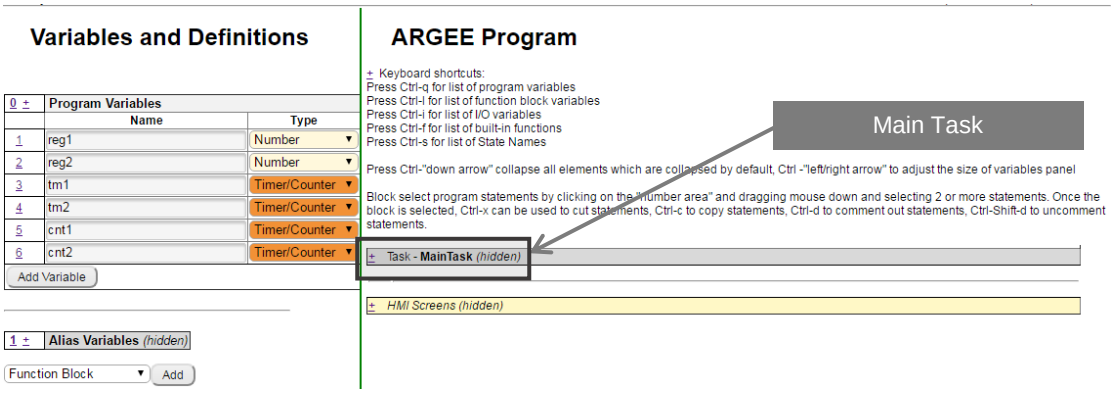


1 ±	Alias Variables	
	Name	IO Point
0	Friendly_IO_Point_Name	IO_Point
1	car_sensor	IO_Basic_Input_Input_value_4
2	greenlight	IO_Basic_Output_Output_value_6
3	PLC_in	IO_PLC_TO_ARGEE_Word4

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*.

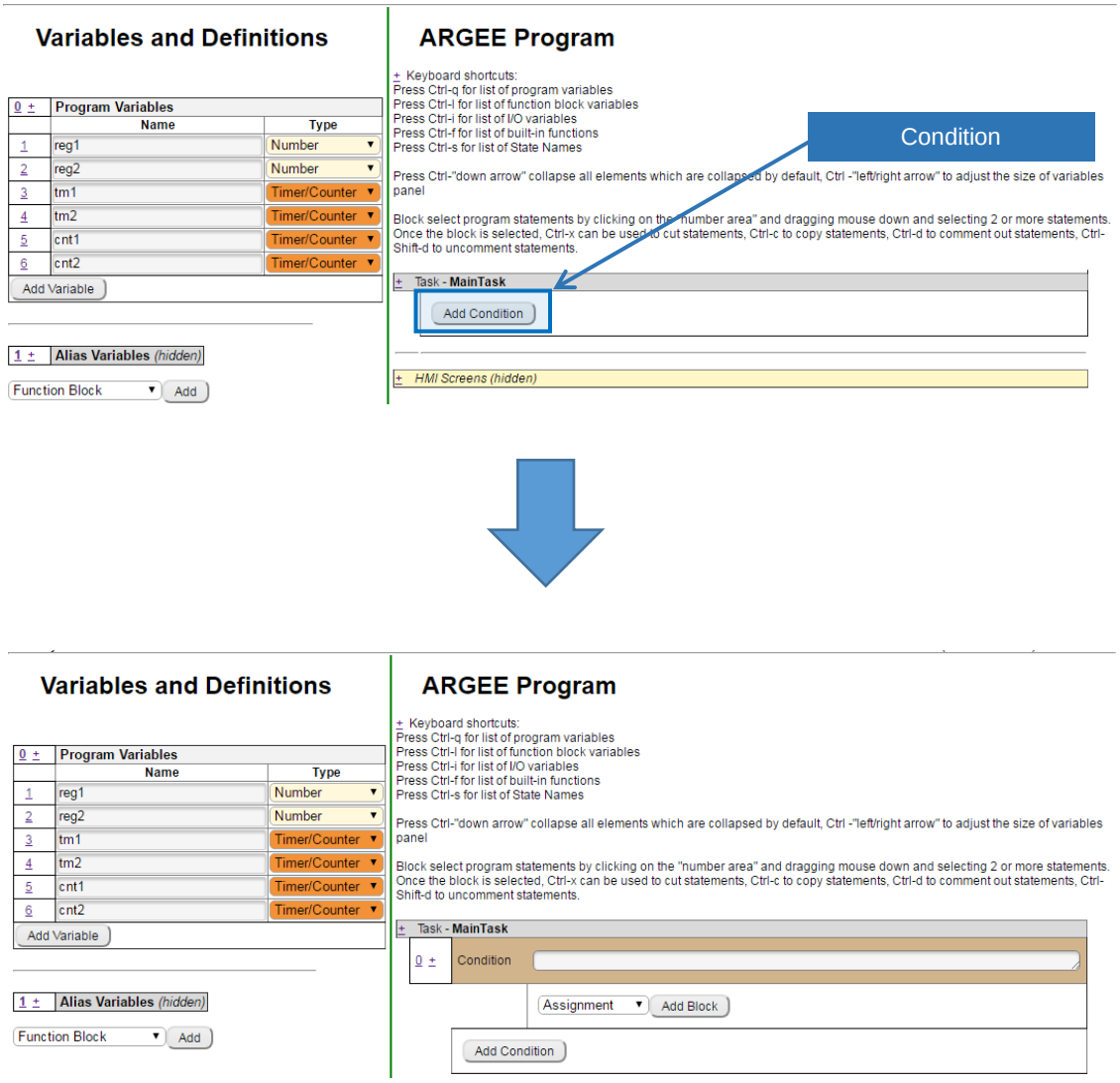


NOTE

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.

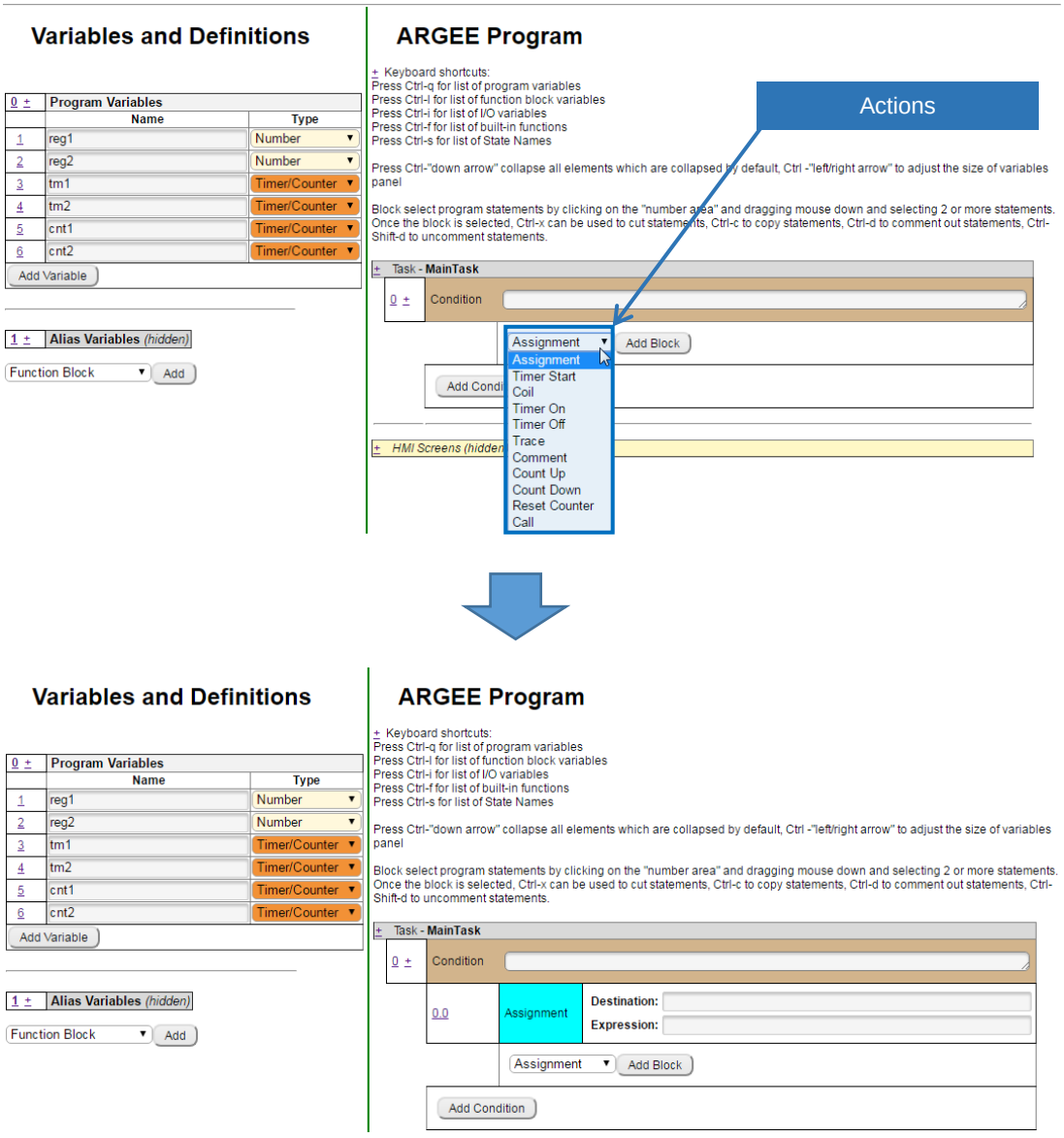


NOTE

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

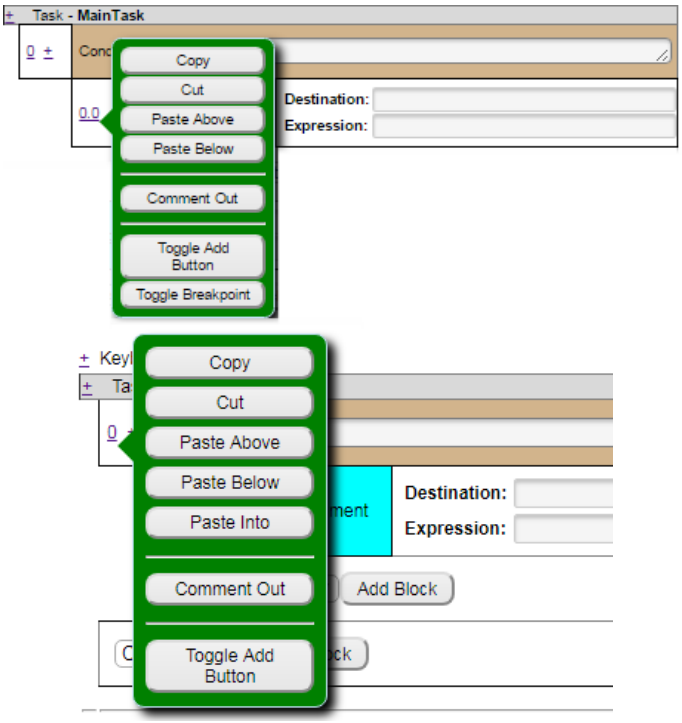
Actions are selected from the *Add Block* drop-down menu. When the desired action is selected, the user can click on the *Add Block* button to add the action to the condition.



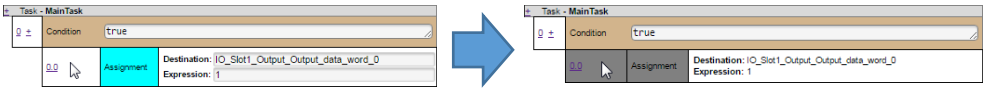
NOTE

Actions are discussed more in this chapter 5.4 Actions.

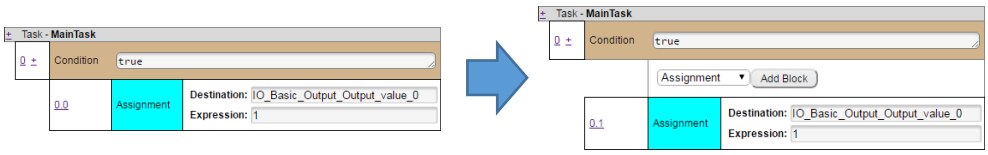
5.7.3 Main Task Context Menu



- **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.

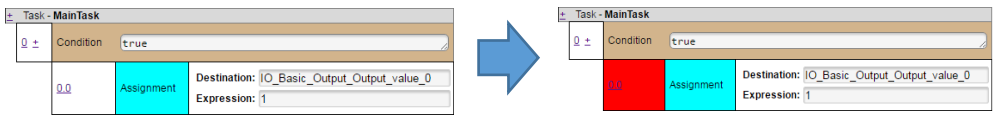


- **Toggle Add Button** - Selecting this will place an “Add Variable” button above the variable.

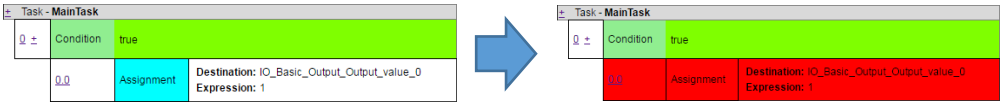


- **Toggle Breakpoint** – The statement becomes a breakpoint when the code is compiled and ran. The program will not progress farther than the selected statement. This can be done from the Edit Code screen or the Debug Code Screen.

  (Edit Code Screen)



(Debug Screen)



NOTE

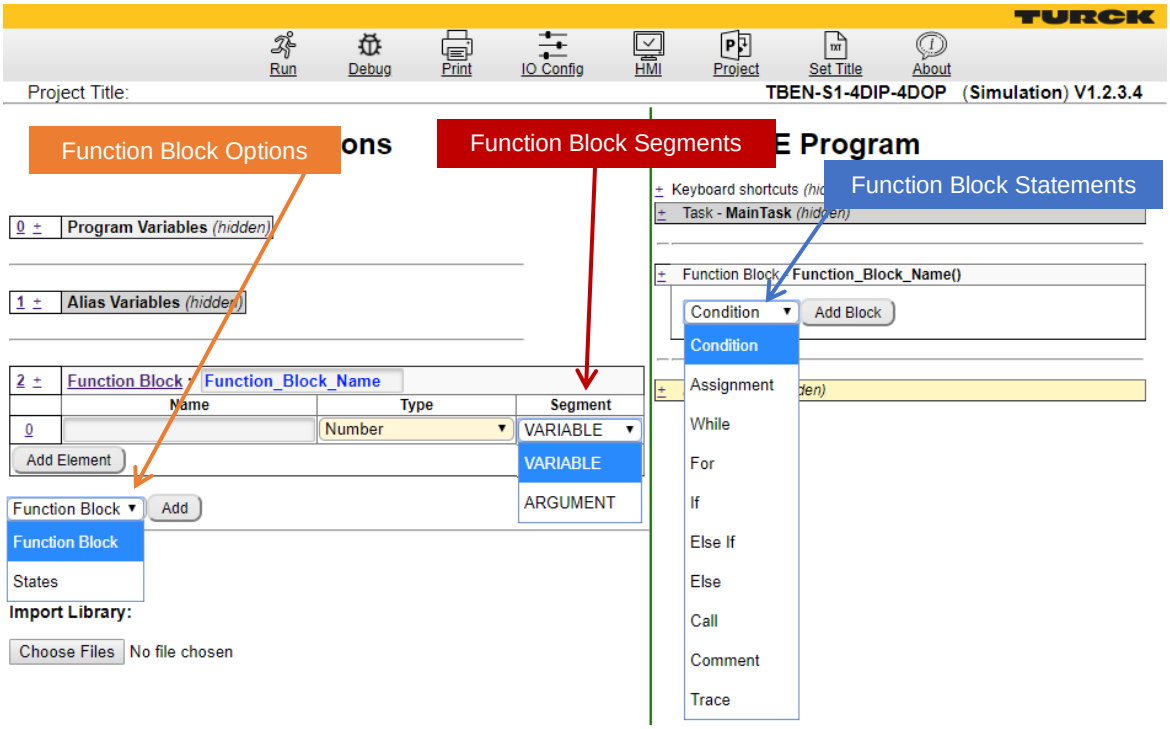
For more information about Debugging, check out Chapter 7 Debugger _ Debugger_1

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 there 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

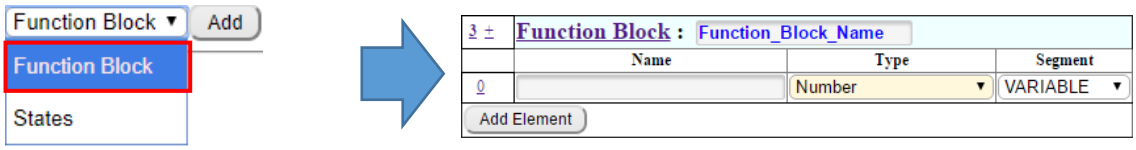
5.8.2 Function Block Options

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.



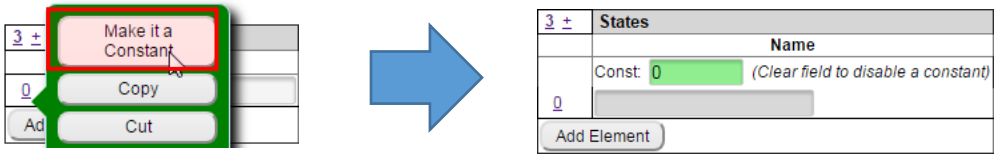
States

- 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.

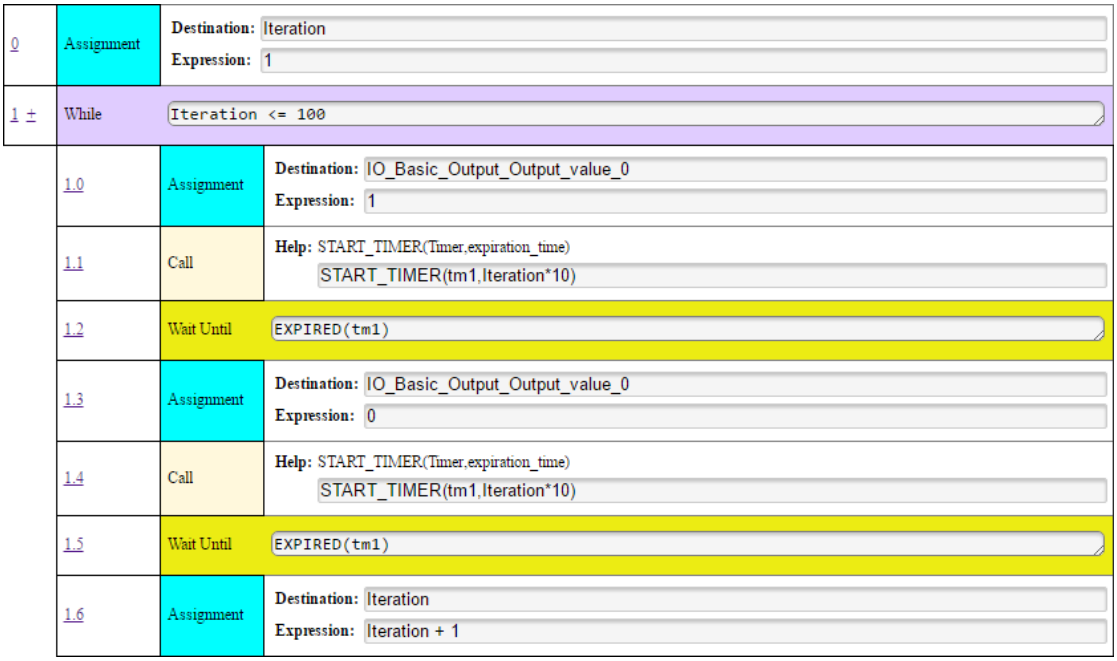
5.8.4 Function Block Statements

If the user wants to use *Statements* in the *Main task,* the user needs to convert their program to ARGEE PRO Advanced Mode.

5.8.4.1 While

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

Example of *While:*



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.

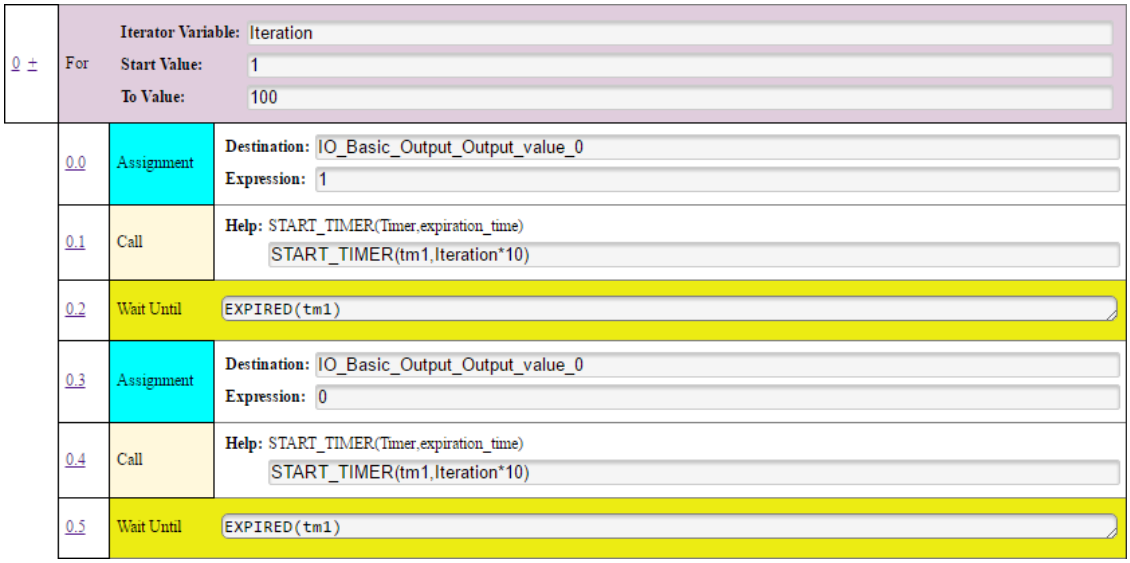
NOTE

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:*



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.

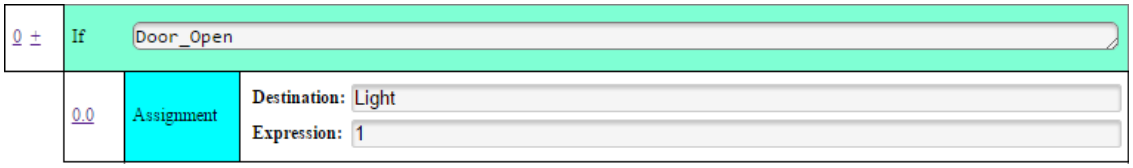
NOTE

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:*

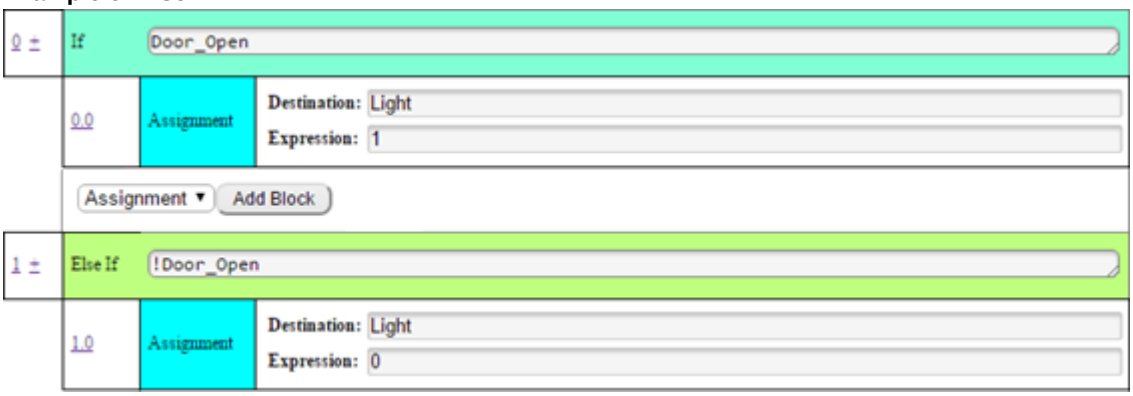


Explaining the Example: If the door is opened, turn on a light.

5.8.4.4 Else If

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

Example of Else If:



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

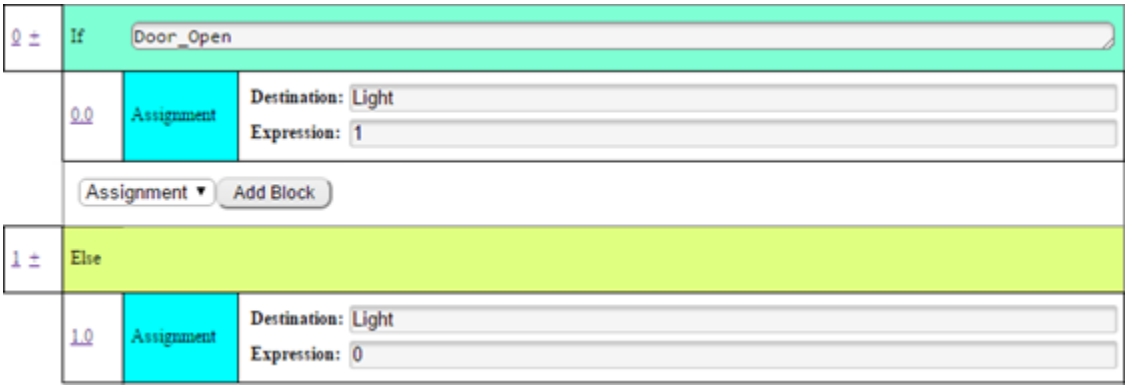
NOTE

"!" 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:*



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

5.9 Libraries

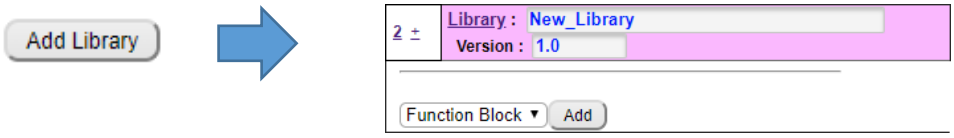
5.9.1 What is a Library?

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.

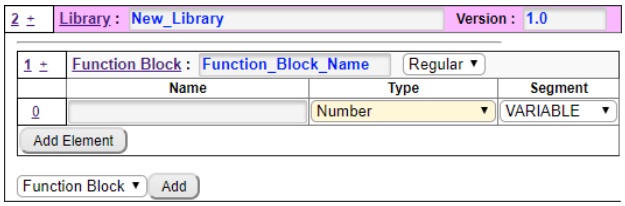
In addition to creating libraries, the user can download official ARGEE libraries from www.turck.com.

5.9.2 Creating a Library

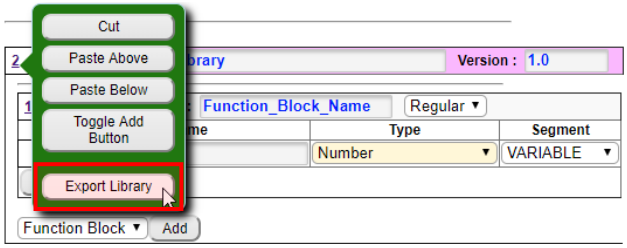
The user can create their own library by first clicking on the *Add Library* button.



The user then creates their desired function blocks.

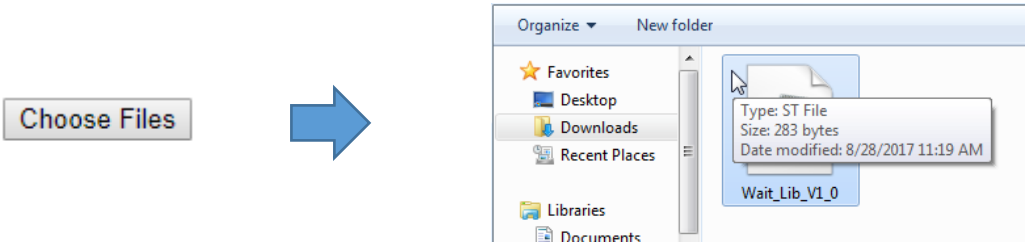


Once the library is complete, the user will select *Export Library* from the library context menu.



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.

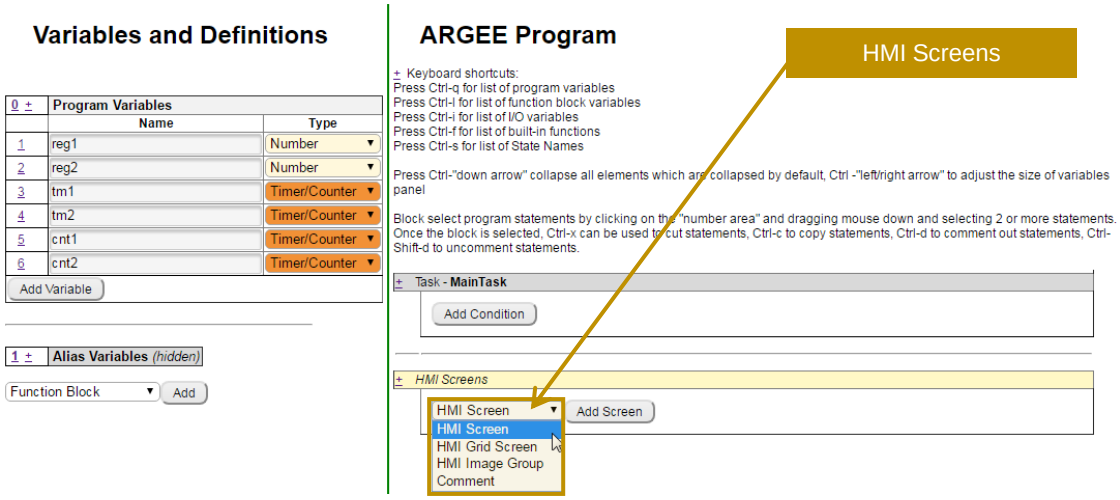
NOTE

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.



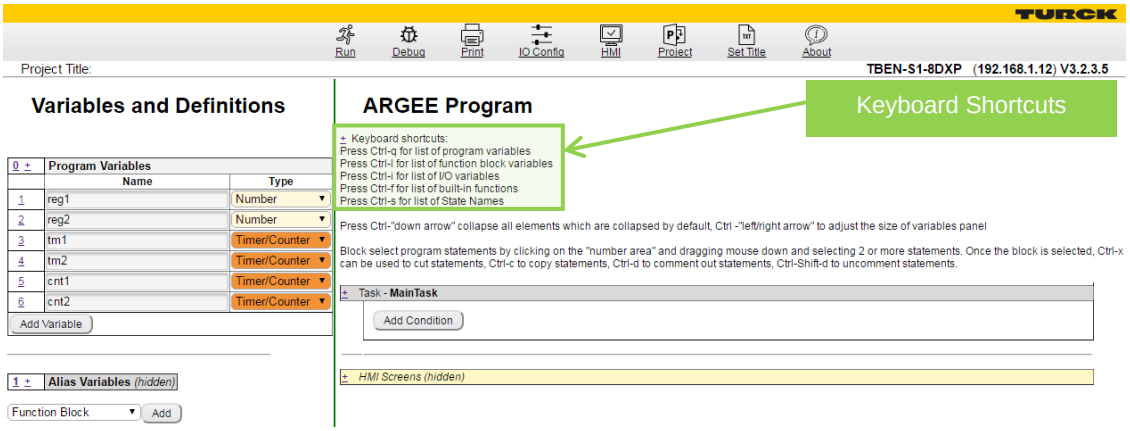
NOTE

Information on the HMI is available in chapter 9 ARGEE HMI.

5.11 Keyboard Shortcuts

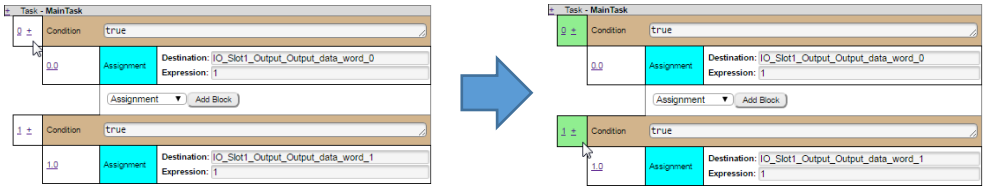
ARGEE 3 has many keyboard shortcuts to help make the user experience much easier. By default, the keyboard shortcuts are collapsed.

➢ Click on the + to expand the keyboard shortcuts.



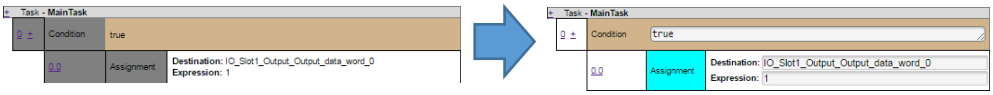
5.11.1 List of Keyboard Shortcuts:

- **Ctrl - q** Brings up a list of Program Variables
- **Ctrl - L** Brings ups 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.

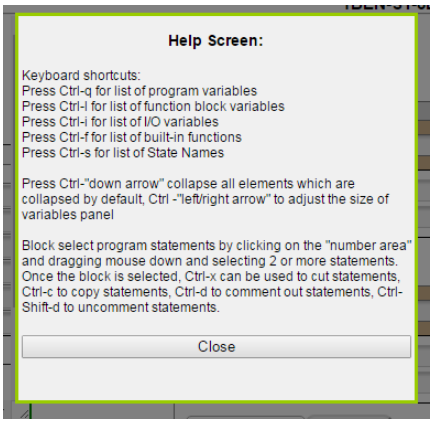


- **Ctrl – x** Cuts the selected statement(s)
- **Ctrl – c** Copies the selected statement(s)
- **Ctrl – z** Undoes the previous action (ARGEE 3 remembers 32 actions)
- **Ctrl - y** Redoes the previous action (ARGEE 3 remembers 32 actions)
- **Ctrl – d** Comment out selected statement(s). This turns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

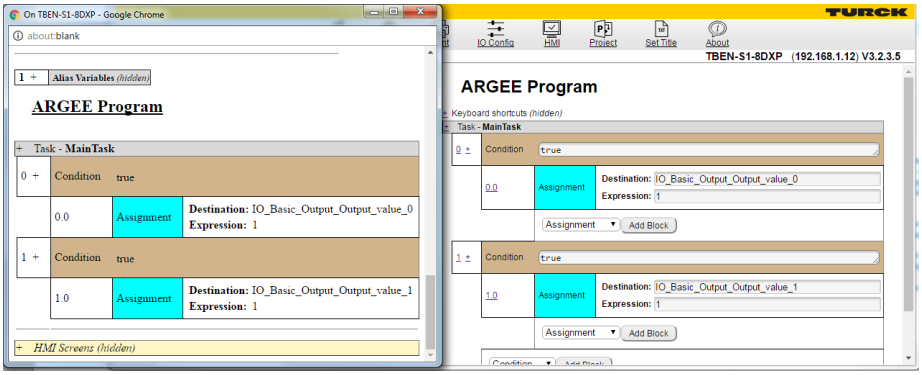
- **Ctrl – Shift – d** Uncomment out selected statement(s). This turns the comments back into statements, and will be compiled when the code is run.



- 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.



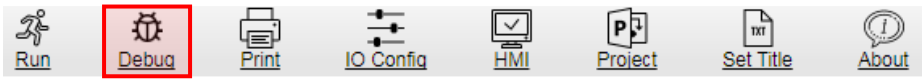
NOTE

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

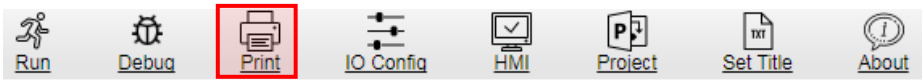
5.12.1 Debug (ARGEE PRO)

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.

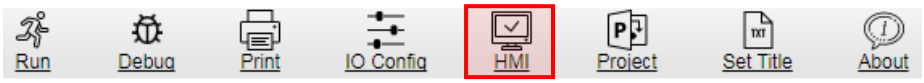


5.12.3 IO Config (I/O Configuration)

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

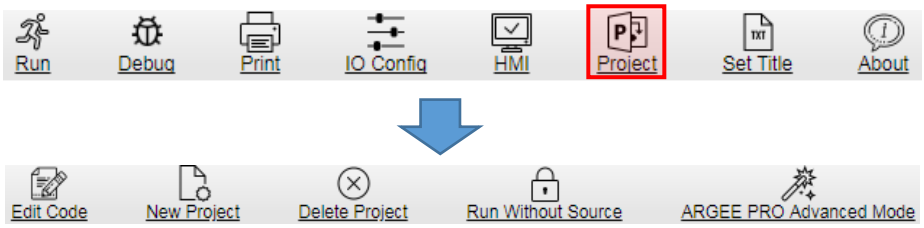
5.12.4 HMI

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.



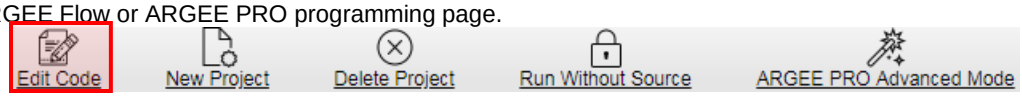
5.12.5 Project

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



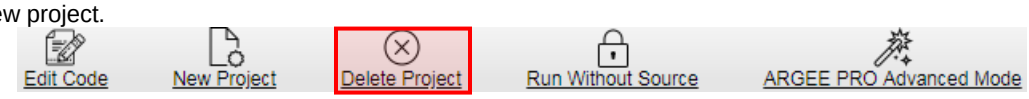
5.12.6 Edit Code

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.



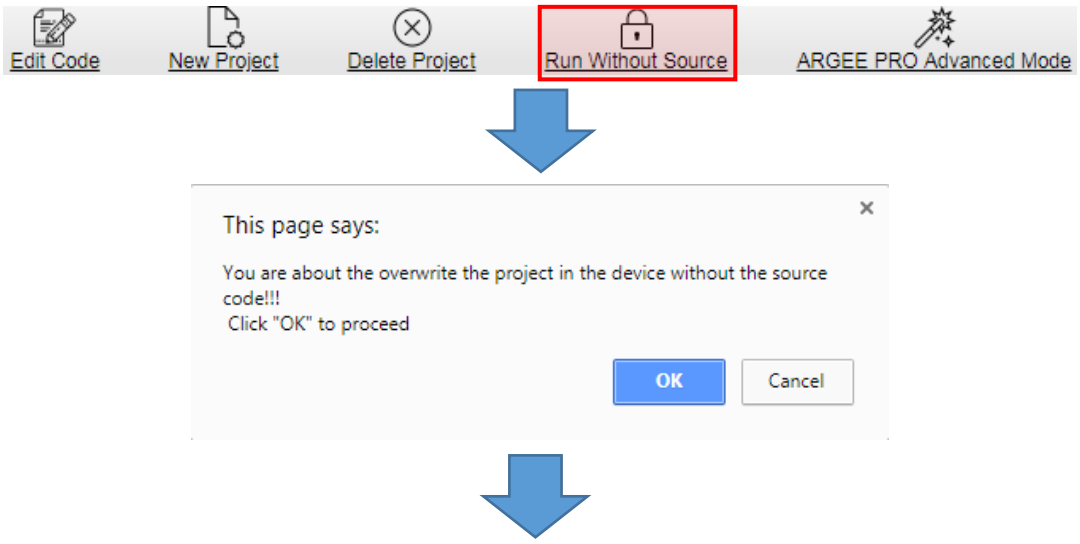
5.12.7 Delete Project

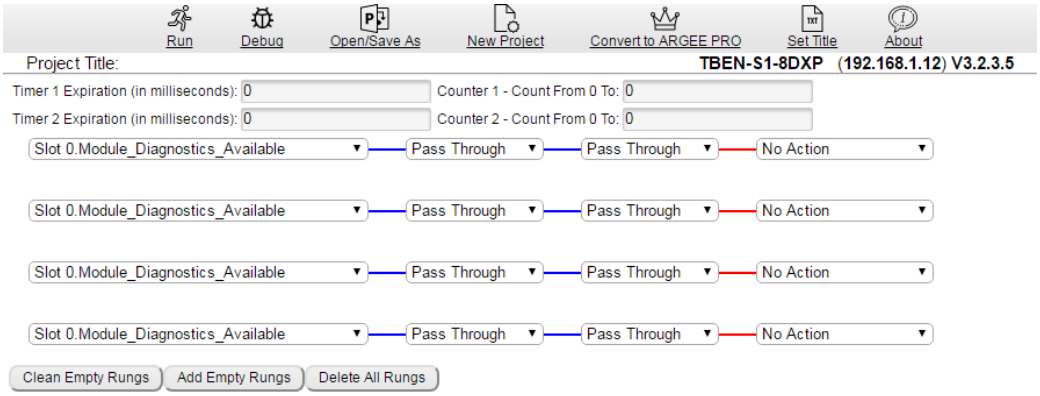
Delete Project is different from *New Project* because it erases the project from the device then starts a new project.



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:

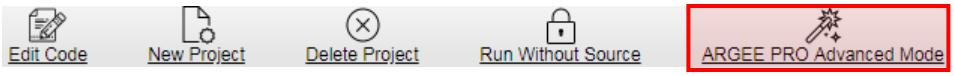
Project without the source code is loaded into the device
Erase it via the web server to be able to load new ARGEE programs!!!!

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

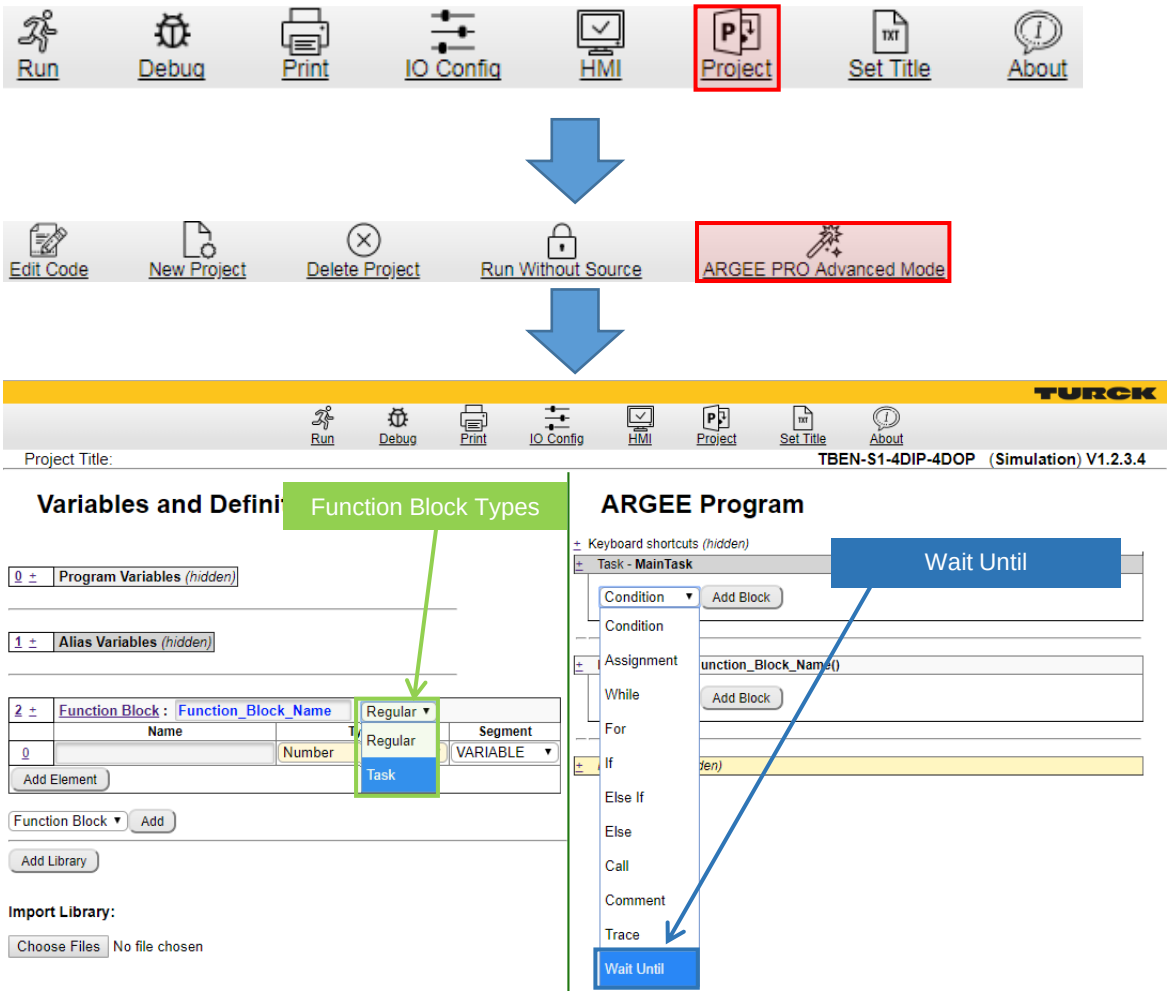
ARGEE PRO Advanced Mode is covered in greater detail in chapter 6 ARGEE PRO Advanced Mode.

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.

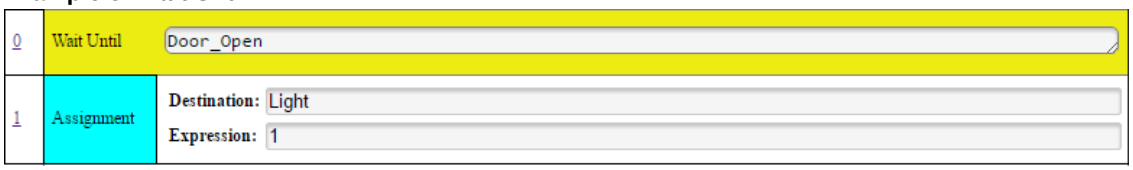
6.2.2 Task (Multitasking)

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

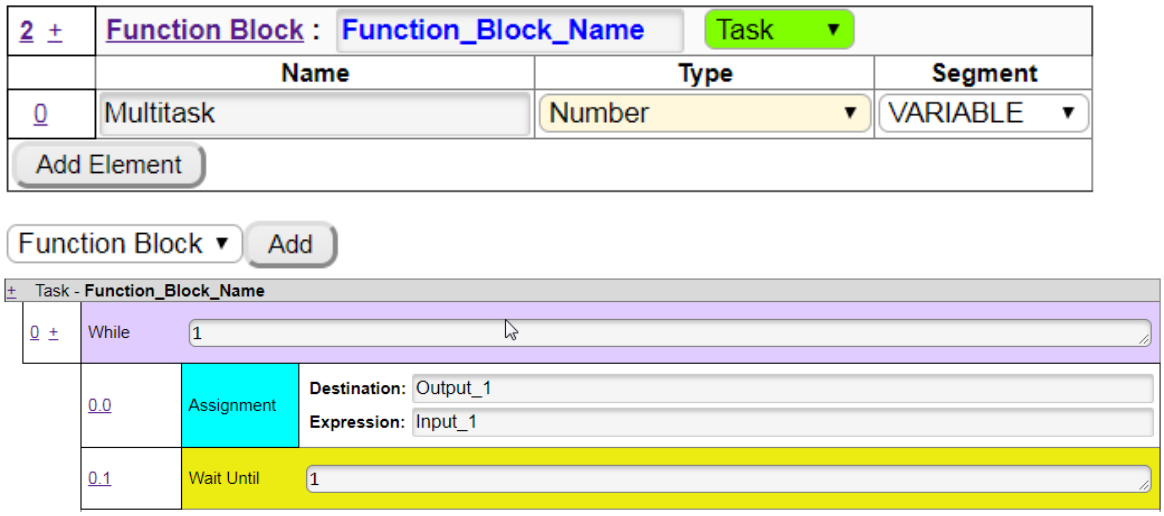
Wait Until is a very powerful statement that halts the execution of a task until a certain condition is met. .

Example of *Wait Until:*



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.



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

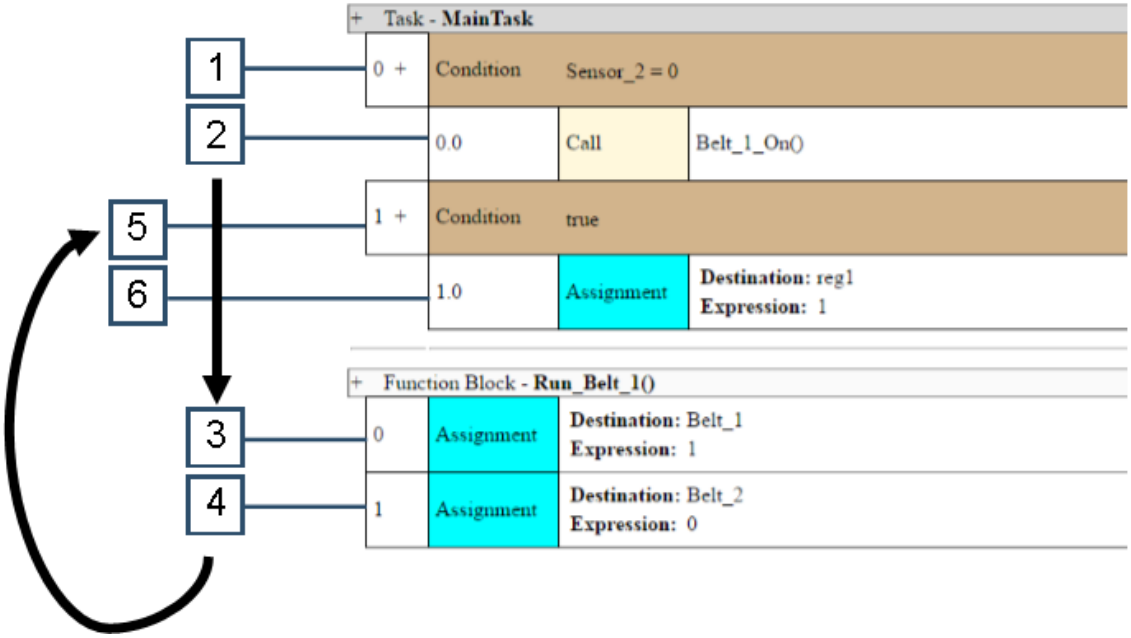
Trace is a very powerful Debug tool. The user will use the Trace function if they want to time stamp exactly when an event occurred. Trace can be used to measure a programs run-time behavior, how long each state takes and even which states were visited in which order.

