MAPware-7000 Getting Started Guide

Your Industrial Control Solutions Source

www.maplesystems.com

For use with the following.

- HMC7000 Series
- HMC3000 Series
- HMC2000 Series
- HMC4000 Series

Maple Systems, Inc. | 808 134th St. SW, Suite 120, Everett, WA 98204 | 425.745.3229

COPYRIGHT NOTICE

WARRANTY

Warranty Statements are included with each unit at the time of purchase and are available at www.maplesystems.com.

TECHNICAL SUPPORT

This manual is designed to provide the necessary information for trouble-free installation and operation of Maple Systems products. However, if you need assistance, please contact Maple Systems.

- Phone. 425-745-3229
- Email. support@maplesystems.com
- Web. www.maplesystems.com

Table of Contents

Introduction

Welcome to Maple Systems' MAPware-7000 programming software, an easy-to-use configuration software for our HMC and MLC products. You can use MAPware-7000 to program a powerful range of automation and display hardware. This guide will introduce you to MAPware-7000 using two short projects to familiarize you with the basic elements necessary to start programming.

If you are still deciding which manufacturer or product line will best suit your needs, you can follow this guide to learn just how easy it is to produce high quality control programs with MAPware-7000. If you are already a Maple Systems customer and are ready to begin working with your new software, use this guide as a starting point. After you have run through the Quick Start Projects, you can use this guide as a reference for building your own projects.

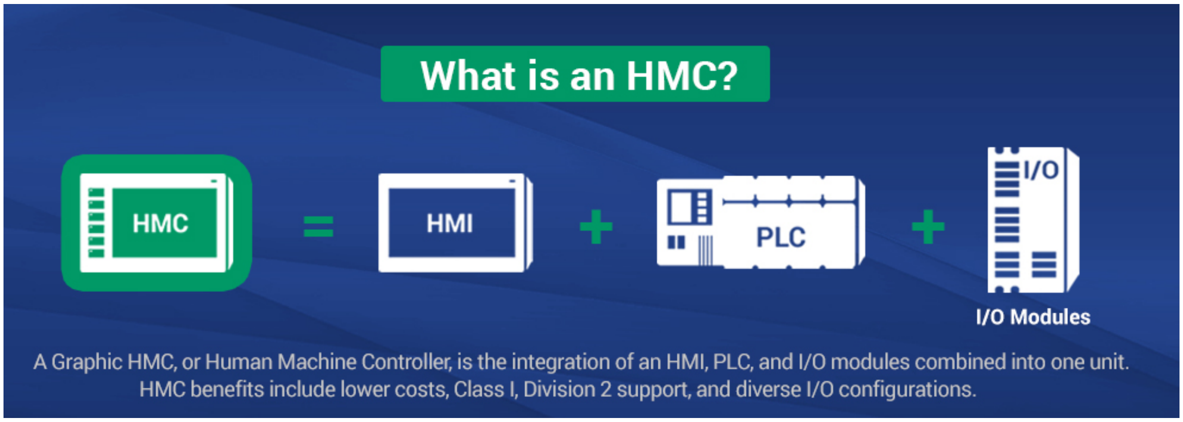
Reading This Manual

This manual is designed as a training manual and reference guide. Some previous exposure to PLC programming and ladder logic is assumed. In this effort, the manual tries to minimize the amount of time required to become familiar with the MAPware-7000 programming software. Additional document resources include the HMC I/O Module Guides, Ladder Logic Guide, IEC Programming Guide, and general MAPware-7000 Programming Manual. Additionally, help files are accessible from the **Help** dropdown menu within MAPware-7000. These are a good source of information when you quickly want to research a particular feature in the software.

Finally, our technical support staff is available by phone or by email to assist you if you run into any problems not covered in this manual. Visit our website at www.maplesystems.com for contact information.

HMC Basics

An HMC is a combination operator-based HMI (human machine interface) with built-in PLC (programmable logic controller) operation and expandable I/O.



A Graphic HMC, or Human Machine Controller, is the integration of an HMI, PLC, and I/O modules combined into one unit. HMC benefits include lower costs, Class I, Division 2 support, and diverse I/O configurations.