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

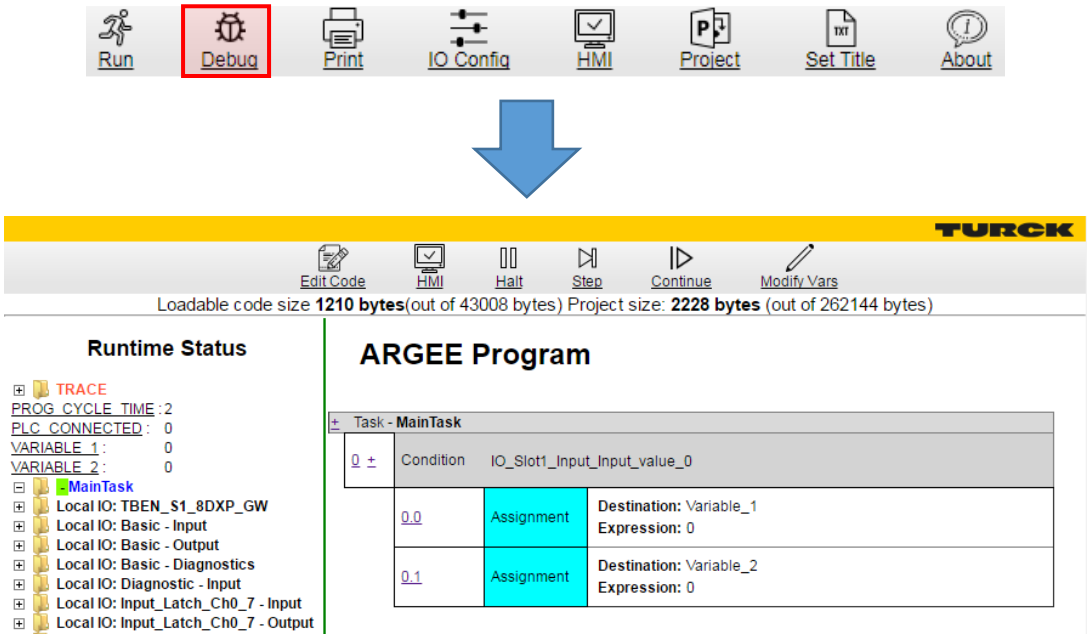
If the user is debugging an ARGEE program that contains function blocks, and is advancing the program one step at a time, they will find that the debugger appears to skip around the program when a Call block is reached. This is because when ARGEE executes a Call block, it jumps down to its function block's 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.



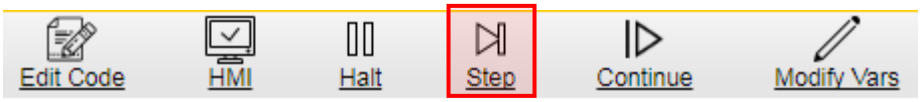
NOTE
The blue links in the left hand column are clickable and will center the window on that specific area of the code. Active conditional statements (while,for,ifs/conditions) show up as green. Inactive conditions show up as gray. Wait_until statements that are actively waiting will show with yellow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

7.2.1 Halt

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

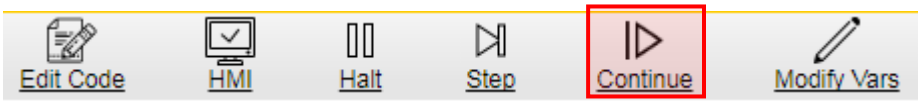
7.2.2 Step

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



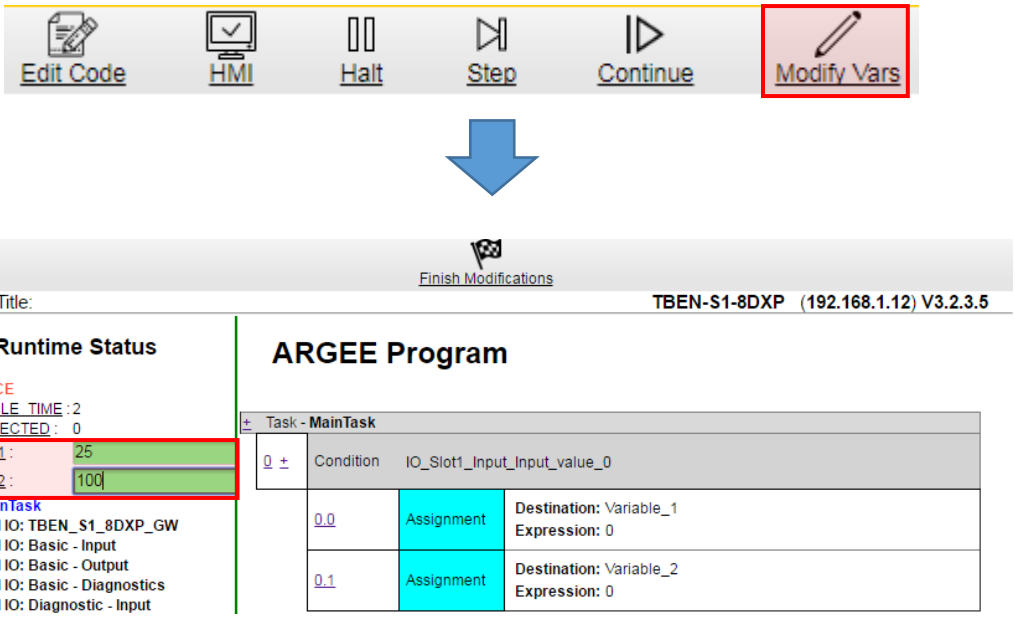
7.2.3 Continue

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



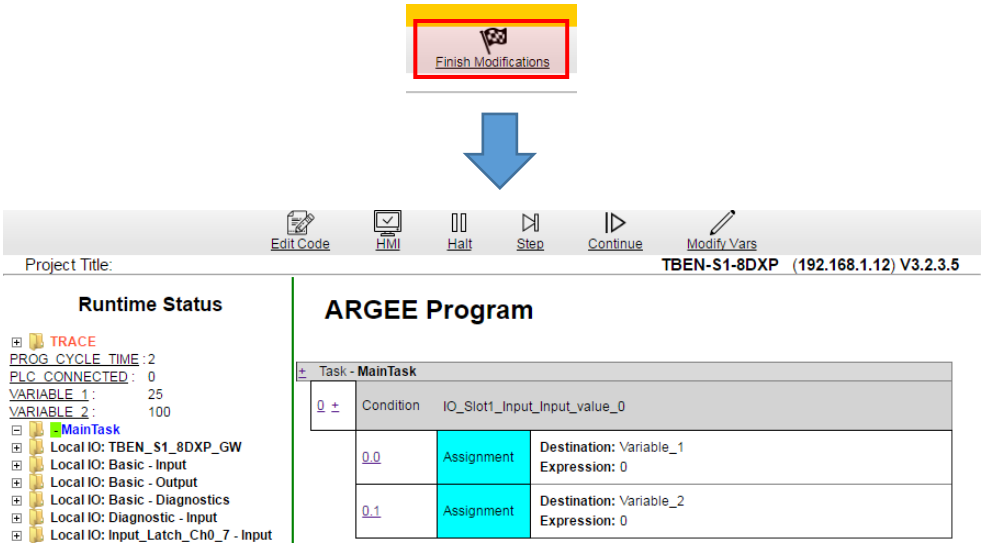
7.2.4 Modify Vars (Modify Variables)

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.



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.

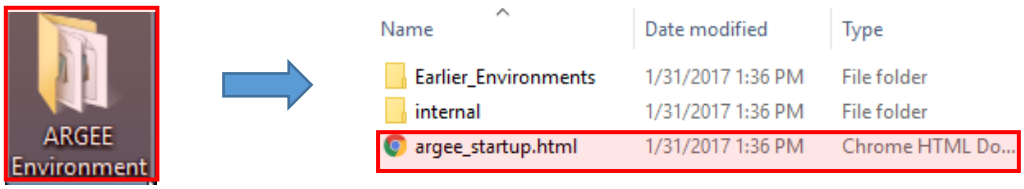


8 ARGEE Simulation Mode

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.

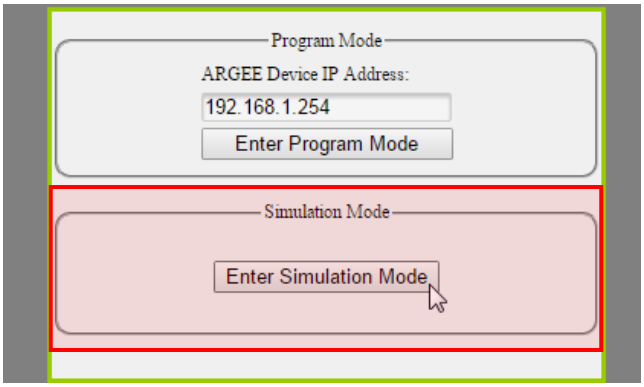


NOTE

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

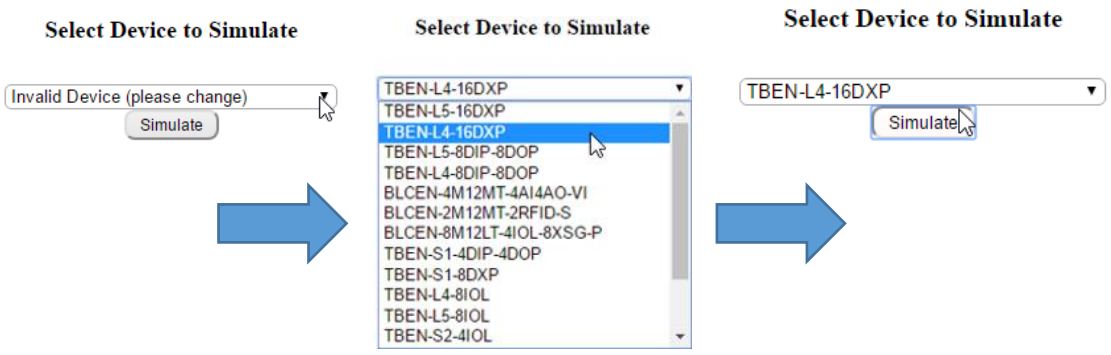
8.2 Logging into Simulation Mode

- Click Enter Simulation Mode.



8.3 Selecting Device to Simulate

Select a device to simulate from the drop down menu.

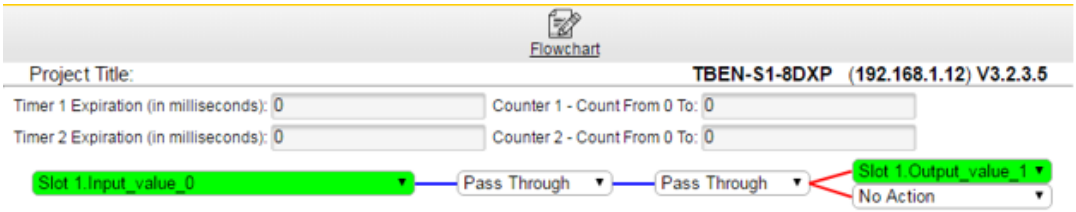


NOTE

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

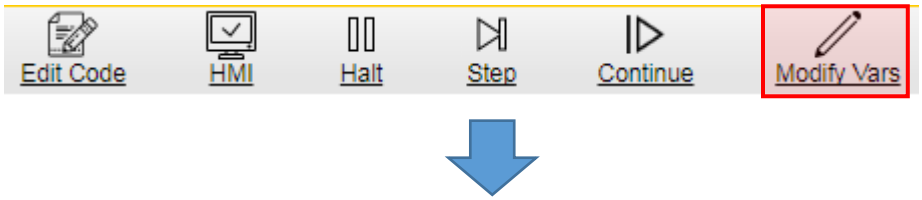
8.3.1 Flow Chart Simulation Mode

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

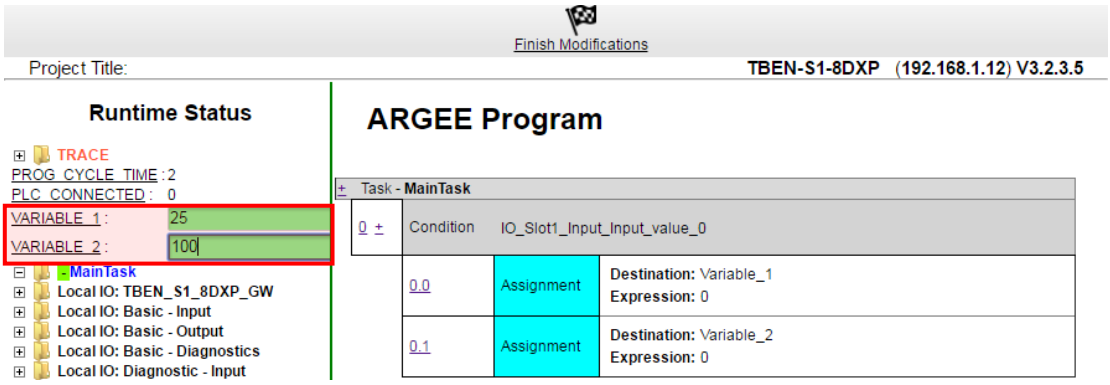


8.3.2 Pro Simulation Mode

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



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



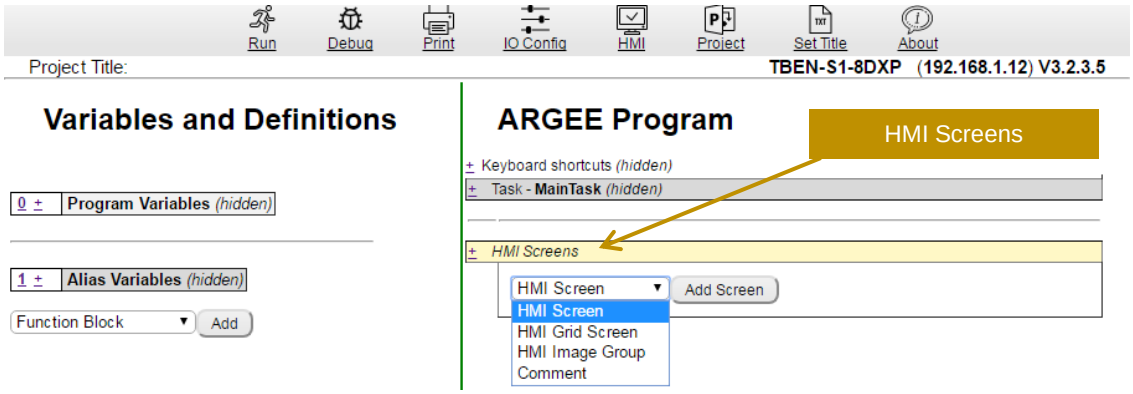
NOTE

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

9 ARGEE HMI

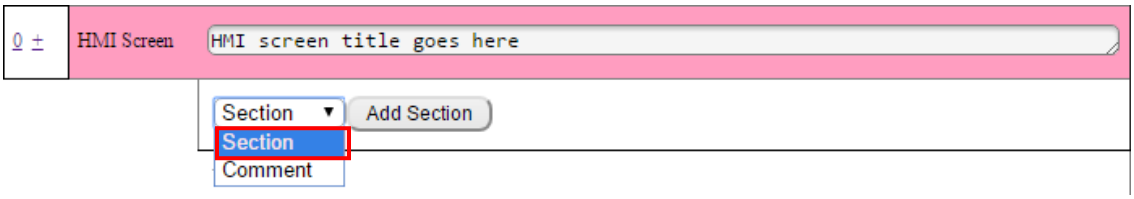
9.1 The Basics

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** in a Google Chrome or Firefox web browser.



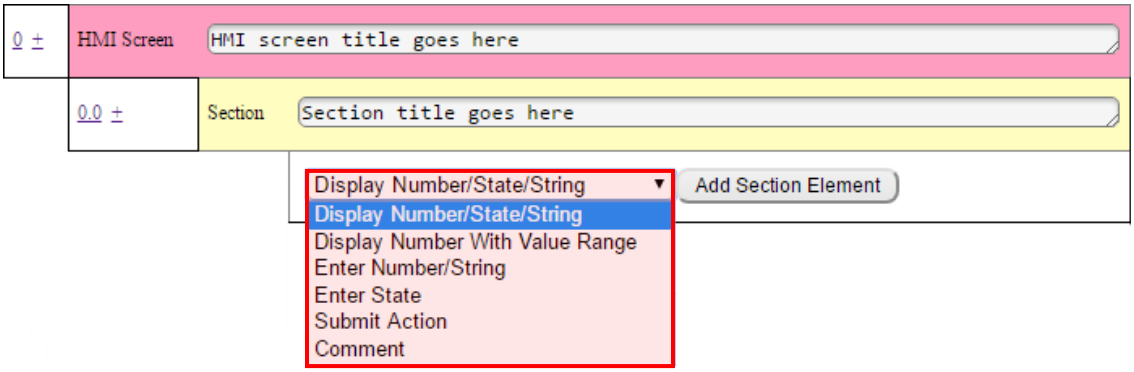
9.2 HMI Screen

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

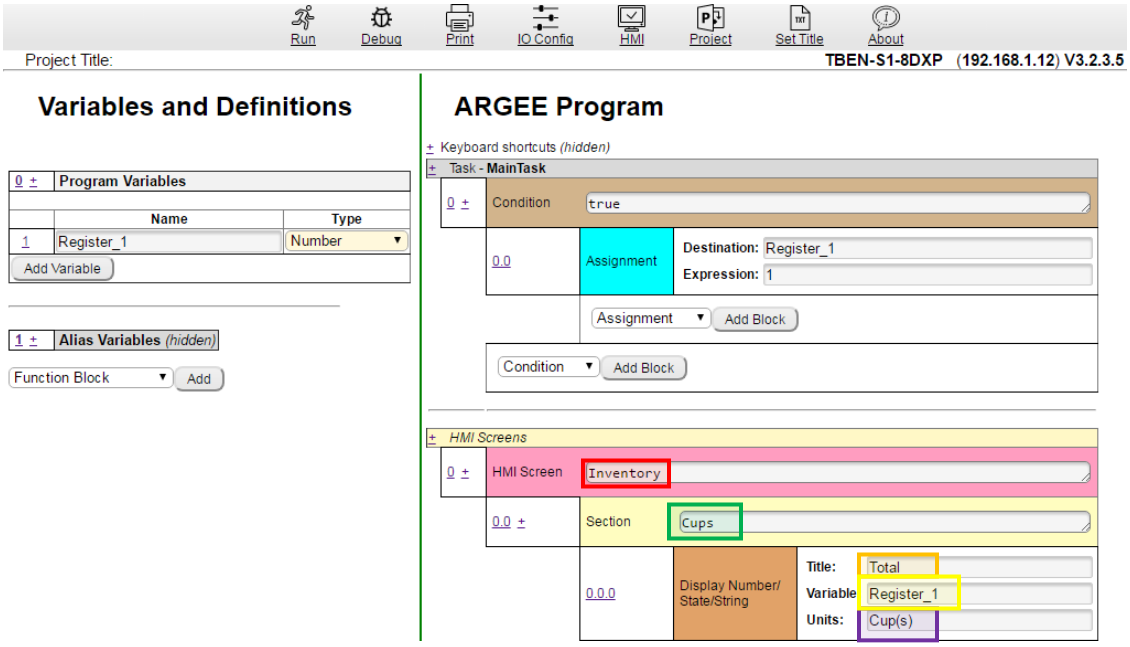
After the user adds a new section, the user will be able to add elements to the HMI screen by highlighting the desired element and clicking the *Add Section Element* button.



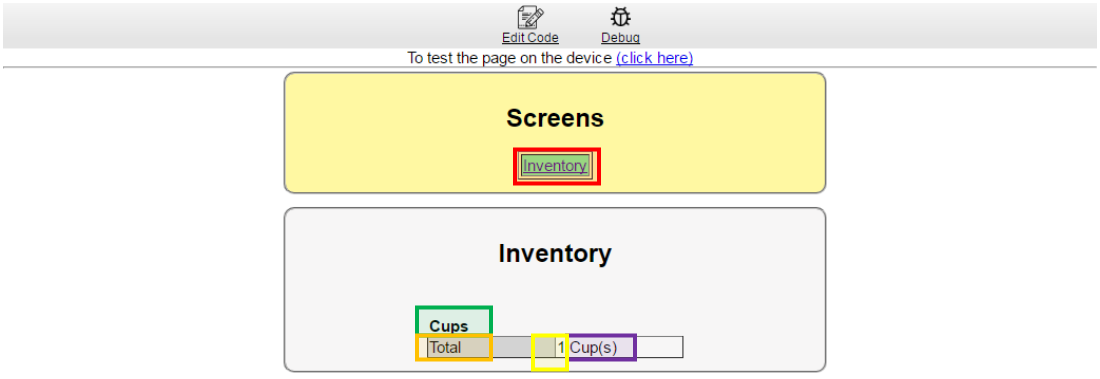
9.2.1.1 Display Number/State/String

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:*



(HMI View):

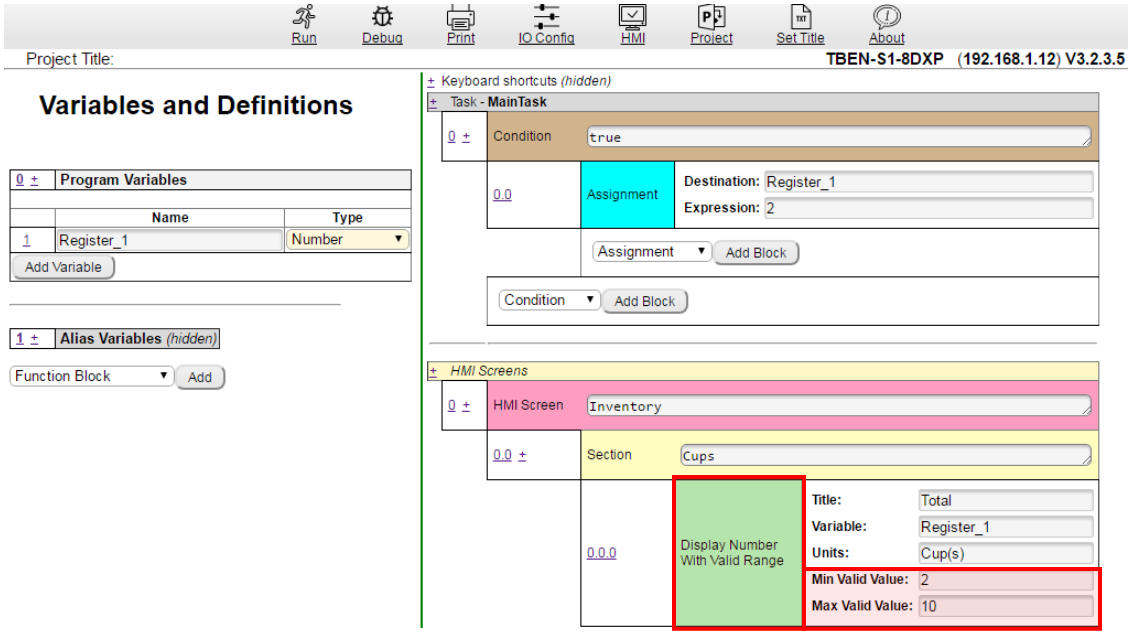


Explaining the Example: The user created an inventory HMI screen that shows how many cups they currently have. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar.

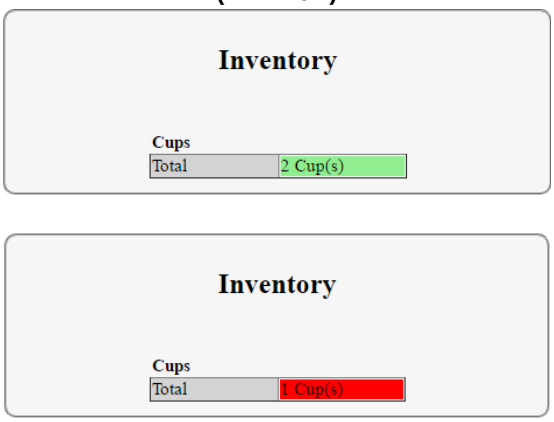
9.2.1.2 Display Number with Valid Range

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:*



(HMI View)

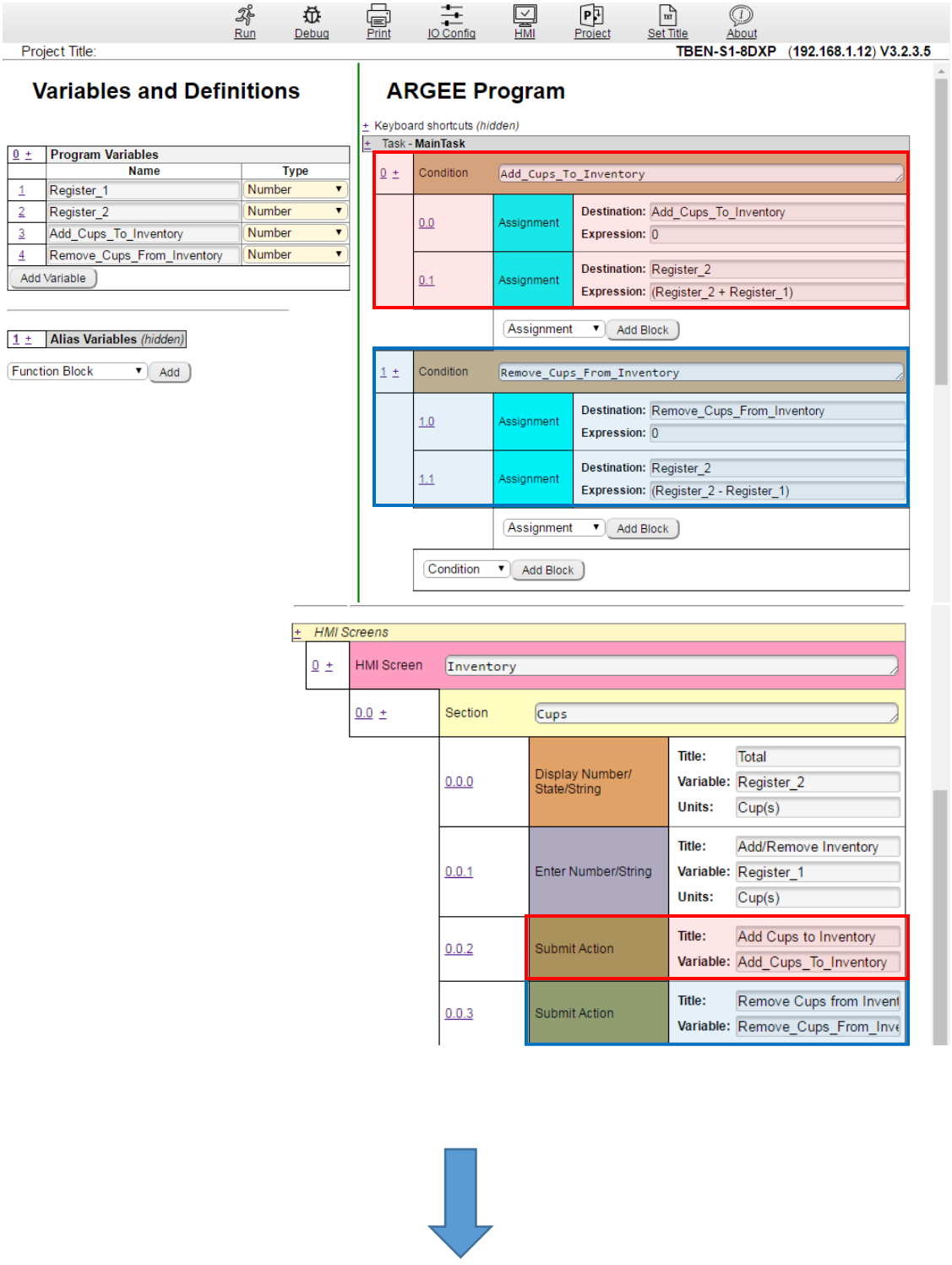


Explaining the Example: The user created an inventory HMI screen that shows how many cups the user currently has in inventory. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar. When there is only one cup left in the inventory, the HMI turns red, letting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

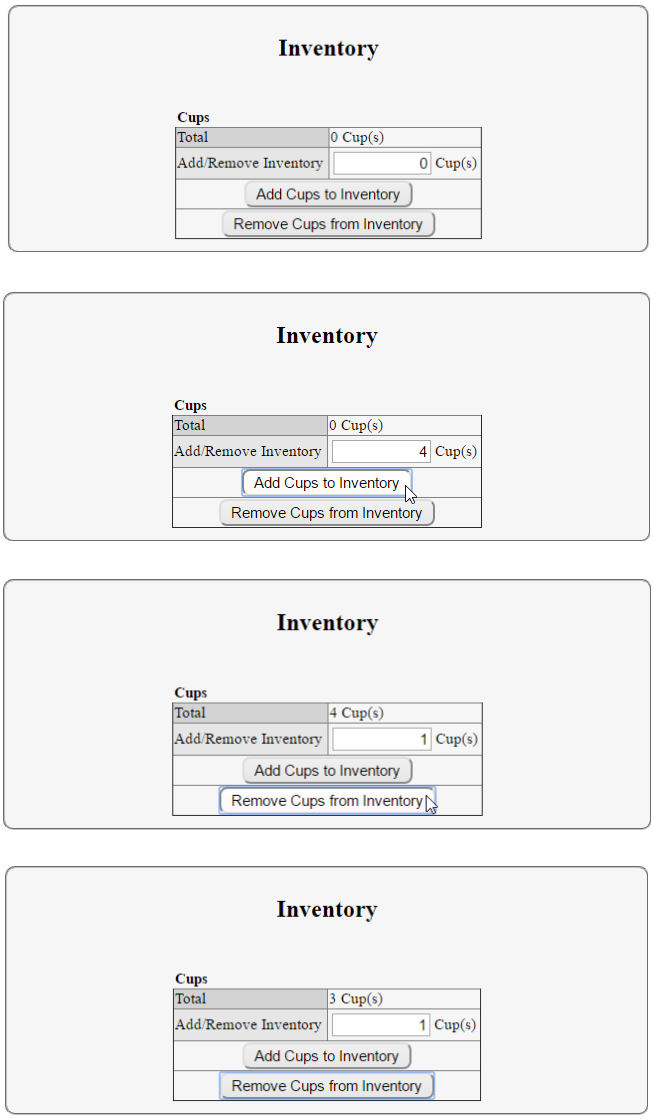
9.2.1.3 Enter Number/String

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:*



(HMI View)



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.

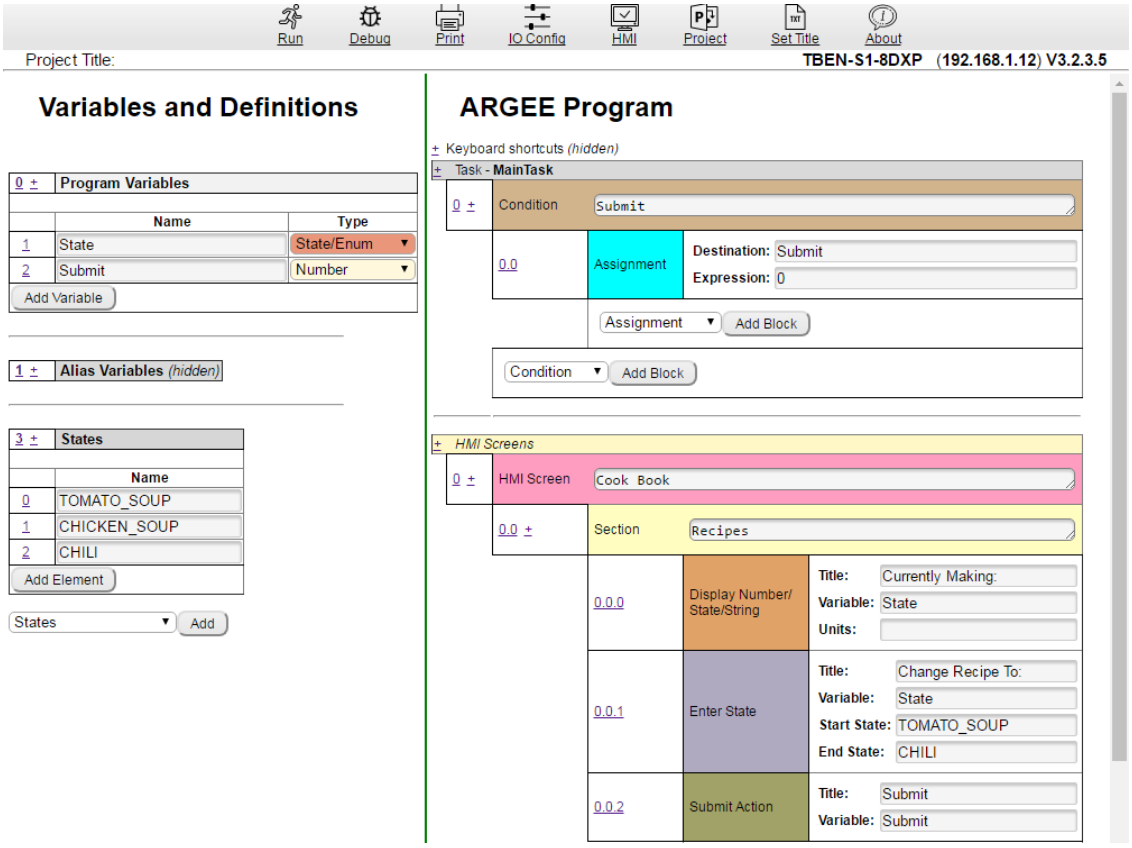
NOTE

The Submit Action element will be talked about later in this chapter in section 9.4.3 Action.

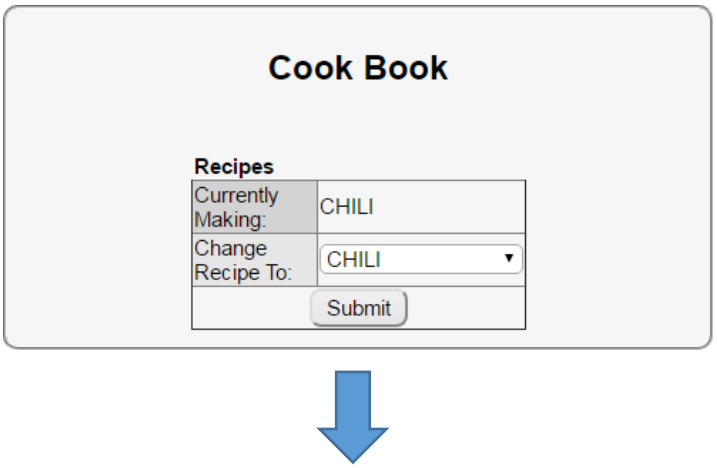
9.2.1.4 Enter State

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

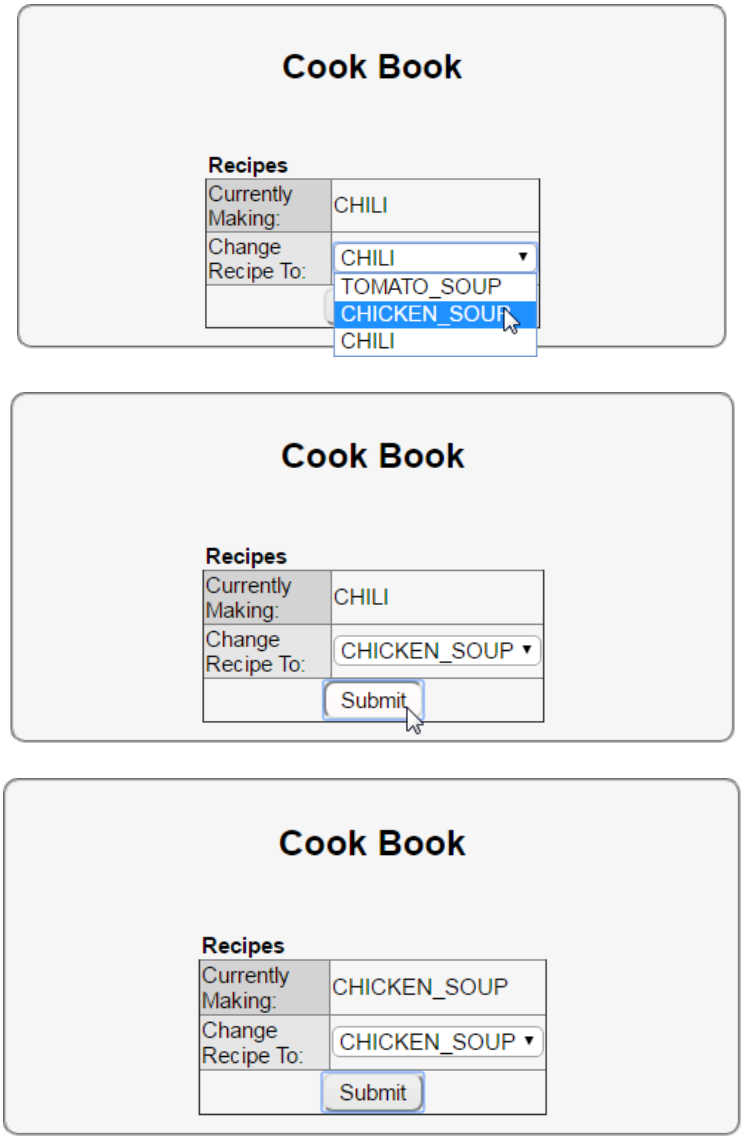
Example of *Enter State:*



(HMI View)



(HMI View)



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.

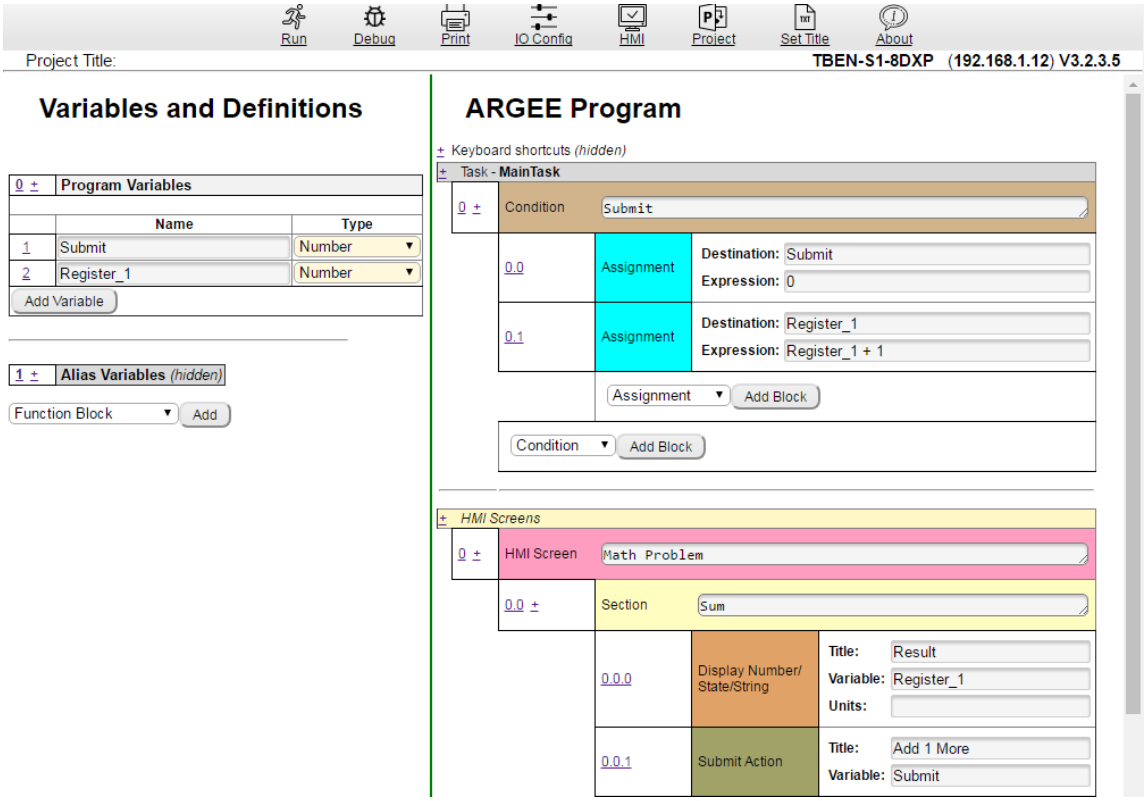
NOTE

The Submit Action element is explained in detail in section 9.2.1.5 Action.

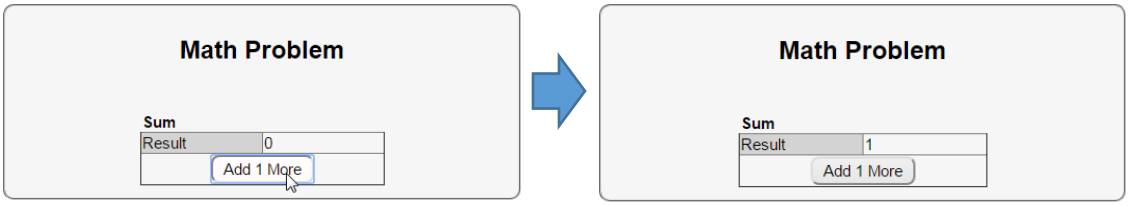
9.2.1.5 Submit Action

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:*



(HMI View)



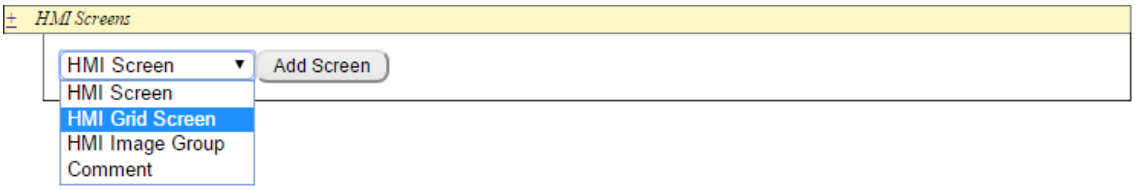
Explaining the Example: The user created a simple HMI which increase the current value in Register_1 by one every time the *Add 1 More* button is pressed. After the user wrote the code, the user clicked *Run* and then *View HMI* in the ARGEE menu bar. The user used the *Display State* element to display 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.

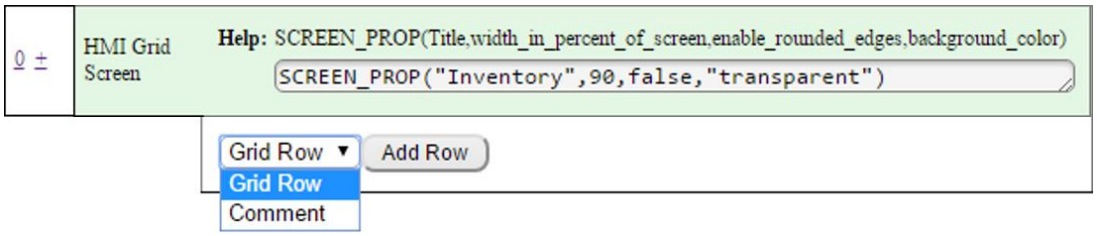


9.3.1 HMI Grid Screen

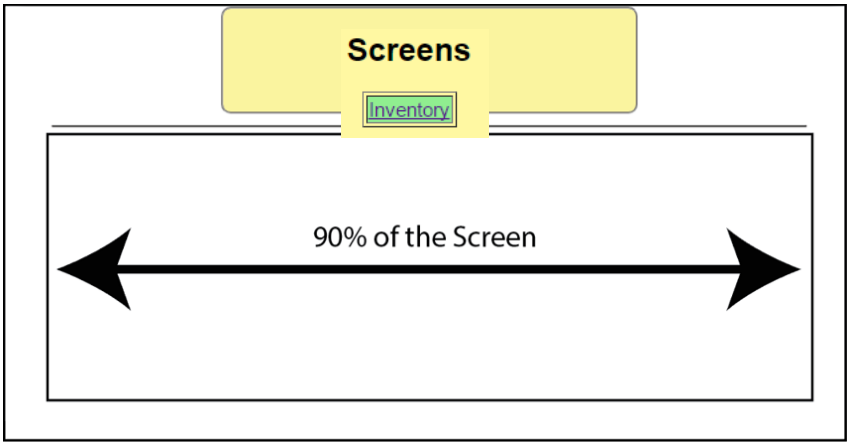
The HMI Grid Screen element has four arguments:

- 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

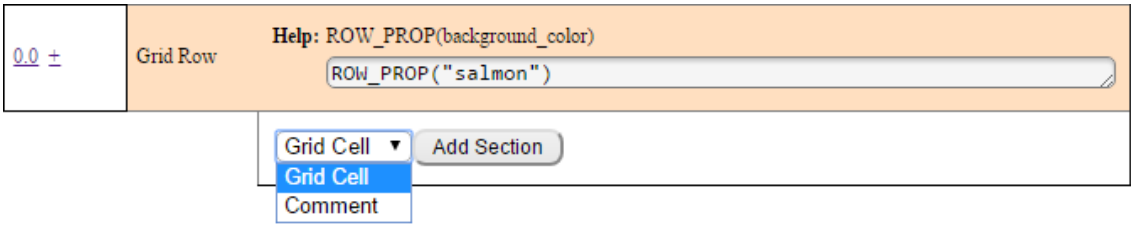
ARGEE supports all colors that your web browser supports. This user can either type in the Hex value or the X11 color name. For example: the user could type “whitesmoke” or “#F5F5F5.”

9.3.2 Grid Row

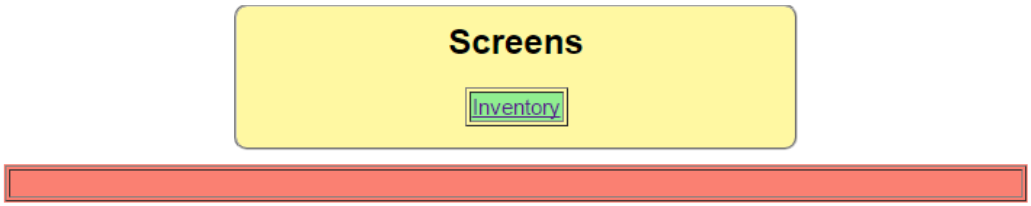
The *Grid Row* element has one argument:

- Background Color

Example of *Grid Row*:



(HMI View)



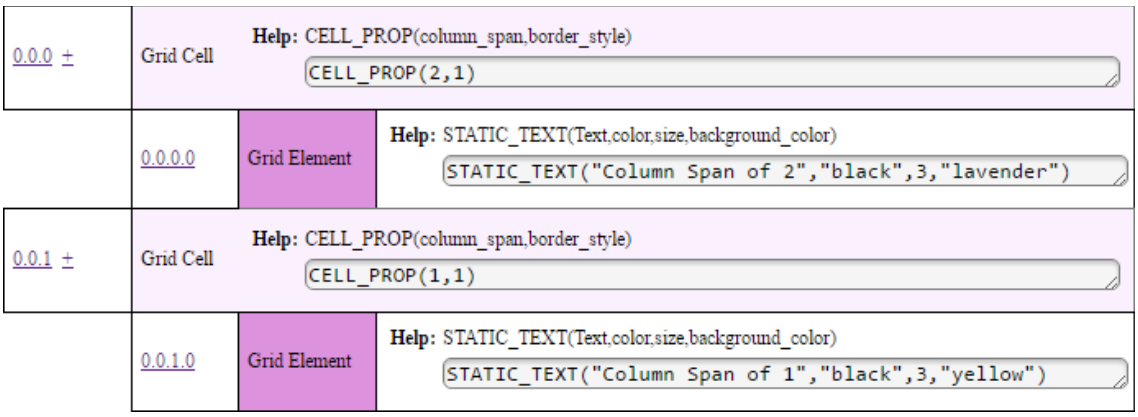
Explaining the example: The user added a row to his “Inventory” screen, and set the background color to salmon.

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*:



(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.

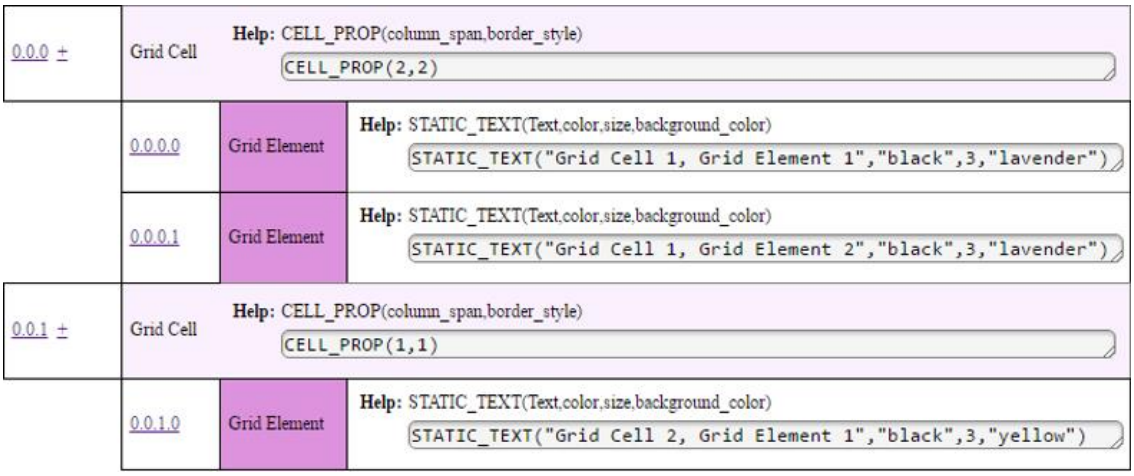
NOTE

The Static Text element will be discussed later in this chapter in section 9.6.4 Static Text.

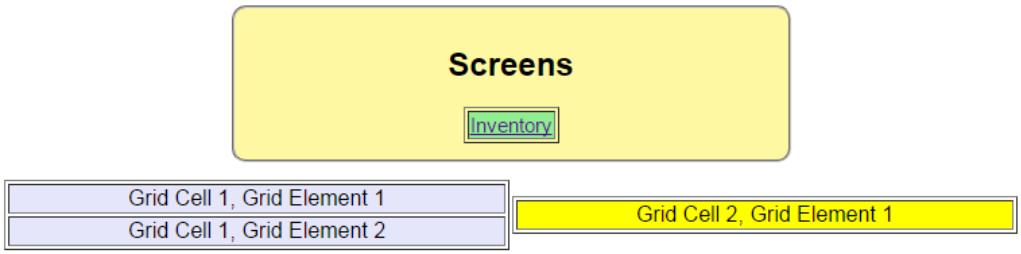
9.3.4 Grid Element

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*:



(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

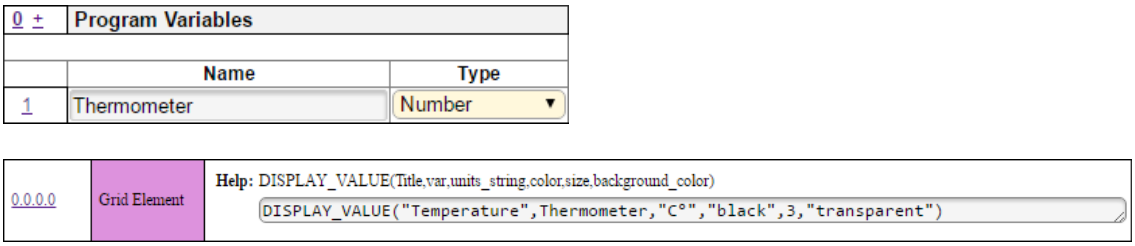
The Static Text element will be discussed later in this chapter in section 9.3.4.4 Static Text.