HMCs offer flexibility. Rather than using fixed I/O, the HMC employs expansion slots to customize your I/O configuration. Maple Systems offers a wide assortment of expansion modules including combination digital input/output modules, digital input only, digital output only, analog modules, and high-speed counter modules. For more information on the different I/O configurations available, see the appropriate HMC family I/O Module Guide for your model.

Backing Up Projects

The MAPware-7000 configuration software does not perform automatic saves of the open project. We recommend that you frequently save your project as you are working on it to ensure that no work is lost in the event of a power failure or computer error. When you have completed a project, archive it to another folder, external network drive, or storage media for safekeeping. To archive your project, backup both the .mpl file and the corresponding file folder located in the same directory.

Although Maple Systems does provide repair support on all of our products, we cannot guarantee that we will be able to restore a project from a damaged unit.

Logic Editor Environment

MAPware-7000 includes a full-featured set of logic editors. There are two options for configuring PLC logic in a MAPware-7000 project: Native Ladder and IEC 61131-3. This selection is made in the **Programming Language** drop down list when a project is first created.

Native Ladder Editor

The Native Ladder Logic editor is a Ladder Diagram editor. Ladder logic is a programming language that represents a program with a graphical diagram based on the circuit diagrams of relay logic hardware. Ladder logic is commonly used to develop software for PLCs used in industrial control applications. The name is based on the observation that programs in this language resemble ladders, with two vertical rails and a series of horizontal rungs between them. Each rung is executed in sequence from left to right.

The programmer can define complex logic operations by building discrete blocks of logic executed either continuously or by function call. A large library of built-in ladder logic instructions is available to perform common automation tasks such as math operations, timers and counters, data manipulation, feedback loops, input scaling, and much more.

🗐 For complete documentation on using the Native Ladder editor, refer to the *MAPware-7000 Ladder Logic Guide*.

IEC 61131-3 Logic Editors

IEC 61131-3 is a section of the International Electro-Technical Committee (IEC) standard that provides a definition for implementing PLC programming software. The goal of the standard is to give automation professionals a familiar environment and set of tools to create PLC programs across vendor platforms. MAPware-7000 has editors implemented for all five programming languages defined by the standard.

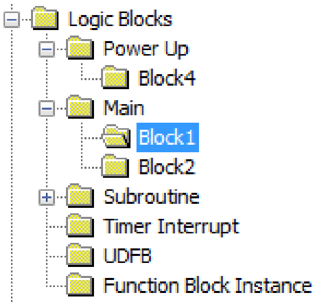
1. **Ladder Diagram (LD)** – Graphical language that simulates an electrical circuit; program instructions are attached as discrete elements in the circuit and are executed when “energized”. Visually and functionally similar to the Native Ladder Logic editor.
2. **Function Block Diagram (FBD)** – Graphical language based on logic diagrams. Functions are represented by blocks; complex operations can be built by interconnecting function blocks.
3. **Structured Text (ST)** – Text-based programming language. Programs are built using keywords, operators and function calls.
4. **Instruction List (IL)** – Text-based procedural programming language.
5. **Sequential Function Chart (SFC) –** Graphical programming language in which program execution is modeled as a flow chart. Programs are developed by adding blocks to the flow chart.

The logic created using these languages is composed of discrete instructions or logic blocks. In addition, the programmer can create their own **User Defined Function Blocks (UDFBs)** to make their logic modular and reusable.

For complete documentation on using the IEC 61131-3 logic editors, refer to the *MAPware-7000 IEC 611313 Programming Guide*.

Logic Blocks and Execution Style

Programs created using the above editors are organized into discrete units called **Blocks**. In addition to the logic they contain, blocks are differentiated by Execution Style. Execution Style determines when and how the block is executed. The available Execution Styles are.