9.3.4.1 Display Value

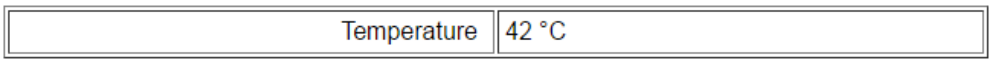
The *Display Value* element has six arguments:

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

Example of *Display Value*:



(HMI View)



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	**Help:** DISPLAY_VALUE(Title,var,units_string,color,size,background_color) `DISPLAY_VALUE("Total Inventory",Total,"units","black",3,"transparent")`
0.0.0.1	Grid Element	**Help:** ENTER_VALUE(Title,var,units_string,color,size,background_color) `ENTER_VALUE("Inventory Added", Inventory,"unit","black",3,"transparent")`
0.0.0.2	Grid Element	**Help:** BUTTON(Title,var,color,size,background_color) `BUTTON("Add to Inventory",Submit,"black",3,"transparent")`

(HMI View)

Total Inventory	0 units
Inventory Added	42 unit
Add to Inventory	

Total Inventory	42 units
Inventory Added	42 unit
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	**Help:** DISPLAY_VALUE(Title,var,units_string,color,size,background_color) `DISPLAY_VALUE("Total",Total,"unit","black",3,"transparent")`
0.0.0.1	Grid Element	**Help:** BUTTON(Title,var,color,size,background_color) `BUTTON("Add 1",Total,"black",3,"transparent")`

(**HMI View)**

Total	0 unit
Add 1	

Total	1 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	**Help:** STATIC_TEXT(Text,color,size,background_color) `STATIC_TEXT("Cell 8 Robot Status","black",3,"transparent")`

(HMI View)

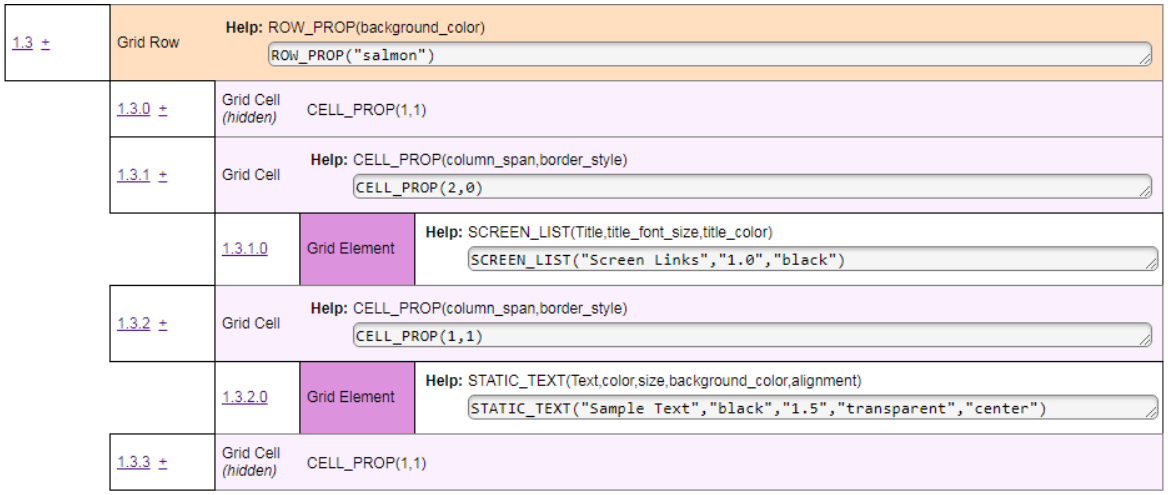
Explaining the example: The user wants to label certain information in the HMI "Cell 8 Robot Status."

9.3.4.5 Screen List

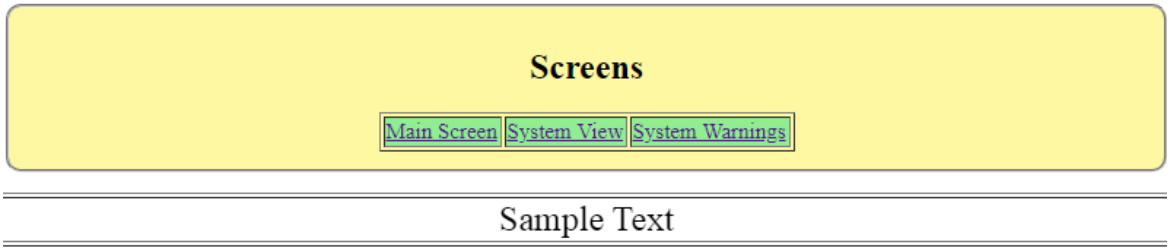
The developer can use the *Screen List* element to have more control over the HMI Screens. They can move the location of where it is displayed, alter the background color, and change the text size.

- Title
- Title Font Size
- Title Color

Example of *Screen List*:



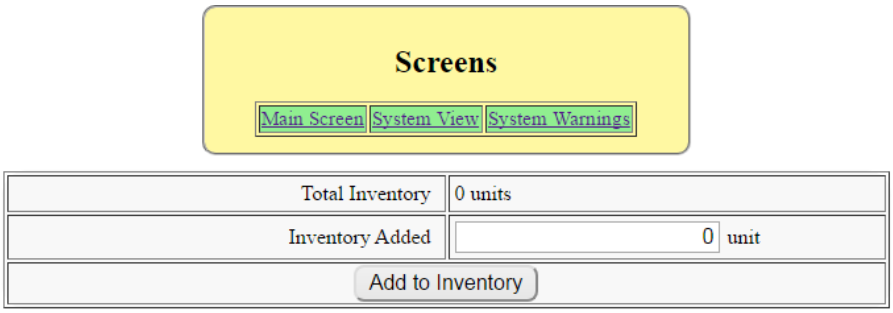
(HMI View without the *Screen List* element)



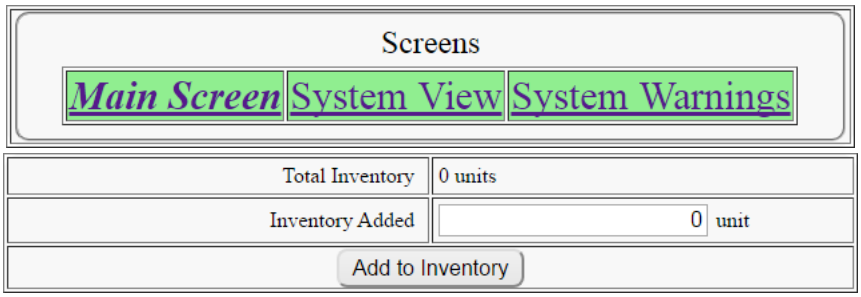
(HMI View with the *Screen List* element)

0.0.0.0	Grid Element	**Help:** SCREEN_LIST(Title,title_font_size,title_color) `SCREEN_LIST("Screens",5,"black")`

(HMI View without the *Screen List* element)



(HMI View with the *Screen List* element)



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	**Help:** 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
- …

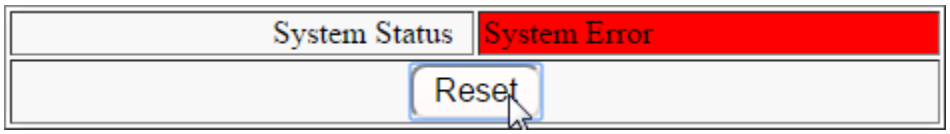
Example of *Multi-State Display String:*

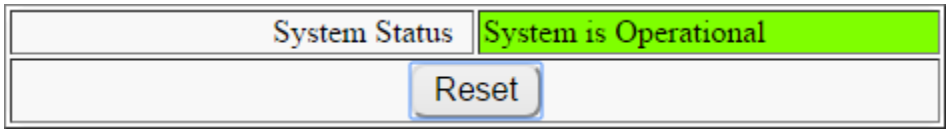
0 +	Program Variables	
	Name	Type
1.0	**INIT** : System_ERROR	
1	System_State	Number
2	Submit	Number

3 +	States
	Name
0	System_OK
1	System_ERROR

0.0.0.0	Grid Element	**Help:** MULTI_STATE_DISPLAY_STRING(Title,var,size,title_color,title_background_color, value1,color1,background1,.....) `MULTI_STATE_DISPLAY_STRING("System Status",System_State,3,"black","transparent", 0, "System is Operational","black","chartreuse", 1, "System Error", "black", "red")`
0.0.0.1	Grid Element	**Help:** BUTTON(Title,var,color,size,background_color) `BUTTON("Reset",Submit,"black",3,"transparent")`

(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
- …

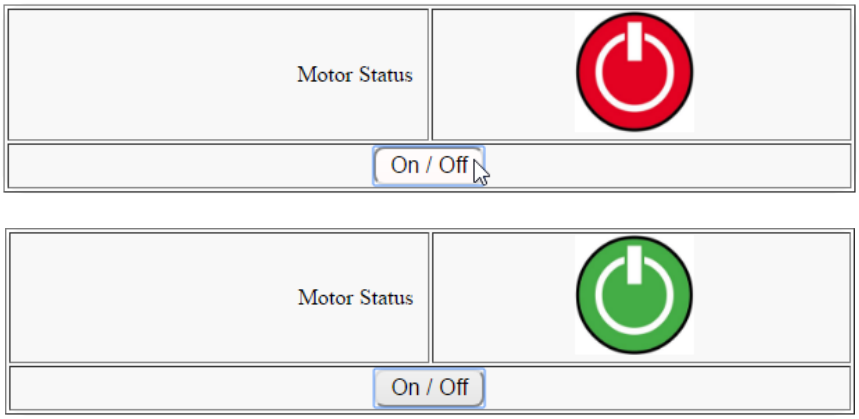
Example of Multi-State Display Graphics:

0 +	Program Variables	
	Name	Type
1	Current_Motor_State	State/Enum
2	Submit	Number

3 +	States
	Name
0	Motor_OFF
1	Motor_ON

0.0.0.0	Grid Element	**Help:** MULTI_STATE_DISPLAY_GRAPHICS(Title,var,title_size,title_color,title_background_color,image_zoom_level,value1,image1,background1...) `MULTI_STATE_DISPLAY_GRAPHICS("Motor Status",Current_Motor_State,3,"black","transparent",30, 0,"Red Button","transparent",1,"Green Button","transparent")`
0.0.0.1	Grid Element	**Help:** BUTTON(Title,var,color,size,background_color) `BUTTON("Submit",Submit,"black",3,"transparent")`

(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

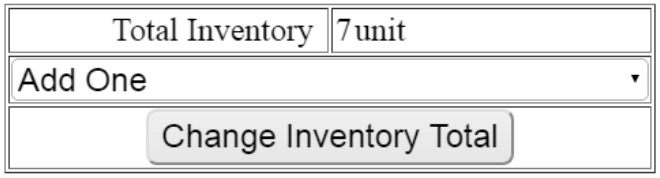
Dropdown List is used to give the user a list of options to change a variable. The *Dropdown List* element has at least 9 arguments, more maybe used depending on how many options the developer needs:

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



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:** DISPLAY_VALUE_WITH_HEALTH(Title,title_color,size,var,units_string,health_var) `DISPLAY_VALUE_WITH_HEALTH("Inventory","black",5,Total,"Cups",Health_Variable)`

(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:*

0.0.0.0	Grid Element	**Help:** LINK(Title_var,value_var,background_color_var,size) `LINK(Fridge_Title_Variable,Fridge_Link_Variable,Fridge_Color_Variable,3)`
0.2.1.0	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

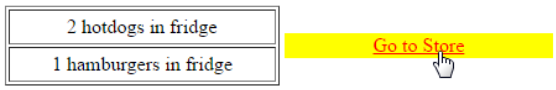
(HMI View)

"Fridge" HMI Screen:

7248 hotdogs in fridge
3624 hamburgers in fridge

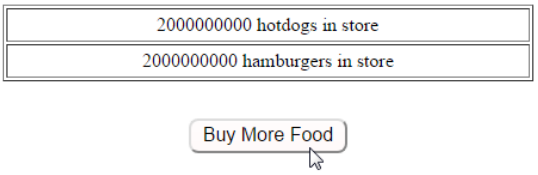
Eating food…

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

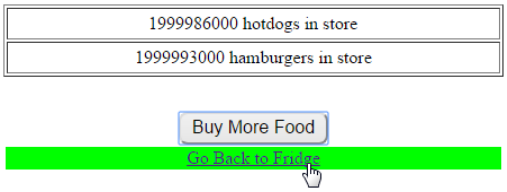


“Store” HMI Screen:

Buying food…



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

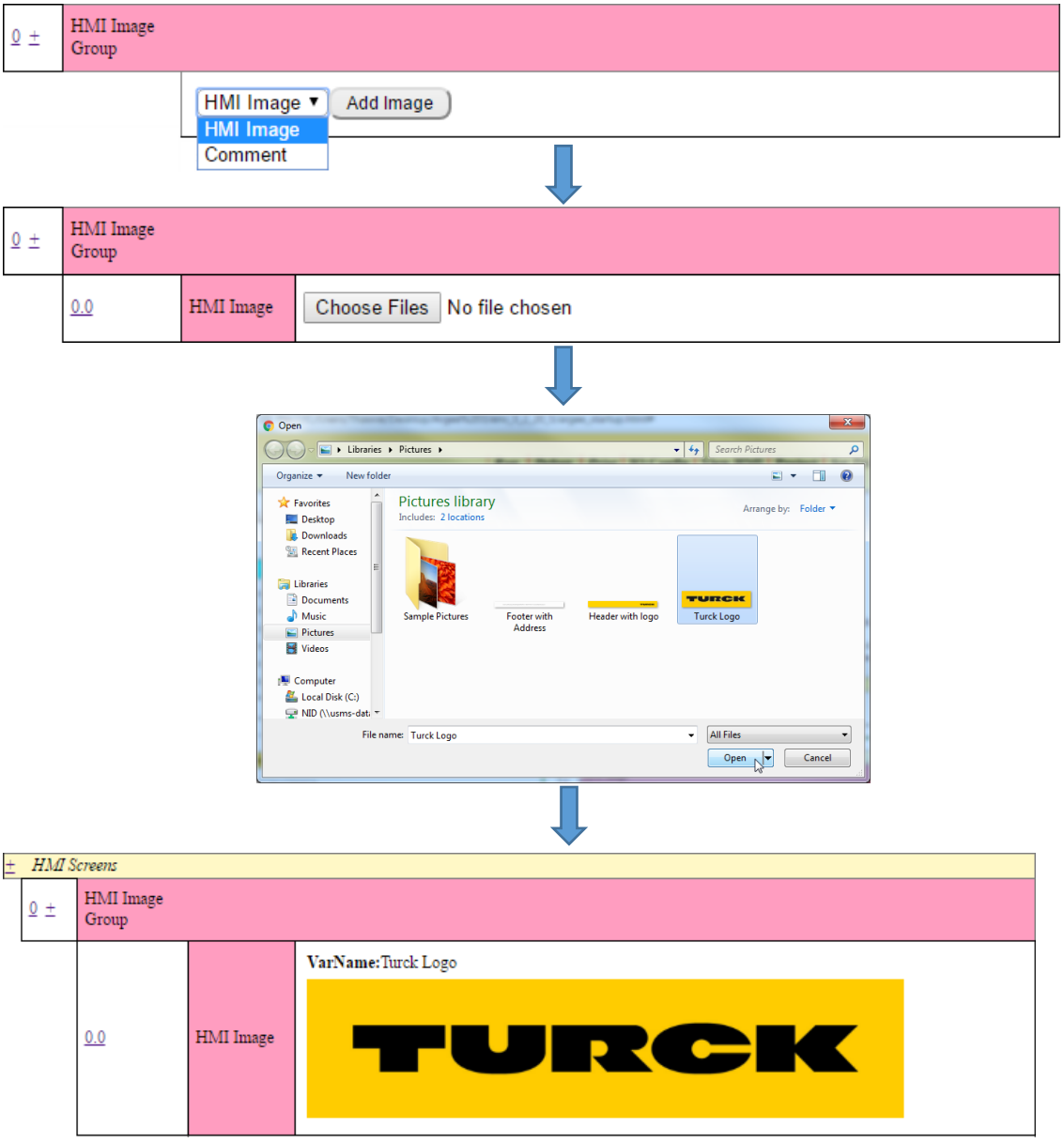


Explaining the example: The user wrote code that decreases the number of hotdogs and hamburgers in their fridge over time. If the count gets too low, the yellow link for the “Store” HMI screen is made visible so more can be bought. After enough food is bought at the store, the green link for the “Fridge”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

1.1 +	Grid Row	**Help:** ROW_PROP(background_color) `ROW_PROP("transparent")`		
1.1.0 +	Grid Cell	**Help:** CELL_PROP(column_span,border_style) `CELL_PROP(1,1)`		
	1.1.0.0	Grid Element	**Help:** DISPLAY_VALUE(Title,var,units_string,color,size,background_color) `DISPLAY_VALUE("Total Inventory",Total," unit","black","1.5","transparent")`	
	1.1.0.1	Grid Element	**Help:** ENTER_VALUE(Title,var,units_string,color,size,background_color) `ENTER_VALUE("Inventory Added",Inventory,"`	
	1.1.0.2	Grid Element	**Help:** BUTTON(Title,var,color,size,background_color) `BUTTON("Submit",Submit,"black","1.5","transparent")`	

Total Inventory	0 unit
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.

1.0 +	Grid Row	**Help:** ROW_PROP(background_color) `ROW_PROP("transparent")`		
1.0.0 +	Grid Cell *(hidden)*	CELL_PROP(1,1)		
1.0.1 +	Grid Cell	**Help:** CELL_PROP(column_span,border_style) `CELL_PROP(1,1)`		
	1.0.1.0	Grid Element	**Help:** DISPLAY_VALUE(Title,var,units_string,color,size,background_color) `DISPLAY_VALUE("Total Inventory",Total," unit","black","1.5","transparent")`	
	1.0.1.1	Grid Element	**Help:** ENTER_VALUE(Title,var,units_string,color,size,background_color) `ENTER_VALUE("Inventory Added",Inventory,"`	
	1.0.1.2	Grid Element	**Help:** BUTTON(Title,var,color,size,background_color) `BUTTON("Submit",Submit,"black","1.5","transparent")`	
	Grid Element ▾ Add Element			
1.0.2 +	Grid Cell *(hidden)*	CELL_PROP(1,1)		

	Total Inventory	0 unit
	Inventory Added	0 unit
	Submit	

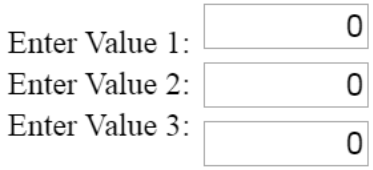
*The spacer cells only have borders so they are easier to view for this documentation, the boarder would usually be set to 0 no border.

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.

1.1 +	Grid Row	**Help:** ROW_PROP(background_color)	`ROW_PROP("transparent")`	
1.1.0 +	Grid Cell	**Help:** CELL_PROP(column_span,border_style)	`CELL_PROP(1,0)`	
1.1.0.0	Grid Element	**Help:** STATIC_TEXT(Text,color,size,background_color,alignment)	`STATIC_TEXT("Enter Value 1:","black","1.5","transparent","center")`	
1.1.0.1	Grid Element	**Help:** STATIC_TEXT(Text,color,size,background_color,alignment)	`STATIC_TEXT("Enter Value 2:","black","1.5","transparent","center")`	
1.1.0.2	Grid Element	**Help:** STATIC_TEXT(Text,color,size,background_color,alignment)	`STATIC_TEXT("Enter Value 3:","black","1.5","transparent","center")`	
1.1.1 +	Grid Cell	**Help:** CELL_PROP(column_span,border_style)	`CELL_PROP(1,0)`	
1.1.1.0	Grid Element	**Help:** ENTER_VALUE(Title,var,units_string,color,size,background_color)	`ENTER_VALUE("",Test,"","black","1.5","transparent")`	
1.1.1.1	Grid Element	**Help:** ENTER_VALUE(Title,var,units_string,color,size,background_color)	`ENTER_VALUE("",Test,"","black","1.5","transparent")`	
1.1.1.2	Grid Element	**Help:** ENTER_VALUE(Title,var,units_string,color,size,background_color)	`ENTER_VALUE("",Test,"","black","1.5","transparent")`	
1.1.2 +	Grid Cell	**Help:** CELL_PROP(column_span,border_style)	`CELL_PROP(8,0)`	



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

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

Enter Value 1: 0

Enter Value 2: 0

Enter Value 3: 0

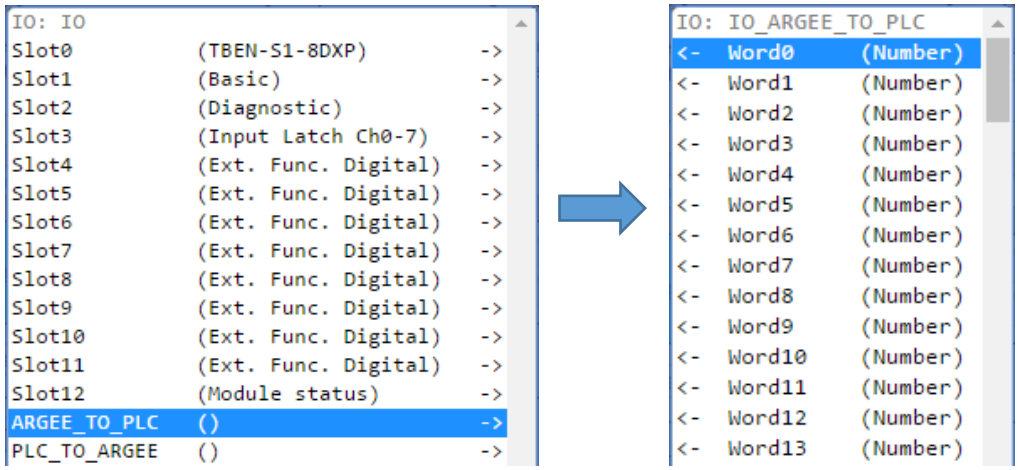
10 PLC Connectivity

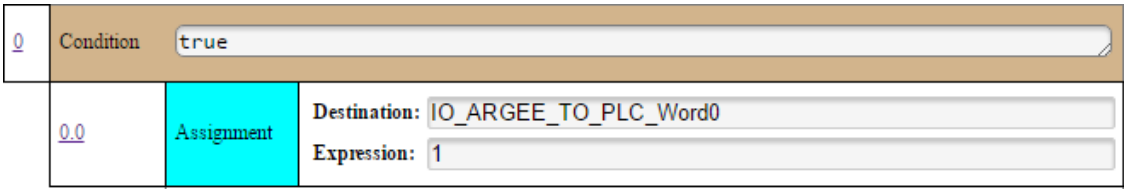
10.1 Communicating with EtherNet/IP Master – RSLogix5000 / Studio5000

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.

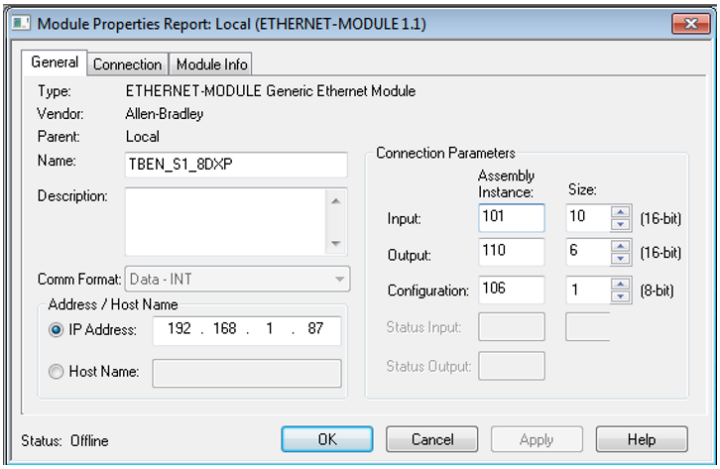
Example of *Communicating with an EtherNet/IP Master:*

(ARGEE Setup)

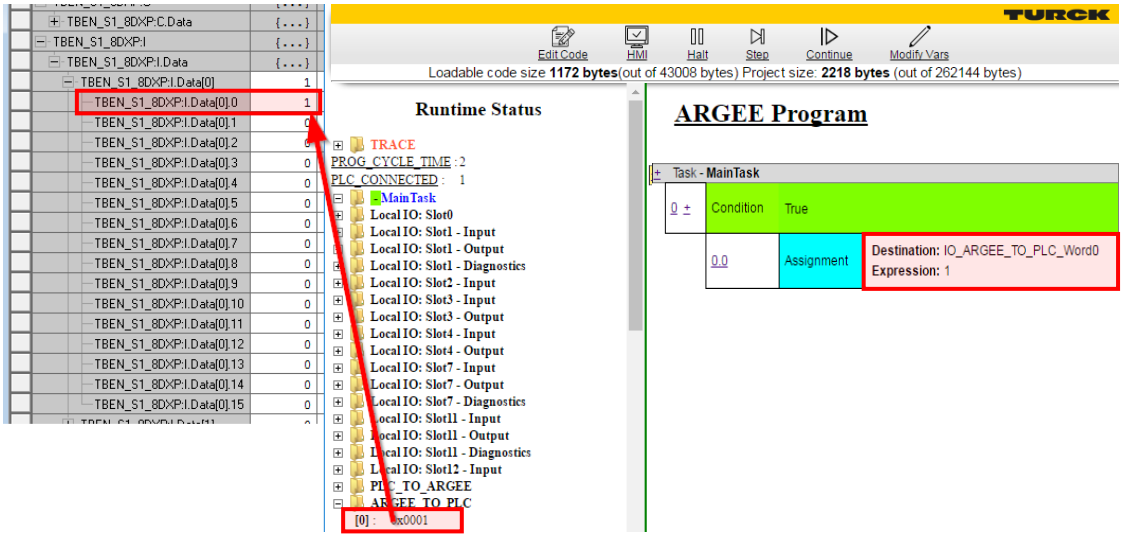


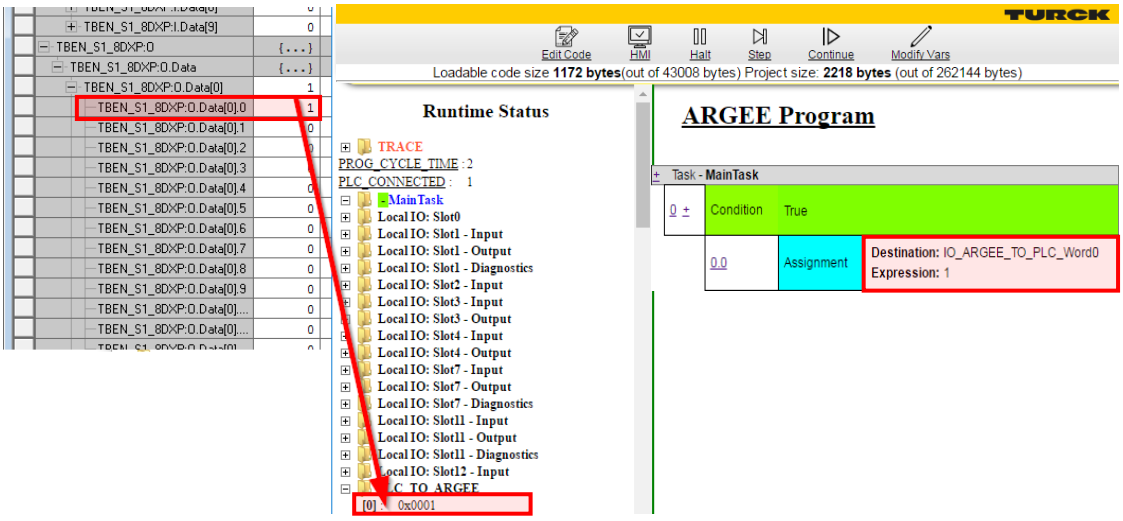


(RSLogix 5000 Setup)



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**.**





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

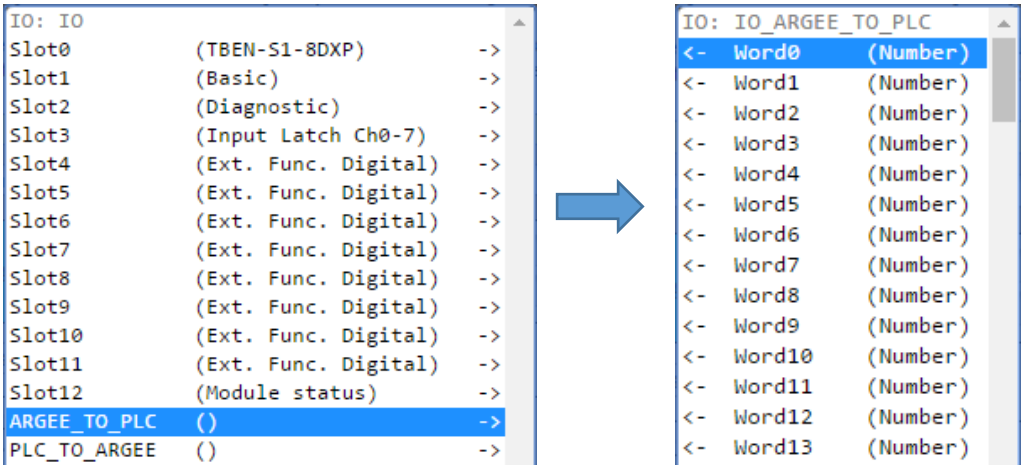
If the user wants to accomplish "bit offsetting," they need to manually adjust *the IO Variable Format* (discussed in 11.4.2 IO Variable Formats). For example, the user wants to force word 0 bit 5 true, the destination variable would be IO_ARGEE_TO_PLC_Word0.5

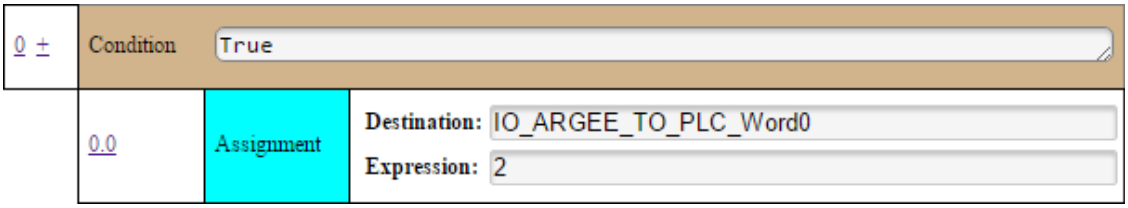
10.2 Communicating with a PROFINET Master – SIMATIC STEP 7

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

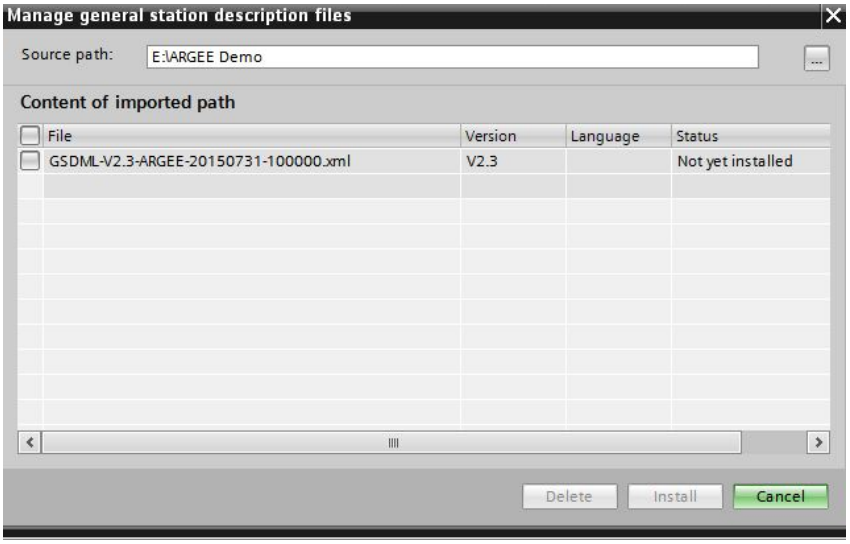
Example of Communicating with a PROFINET Master:

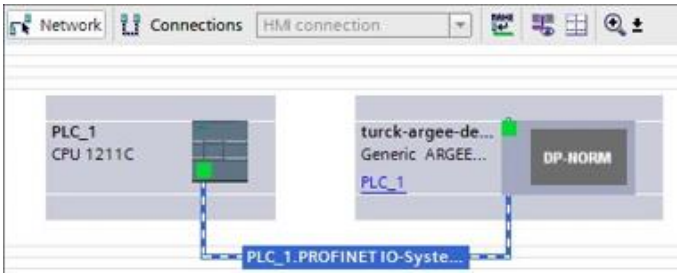
(ARGEE Setup)





(SIMATIC STEP 7 Setup)

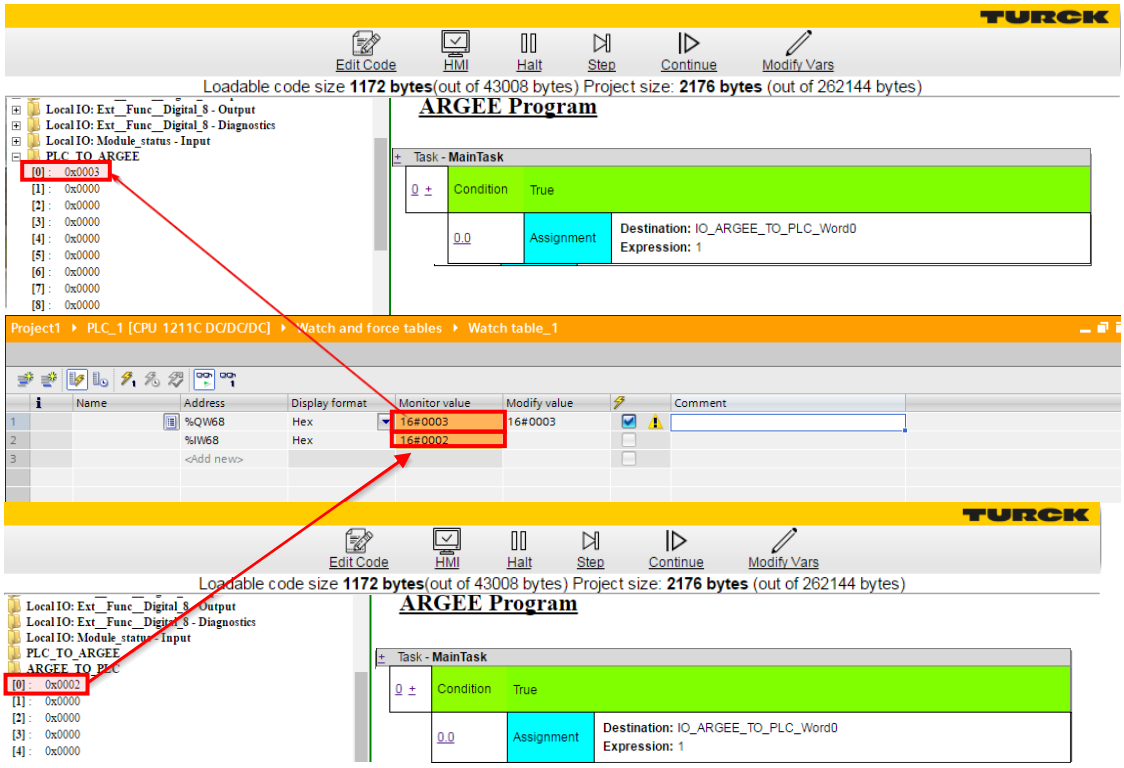




Device overview

	Module	Rack	Slot	I address	Q address	Type
	argee	0	0			ARGEE PN Device
	PN-IO	0	0 X1			turck-argee-device
	IN 8 WORD_1	0	1	68...83		IN 8 WORD
	OUT 8 WORD_1	0	2		68...83	OUT 8 WORD

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_PLC register. The user then defines the "I address" and "Q address" from the Step 7 Device overview screen.



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_PLC_Word0.5

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

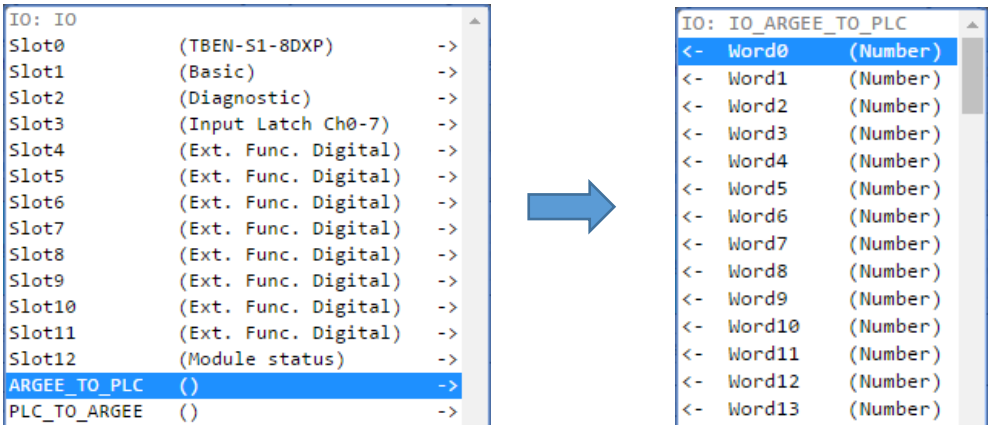
ARGEE blocks have the ability to communicate with a Modbus TCP/IP Master. The Modbus Master can establish communication via registers 0x4000 (register 16384 in decimal) and 0x4400 (register 17408 in decimal). 0x4000 is a read-only register, while 0x4400 is a read/write register.

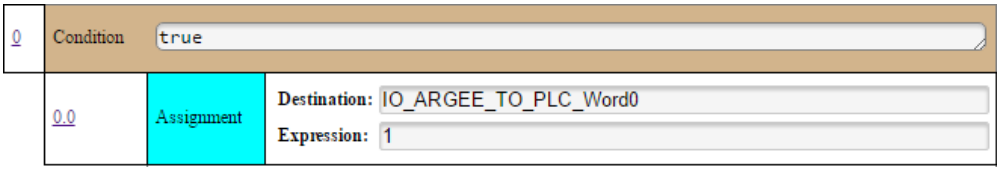
NOTE

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.

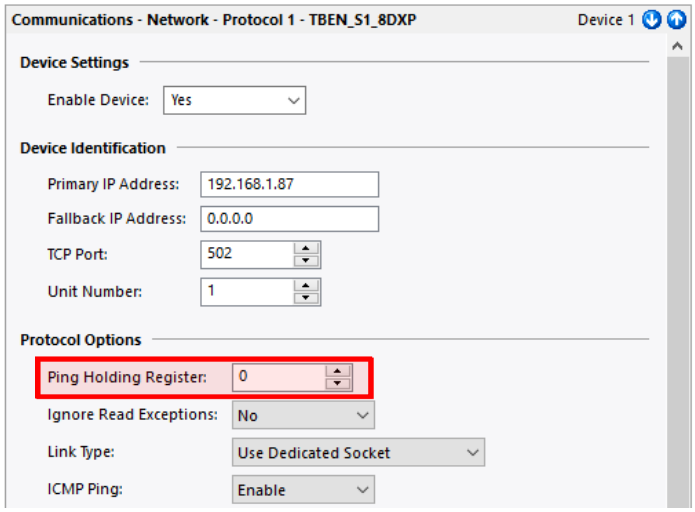
Example of *Communicating with a Modbus TCP/IP Master:*

(ARGEE Setup)



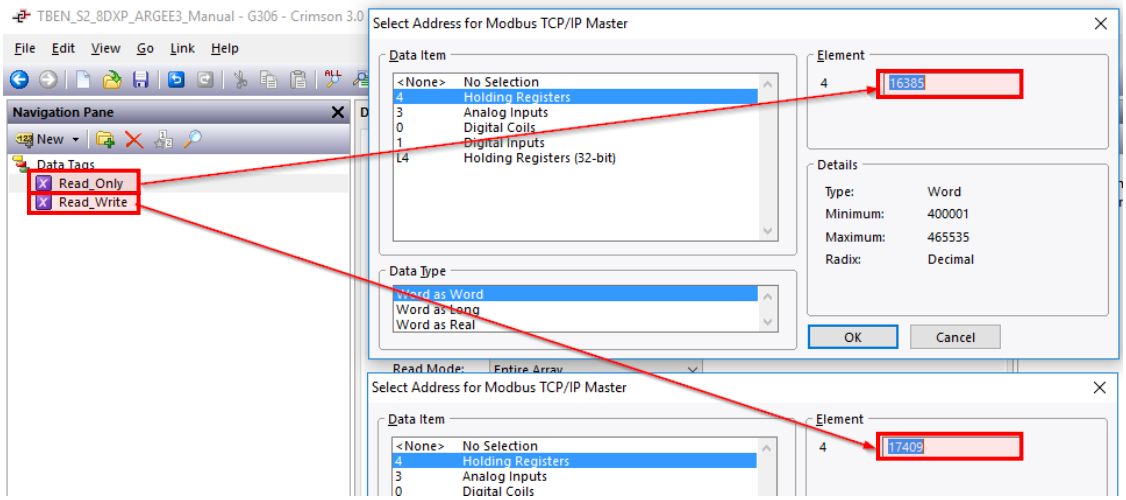


(Crimson 3 Setup)



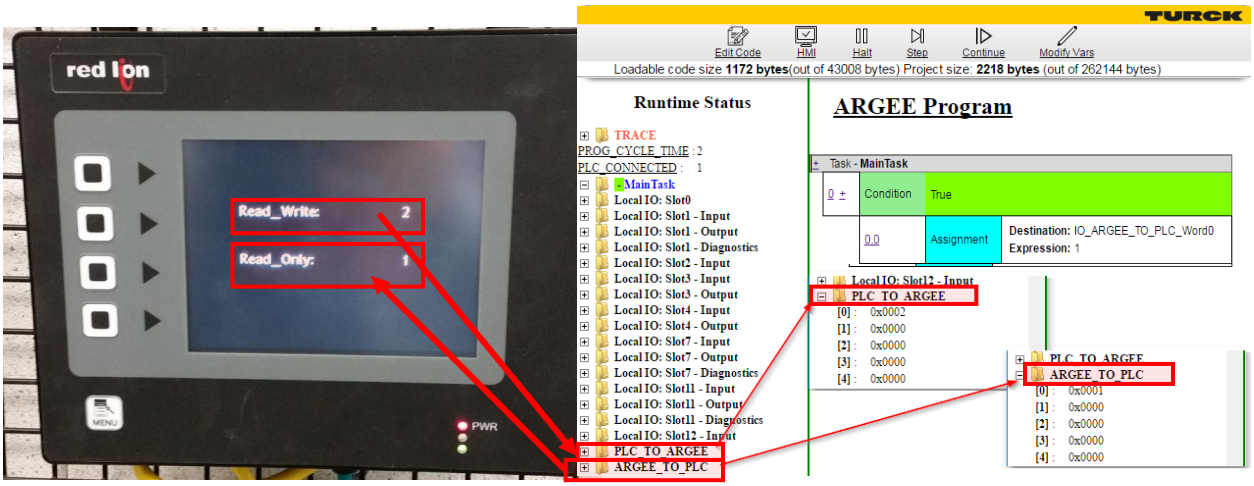
NOTE

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.

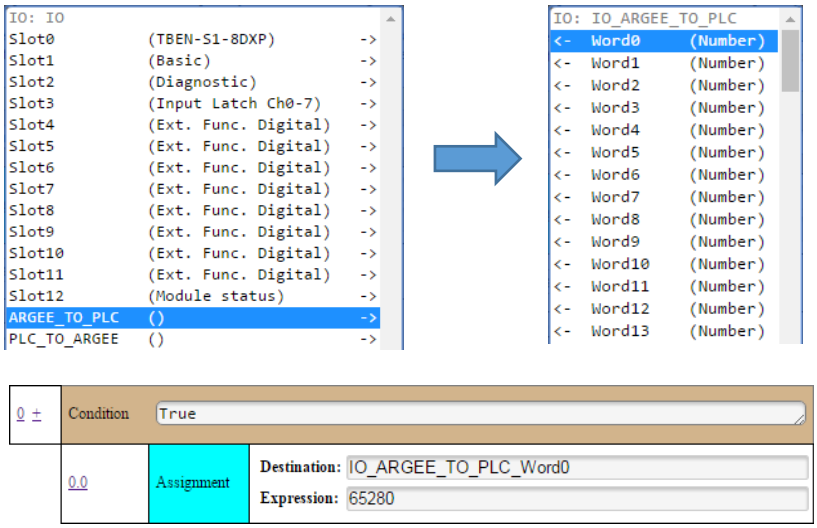
10.4 Communicating with a Turck PLC or TX500 Series HMI – CODESYS 3

10.4.1 EtherNet IP

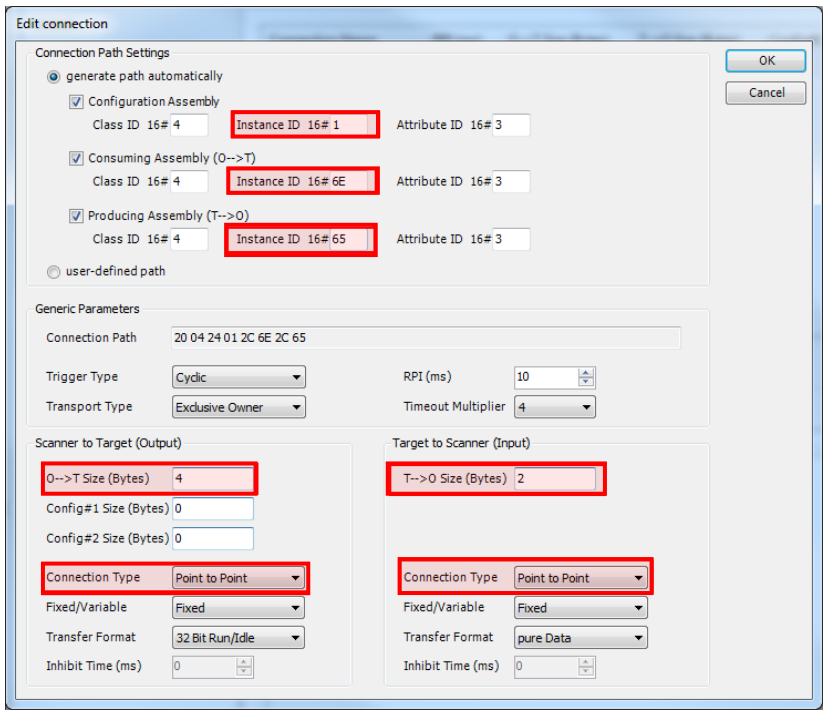
ARGEE blocks have the ability to communicate with an EtherNet IP Scanner communication via tags. The input assembly instance is 101 (0x65), the output assembly instance is 110 (0x6E), and the configuration assembly instance is 01 (0x01). The size of the input and output assemblies (in bytes) is defined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

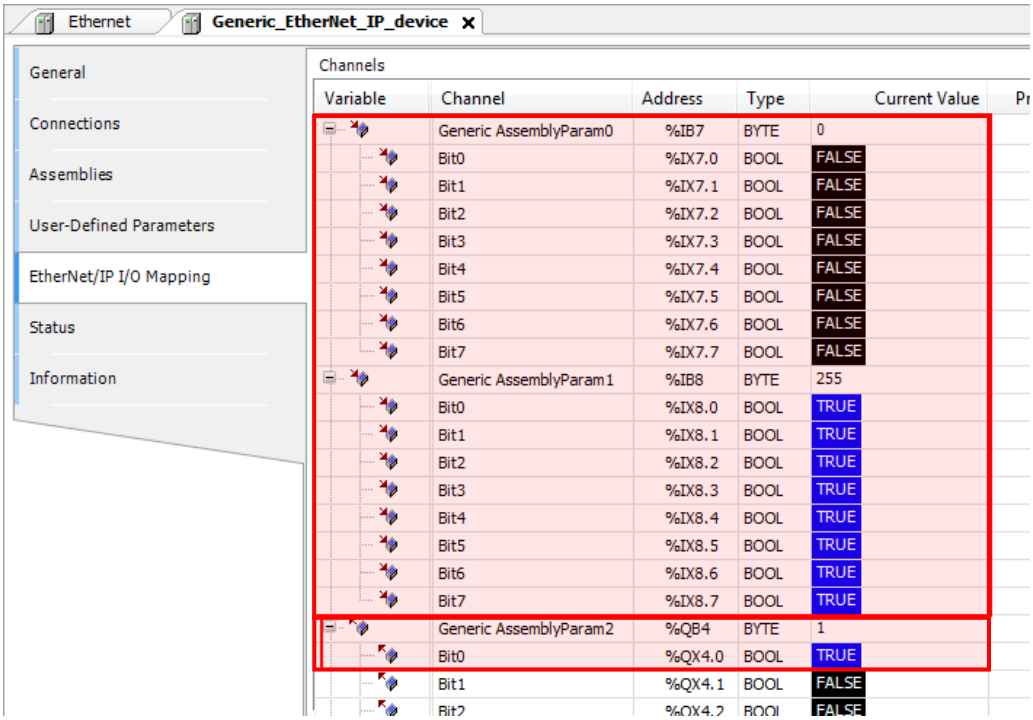
Example of Communicating with a *Turck PLC or TX500 Series HMI:*

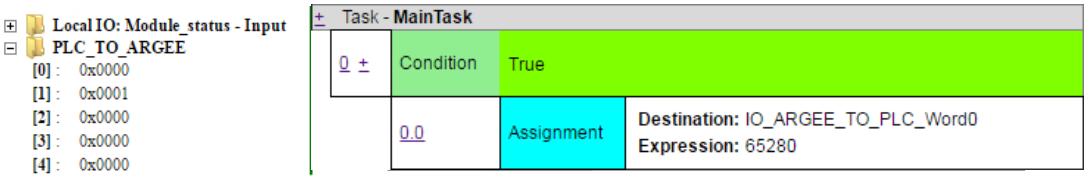
(ARGEE Setup)



(CODESYS 3 Setup)







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

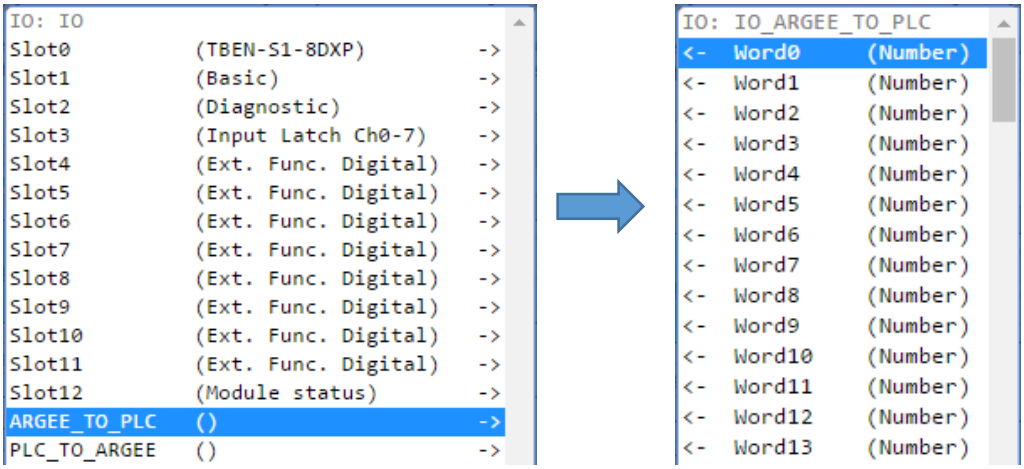
If the user wants to accomplish “bit offsetting,” they need to manually adjust the IO Variable Format (discussed in 11.4.2 IO Variable Formats). For example, if the user wants to force word 0 bit 5 true, the destination variable would be IO_ARGEE_TO_PLC_Word0.5.

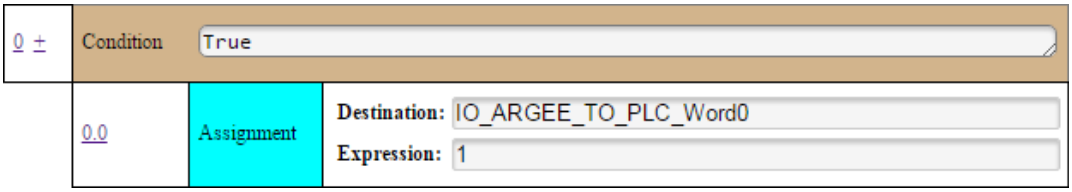
10.4.2 PROFINET

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)



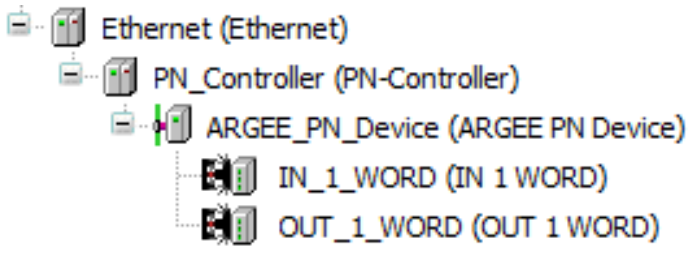


NOTE

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

---

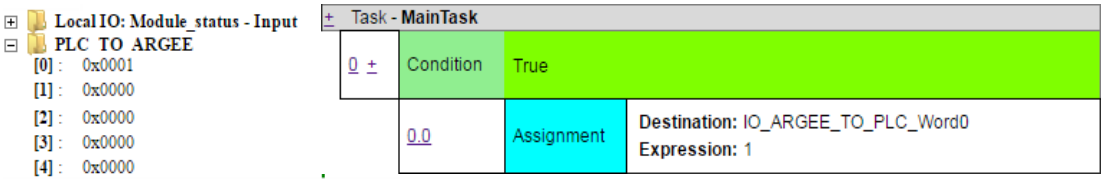
(CODESYS 3 Setup)



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



Explaining the example: The code loads a 1 into *IO_ARGEE_TO_PLC_WORD0*. The PLC loads a 1 into word 1 output.

NOTE

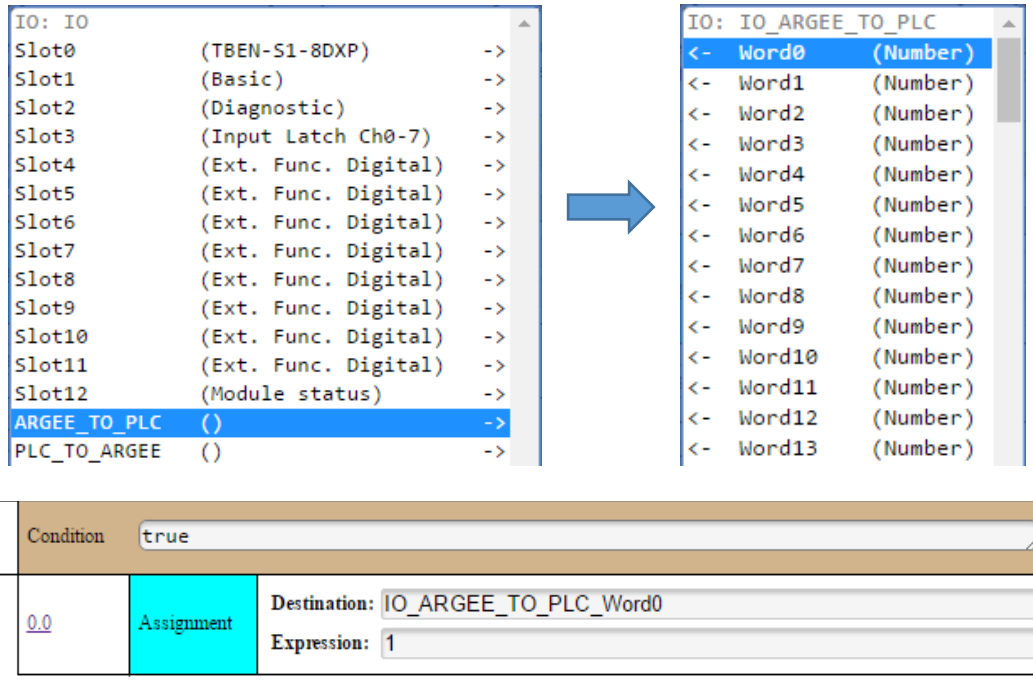
If the user wants to accomplish “bit offsetting,” they need to manually adjust the IO Variable Format (discussed in 11.4.2 IO Variable Formats). For example, if the user wants to force word 0 bit 5 true, the destination variable would be IO_ARGEE_TO_PLC_Word0.5.

10.4.3 Modbus TCP/IP

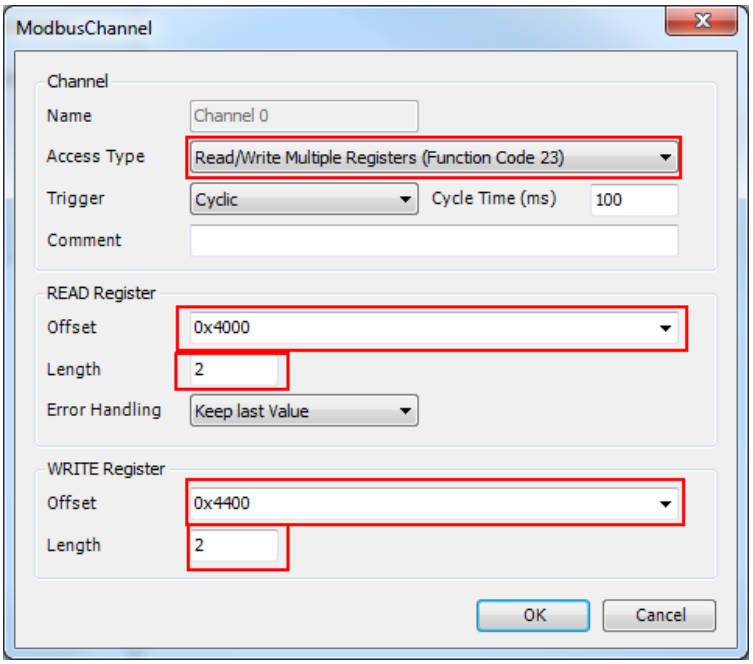
ARGEE blocks have the ability to communicate with a Modbus TCP/IP Master. The Modbus Master can establish communication via registers 0x4000 (register 16384 in decimal) and 0x4400 (register 17408 in decimal). 0x4000 is a read-only register, while 0x4400 is a read/write register.

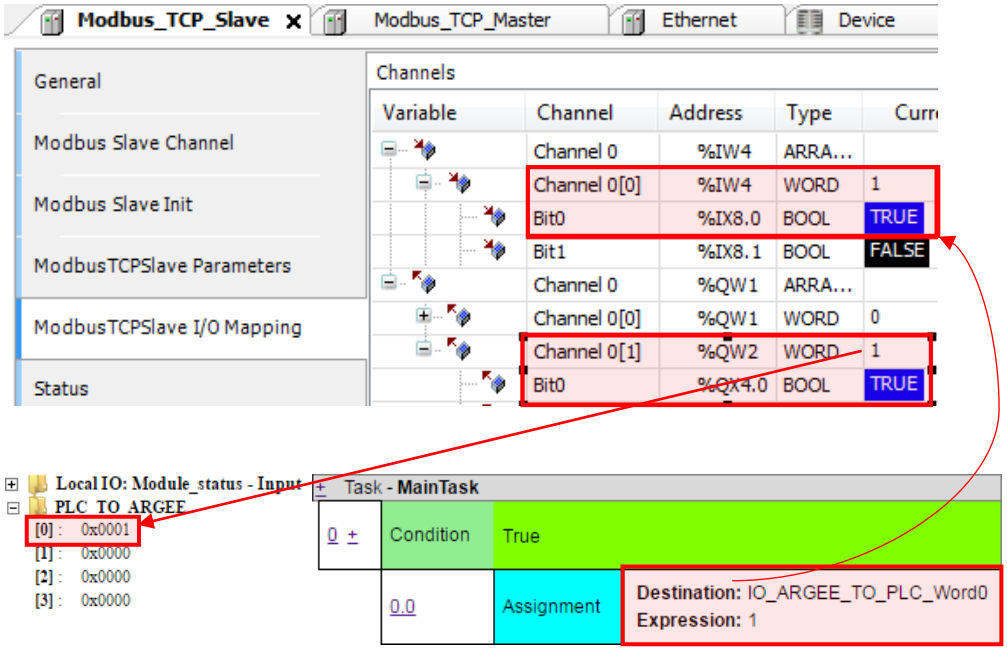
Example of Communicating with a *Turck PLC or TX500 Series HMI:*

(ARGEE Setup)



(CODESYS 3 Setup)





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

NOTE

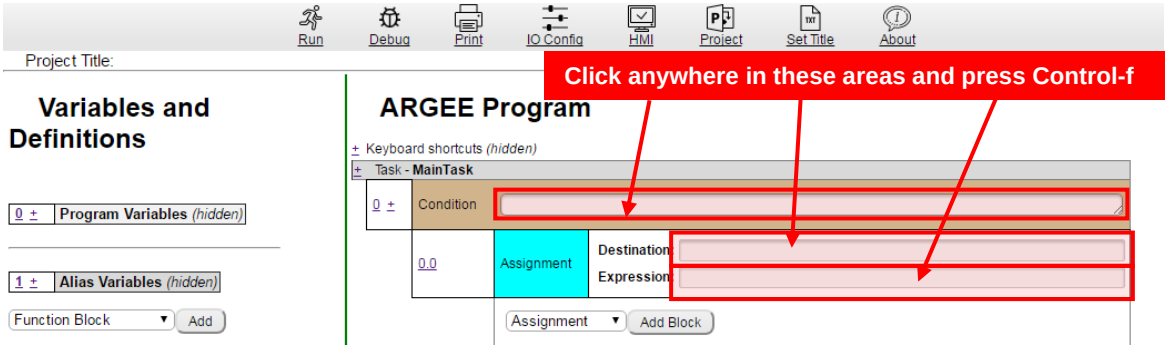
If the user wants to accomplish “bit offsetting,” they need to manually adjust the IO Variable Format (discussed in 11.4.2 IO Variable Formats). For example, if the user wants to force word 0 bit 5 true, the destination variable would be IO_ARGEE_TO_PLC_Word0.5.

---

11 Appendix I - Definitions

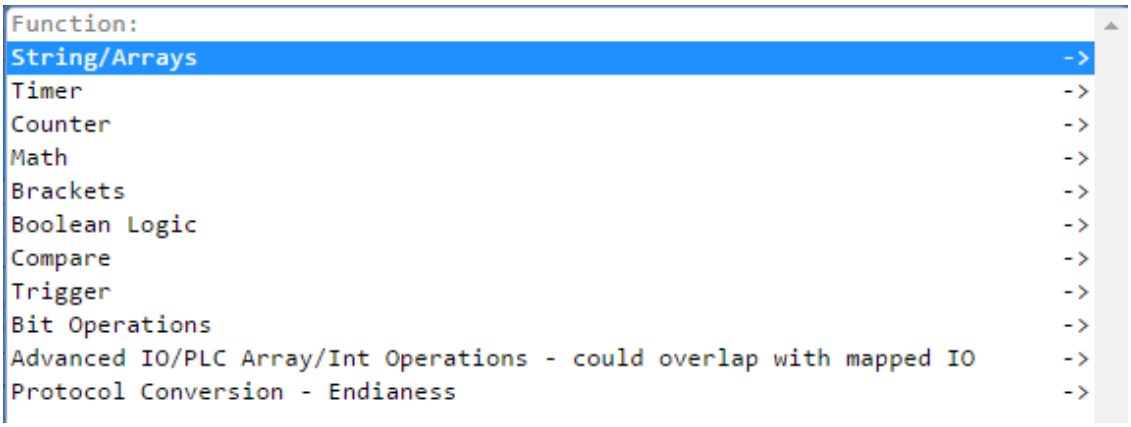
11.1 Built-in Functions (Ctrl-f)

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



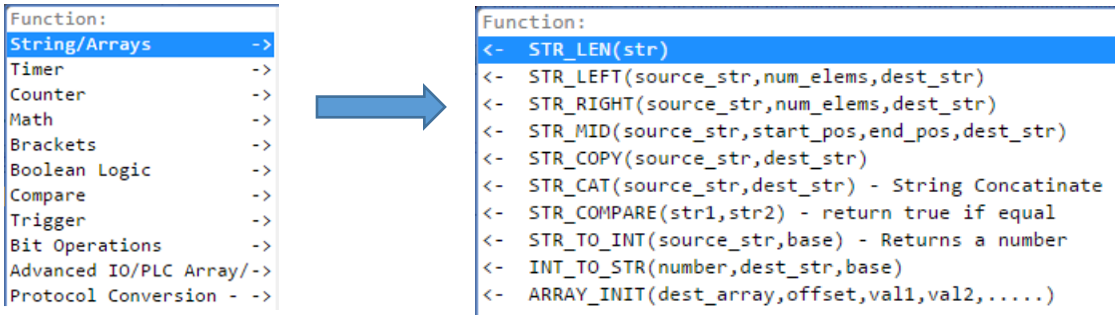
11.2 Built-in Functions Menu

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

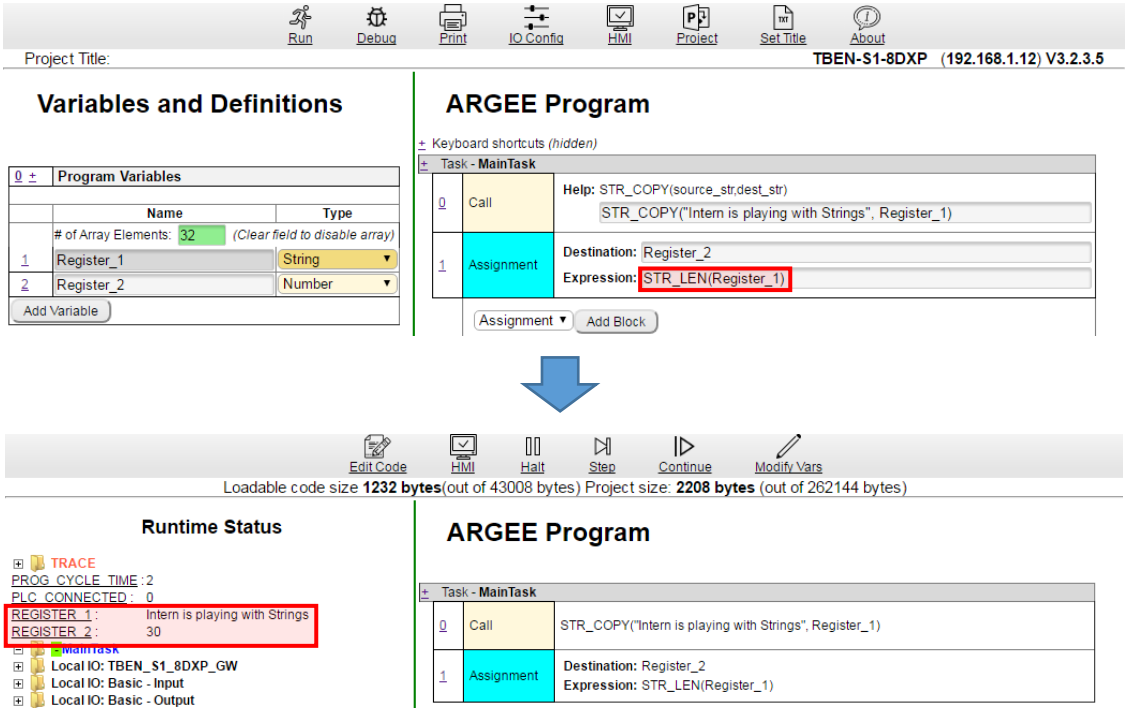
To access the String/Arrays functions, highlight *String/Arrays,* and press "→" on the keyboard or click "->" with the mouse to advance to the next or previous level.



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:*



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 “ ”.

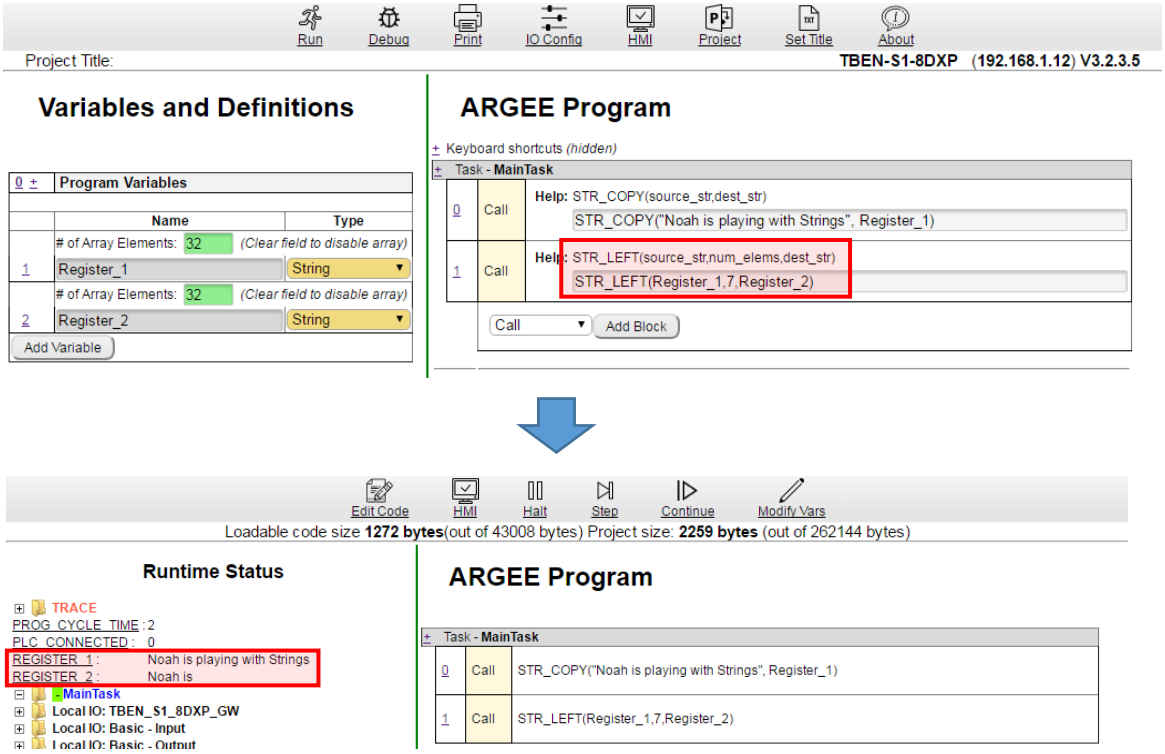
NOTE

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:*



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 “ ”.

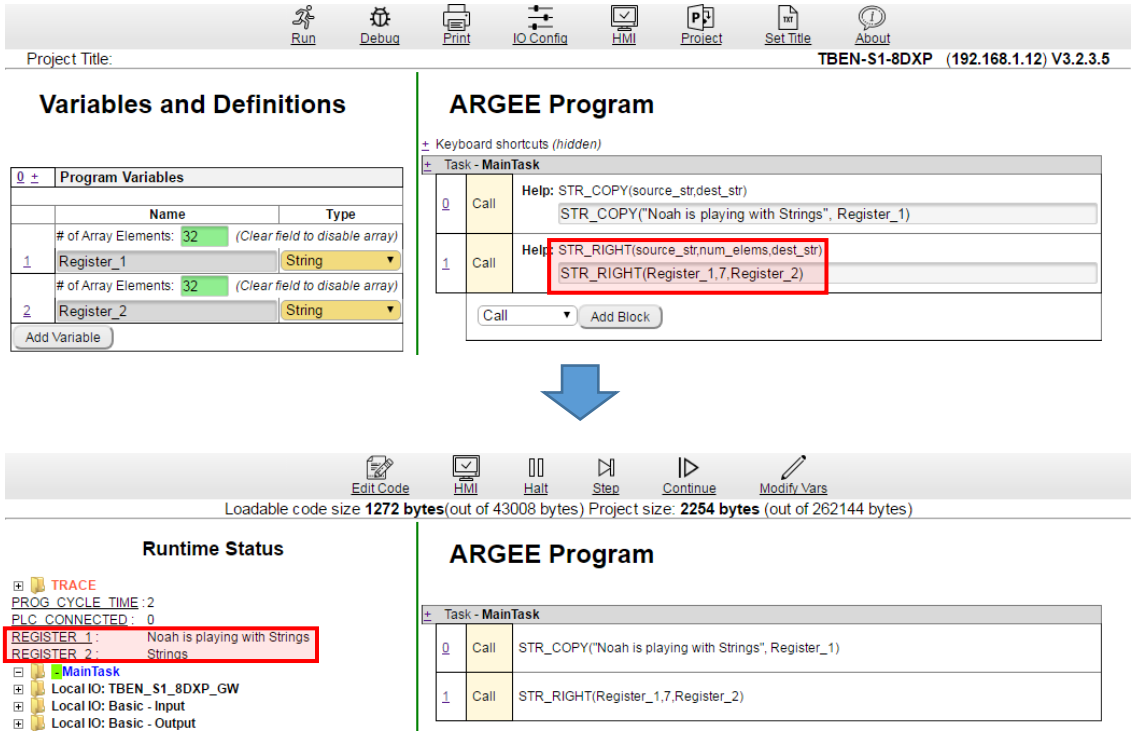
NOTE

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:*



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 “ ”.

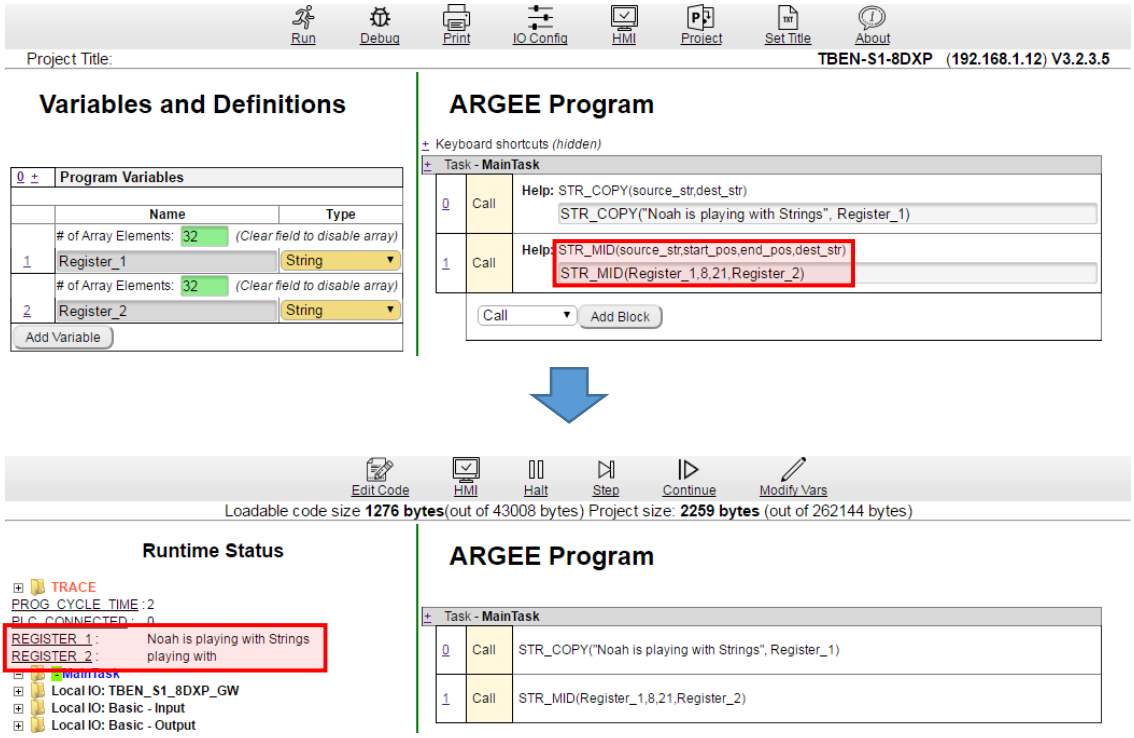
NOTE

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:*



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 “ ”.

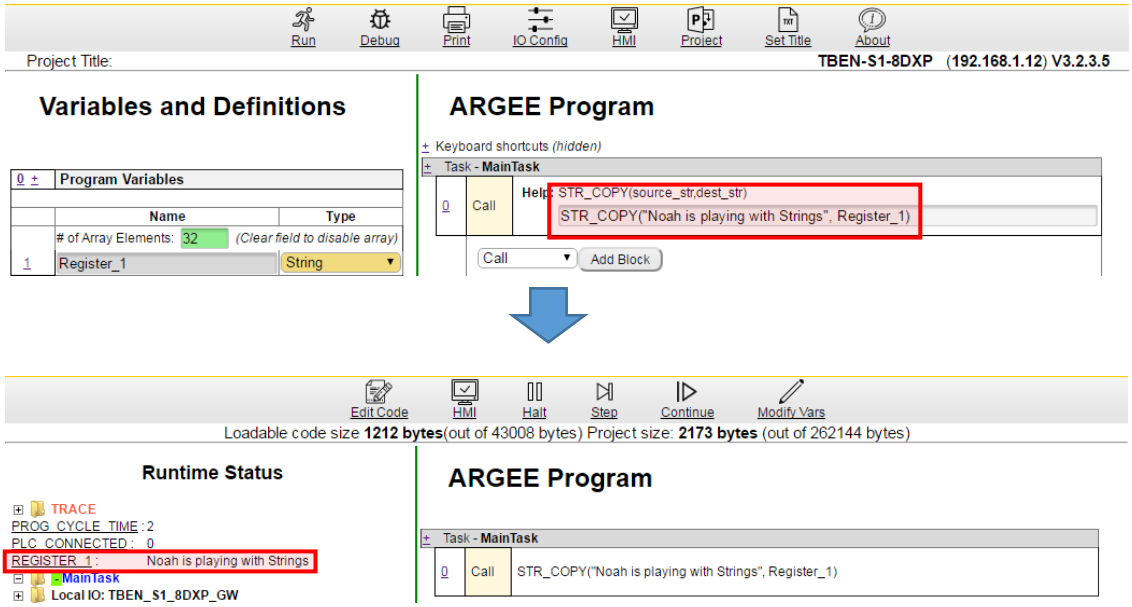
NOTE

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:*



Explaining the Example: STR_COPY copies the string “Noah is playing with Strings” into Register_1.

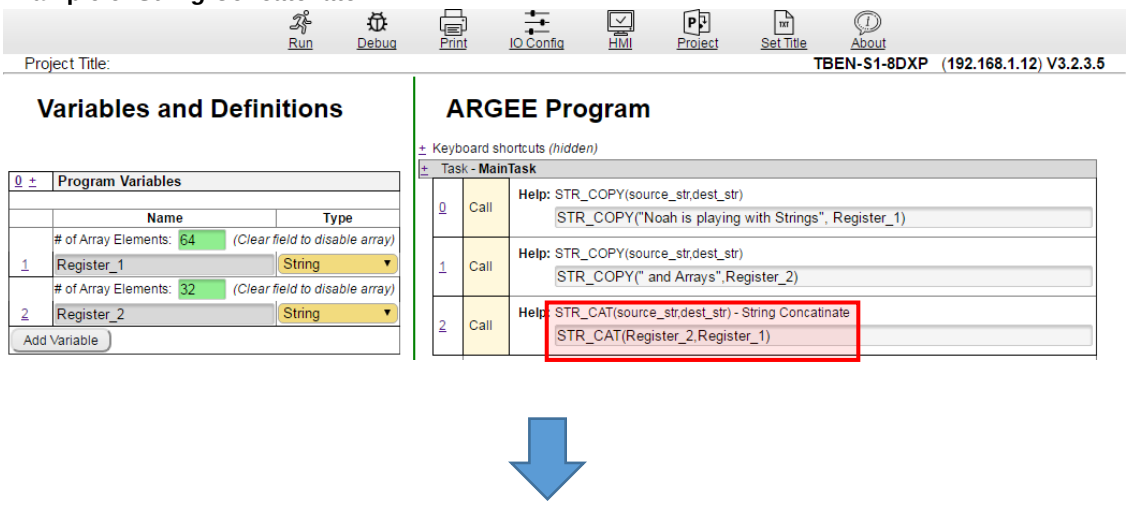
NOTE

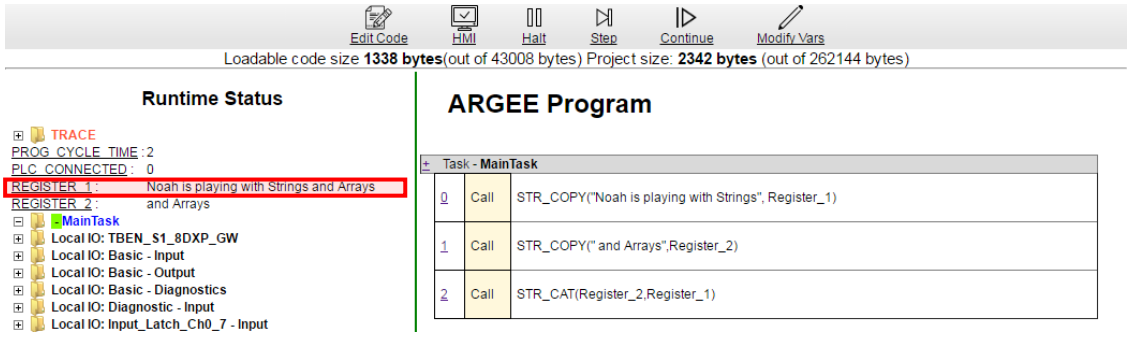
Strings must be one element larger than the number of characters you want to store, and must be surrounded by quotations “ ”.

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:*





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.”

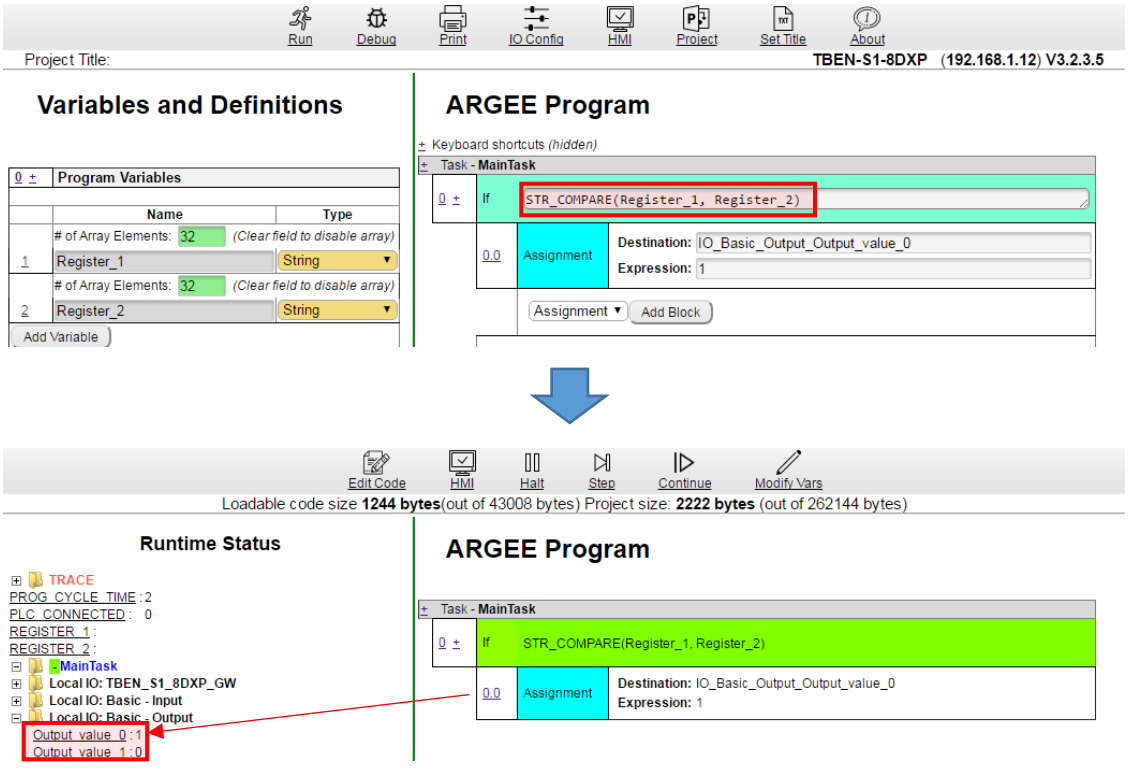
NOTE

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:*



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.

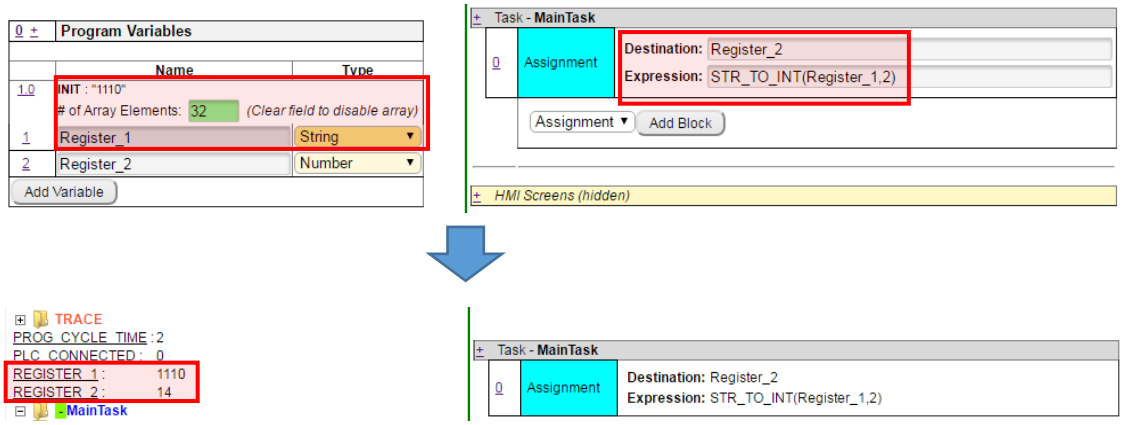
NOTE

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

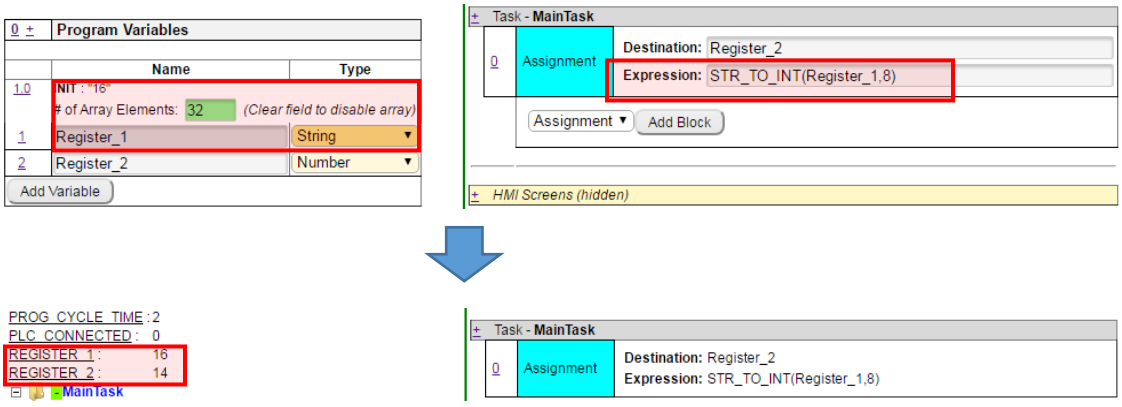


Explaining the Example: Register_1 is initialized to value “1110.” STR_TO_INT takes the binary string in Register_1, converts it into a decimal integer and puts it into Register_2.

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

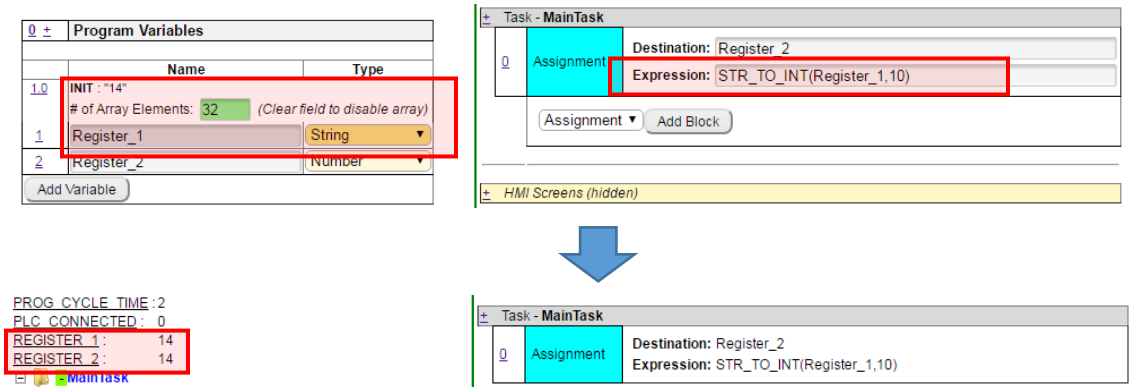


Explaining the Example: Register_1 is initialized to value “16.” STR_TO_INT takes the octal string in Register_1, converts it into a decimal integer and puts it into Register_2.

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



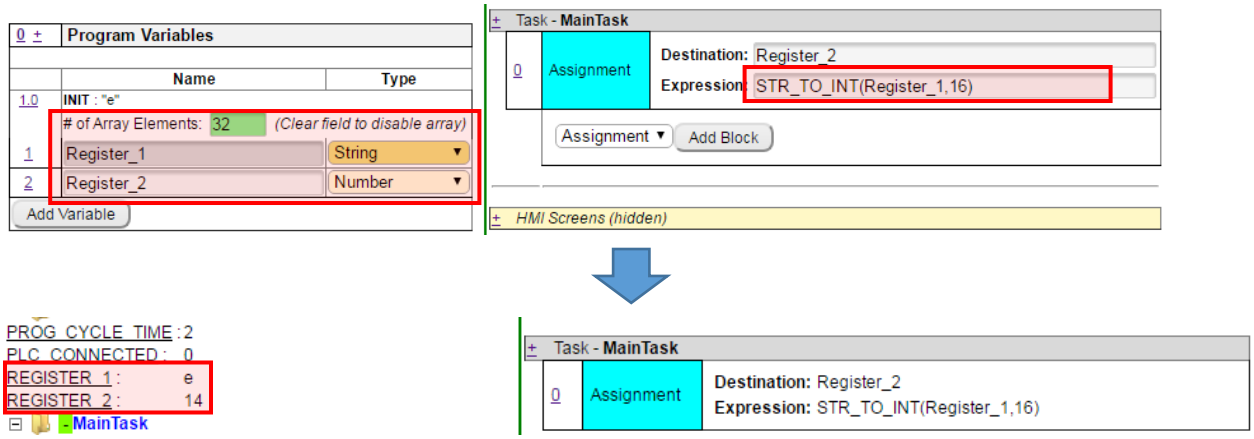
Explaining the Example: Register_1 is initialized to value “14.” STR_TO_INT takes the decimal string in Register_1, converts it into a decimal integer and puts it into Register_2.

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:*



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.

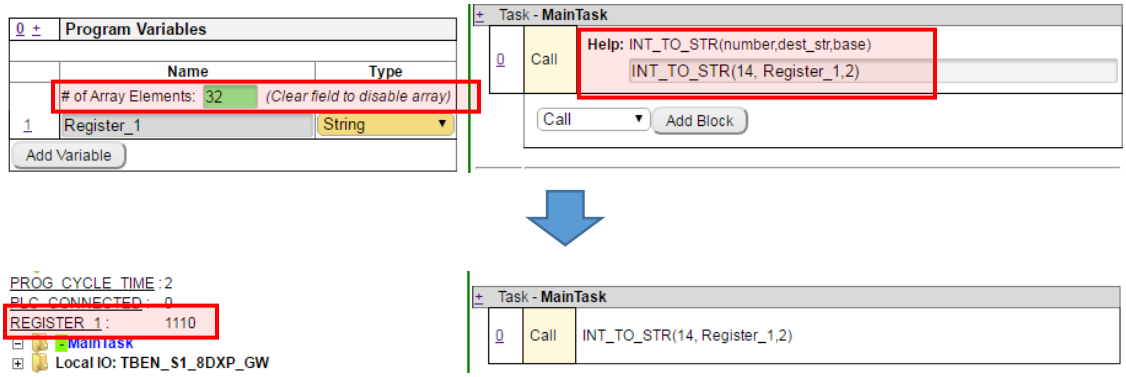
NOTE

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

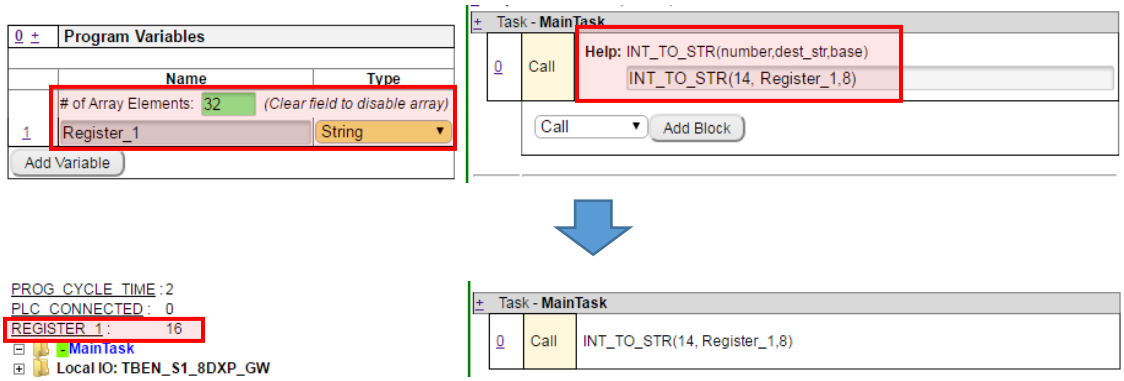
The user will use *INT_TO_STR* when the user wants to move an integer into a string. The user can also convert the integer into a binary, octal, decimal or hexadecimal base.

11.2.1.9.1 Integer to String – Base 2 – Binary



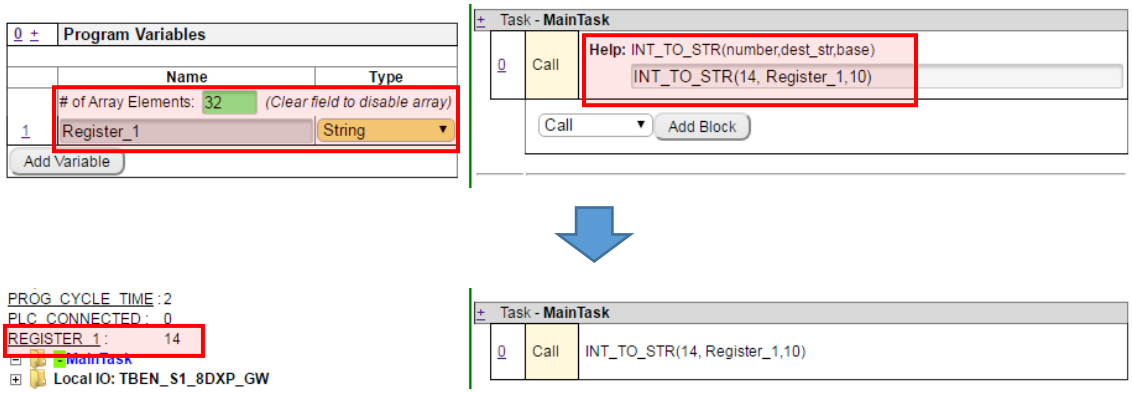
Explaining the Example: INT_TO_STR converts the decimal integer 14 into binary and puts that value into Register_1.

11.2.1.9.2 Integer to String – Base 8 – Octal



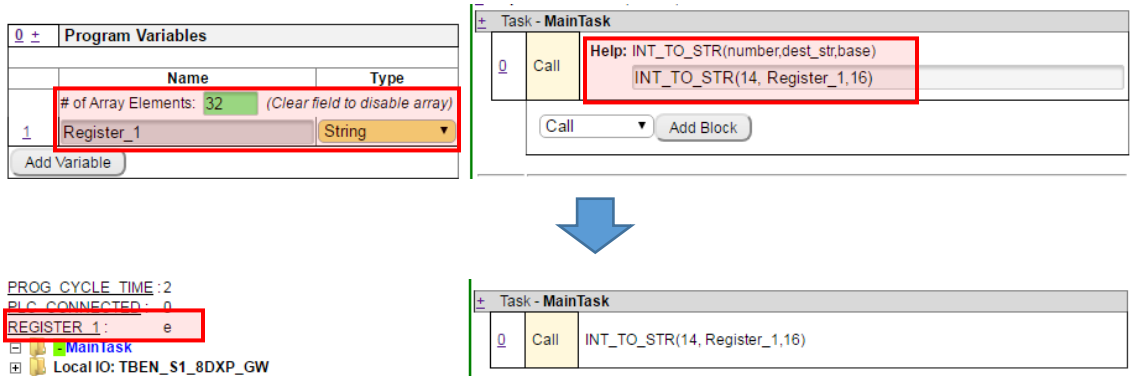
Explaining the Example: INT_TO_STR converts the decimal integer 14 into octal and puts that value into Register_1.

11.2.1.9.3 Integer to String – Base 10 – Decimal



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



Explaining the Example: INT_TO_STR converts the decimal integer 14 into hexadecimal and puts that value into Register_1.

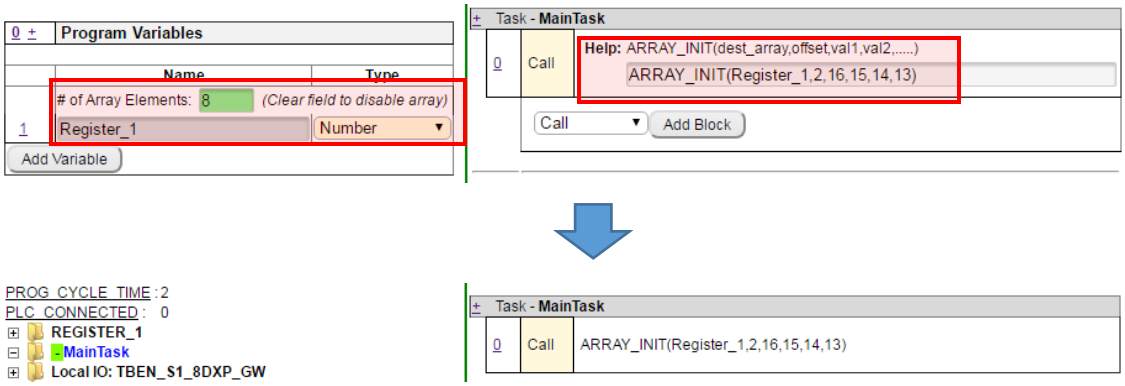
NOTE

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.

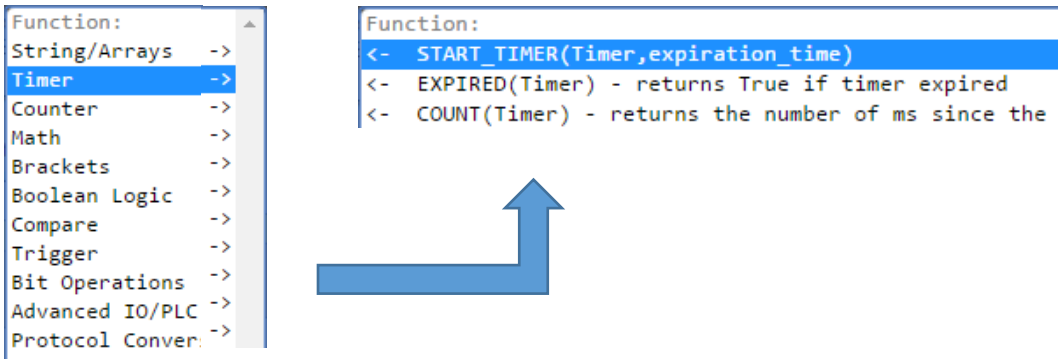
Example of *Array Initialize:*



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

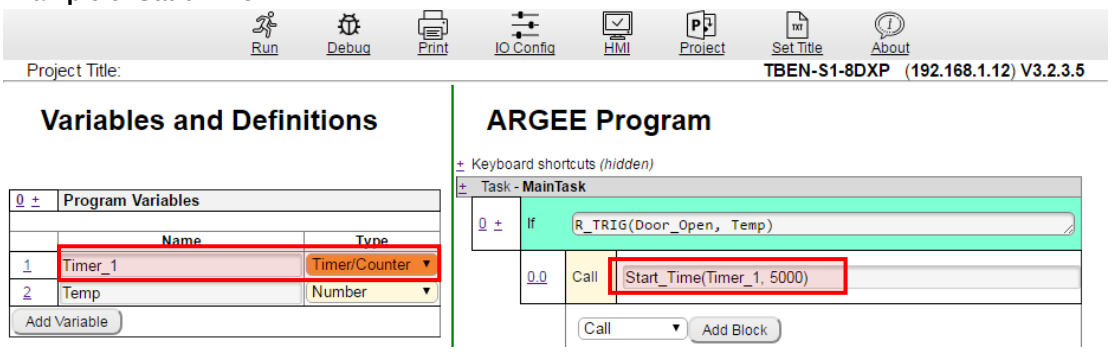
To access the Timer functions, highlight *Timer* and press "→" on the keyboard or click "->" with the mouse to advance to the next or previous level.



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:*



Explaining the Example: If the door opens, Timer_1 starts counting. Timer_1 expires after 5000ms (or 5 seconds).

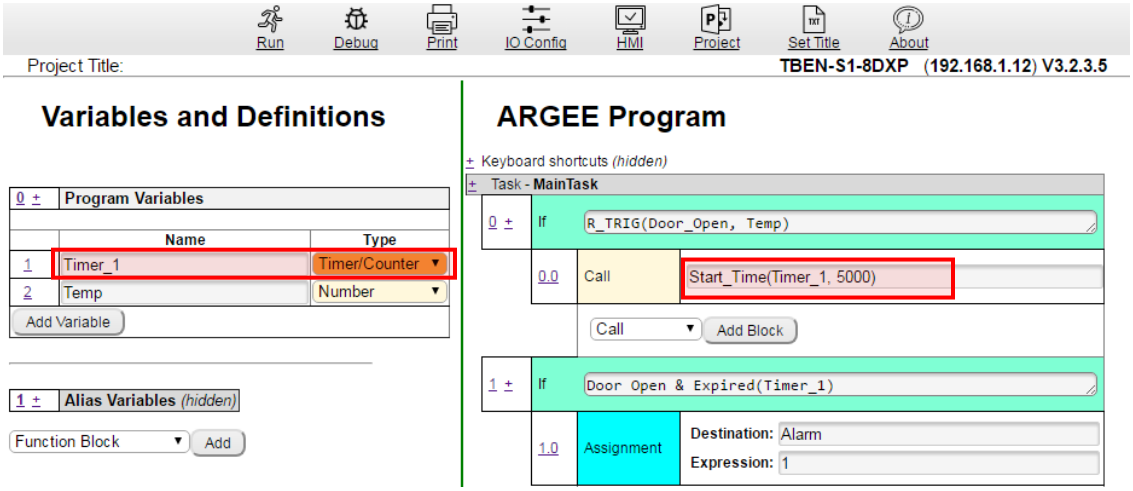
NOTE

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:*



Explaining the Example: When the door opens, Timer_1 starts. If the door is still open when Timer_1 expires, the alarm turns on.

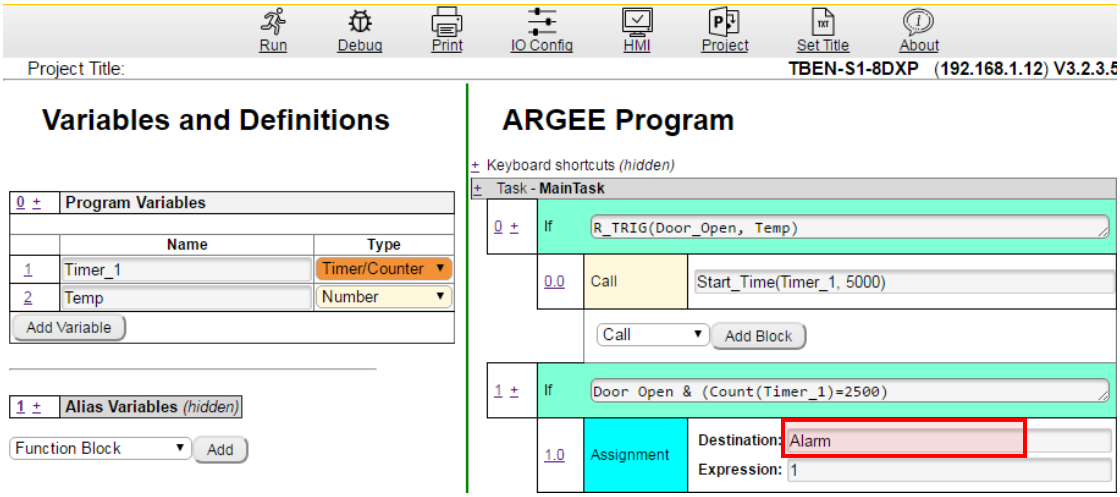
NOTE

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:*



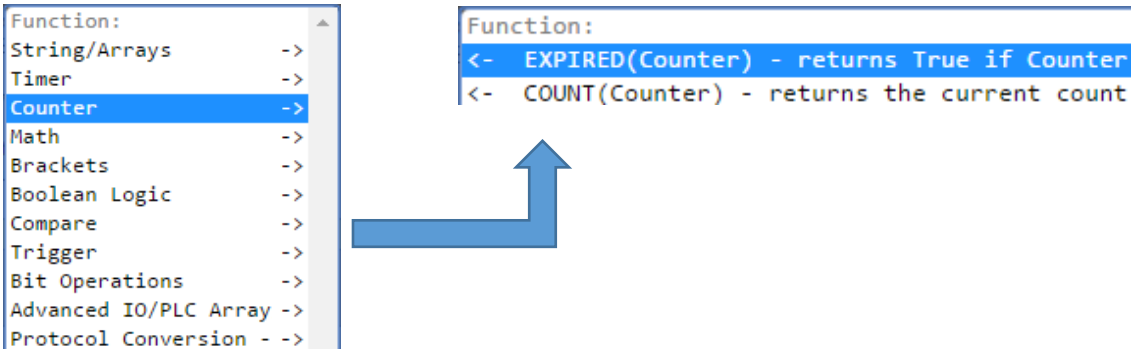
Explaining the Example: When the door opens, Timer_1 starts. If the door is still open after 2500ms (2.5 seconds), a light will turn on.

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

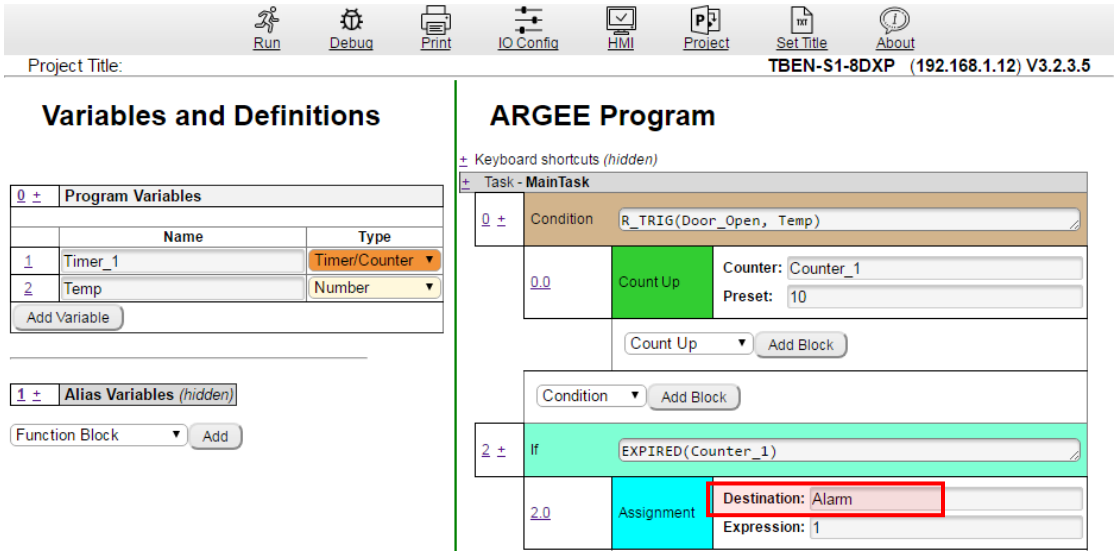
To access the Counter functions, highlight *Counter* and press "->" on the keyboard or click "->" with the mouse to advance to the next or previous level.



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:*



Explaining the Example: When the door opens, Counter_1 counts up one time. Couner_1 expires after 10 counts. If Counter_1 expires, an alarm turns on.

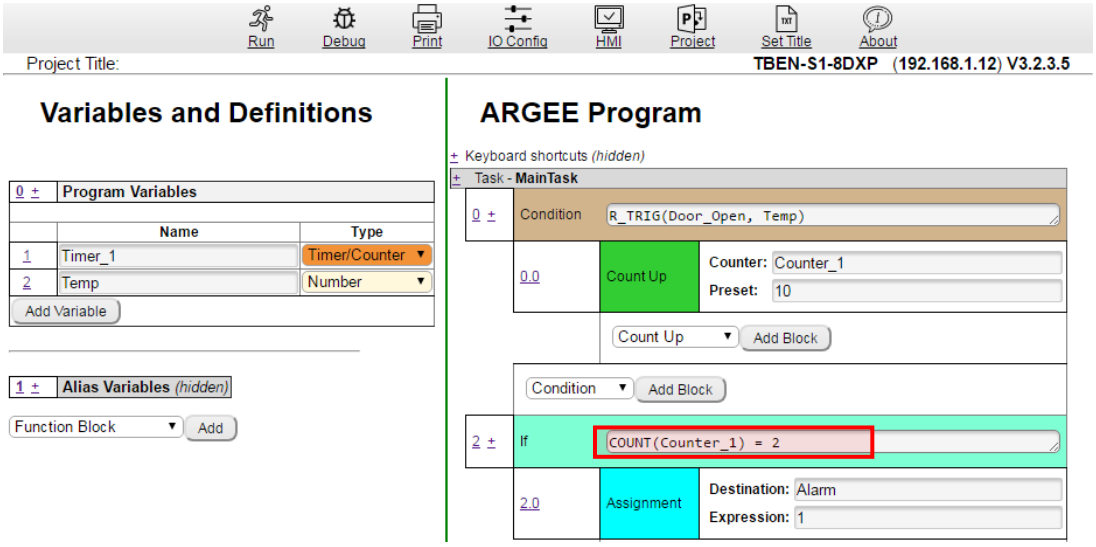
NOTE

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:*



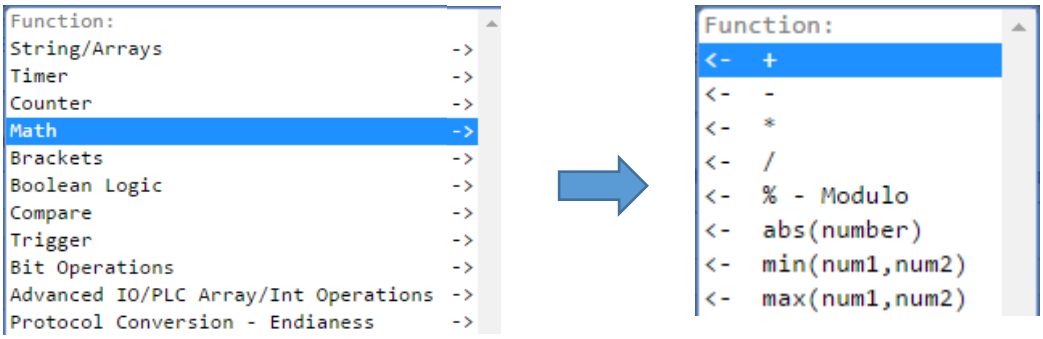
Explaining the Example: When the door opens, Counter_1 counts up one time. Counter_1 expires at 10 counts. If the door is opened 2 times, a light turns on.

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

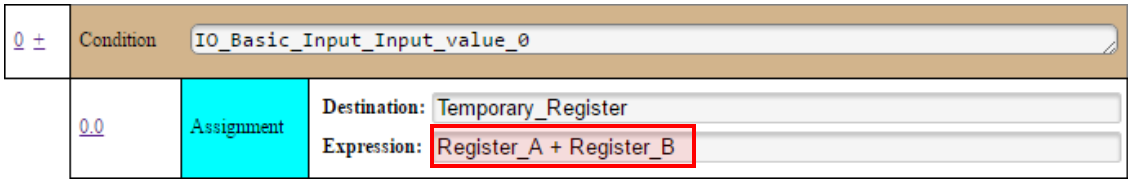
The user will use Math Operations if they want to monitor, compare, or combine data from different registers. To access the Math functions, highlight Math and press "→" on the keyboard or click "->" with the mouse to advance to the next or previous level.



11.2.4.1 Addition

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

Example of *Add:*

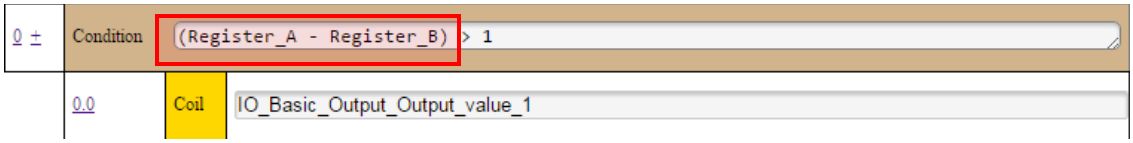


Explaining the Example: When Input_value_0 goes true, the value in Register_A will be added to the value in Register_B. The result is placed in Temporary_Register.

11.2.4.2 Subtraction

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

Example of *Subtraction:*

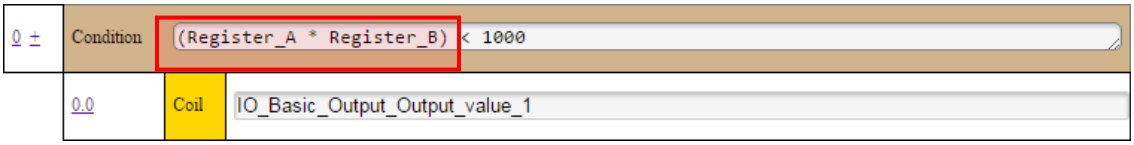


Explaining the Example: The user is subtracting the value in Register_A from the value in Register_B. When Register_A minus Register_B is greater than 1, the user Coils on Output_value_1.

11.2.4.3 Multiplication

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

Example of *Multiplication:*

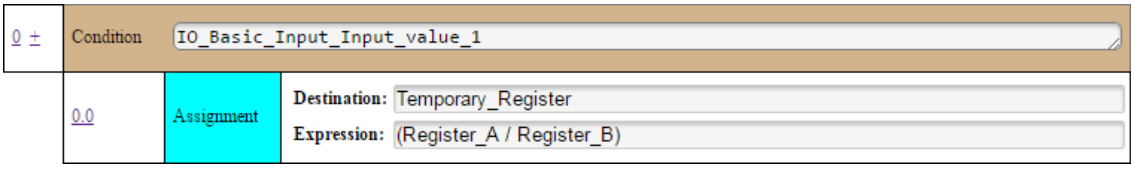


Explaining the Example: The user is multiplying the value in Register_A with the value in Register_B. If Register_A times Register_B is less than 1000, the user Coils on Output_value_1.

11.2.4.4 Division

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

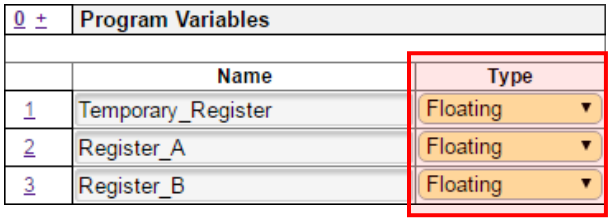
Example of *Division:*



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



If the registers are not set to floating, ARGEE will drop the fraction and just display the whole number.

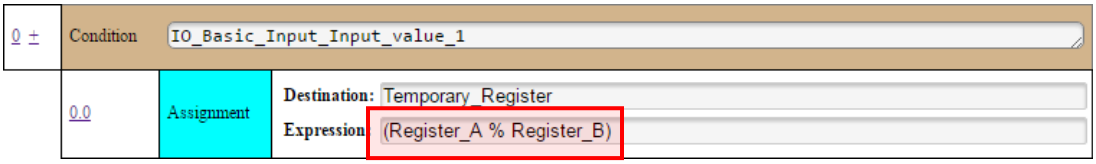
For example:

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:*



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.

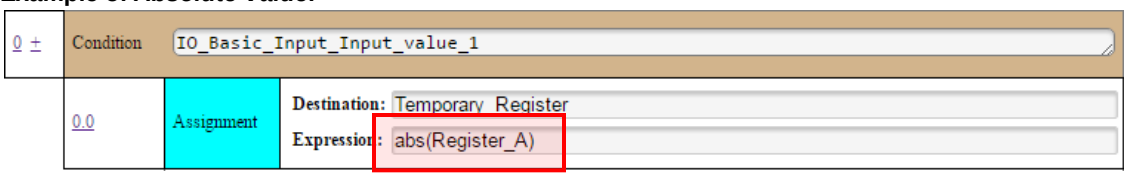
For example:

36 / 6 = “6” with a remainder of “0” → ARGEE displays “0”
34 / 6 = “5” with a remainder of “4” → ARGEE displays “4”
6 / 36 = “0” with a remainder of “6” → ARGEE displays “6”

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:*

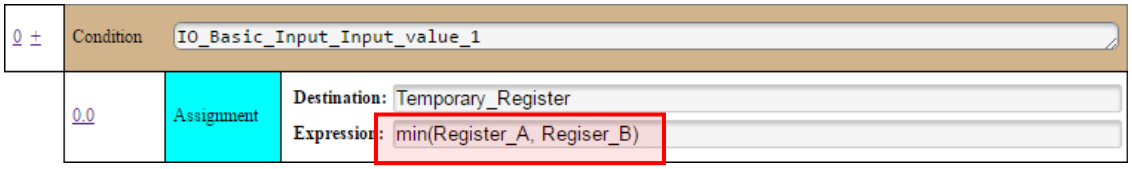


Explaining the Example: When Input_value_1 goes true, ARGEE will take the absolute value of the integer in Register_A, and place into Temporary_Register.

11.2.4.7 Minimum Value

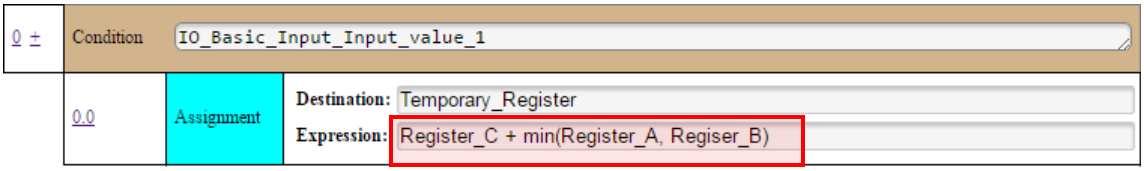
The user will use the minimum value (min) to compare multiple registers and place the smallest value in to the destination register. The user can also use the minimum value (min) to compare multiple registers and use the smallest value in a math operation.

Example of *Minimum Value:*



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

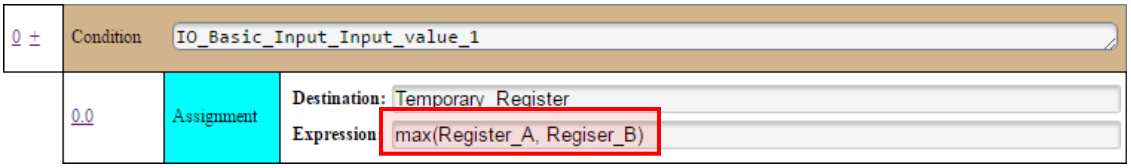


Explaining the Example: When Input_value_1 goes true, ARGEE will take the smallest value between Register_A and Register_B and place that value into the addition operation. The result will be put into Temporary_Register.

11.2.4.8 Maximum Value

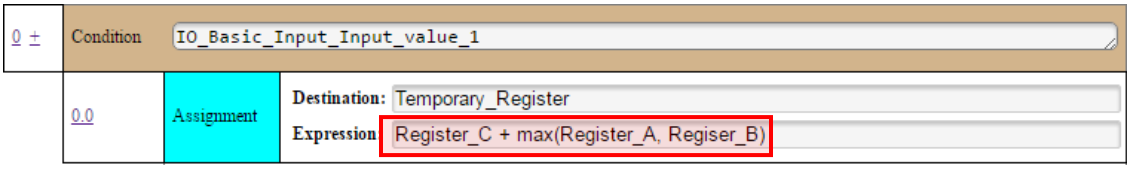
The user will use the maximum value (max) to compare multiple registers and place the largest value into the destination register. The user can also use the maximum value (max) to compare multiple registers, and use the largest value in a math operation.

Example of *Maximum Value:*



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.

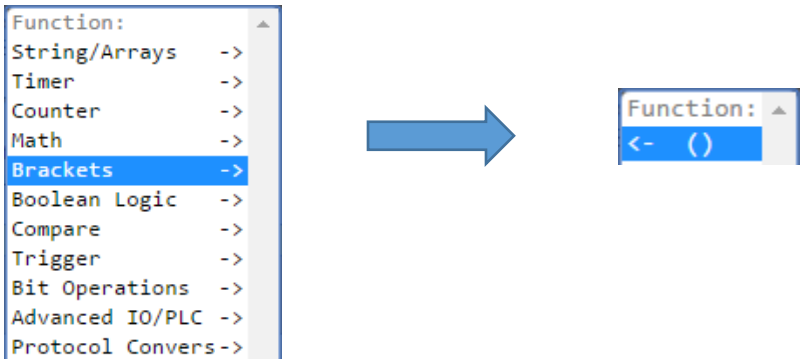
OR



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.

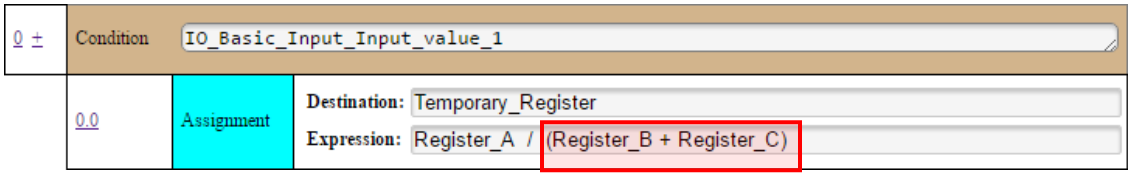
11.2.5 Brackets

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.

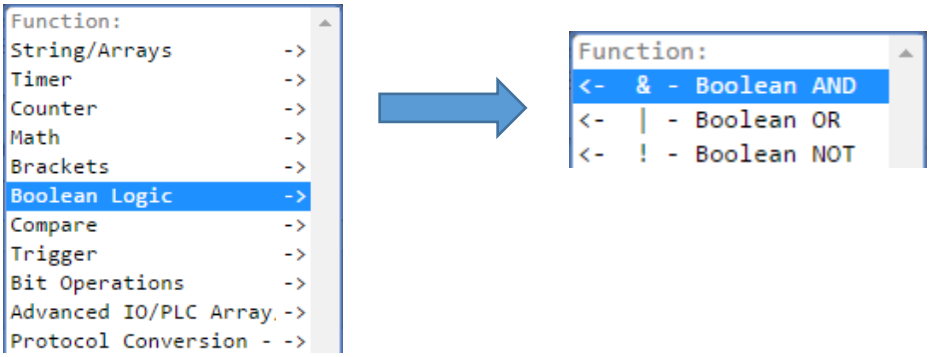
Example of *Brackets:*



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 (I) 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.



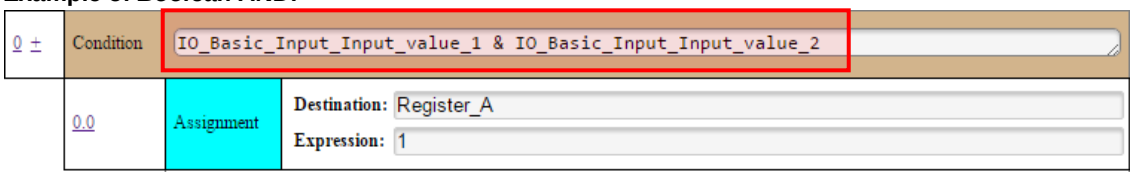
NOTE

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:*

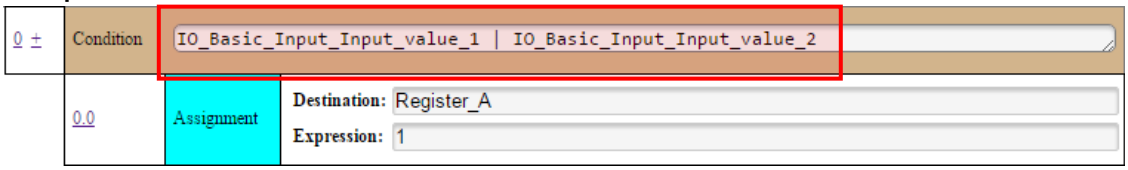


Explaining the Example: When both Input_value_1 AND input_value_2 are true, load the value "1" into Register_A.

11.2.6.2 Boolean OR

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

Example of *Boolean OR:*

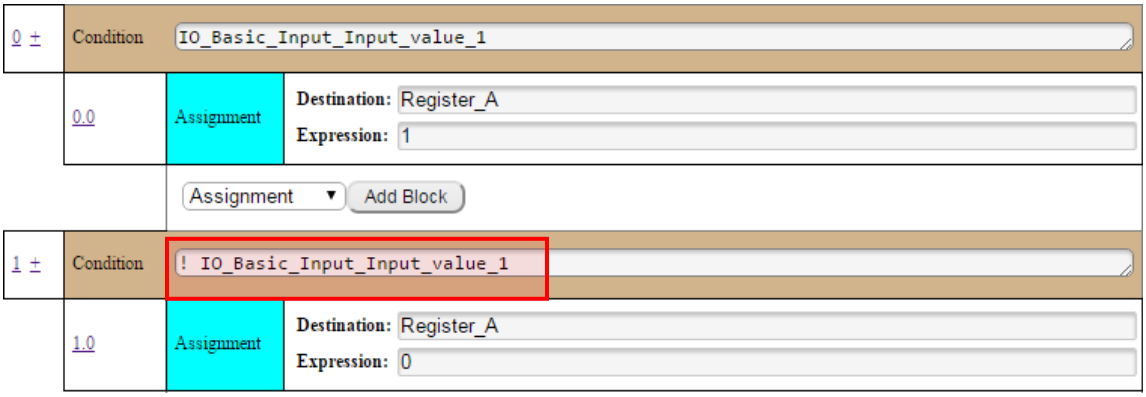


Explaining the Example: When either Input_value_1 OR input_value_2 are true, load the value "1" into Register_A.

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.

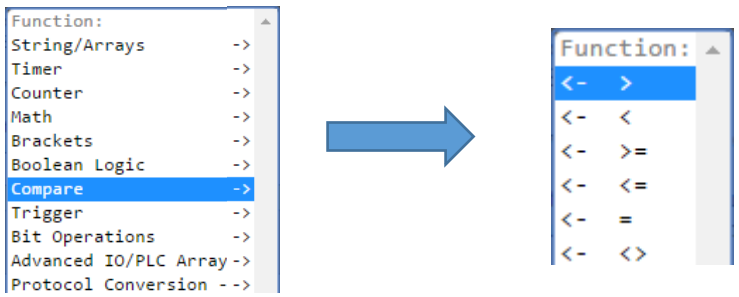
Example of *Boolean NOT:*



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

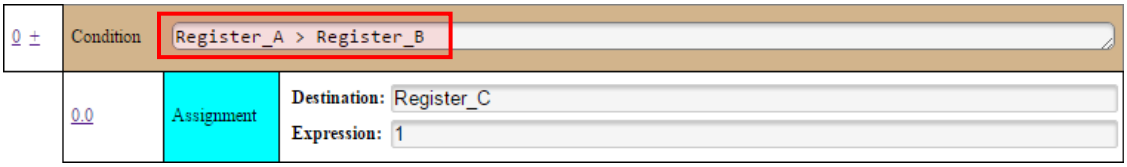
The user will select the Compare function if he needs to compare two numbers and find the smallest or largest, or see if they are equal or unequal. To access the Compare functions, highlight Compare and press "->" on the keyboard or click "->" with the mouse to advance to the next or previous level.



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:*

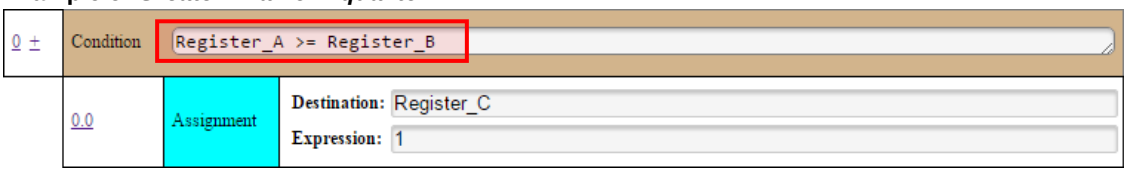


Explaining the Example: When the value in Register_A is greater than the value in Register_B, the value "1" will be loaded into Register_C.

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:*

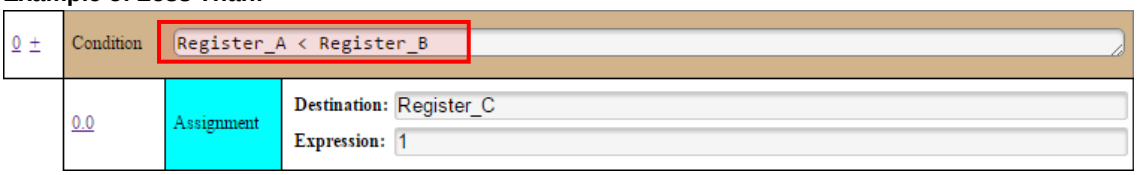


Explaining the Example: When the value in Register_A is greater than or equal to the value in Register_B, the value "1" will be loaded into Register_C.

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:*

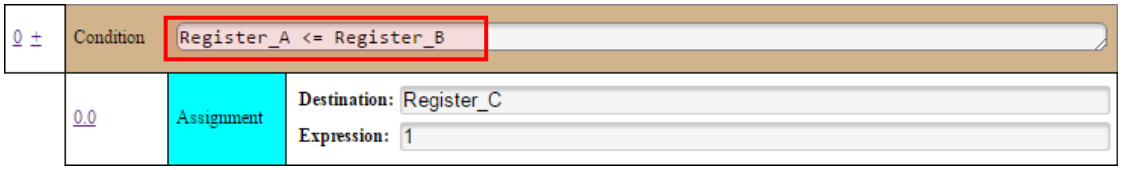


Explaining the Example: When the value in Register_A is less than the value in Register_B, the value "1" will be loaded into Register_C.

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:*

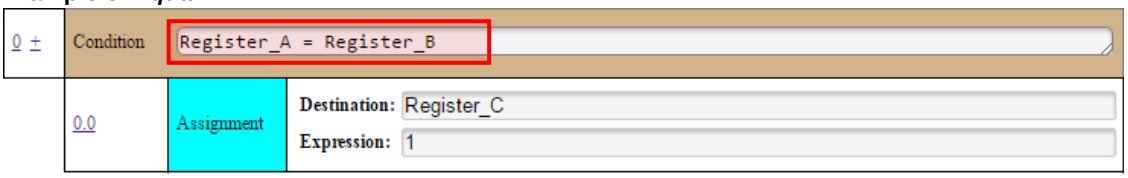


Explaining the Example: When the value in Register_A is less than or equal to the value in Register_B, the value "1" will be loaded into Register_C.

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:*

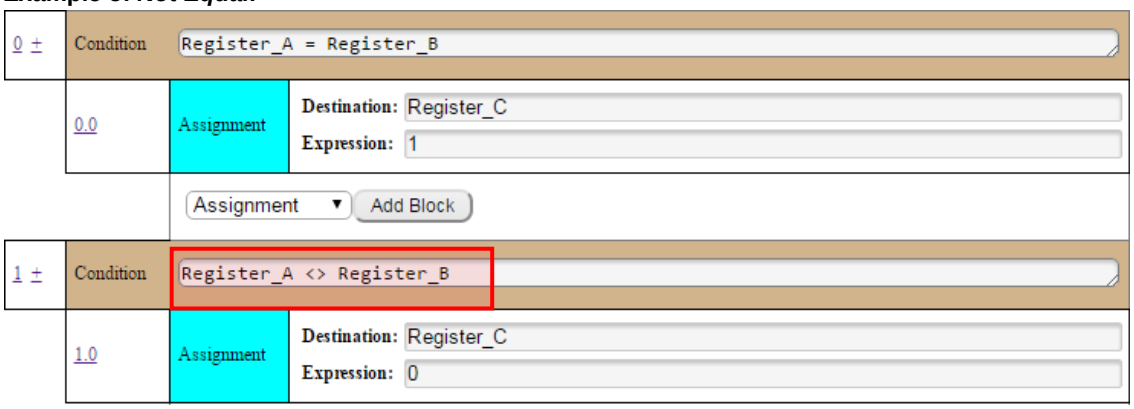


Explaining the Example: When the value in Register_A is equal to the value in Register_B, the value "1" will be loaded into Register_C.

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.

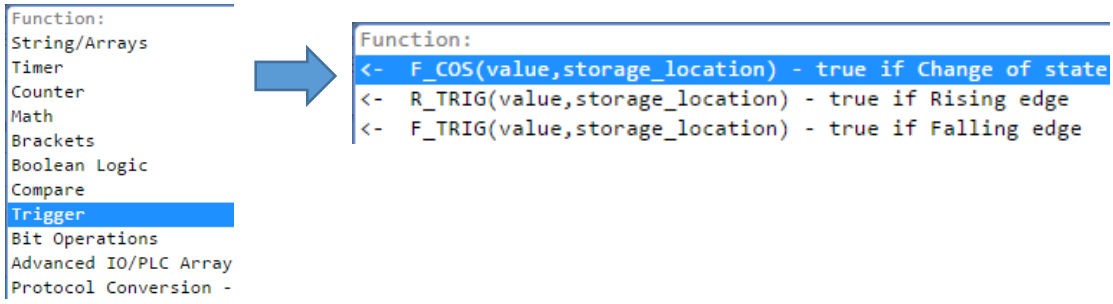
Example of *Not Equal:*



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.

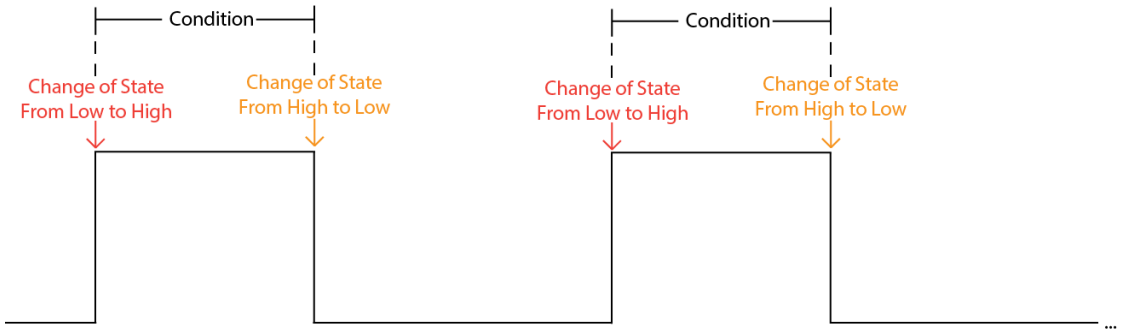
11.2.8 Trigger

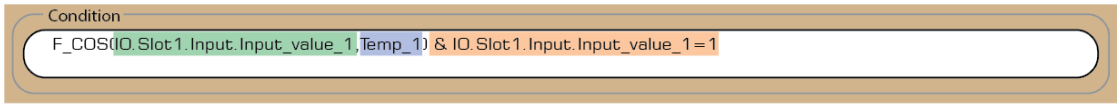
To access the Trigger functions, highlight *Trigger* and press "->" on the keyboard or click "->" with the mouse to advance to the next or previous level.



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.



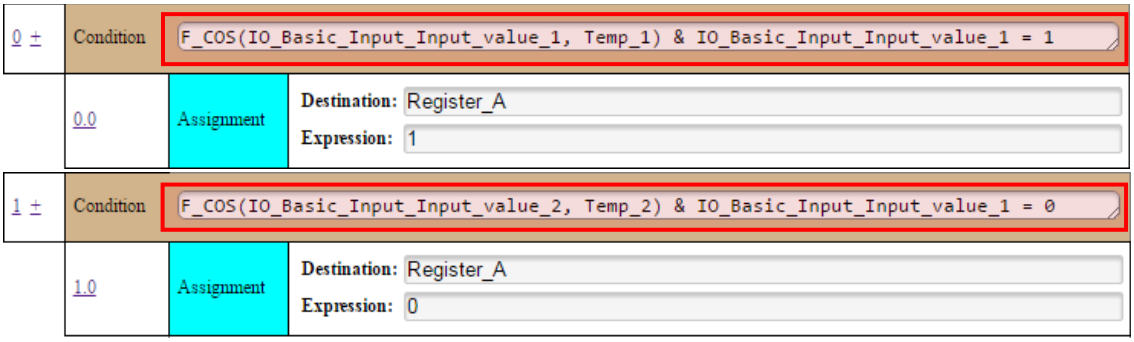


= The *Condition* that is being monitored for a *Change of State.*

= 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):*



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.

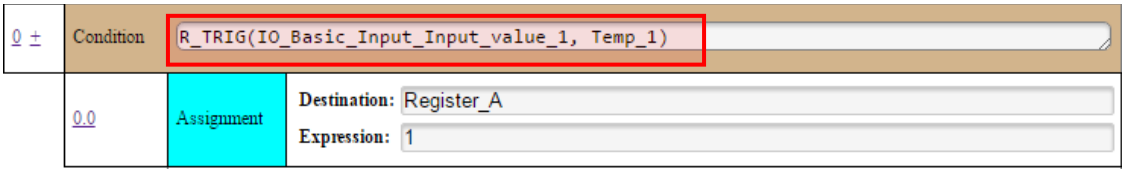
NOTE

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:*



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.

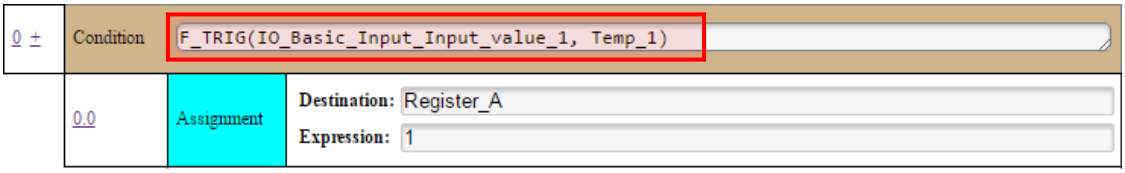
NOTE

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:*



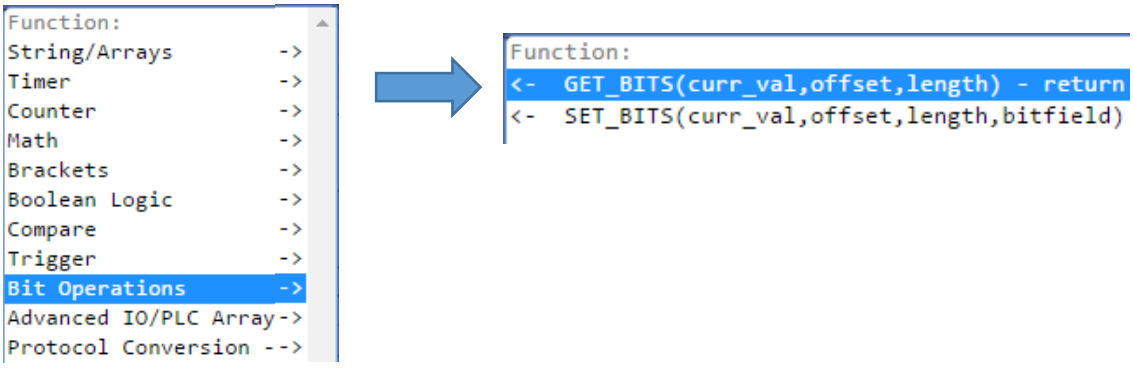
Explaining the Example: When Input_value_1 does a Change of State from high (1) to Low (0), the value "1" is loaded into Register_A

NOTE

Each monitored condition requires its own temp register.

11.2.9 Bit Operations

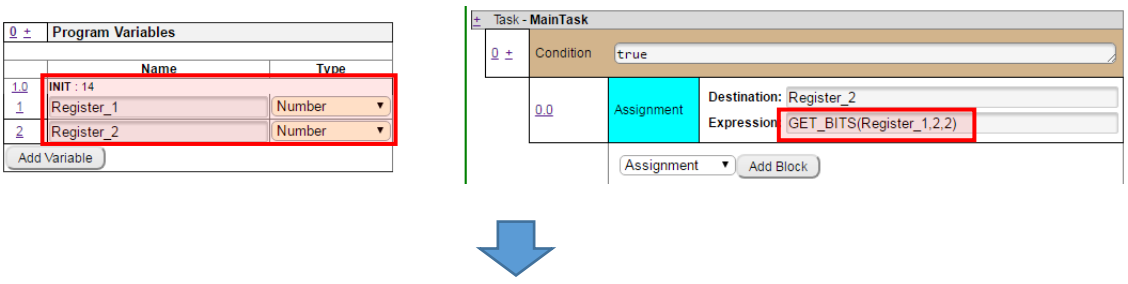
To access the Bit Operations functions, highlight *Bit Operations* and press "→" on the keyboard or click " ->" with the mouse to advance to the next or previous level.

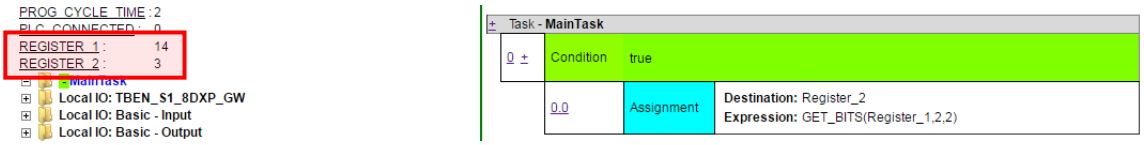


11.2.9.1 Get Bits

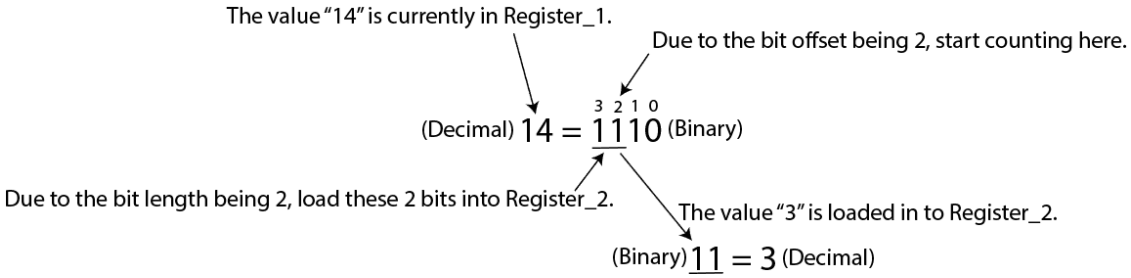
The user will use GET_BITS if the user wants to get bits from a certain register, and put them into another register.

Example of *GET_BITS (Target Register, Bit Offset, Bit Length):*





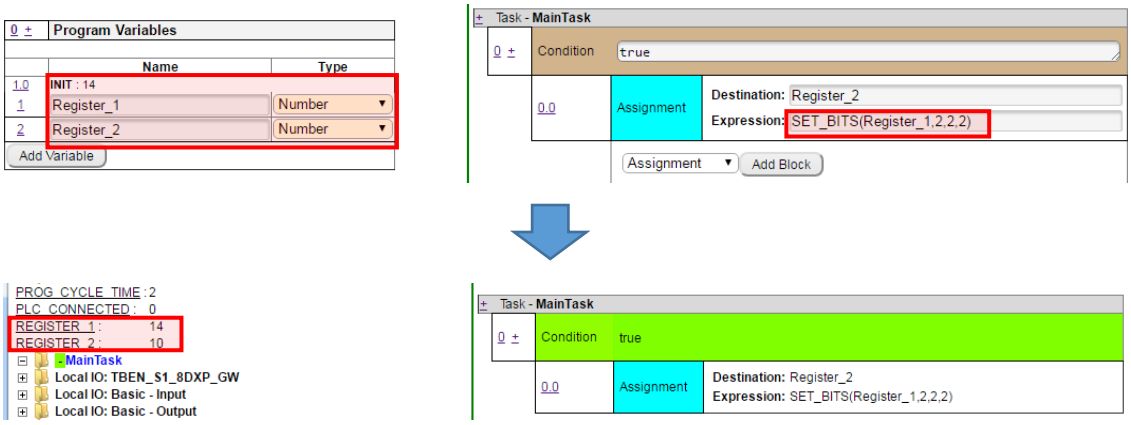
Explaining the Example: The value "14" is loaded into Register_1. Due to the bit offset being 2, ARGEE starts counting at bit 2. Due to the bit length being 2, ARGEE takes the next 2 bits, converts them to decimal, and places the value "3" in Register_2. (View the below example)



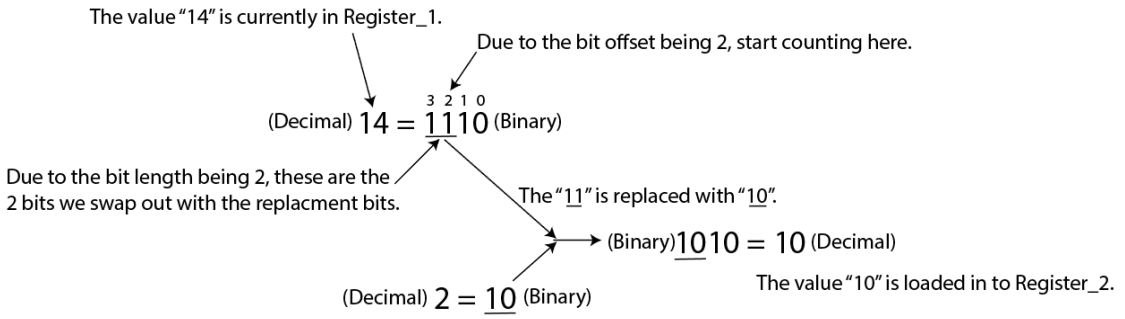
11.2.9.2 Set Bits

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):*

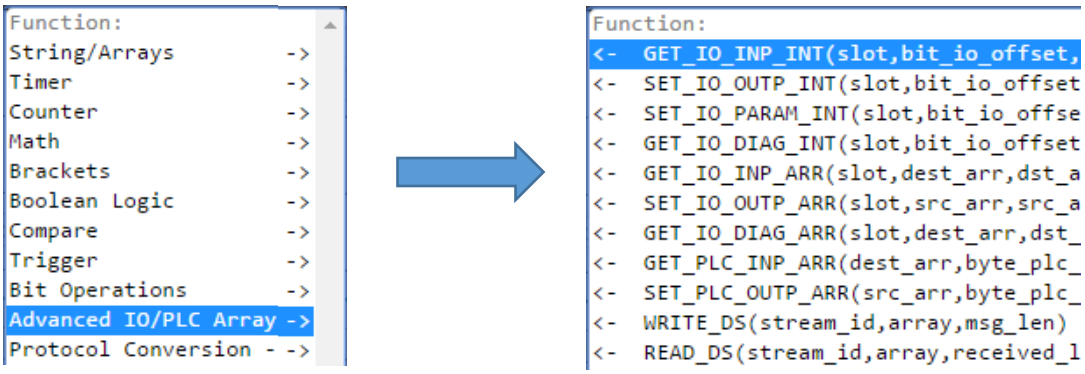


Explaining the Example: The value "14" is loaded into Register_1. Due to the bit offset being 2, ARGEE starts counting at bit 2. Due to the bit length being 2, ARGEE takes the next 2 bits, replaces those bits with the replacement value (a binary "2"), converts the new number to decimal and place that value in Register_2.



11.2.10 Advanced IO/PLC Array

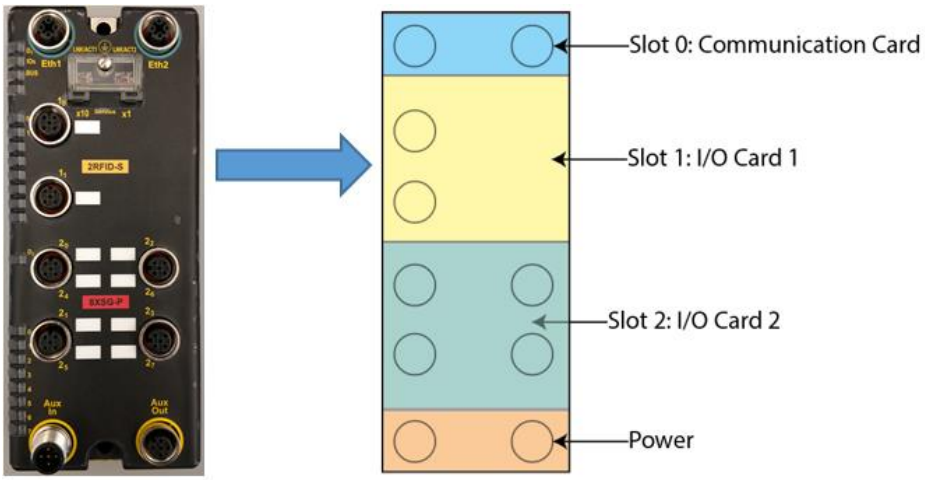
To access the Advanced IO/PLC Array functions, highlight *Advanced IO/PLC Array* and press "→" on the keyboard or click "->" with the mouse to advance to the next or previous level.



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.



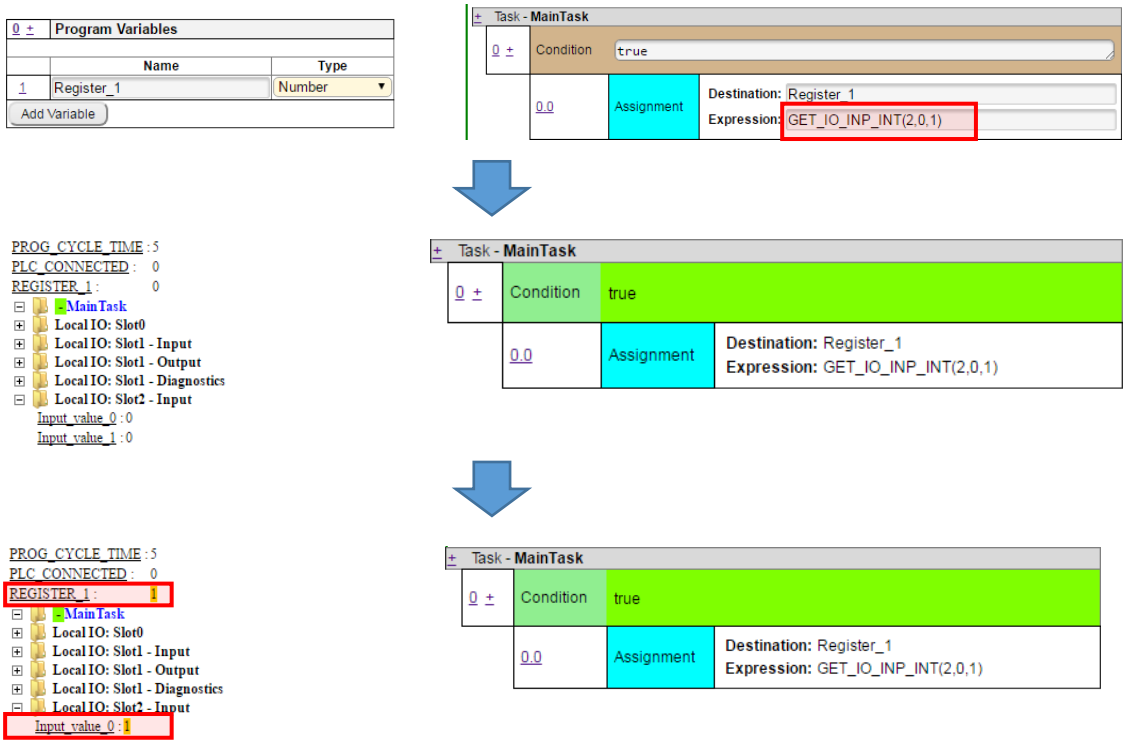
11.2.10.1 Get IO Input Integer

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$_7$	DI 2$_6$	DI 2$_5$	DI 2$_4$	DI 2$_3$	DI 2$_2$	DI 2$_1$	DI 2$_0$
	25	-	-	-	-	-	-	-	-



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.

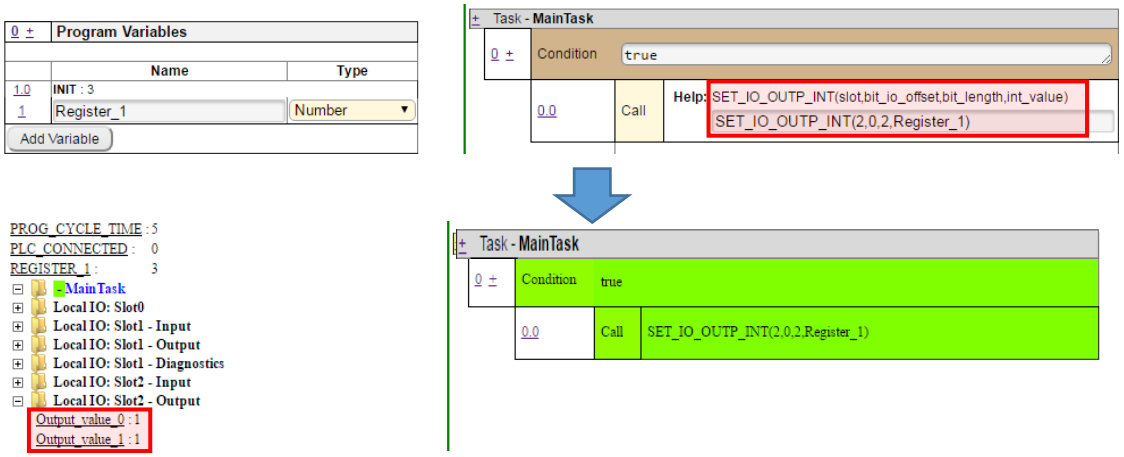
11.2.10.2 Set IO Output Integer

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_7	DO 2_6	DO 2_5	DO 2_4	DO 2_3	DO 2_2	DO 2_1	DO 2_0
	25	-	-	-	-	-	-	-	-



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

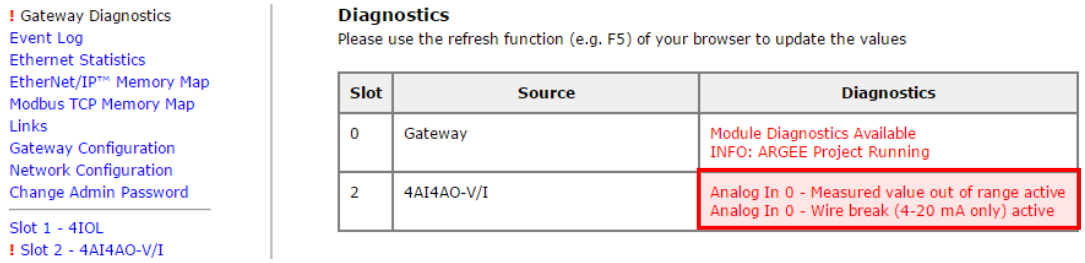
The user will use GET_IO_DIAG_INT if the user wants get diagnostic bits from a certain register and put them into another register.

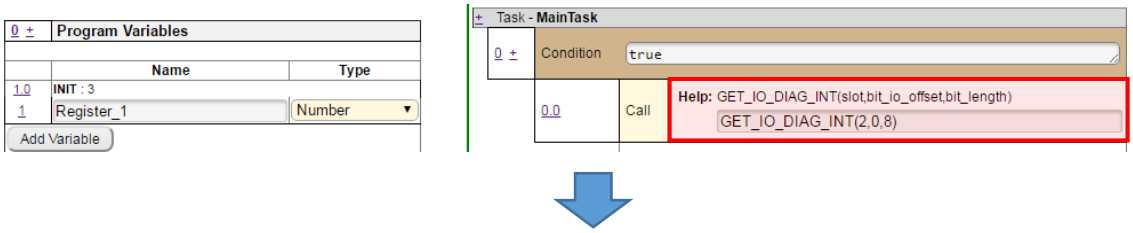
Example of *GET_IO_DIAG_INT (Target Slot, Bit Offset, Bit Length):*

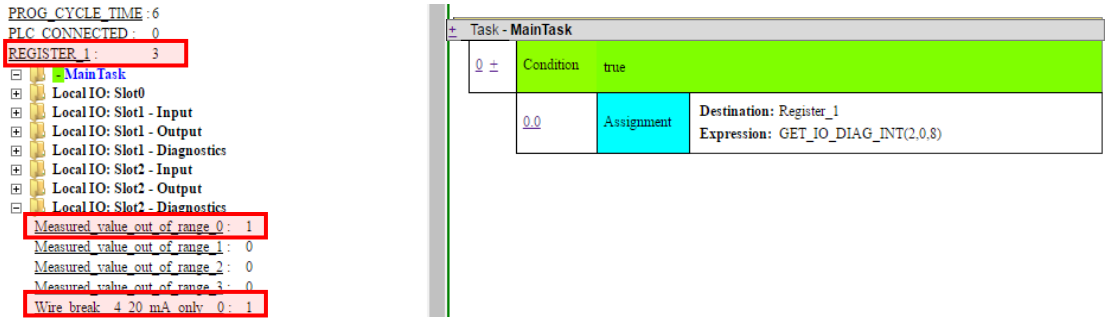
BLCEN-8M12LT-4IOL-4AI4AO-VI Diagnostic Data Map

INPUT	BYTE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Diagnostics	34	Hardware Failure	-	-	-	AI 2_0 Over-flow/Under-flow	-	Wire Break AI 2_0 (4…20 mA range only)	Range Error AI 2_0

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







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.

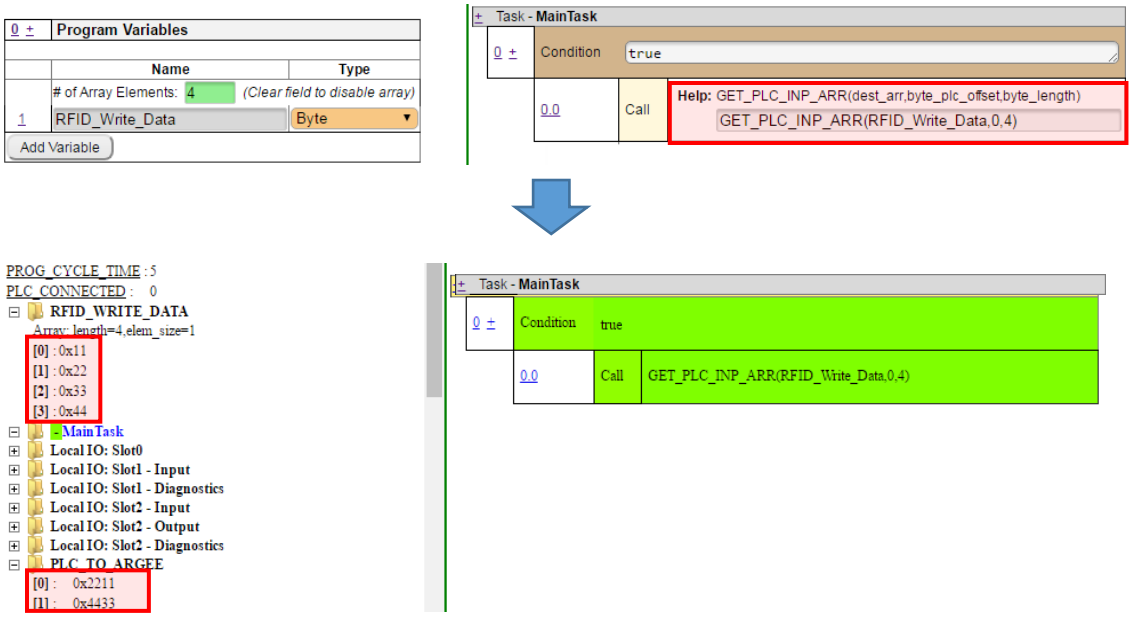
11.2.10.7 Get IO Diagnostics Array

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):*

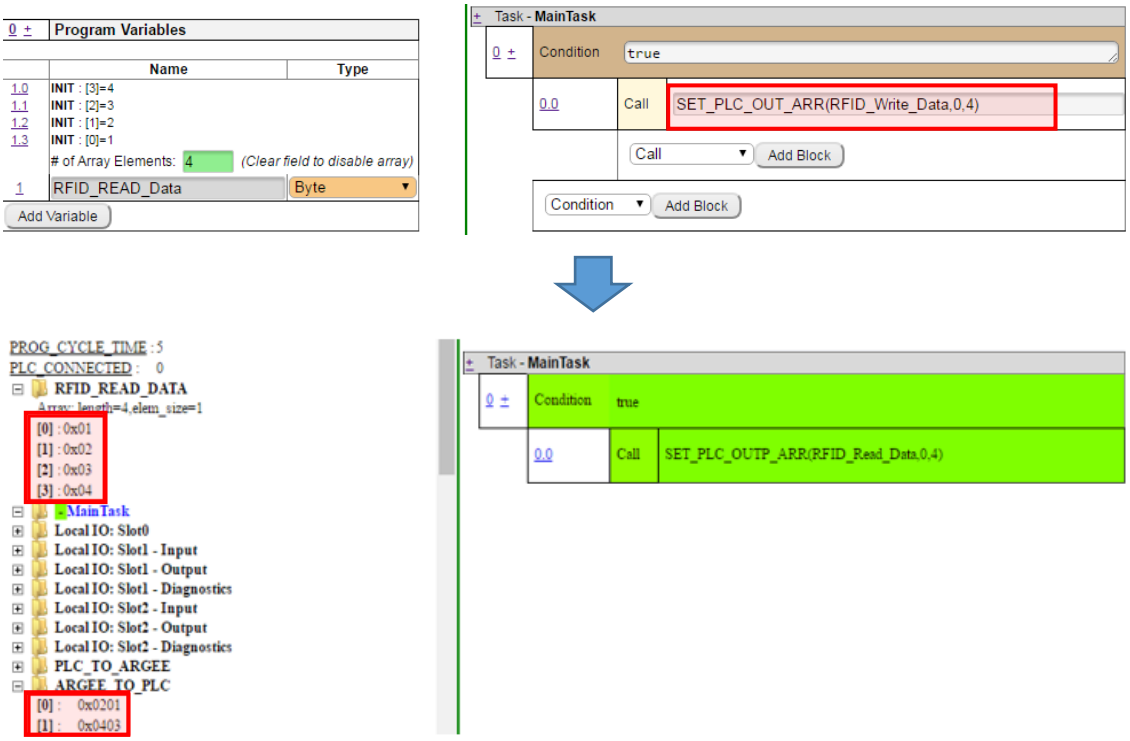


Explaining the Example: RFID write data is sent from the PLC and loaded into program variable RFID_Write_Data. The user uses the GET_PLC_INP_ARR command, sets an offset of 0 and transfers 4 bytes (or two words) from the PLC to the devices.

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):*



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

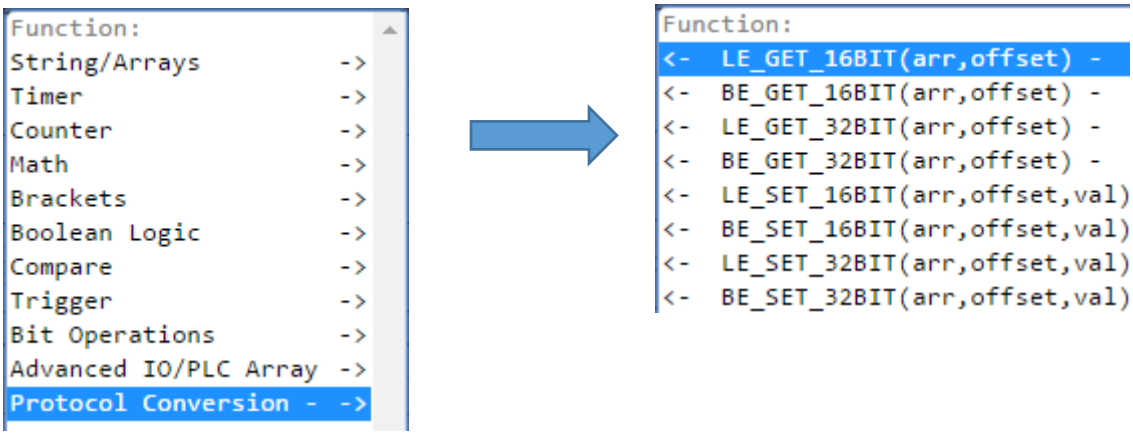
The user will use the WRITE_DS command if the user is working with acyclic messaging. This command is primarily used with IO-Link modules. Turck recommends that the user uses the ARGEE libraries when working with IO-Link. If the user wants to use this feature, please contact Turck for more information.

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

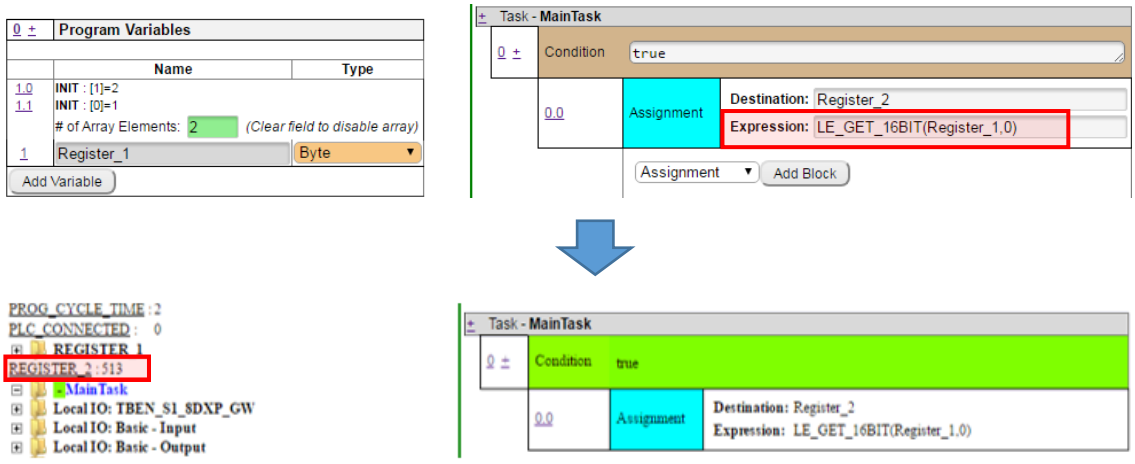
To access the Protocol Conversion functions, highlight *Protocol Conversion* and press "→" on the keyboard or click "->" with the mouse to advance to the next or previous level.



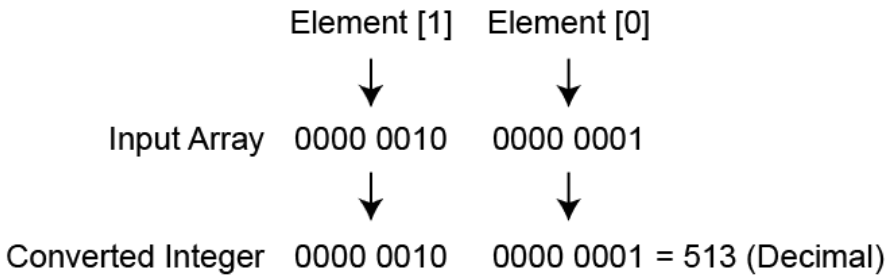
11.2.11.1 Little-endian, Get 16 Bits

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):*



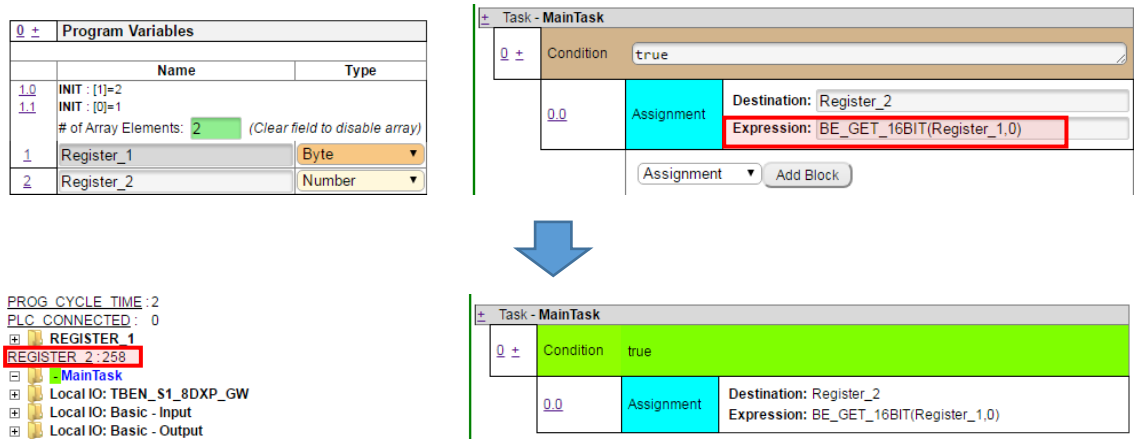
Explaining the Example: This example does not actually do anything special because the user is converting Little-endian to Little-endian. The value "1" is loaded into Register_1 position zero and the value "2" is loaded into Register_1 position one.



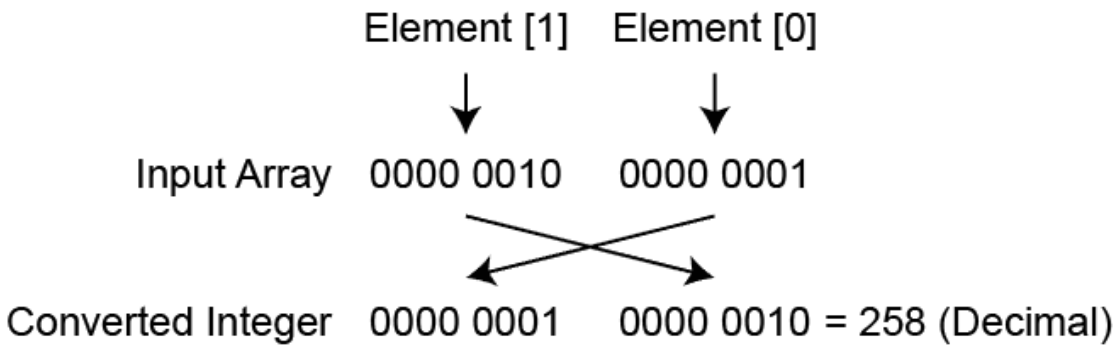
11.2.11.2 Big-endian, Get 16 Bits

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):*



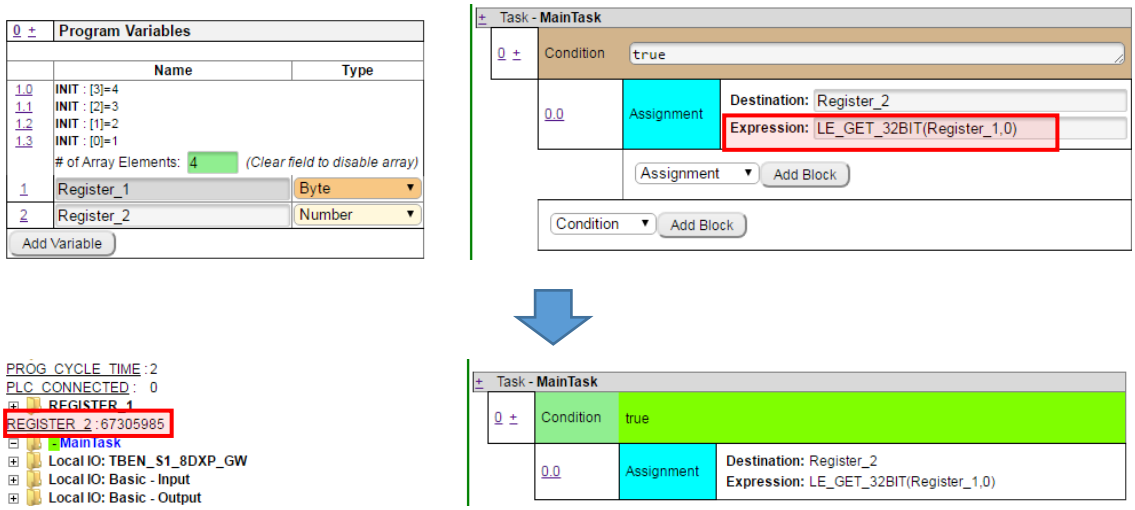
Explaining the Example: The value “1” is loaded into Register_1 position zero and the value “2” is loaded into Register_1 position one. The BE_GET_16BIT command swaps byte 1 with byte 2 and loads the value “258” into Register_2.



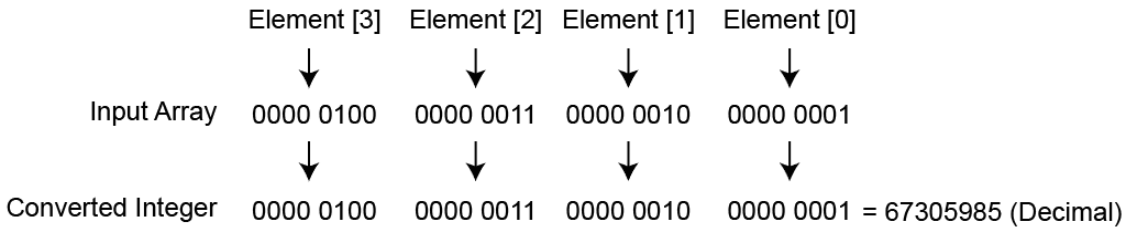
11.2.11.3 Little-endian, Get 32 Bits

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):*



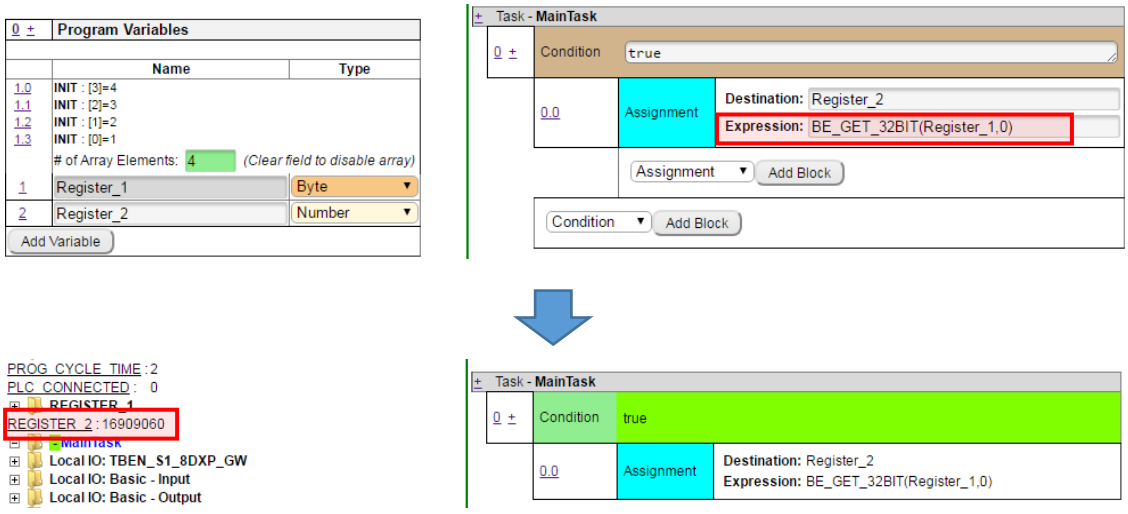
Explaining the Example: This example does not do anything special because the user is converting Little-endian to Little-endian. The value “1” is loaded into Register_1 position zero, the value “2” is loaded into Register_1 position one, the value “3” is loaded into Register_1 position two and the value “4” is loaded into Register_1 position three.



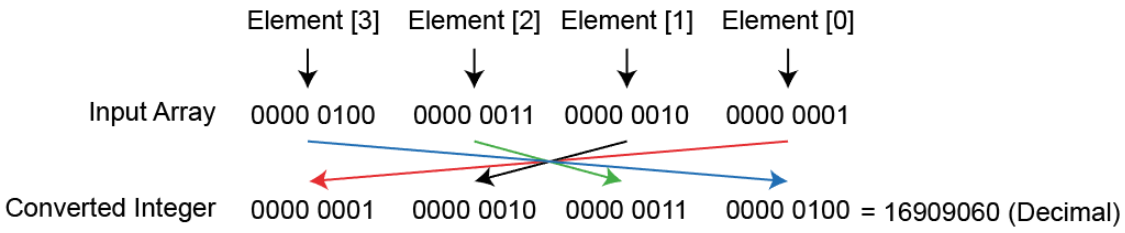
11.2.11.4 Big-endian, Get 32 Bits

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):*



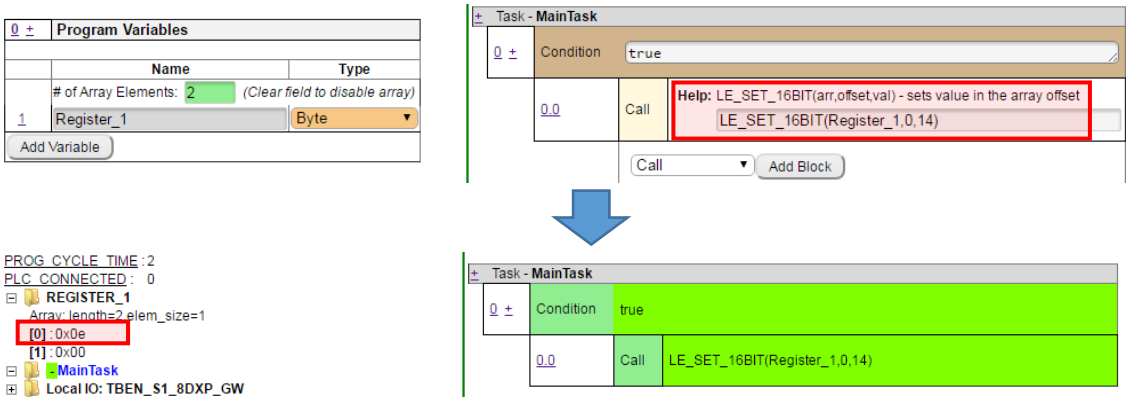
Explaining the Example: The value “1” is loaded into Register_1 position zero, the value “2” is loaded into Register_1 position one, the value “3” is loaded into Register_1 position two and the value “4” is loaded into Register_1 position three. The BE_GET_32BIT command swaps all four bytes and loads the value “16909060” into Register_2.



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):*

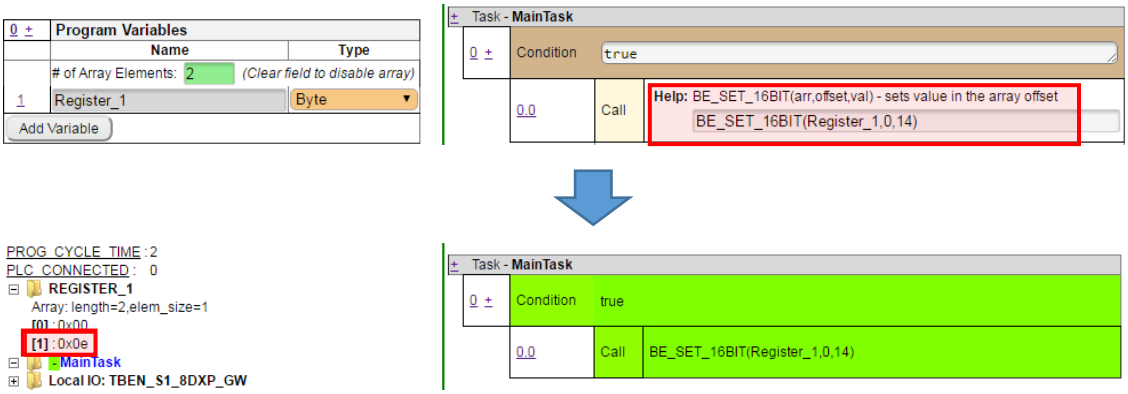


Explaining the Example: The user loads the value “14” (or Hex “e”) into Register_1. Little-endian counts bytes from right to left so the hex value “0x0e” (or decimal 14) is placed in Register_1 position zero.

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):*

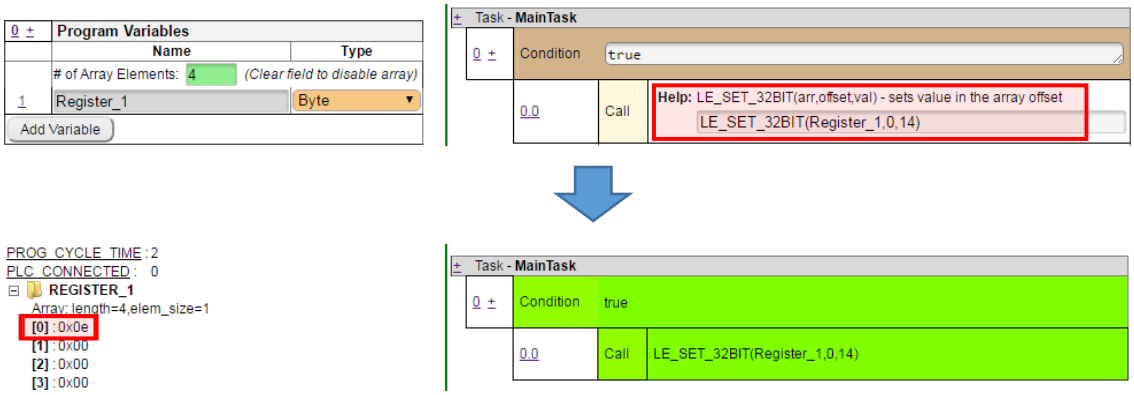


Explaining the Example: The user loads the value “14” (or Hex “e”) into Register_1. Big-endian counts bytes from left to right so the hex value “0x0e” (or decimal 14) is placed in Register_1 position one.

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):*

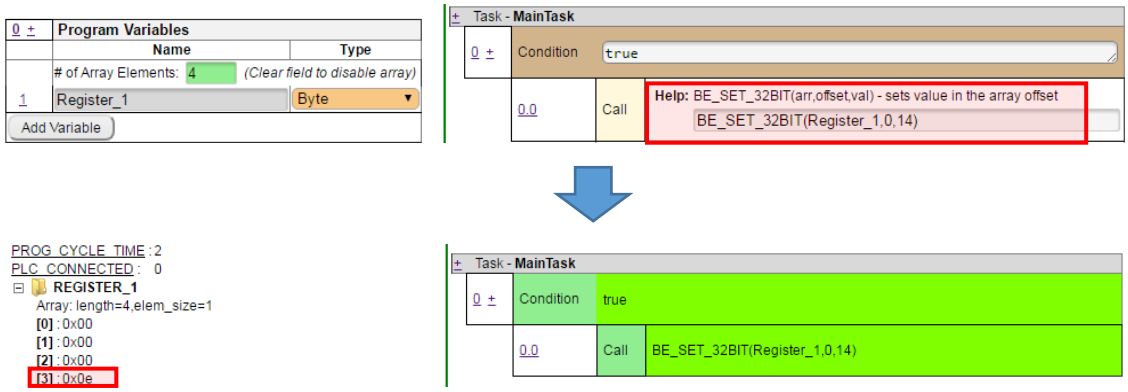


Explaining the Example: The user loads the value “14” (or Hex “e”) into Register_1. Little-endian counts bytes from right to left so the hex value “0x0e” (or decimal 14) is placed in Register_1 position zero.

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):*



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:

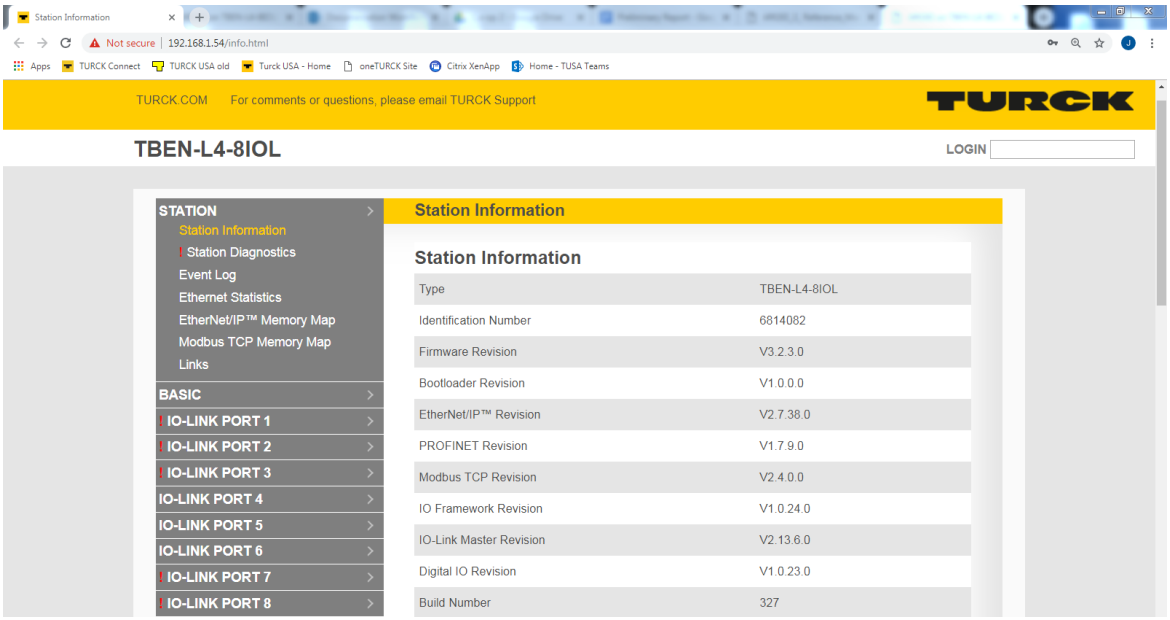
- The regular PROFINET module ID is accessible. ARGEE PROFINET connection is not allowed. If the ARGEE environment attempts to load an ARGEE code when a standard PROFINET connection is establish, the ARGEE environment will block the upload.

NOTE

PLC Connection examples can be found in chapter 10 PLC Connectivity.

11.3.3 Password Protection – ARGEE Environment

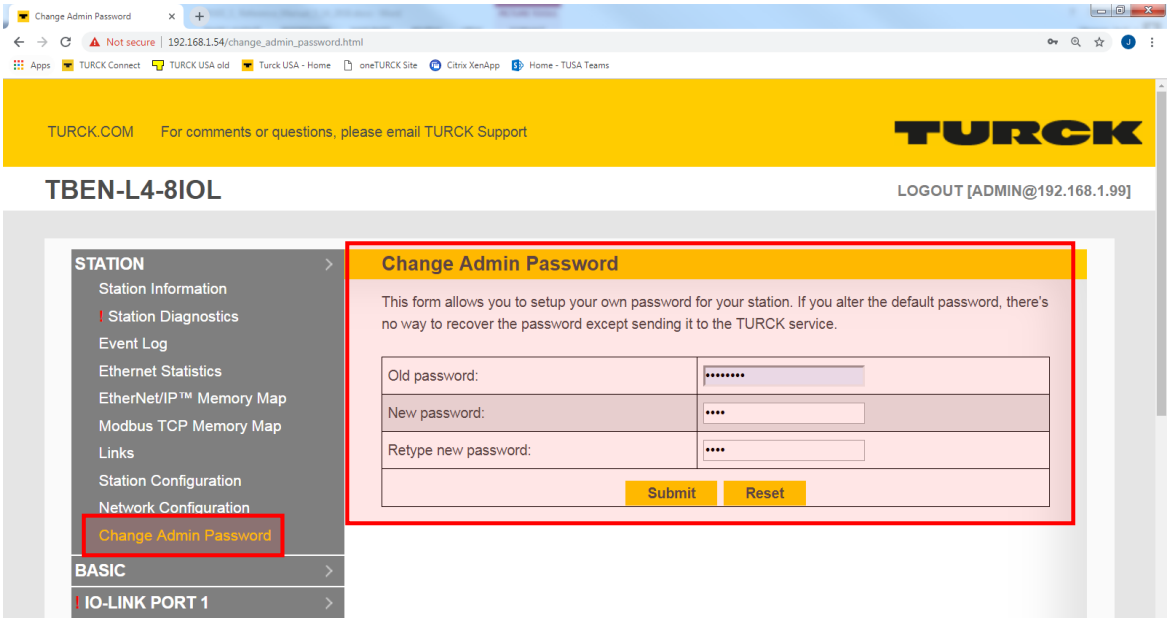
All Turck block devices support a password-protected webserver. To access the block's webserver, the user needs to type the block's IP address into any HTML5-compatible web browser.



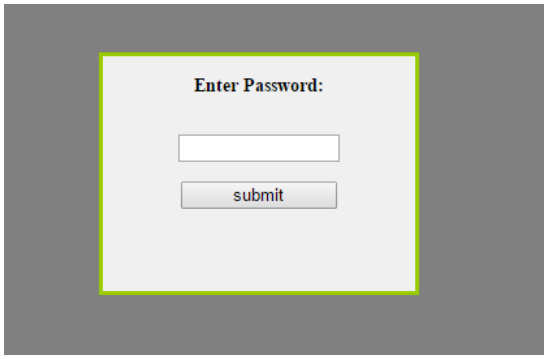
NOTE

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.



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



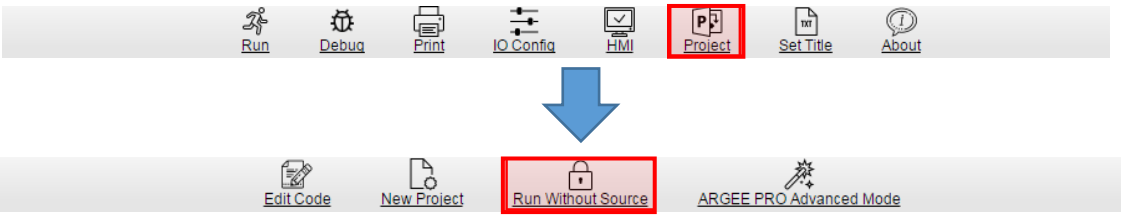
NOTE

To remove this protection, the user can simply change their webserver password back to “password”.

11.3.4 Source Code Protection – Run Without Source

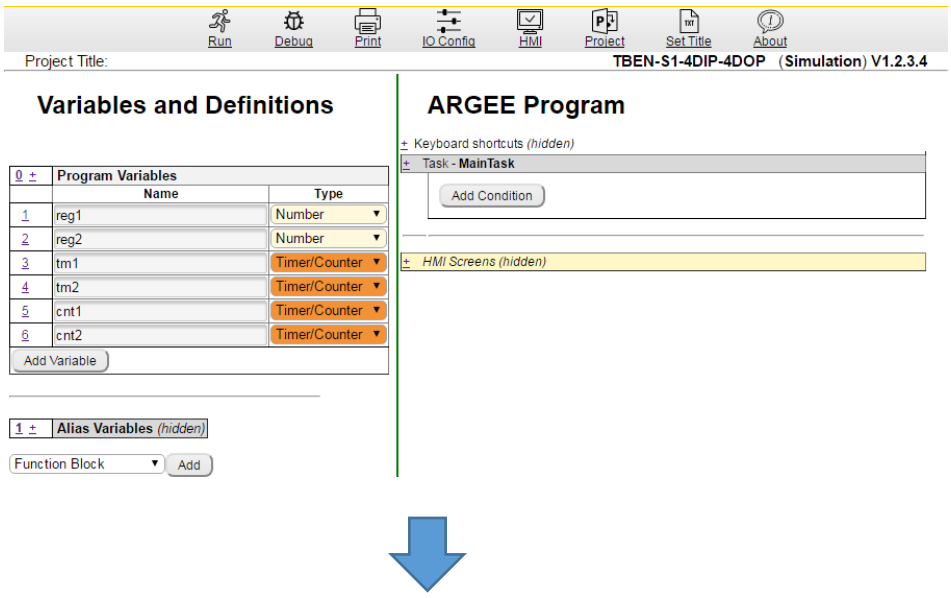
If a user wants to prevent “end users” from logging into the block and seeing or modifying code, the user will want to use the *Run Without Source* feature.

The access *Run Without Source*, the user must first click on the *Project* link in the ARGEE menu bar.

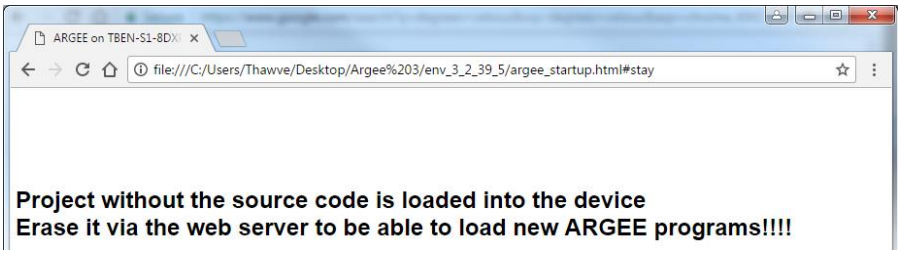


If the user clicks on *Run Without Source* and then logs out of the environment, the ARGEE program code will be hidden to anyone who tries to log into the block.

Logging in before clicking Run Without Source:



Logging in after the user click Run without Source:



NOTE

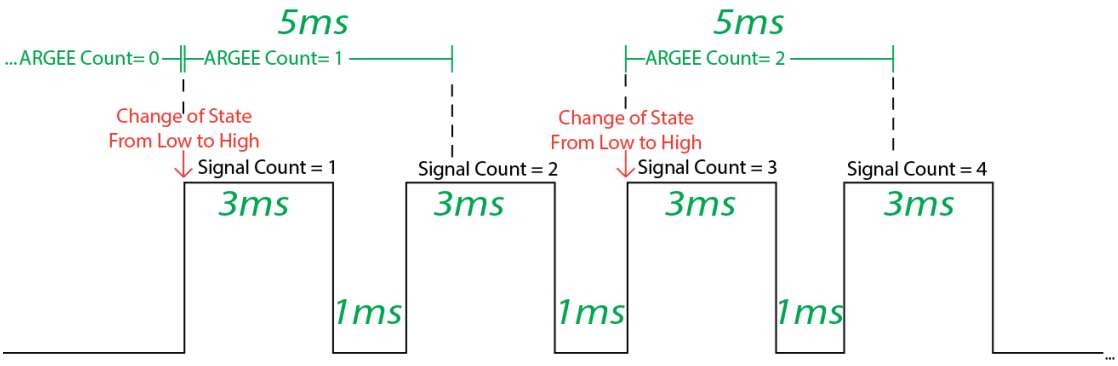
The user needs to save a Master Copy of the program before the user logs out of the environment if the user wants to view or edit the code in the future

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:



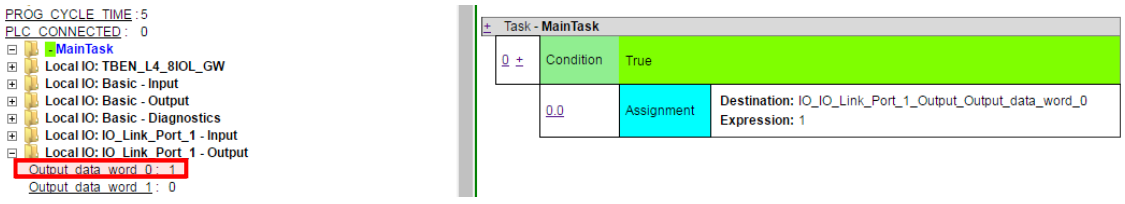
Explaining the Example: In this example, the user is hammering ARGEE with repeated 3 ms signals. Notice that ARGEE does not catch all the signals, because the signal is occurring faster than ARGEE's Scan Cycle.

NOTE

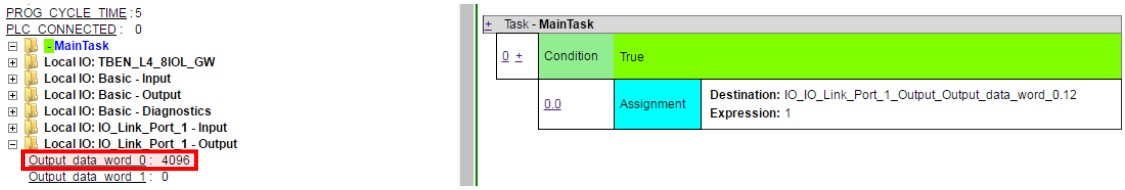
ARGEE is not suited for high speed motion applications.

11.4.2 IO Variable Formats

IO Variable Formats are normally used when working with IO-Link or transferring data with a PLC.

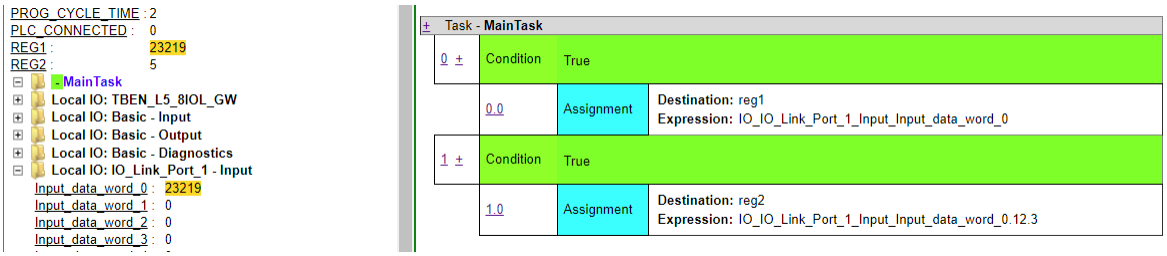


Explaining the Example: The user set IO-Link Port 1 (bit 0) true.

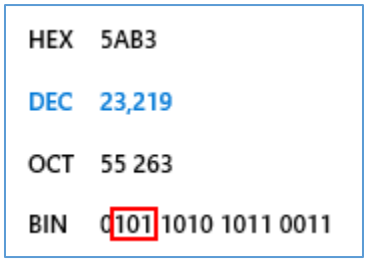


Explaining the Example: The user set IO-Link Port 1 (bit 12) true.

The user can also target bits in a word by using Word.Offset.BitLength.



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.



11.4.3 Defining Variable Types – (Advanced Definitions)

Type	Description	Type	Allowed arithmetic expressions	Specific actions	Size
Number	A 32-bit signed integer to be used for arithmetic	32-bit signed integer	All integer arithmetic	Assignment	4 bytes
Floating	Single precision floating point. Only available in TBEN and FEN20-4DIP-4DXP	32-bit signed integer	All integer arithmetic	Assignment	4 bytes
String	Null-terminated array of ASCII character values stored as bytes				X
Byte	One unsigned byte.		All integer arithmetic	Assignment	1 byte
Word	One unsigned word.		All integer arithmetic	Assignment	2 bytes
Timer/ Counter	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
State/ Enum	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
Retain Number	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)
Retain Float	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
PLC Variables	Variables mapping upper level PLC (Modbus/TCP, EtherNet/IP or PROFINET) exchange data to an integer variable accessible in the program.	They are mapped to integer variables in the program	All integer arithmetic	Assignment	8 bytes (4 bytes of data, 4 bytes of additional information)
Local IO	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)
System Variables – PLC Connected	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)
System Variables – Program Cycle Time	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

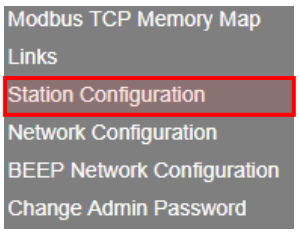
The user can find their device's IP Address on the block itself, located in the hatch, set by rotary dials, or by using the Turck Service Tool application.

- 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.

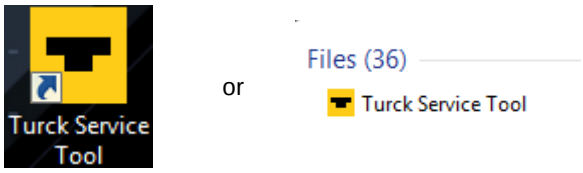


➢ At the bottom of the page click Erase ARGEE Program

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

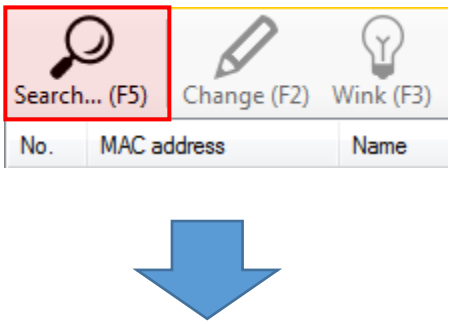


NOTE

The Turck Service Tool is available for download at www.turck.us

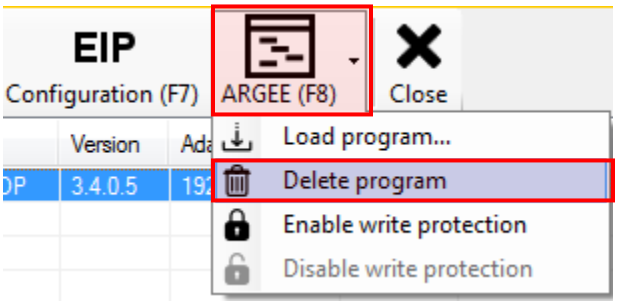
---

➢ Click the search tab to find your device.

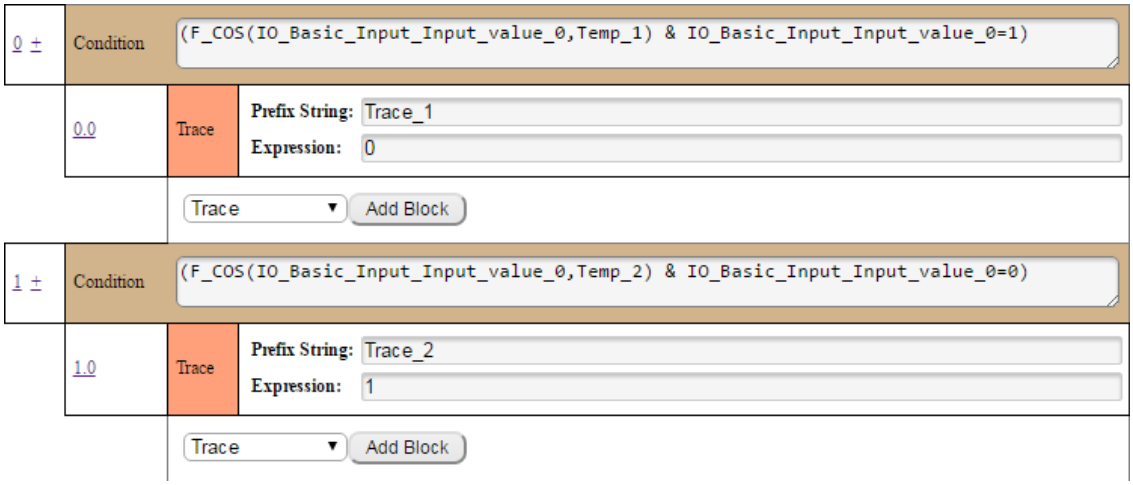


➢ Enable Expert View.

➢ 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



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.

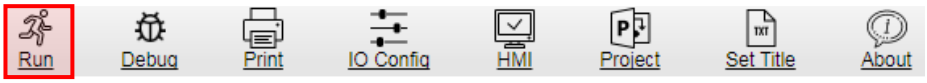
NOTE

An example of *Trace* can be found in the Appendix 11.2.8.1 Change of State (F_COS).

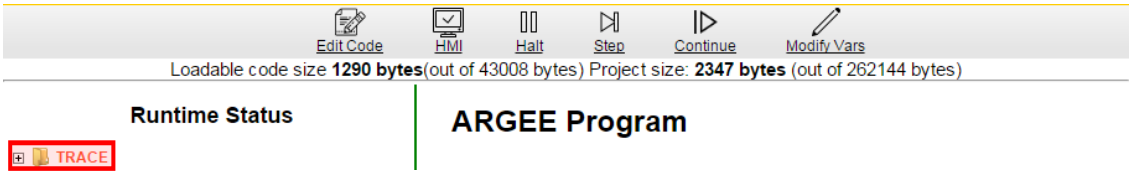
The Trace example is continued on the next page.

Trace Example (Continued):

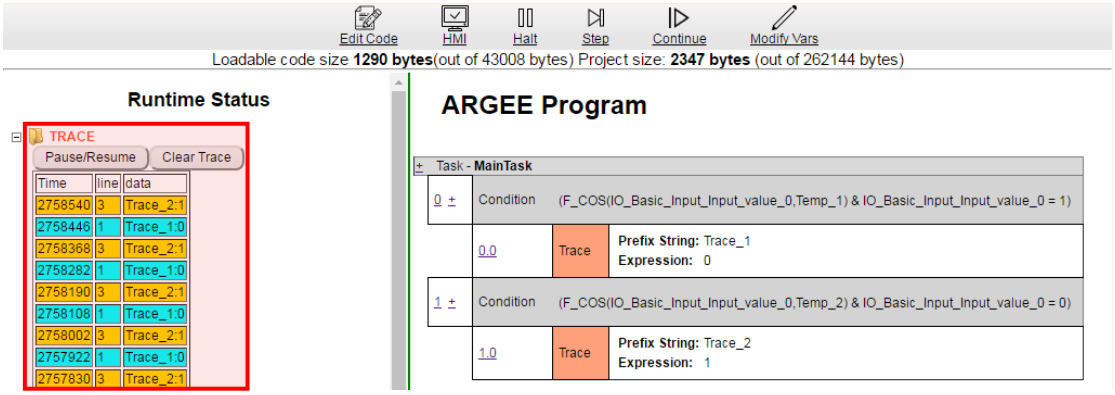
- Once the user has written the code, the user will click Run.



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.

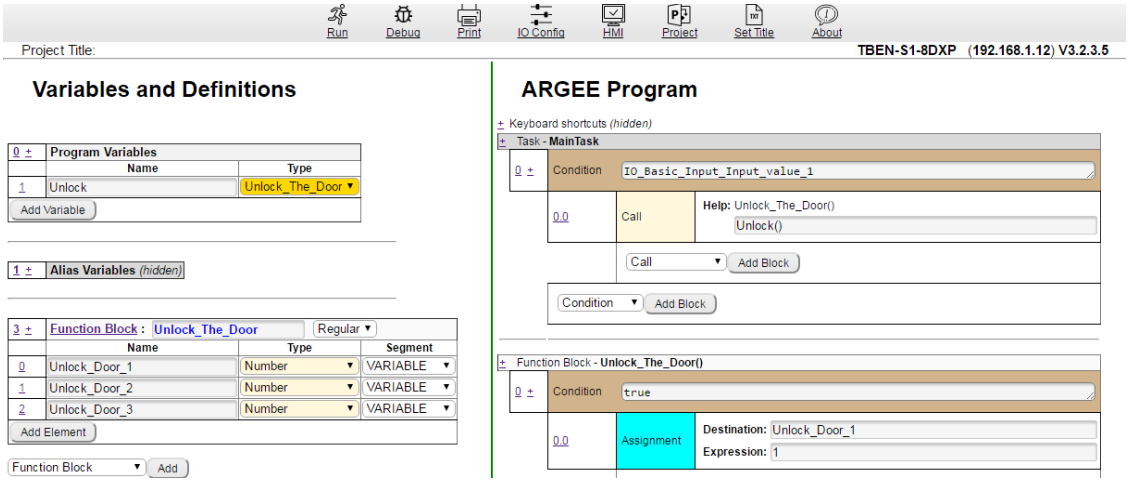


NOTE

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:



Explaining the Example: When Input_value_1 goes true, the function block *Unlock_The_Door* will be called.

NOTE

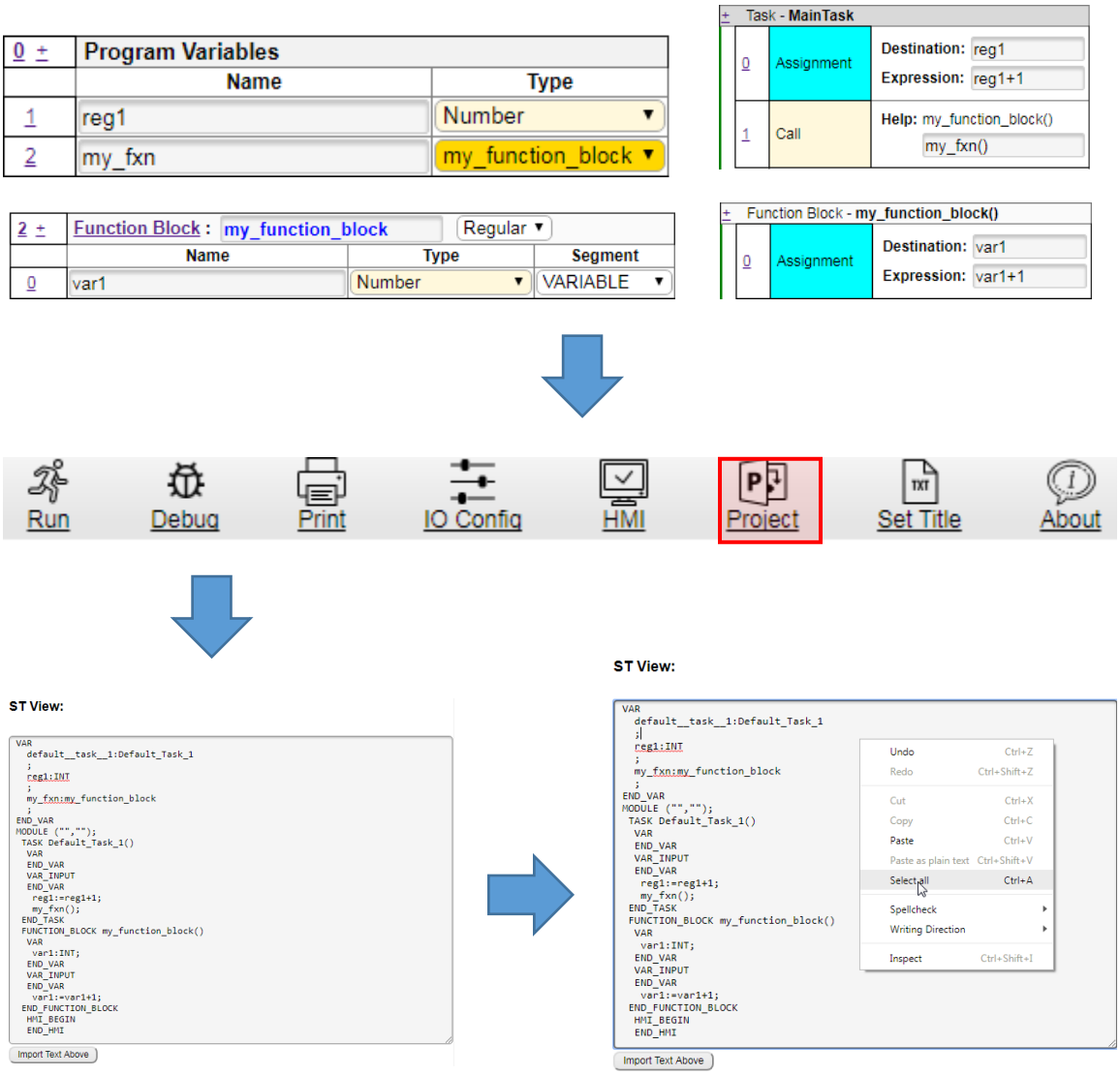
Function blocks are explained in Chapter 5.8 Function Blocks.

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)



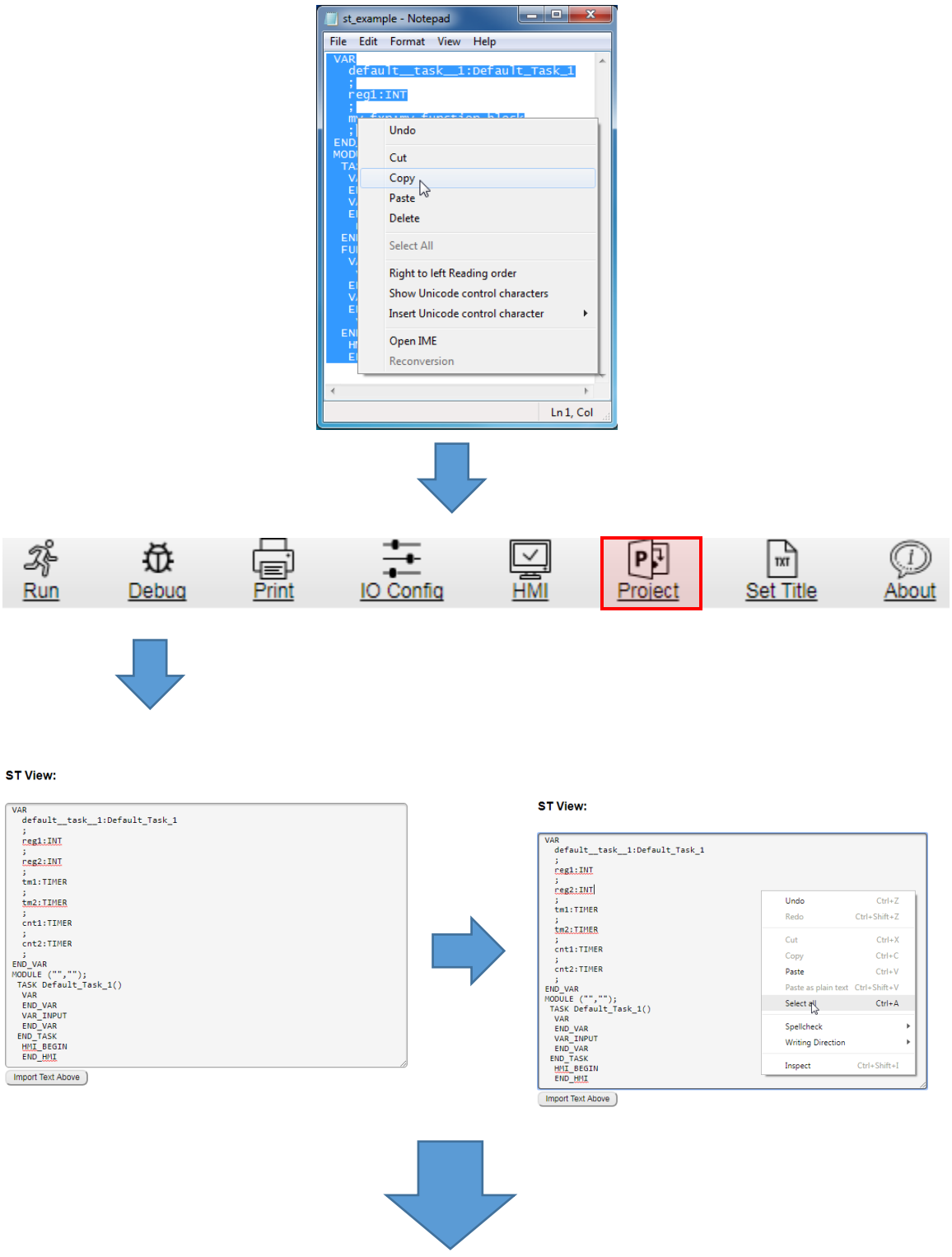
NOTE

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 Notpad.

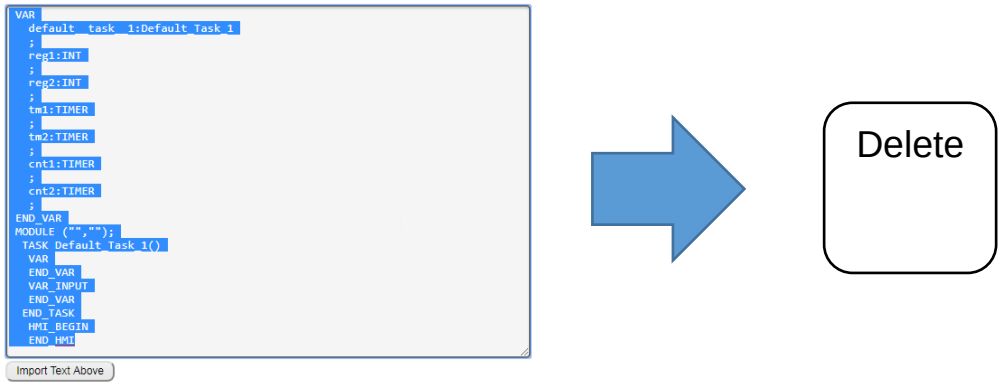
Explaining the Example: An ARGEE project's Structure Text was copied, pasted, and saved into a .txt file.

12.4.2 Example of Importing Structure Text and Converting it into an ARGEE Project.

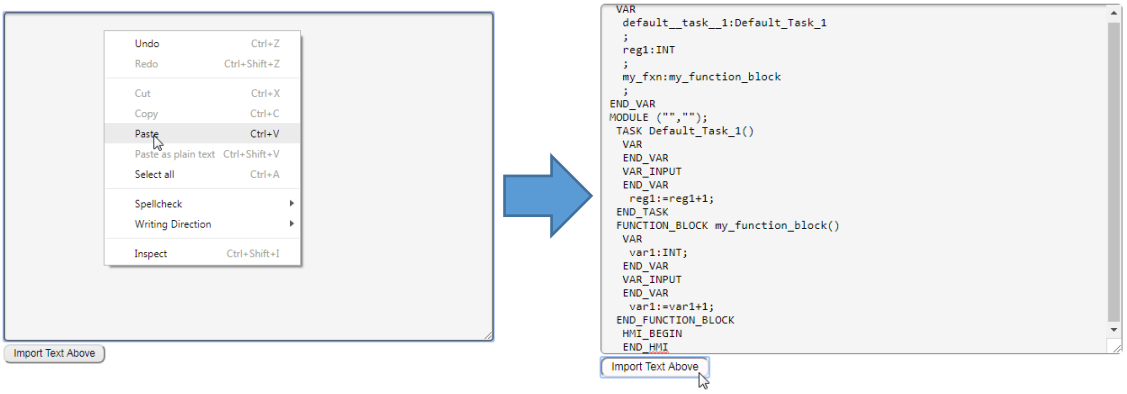
Open the file where the Structure Text is to be copied from. In this case, a .txt file created with Notepad was used to store the Structure Text from the previous example. Select it all, copy it all, and then switch to your open ARGEE environment:

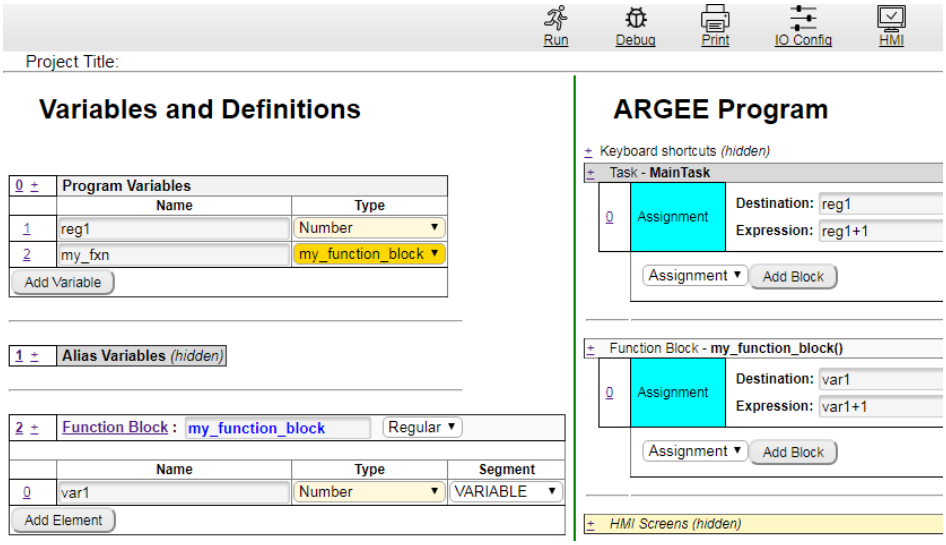


ST View:



ST View:





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.*

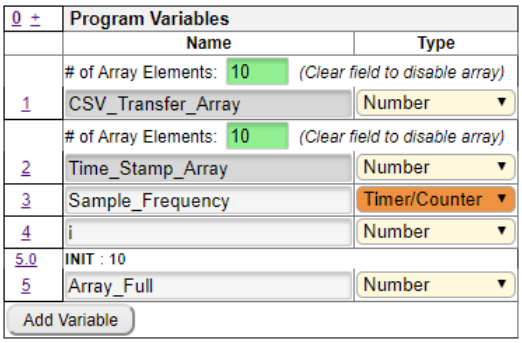
12.5 How to Export a CSV File

12.5.1 HMI export of arrays

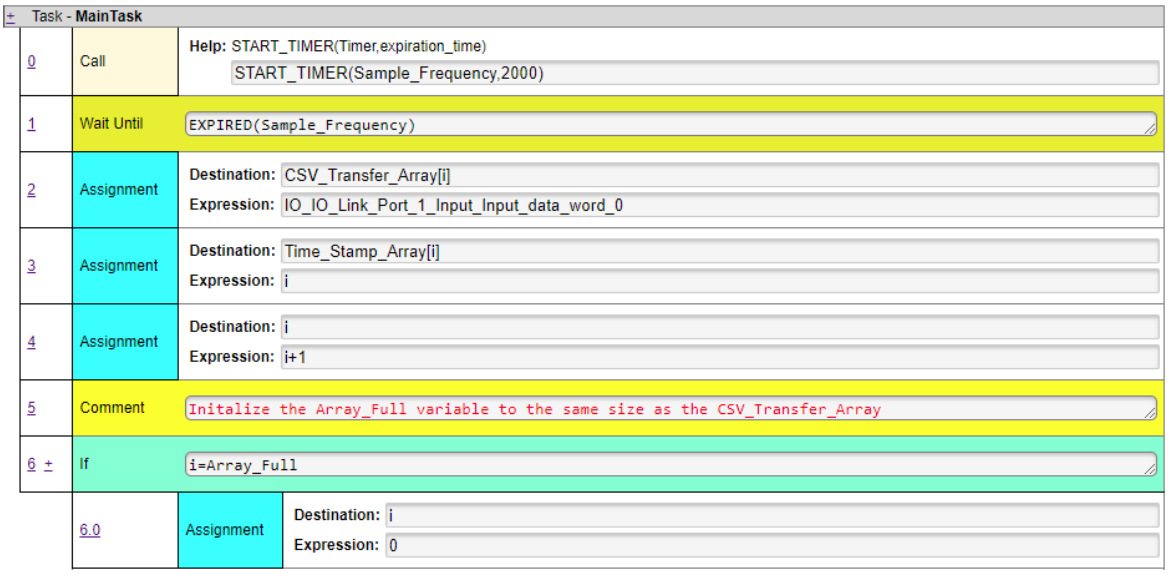
The HMI can export a CSV file with the Submit Action. The CSV is saved to the Downloads folder on the connected computer. This action requires the following arguments.

- 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.

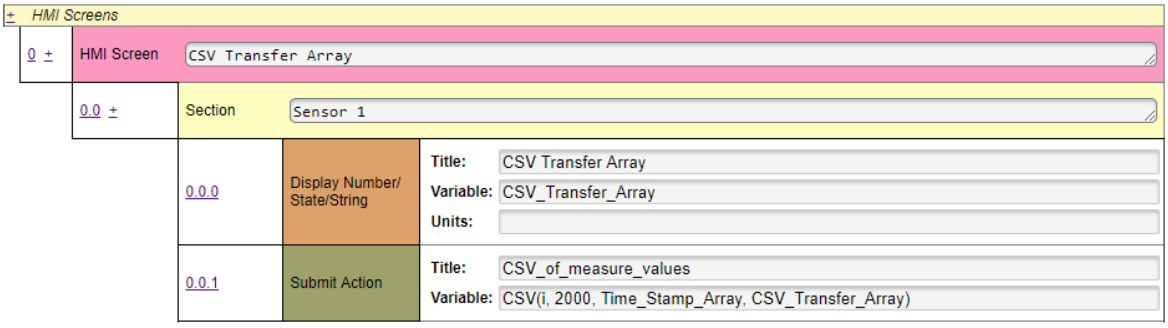
12.5.2 Example of Exporting a CSV



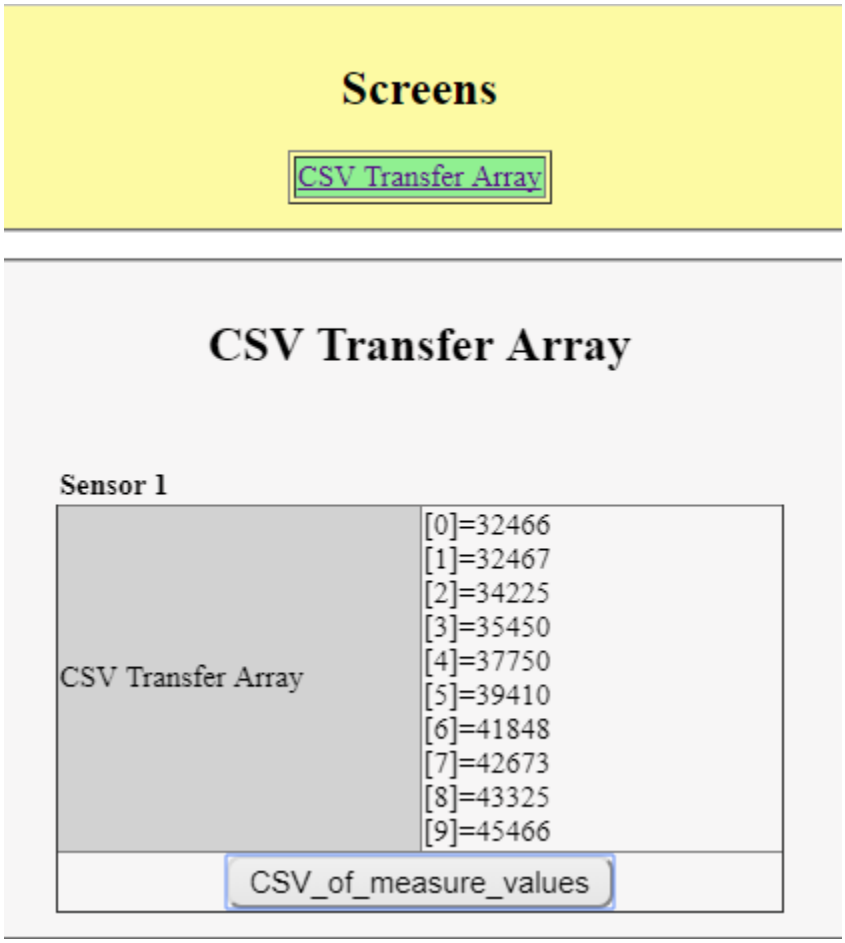
The user creates arrays for measurements and the timestamp.



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.



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

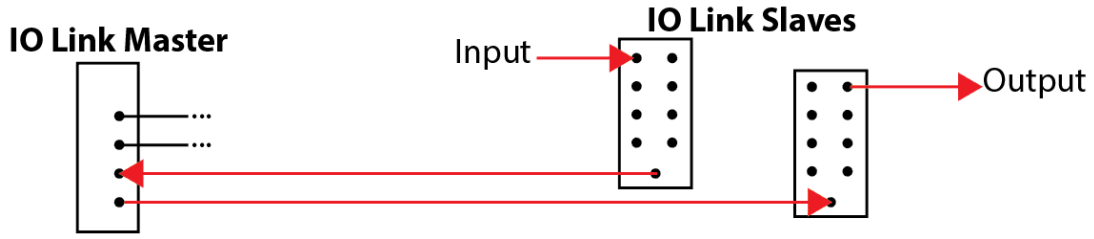
12.6.1 Working with IO-Link

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.

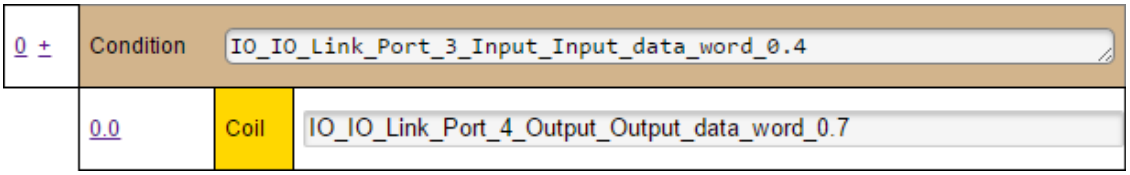
12.6.1.1 Working with IO-Link

Example of *IO-Link:*

(Customer's Application)



(ARGEE Setup)



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

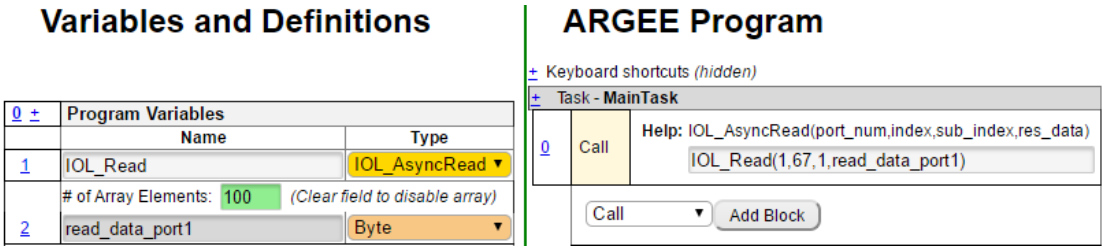
When working with acyclic communication, the first thing the user needs to do is import the IO-Link libraries (Importing libraries is discussed in Chapter 5.9.3 Importing a Library).

Example of *Acyclic Communication – Read:*

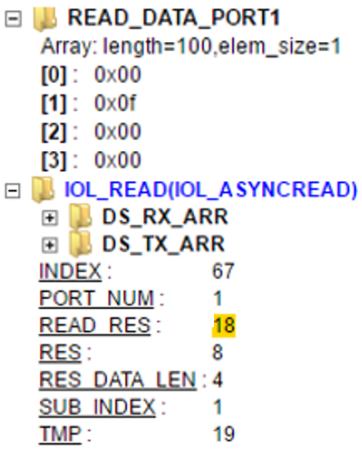
(IODD file for an IO-Link Device)

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

(ARGEE Setup)



(ARGEE Debug Screen)



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

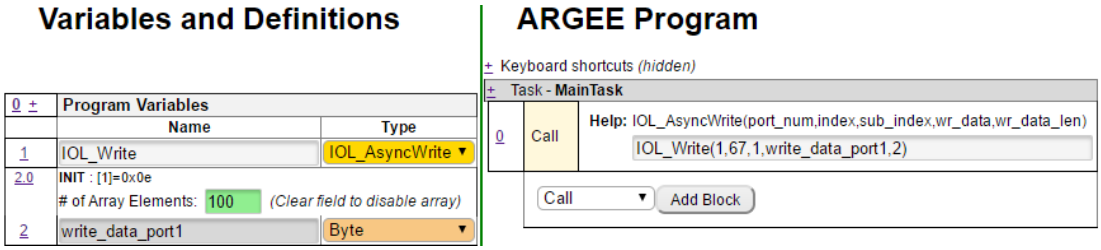
When working with acyclic communication, the first thing the user needs to do is import the IO-Link libraries (Importing libraries is discussed in chapter 5.9.3 Importing a Library).

Example of *Acyclic Communication –Write:*

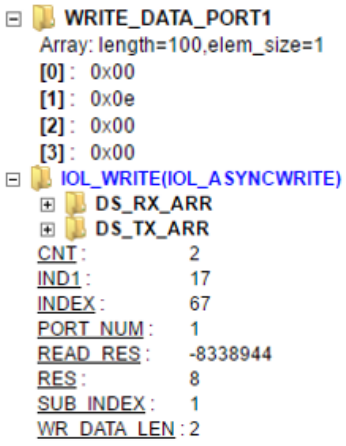
(IODD file for an IO-Link Device)

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)



(ARGEE Debug Screen)



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

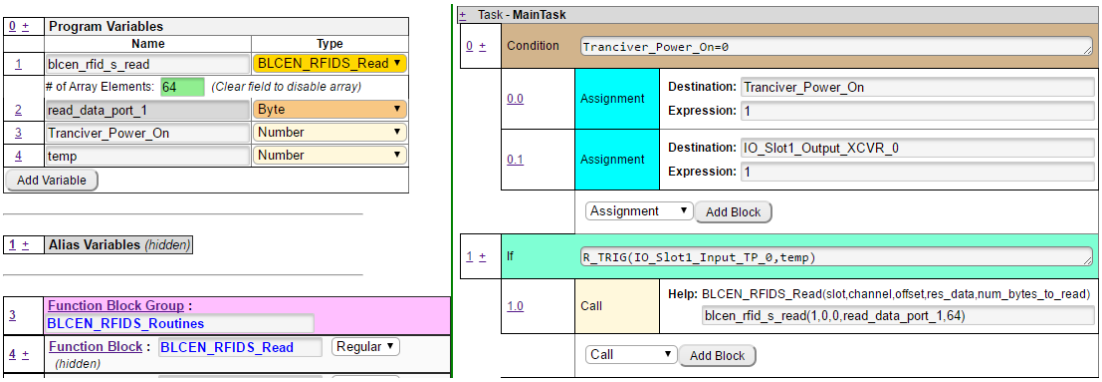
Many factors influence RFID Read/Write applications. The user can reference the RFID user manual for more information about RFID.

12.6.2.1.1 RFID Communication – Read

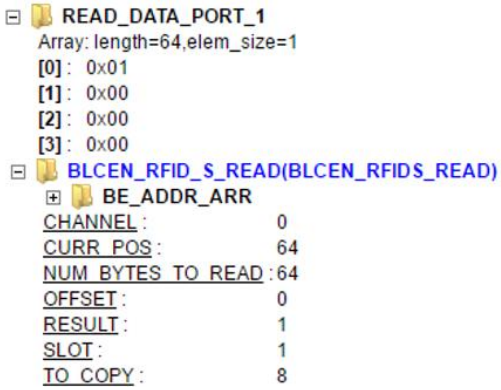
When working with RFID, the first thing the user needs to do is import the RFID libraries (Importing libraries is discussed in chapter 5.9.3 Importing a Library).

Example of *RFID Communication – Read:*

(ARGEE Setup)



(ARGEE Debug Screen)



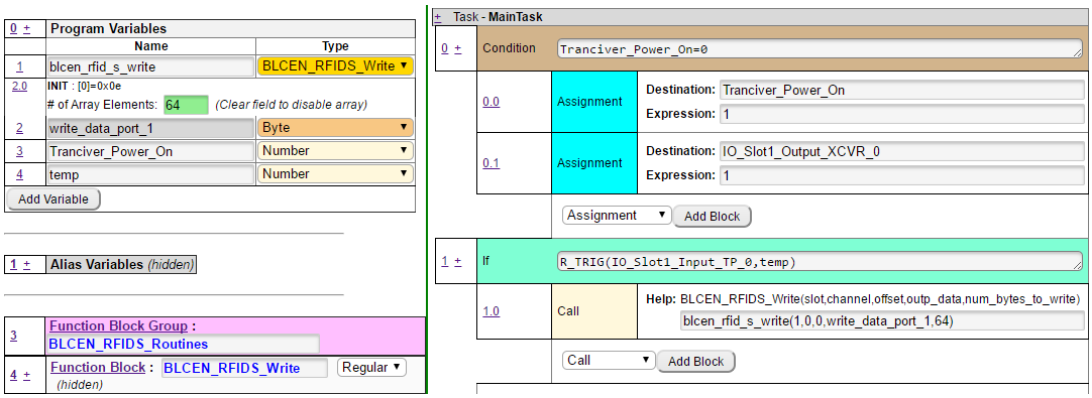
Explaining the example: The user input five arguments into the *BLCEN_RFIDS_Read* function block: Slot number, channel number, bit offset, result data location, and number of bytes to read. *Condition statement 0* in the code is used to power up the transceiver. *If statement 1* says whenever the tag 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

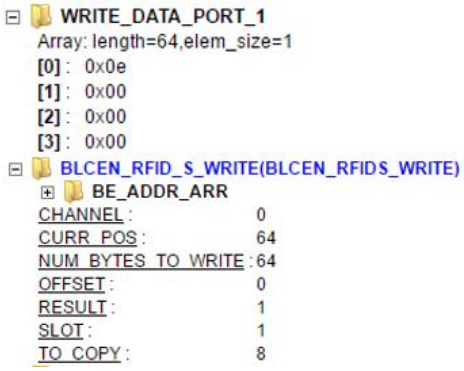
When working with RFID, the first thing the user needs to do is import the RFID libraries (Importing libraries is discussed in chapter 5.9.3 Importing a Library).

Example of *RFID Communication – Write:*

(ARGEE Setup)



(ARGEE Debug Screen)



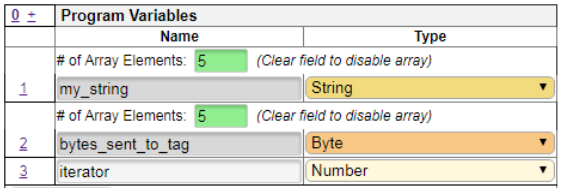
Explaining the example: The user input five arguments into the *BLCEN_RFIDS_Write* function block: Slot number, channel number, bit offset, output data location, and number of bytes to write. *Condition statement 0* in the code is used to power up the transceiver. *If statement 1* says, whenever the 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

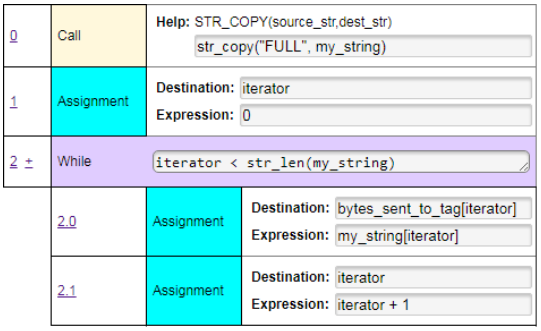
When working with RFID, the first thing the user needs to do is import the RFID libraries (Importing libraries is discussed in Chapter 5.9). Strings cannot be written or read from RFID tags directly. If writing, the user's string must be converted to a byte array, then written to the tag. If reading, 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)



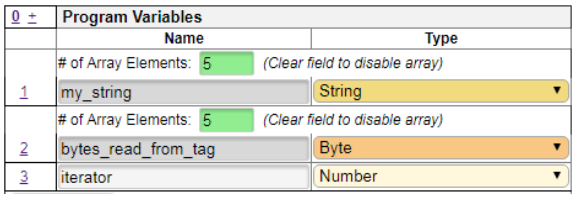
(ARGEE Code)



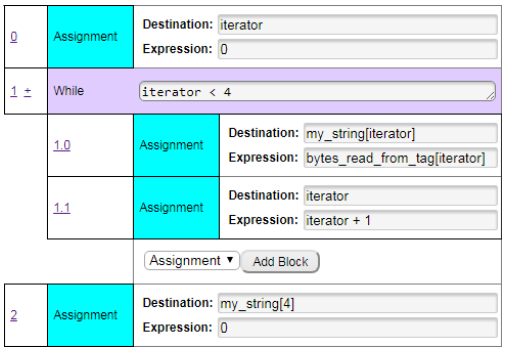
Explaining the example: The user wants to write the string "FULL" to an RFID tag. The characters "FULL" are stored in my_string, then my_string is copied element-by-element to the byte array called bytes_sent_to_tag. The data in bytes_sent_to_tag is now ready to be written to the tag, using the appropriate Write function from the Turck RFID library (not shown).

12.6.2.1.3.2 Example of RFID Communication – Reading Strings

(ARGEE Setup)



(ARGEE Code)



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 " "

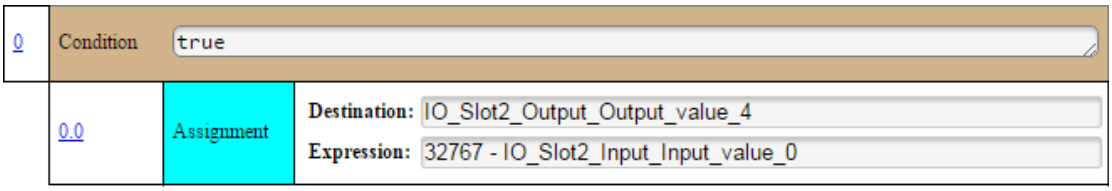
12.6.3 Working with Analog

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)

(ARGEE Setup)



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 – 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.

(ARGEE Setup)

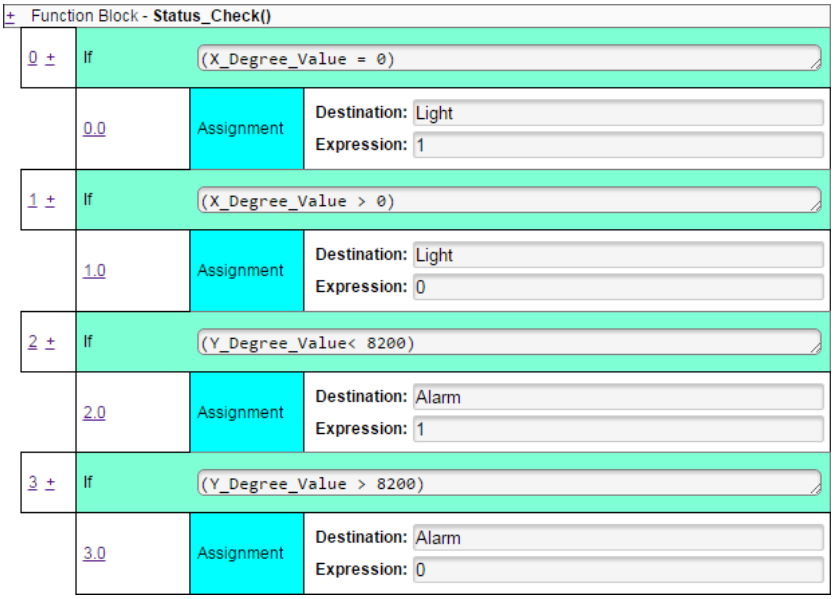
0 +	Program Variables	
	Name	Type
1	Conversion	Convert
2	Status	Status_Check

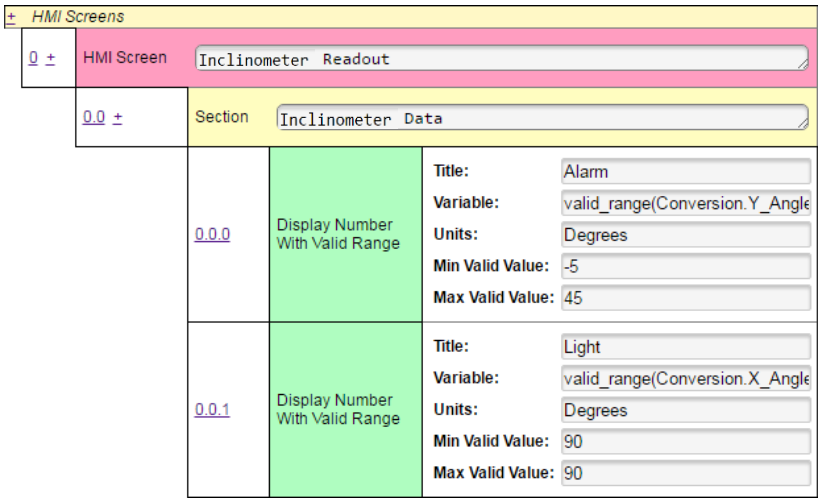
1 +	Alias Variables	
	Name	IO Point
0	X_Degree_Value	IO_Slot1_Input_Input_value_2
1	Y_Degree_Value	IO_Slot1_Input_Input_value_0
2	Light	IO_Slot2_Output_Output_value_0
3	Alarm	IO_Slot2_Output_Output_value_4

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)

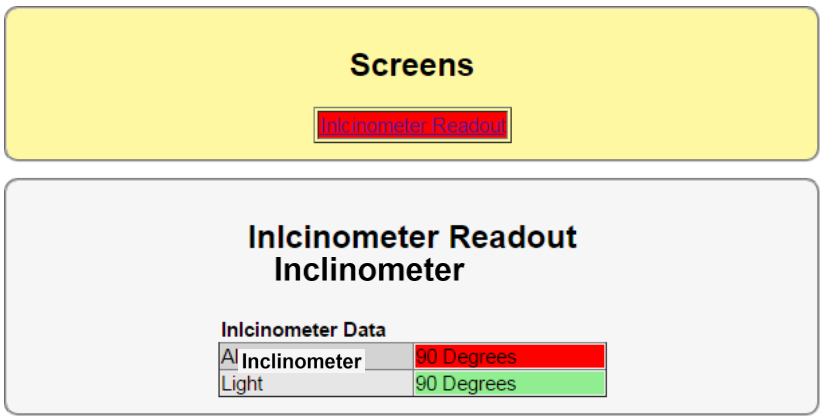
+	Task - **MainTask**	
0	Call	**Help:** Convert() Conversion()
1	Call	**Help:** Status_Check() Status()

+	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

(ARGEE HMI)


Screens
Inlcinometer Readout
Inlcinometer Readout
Inclinometer
Inlcinometer Data
Al Inclinometer
90 Degrees
Light
90 Degrees

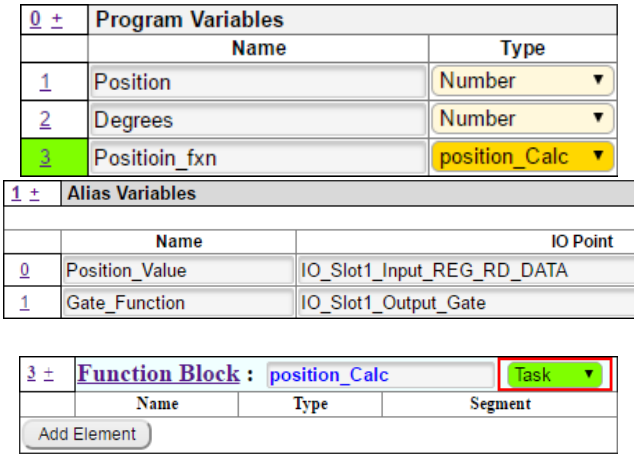
Explaining the example: The user wrote the code to monitor the angle of the boom in both the X and Y axis. The angle of the boom in the Y axis is sounding an Alarm while the angle in the X axis is appropriate for operation.

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.

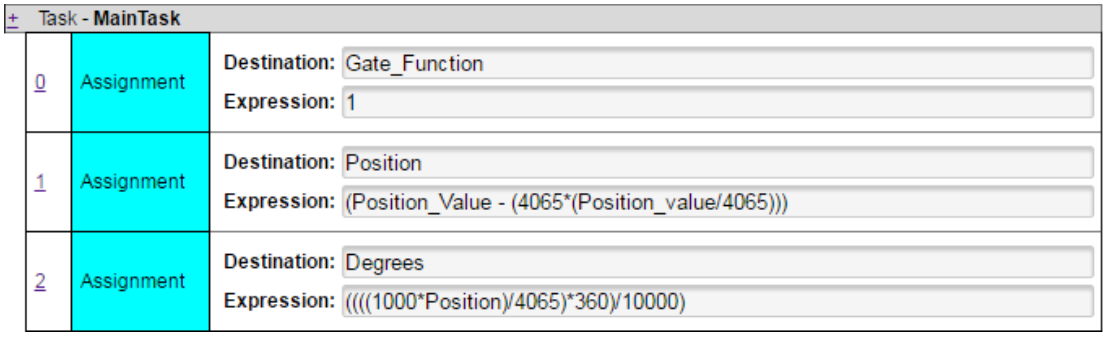
Example of *Working with Encoders:*

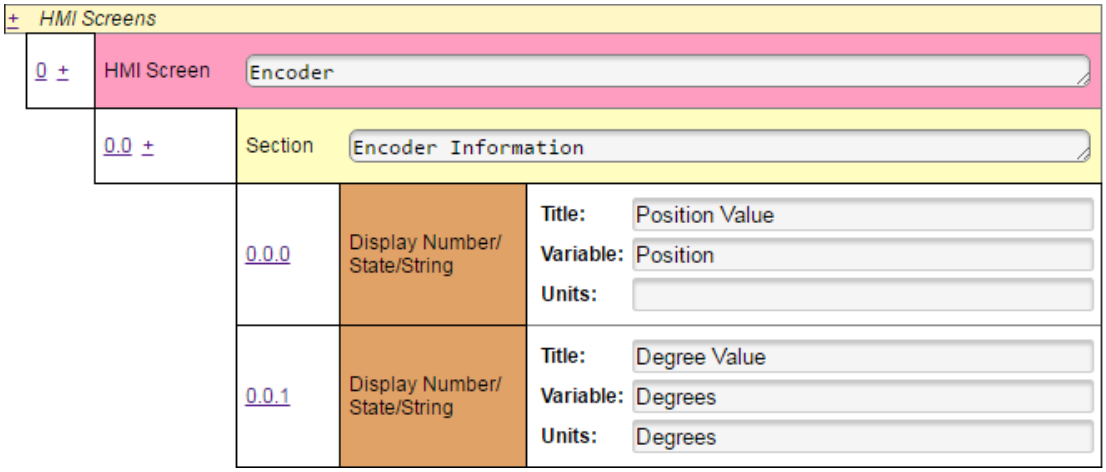
(ARGEE Setup)



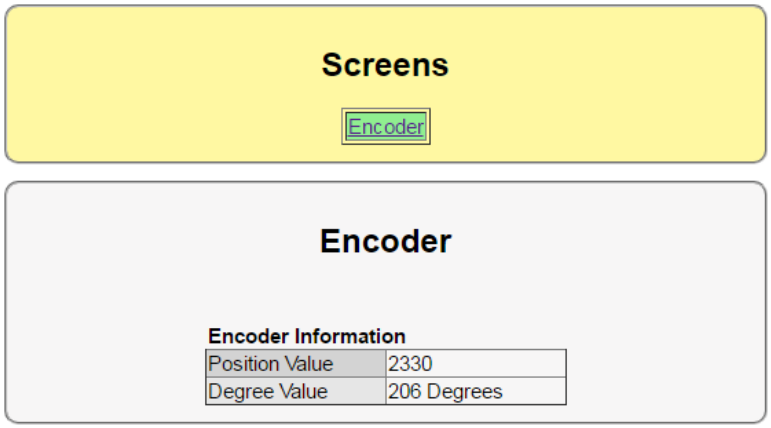
NOTE

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.





(ARGEE 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 to learn more about encoder settings.

12.7.2 Working with State Variables

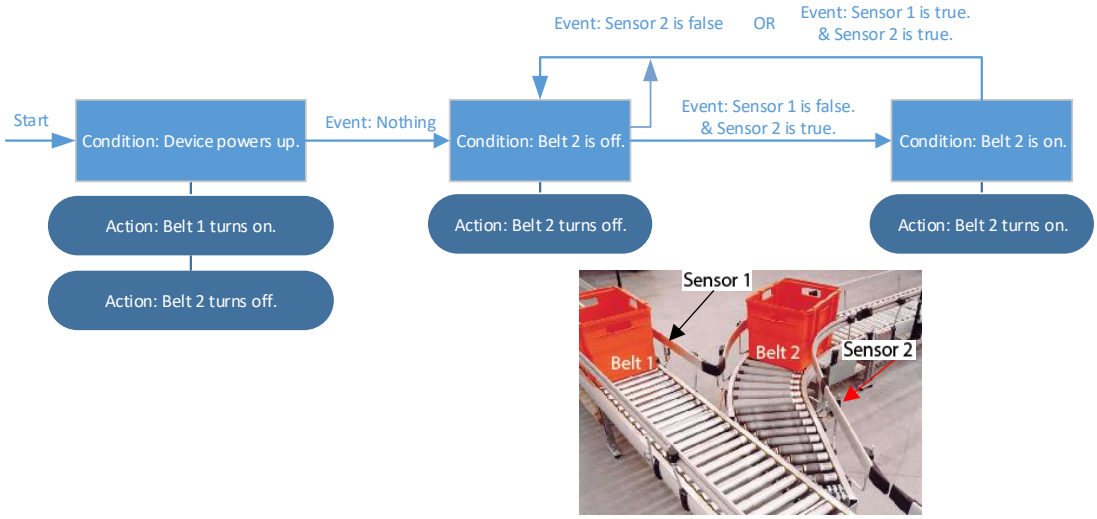
State Variables are helpful in keeping track of the signal as it steps through the code. Before the user creates State Variables, it is a good idea to create a State Machine.

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.

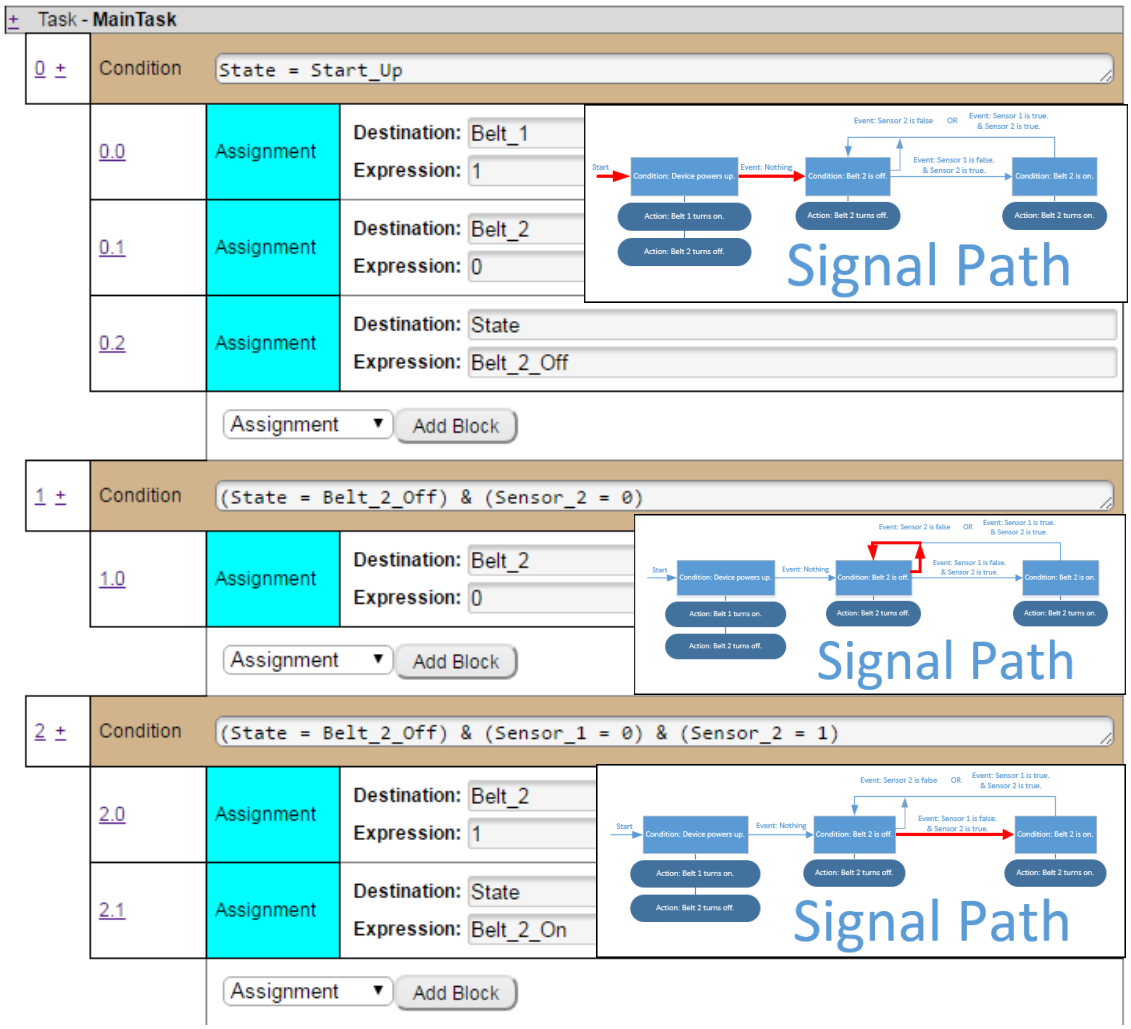
0 +	Program Variables	
	Name	Type
1.0	INIT : Start_Up	
1	State	State/Enum

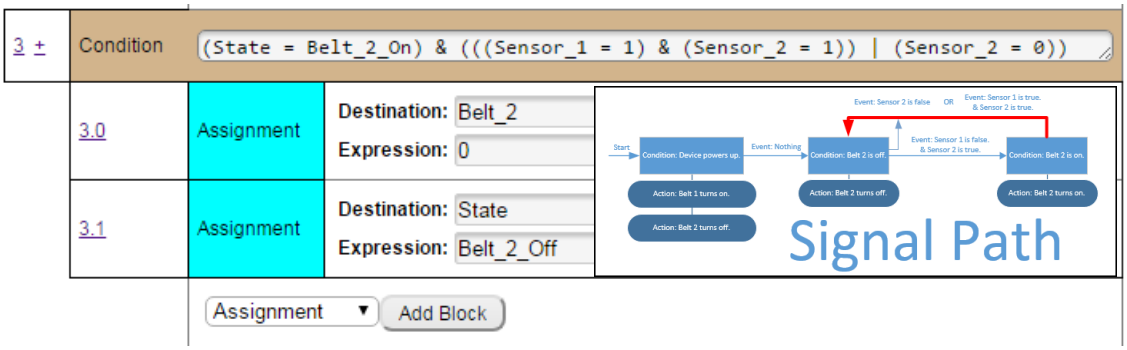
3 +	States
	Name
0	Start_Up
1	Belt_2_Off
2	Belt_2_On

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)

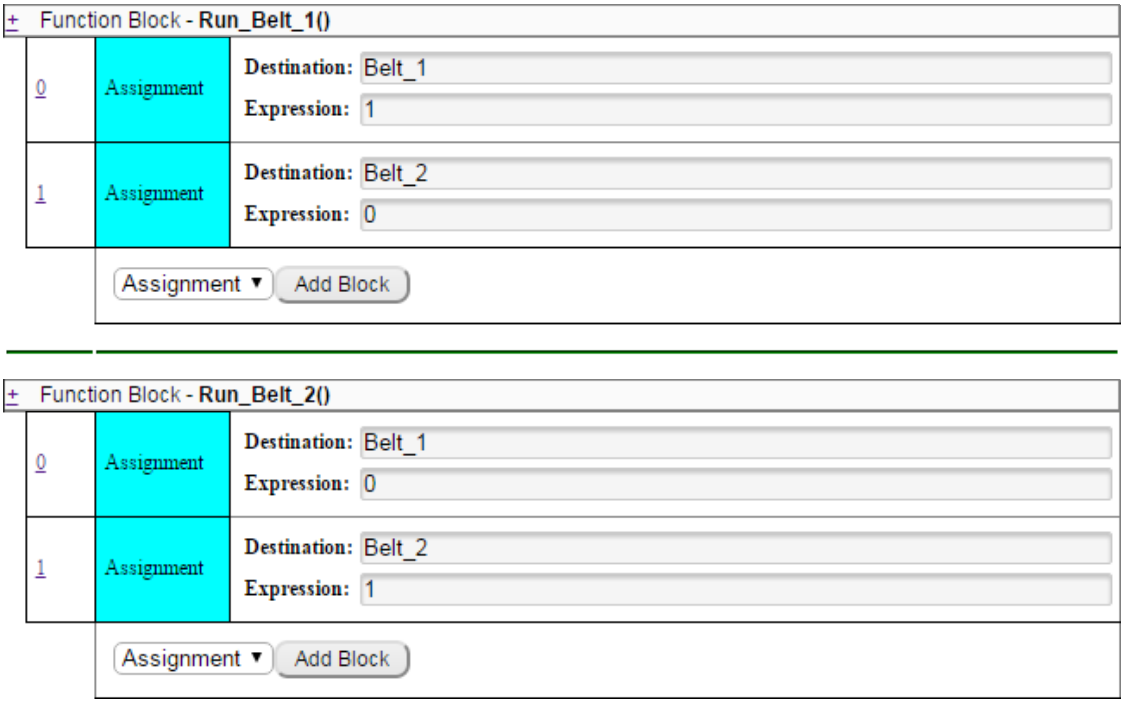
0 +	Program Variables	
	Name	Type
1	Belt_1_On	Run_Belt_1 ▼
2	Belt_2_On	Run_Belt_2 ▼

3 +	Function Block : Run_Belt_1	Regular ▼	*(hidden)*
4 +	Function Block : Run_Belt_2	Regular ▼	*(hidden)*

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

+ Task - **MainTask**

0 +	Condition	`Sensor_2 = 0`
0.0	Call	**Help:** Run_Belt_1() Belt_1_On()
	Call ▼	Add Block
1 +	Condition	`(Sensor_1 = 1) & (Sensor_2 = 1)`
1.0	Call	**Help:** Run_Belt_1() Belt_1_On()
	Call ▼	Add Block
2 +	Condition	`(Sensor_1 = 0) & (Sensor_2 = 1)`
2.0	Call	**Help:** Run_Belt_2() Belt_2_On()
	Call ▼	Add Block

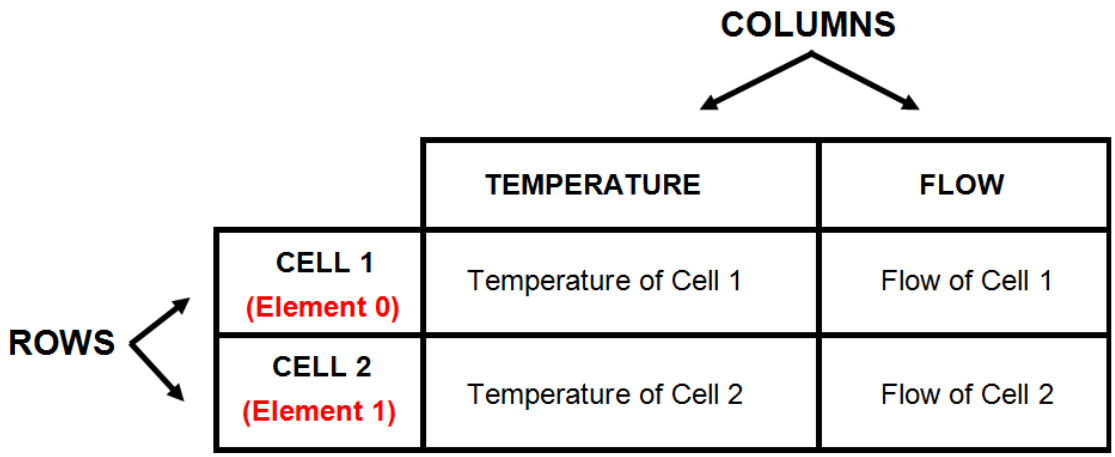


Explaining the example: If Sensor_1 is true or false and Sensor_2 is false, turn on Belt_1 and turn off Belt_2. If both sensors are true, turn on Belt_1 and turn off Belt_2. If Sensor_1 is false and Senor_2 is true, Turn off Belt_1 and turn on Belt_2.

12.7.3 Working with User-Defined Data Types

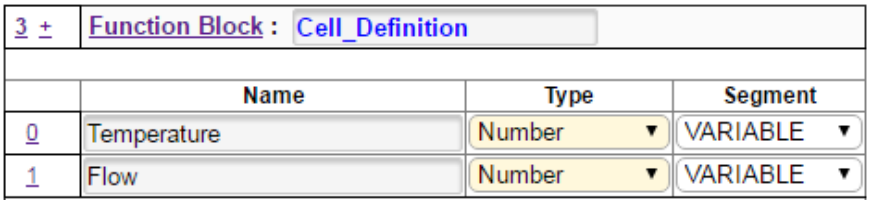
A User-Defined Data Type (UDT) is a function block which contains variables but no code. A user would create a UDT if they were dealing with multiple objects with multiple properties.

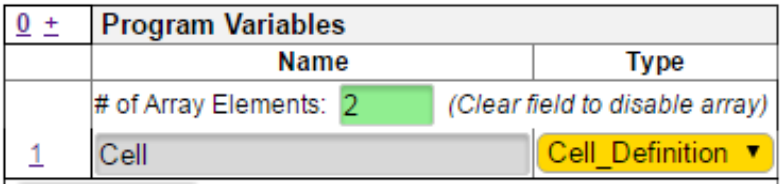
Example of *User-Defined Data Types:*
Suppose the user has 2 cells, and each cell has 2 properties: temperature and flow. This is best illustrated as a matrix:

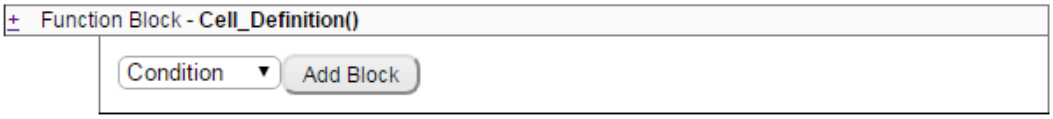


To express this in ARGEE, the user will create a Function Block with variables (the columns), and then create an array of this Function Block (the rows). No code goes into the Function Block; its only purpose is to contain variables.

(ARGEE Setup)







NOTE

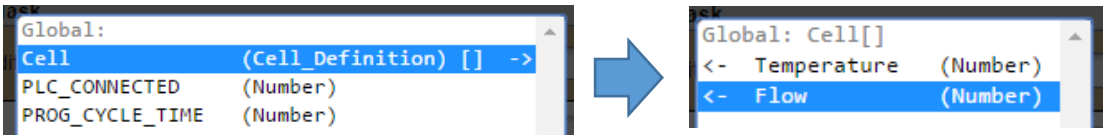
If the user wanted to add more rows to this matrix, he would increase the size of the array. If the user wanted to add more columns, he would create more Function Block variables.

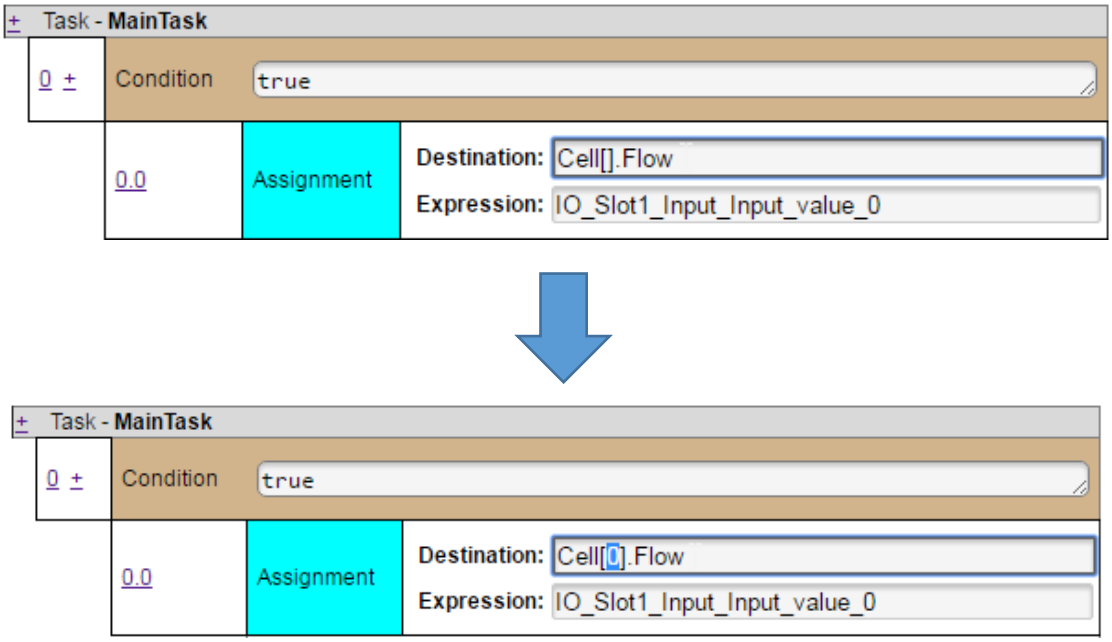
12.7.3.1 Referencing Internal Function Block Variables

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

(ARGEE 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

(ARGEE 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

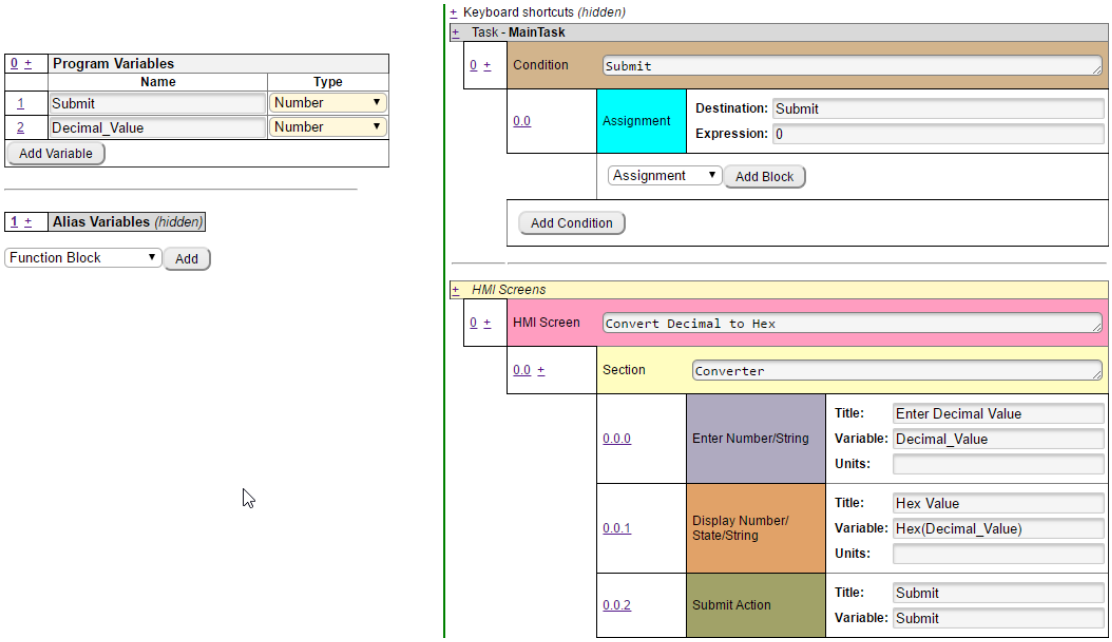
Explaining the example: The user wants to convert raw data from their IO-Link temperature sensor to degrees Celsius, and store it as a variable inside a user-defined data type (UDT). They pass a raw value and the name of their UDT into their temperature conversion Function Block, which converts the value 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

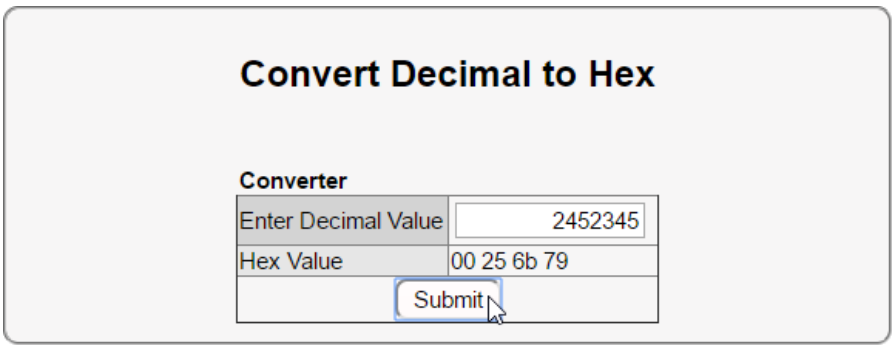
ARGEE can easily convert any value to hex. For example: if the user types “Hex(12),” then the value “c” will be returned.

Example of *Working with Hex Values:*

(ARGEE Setup)



(HMI View)



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?

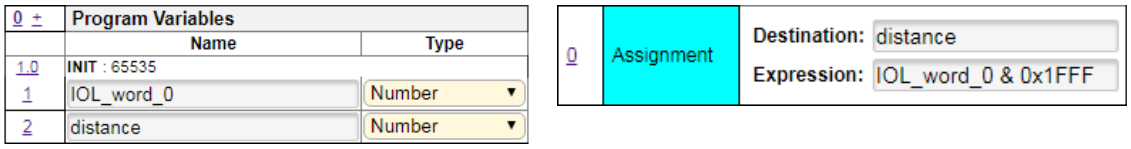
A *bitwise operation* is a Boolean operation that compares variables' bits against each other, instead of comparing the variables' values. ARGEE has bitwise OR, AND, NOR, and NAND, though AND is the only operation with a practical use, which is bit masking.

12.7.5.2 What is Bit Masking?

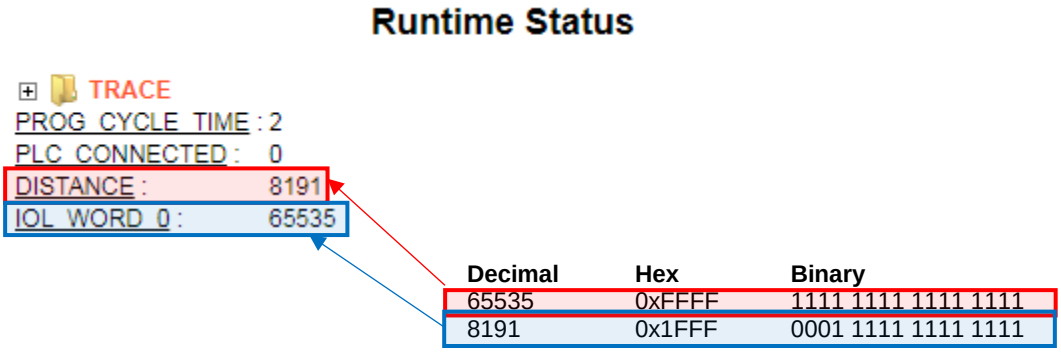
Suppose you have an IO-Link laser distance sensor that outputs one word of process data; the first 15 bits are dedicated to distance data, and the last 3 bits are status bits. To use the distance as a number in your ARGEE code, you want to ignore the status bits, and just work with the distance data, 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)



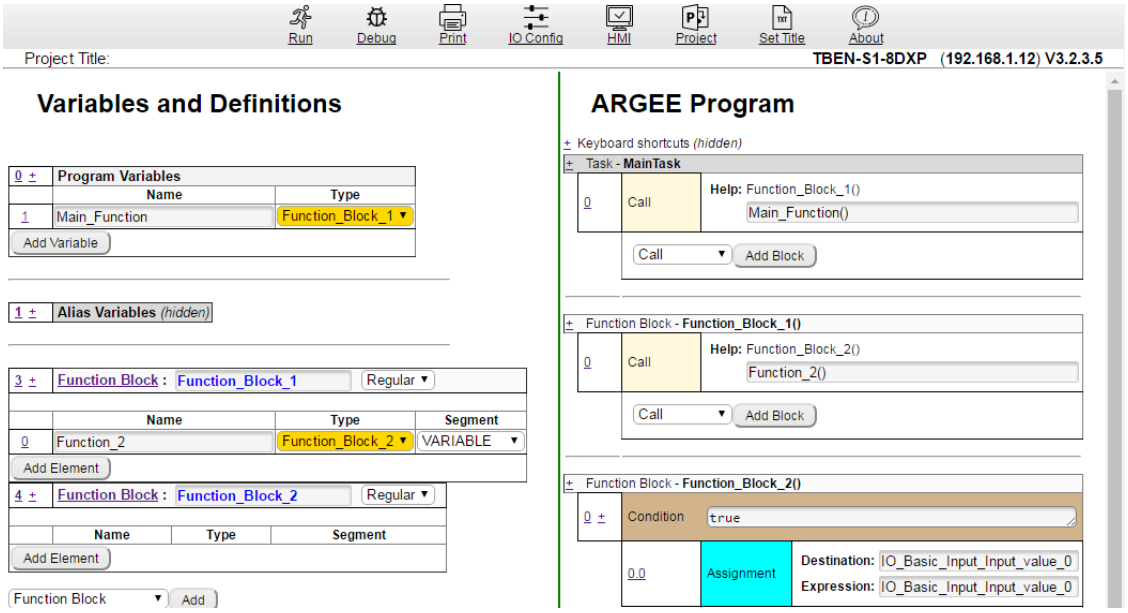
(ARGEE Debug Screen)



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.



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
	Name
0	High
1	Crit_High
2	OK
3	Low
4	Crit_Low

3 +	Function Block : Tank_Status	Regular ▾
	Name	Type	Segment
0	Tank_State	State/Enum ▾	VARIABLE ▾

+	Task - MainTask	
0	Call	Help: Tank_Status() Tank_Monitoring()

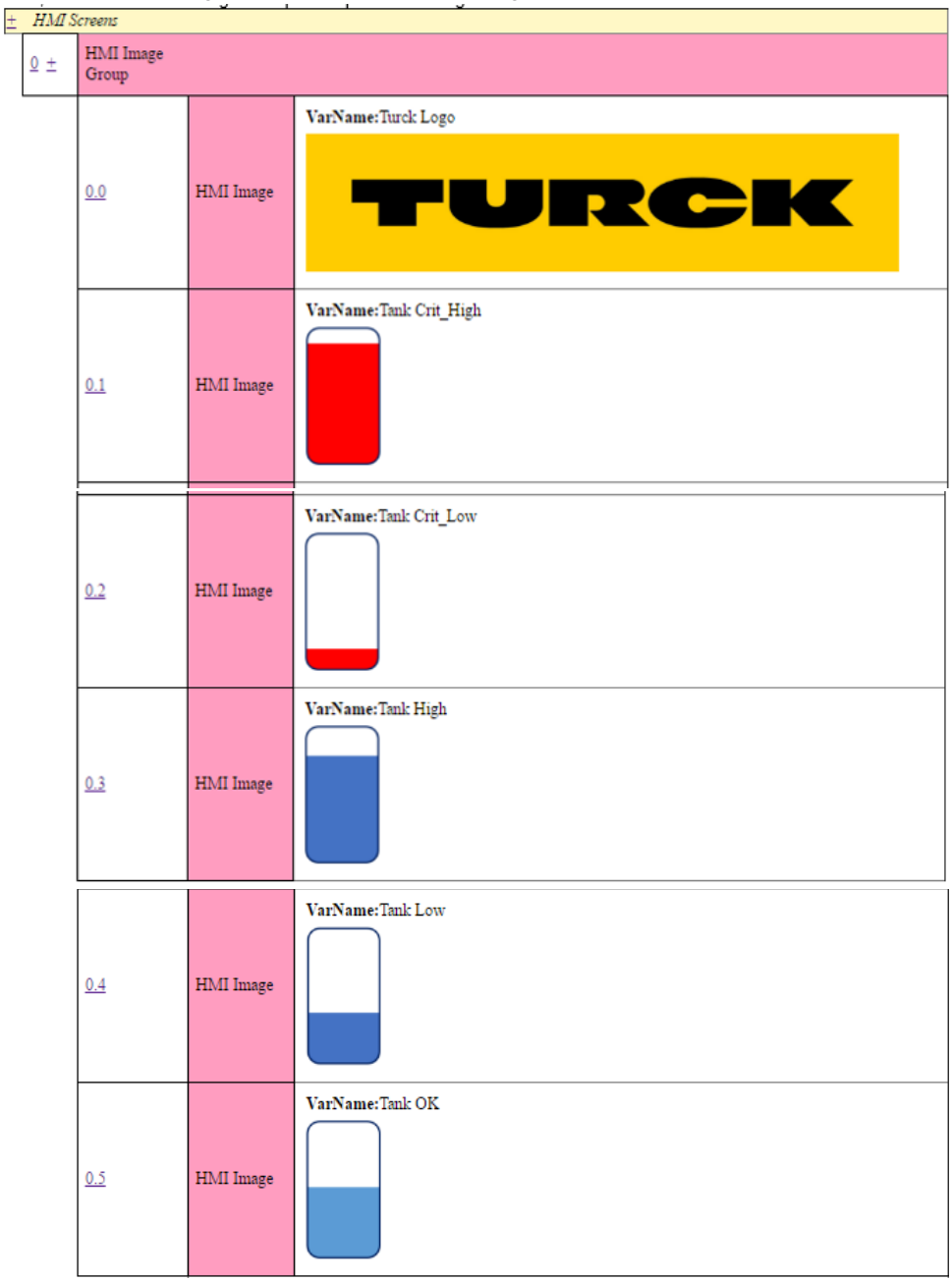
+	Function Block - Tank_Status()		
0 +	If	Tank_Sensor < 1900 & (1401 < Tank_Sensor)	
0.0	Assignment	Destination: Tank_State	Expression: Crit_Low
1 +	If	Tank_Sensor < 1400 & (1001 < Tank_Sensor)	
1.0	Assignment	Destination: Tank_State	Expression: Low
2 +	If	Tank_Sensor < 1000 & (801 < Tank_Sensor)	
2.0	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

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: *ARGEEE HMI* for more details.

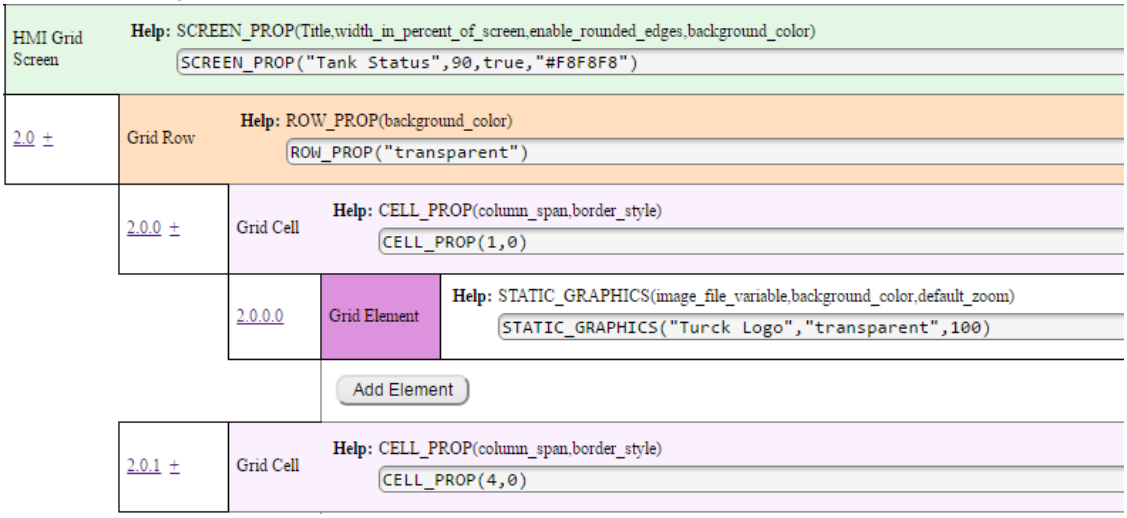
First, we’ll add an HMI Image Group and upload our images.



NOTE

The user can upload any image. Keep file size below 20kb.

Now let's add a grid screen.

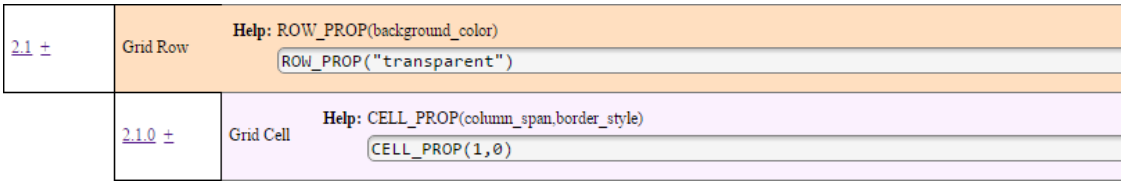


Our first row is just to display the logo in the top left of the screen. We added two *Grid Cells* one that spans 1/5 of the screen and the other 4/5 screen. We then added a *Grid Element* to the first *Grid Cell* and used the STATIC_GRAPHICS function to place our logo, using its variable name.

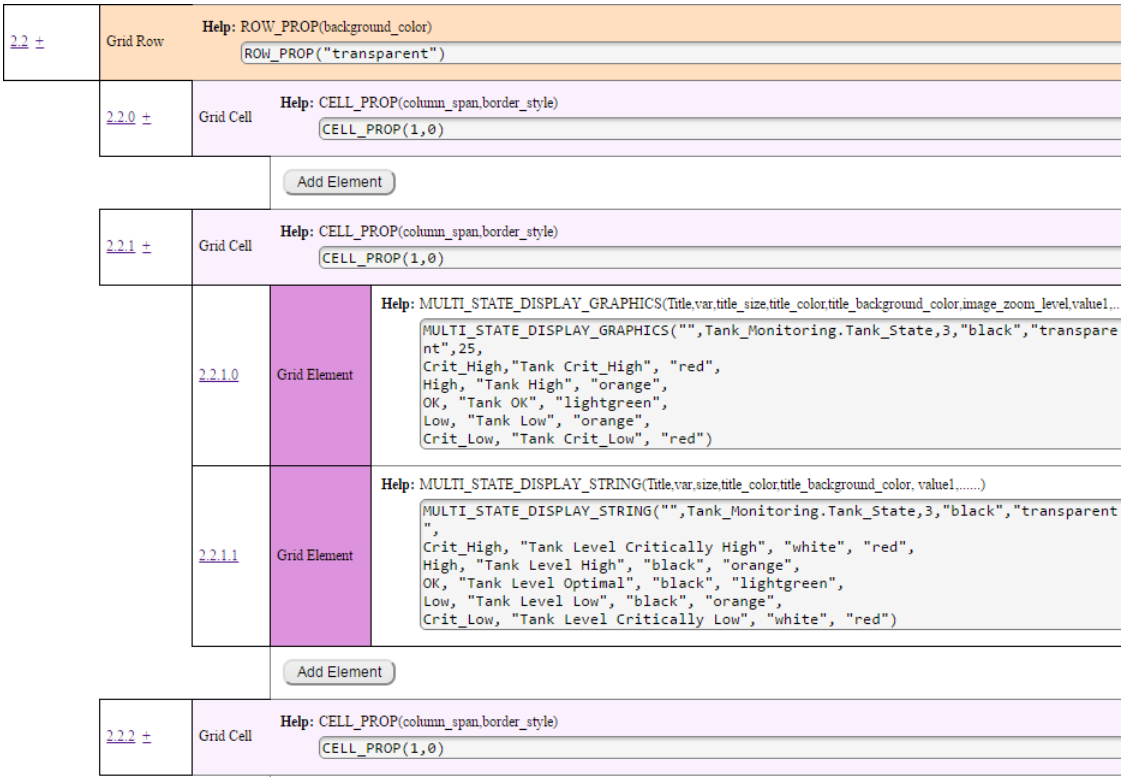
NOTE

HMI functions are available by hitting Ctrl-f, and image file variables are available by hitting Ctrl-i.

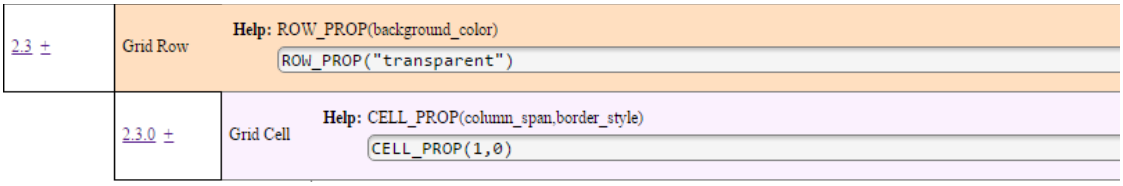
Our second row is empty; it will be used as a spacer.



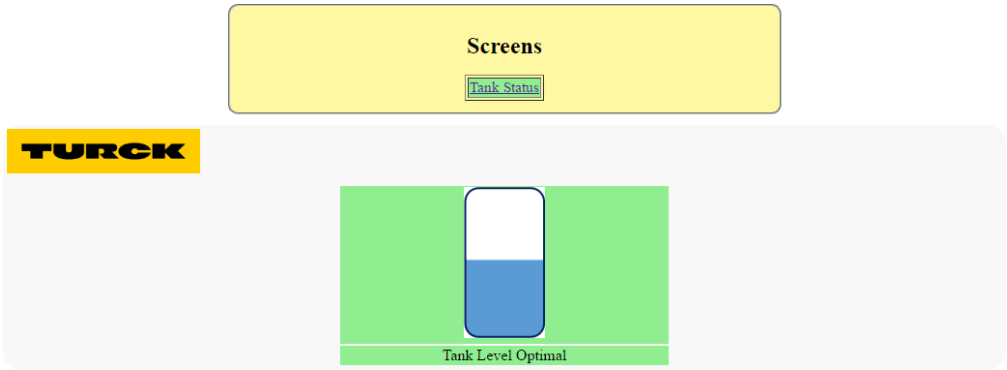
Our third row will have three cells, and the middle cell will have the *Grid Elements* Multi State Display Graphics and Multi State Display Strings. Both of these functions will use different background colors for each state.

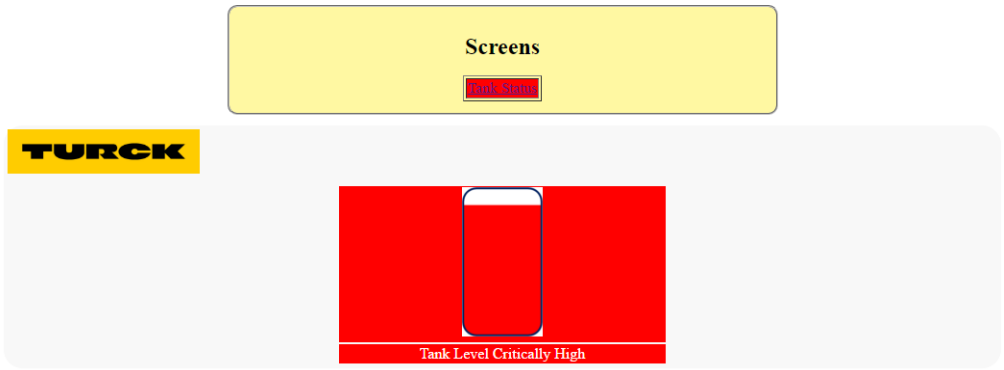


The last row will be empty and used as a spacer.



(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 bellow.

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.

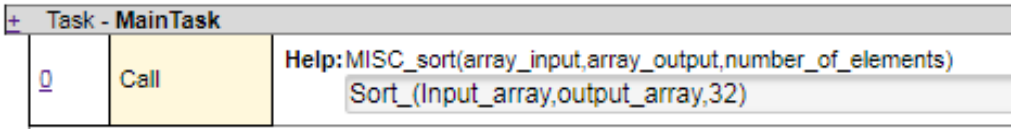
The sort function block has 2 number arrays (one for the input and one for the output), and a number that represents the length of the input array as arguments.

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

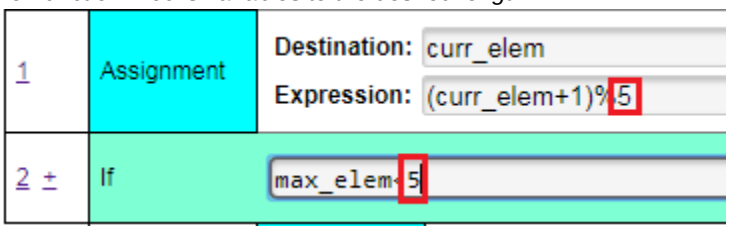
How to Call: The call needs a number array for the input, a number array to hold the output, and number to represent the length of the input array.



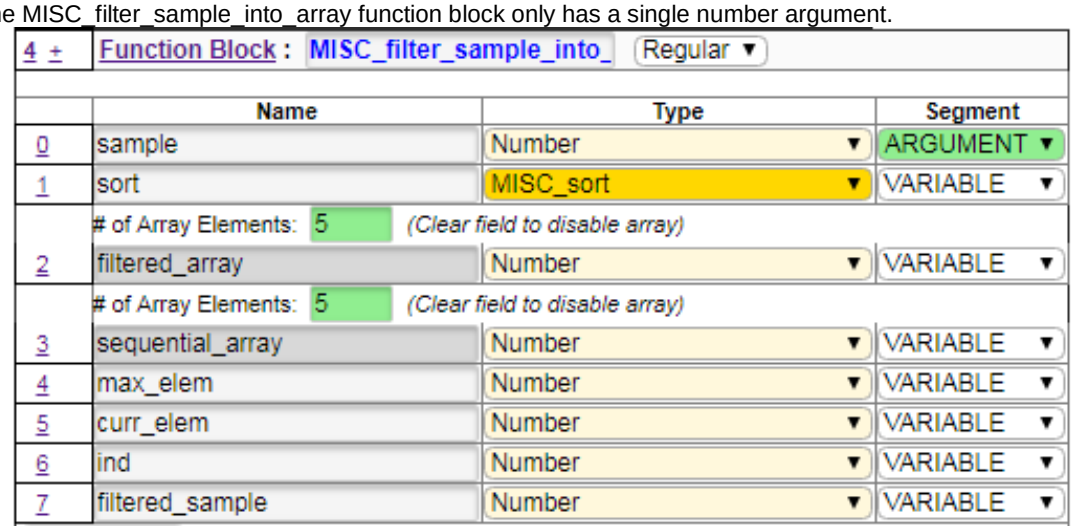
13.1.4 MISC_filter_sample_into_array

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 5th will overwrite the first data values stored.

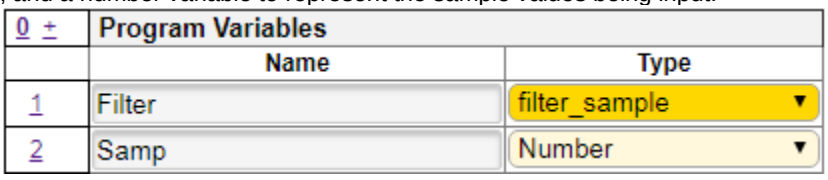
*****_To change the array length the user will need to alter the code in the function block change the 5s highlighted bellow to the desired length, and change the # of Array Elements of filtered_arr and seq_arr in the Function Blocks variables to the desired length._



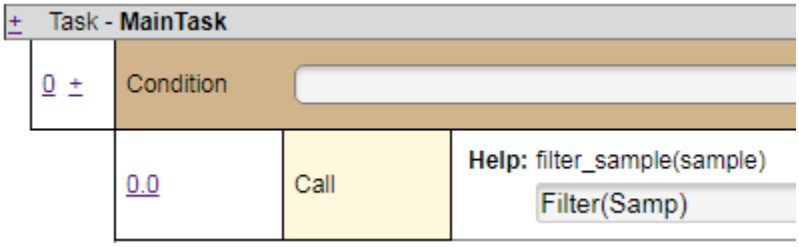
The MISC_filter_sample_into_array function block only has a single number argument.



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.



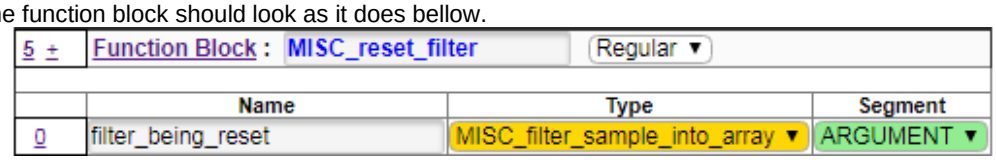
How to Call: For MISC_filter_sample_into_array to be called successfully it cannot be continuously called, so it needs to be in a condition block. If it is called continuously the filtered array values will be filled with repeats of the current sample value, and not populated with 5 unique sample values.



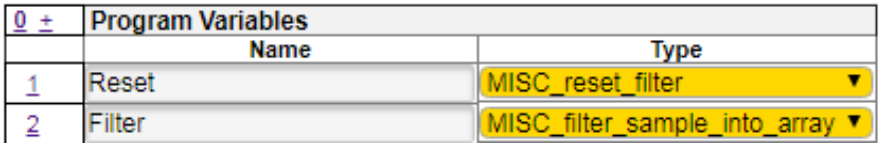
13.1.5 MISC_reset_filter

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.

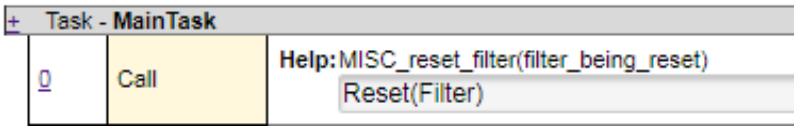
The function block should look as it does bellow.



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.



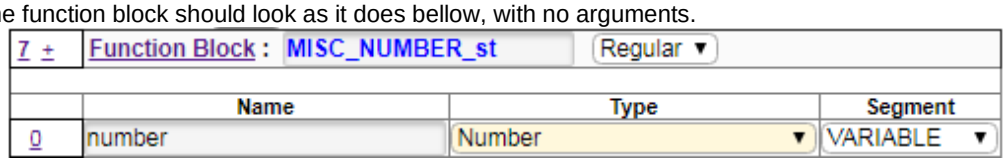
How to Call: When calling MISC_reset_filter the argument needs to be a MISC_filter_sample_into_array variable.



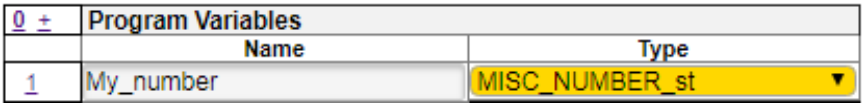
13.1.6 MISC_NUMBER_st

Function: The function of MISC_NUMBER_st is to pass a number to a function block.

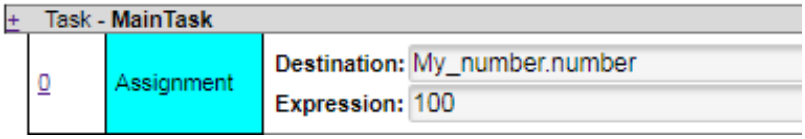
The function block should look as it does bellow, with no arguments.



Program Variables: The only variable needed is a MISC_NUMBER_st variable.



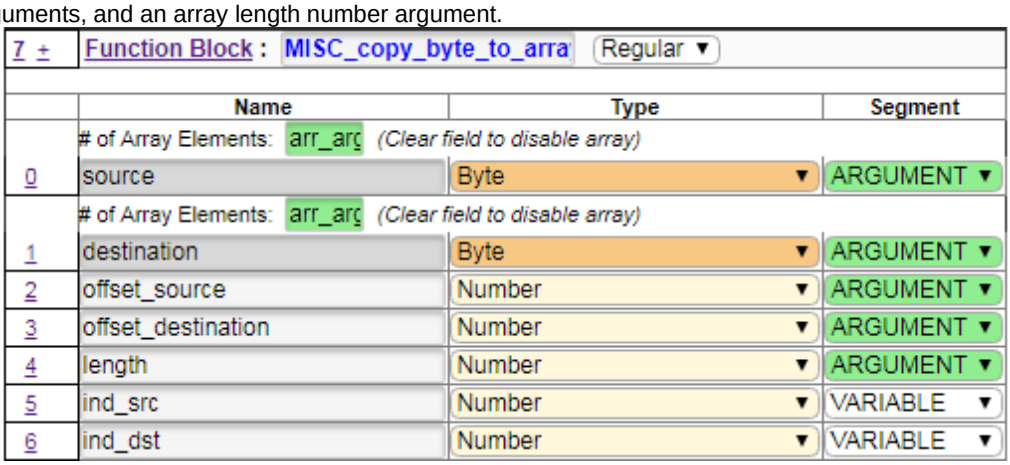
How to Call: This function block is not really called instead a number is assigned to the variable in the function block as shown below.



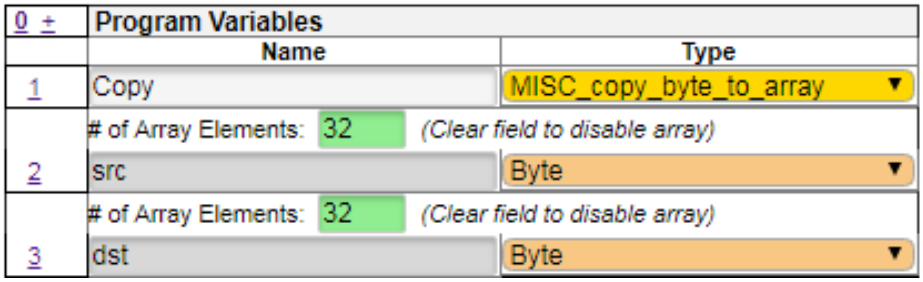
13.1.7 MISC_copy_byte_to_array

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.

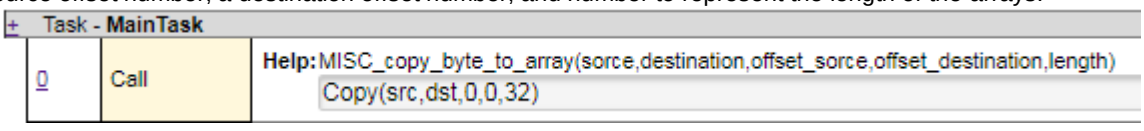
The Function Block has source and destination Byte array arguments, source and destination offset number arguments, and an array length number argument.



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.



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.



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

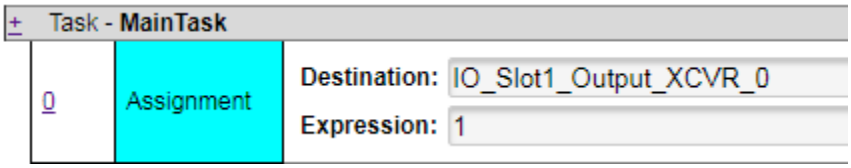
How to Call: To call Float_to_String the following arguments must be satisfied; a float variable that holds the float being input, the number of decimal places the float variable has, and the string that the is being output with the value of the float variable.

+	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

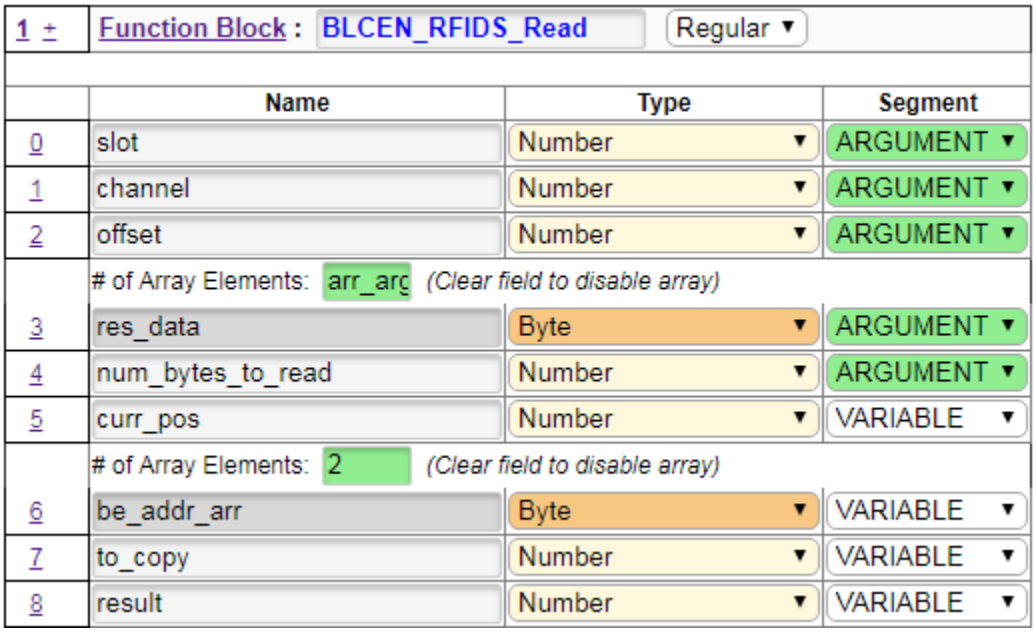
For BLCEN-RFIDS devices to read or write the transceiver needs to be turned on. This is done as shown below.



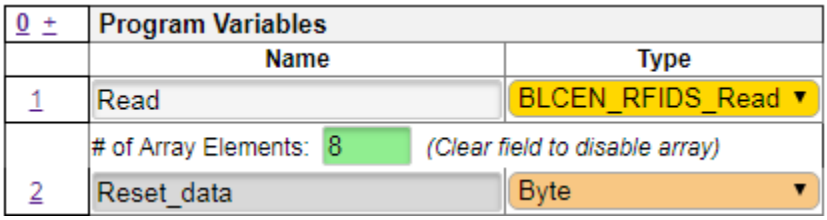
13.2.2 BLCEN_RFIDS_Read

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.

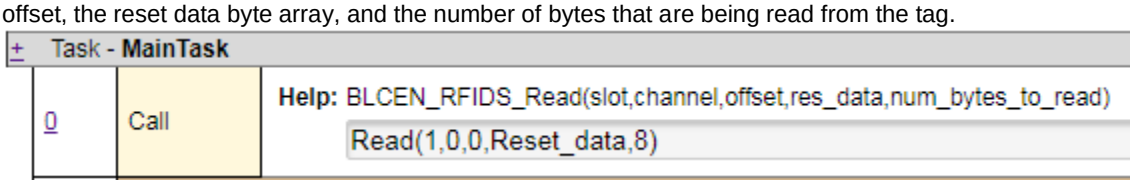
The Function Block should look as it does bellow.



Program Variables: To call BLCEN_RFIDS_Read a BLCEN_RFIDS_Read variable, and a byte array are needed.



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.



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	**Help:** 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 : TBEN_S2_RFID_READ	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	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: 32 *(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_IOLAsyncRead 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 : TBEN_IOL_AsyncRead	Regular ▼	
	Name	**Type**	**Segment**
0	port_num	Number ▼	ARGUMENT ▼
1	index	Number ▼	ARGUMENT ▼
2	sub_index	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg *(Clear field to disable array)*		
3	reset_data	Byte ▼	ARGUMENT ▼
4	reset_data_length	Number ▼	VARIABLE ▼
	# of Array Elements: 32 *(Clear field to disable array)*		
5	ds_tx_array	Byte ▼	VARIABLE ▼
	# of Array Elements: 32 *(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: 8 *(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 : TBEN_IOL_AsyncWrite	Regular ▼	
	Name	**Type**	**Segment**
0	port_num	Number ▼	ARGUMENT ▼
1	index	Number ▼	ARGUMENT ▼
2	sub_index	Number ▼	ARGUMENT ▼
	# of Array Elements: arr_arg *(Clear field to disable array)*		
3	write_data	Byte ▼	ARGUMENT ▼
4	write_data_length	Number ▼	ARGUMENT ▼
5	index1	Number ▼	VARIABLE ▼
	# of Array Elements: 32 *(Clear field to disable array)*		
6	ds_tx_array	Byte ▼	VARIABLE ▼
	# of Array Elements: 32 *(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: 8 *(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	**Help:** 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.

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