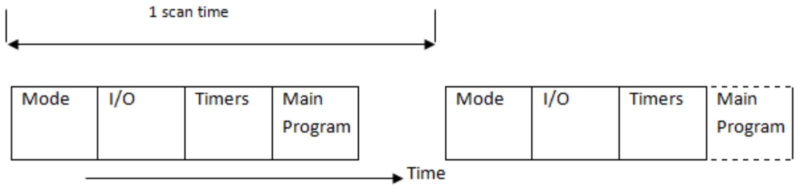
- Power Up
- Main
- Subroutine
- Timer Interrupt

The current Execution Style of a logic block is indicated by the block's location within the **Logic Blocks** folder of the project tree.

Main Program Blocks

Main program blocks are the core of the user program. They are executed once during each scan.

Multiple logic blocks can be created (up to 256) and used as Main Program blocks. During execution, the HMC starts with the first block listed. When completed, it will execute each block in sequence. The figure below shows a typical scan sequence.



Where:

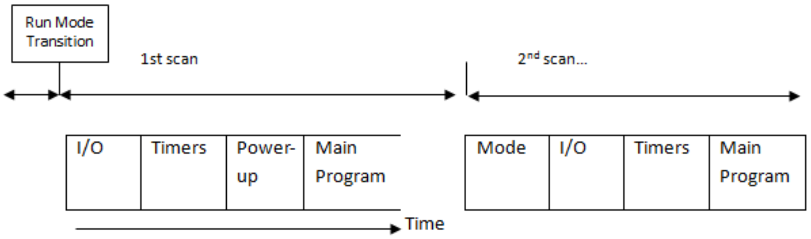
- Mode – Determines mode of operation (Run, Halt, etc.)
- I/O – Update and process all inputs and outputs
- Timer – Update all running timers
- Main Program – All logic blocks created under Main

The length of a scan is not deterministic. It depends on what blocks are executed, etc. If precise timing is required, use a timer interrupt routine.

Power Up Blocks

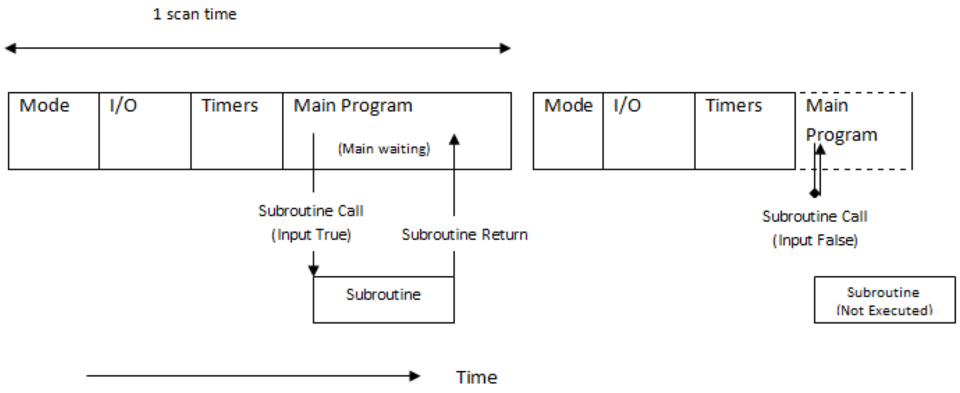
If Power Up blocks are present, they are executed <u>once</u> at the beginning of the first scan (before main block execution). Therefore, Power Up blocks can be used to set initial values into registers.

The figure below shows the first scan operation.



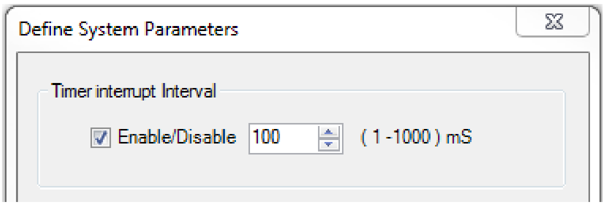
Subroutine Blocks

Subroutines are not executed unless specifically called by another logic block. Subroutines are useful when you have a set of commands that should be executed only under certain conditions. A maximum of 256 subroutines can be created (dependent upon total memory available).



Timer Interrupt Blocks

Timer interrupt logic blocks are given the highest priority when the MAPware-7000 program is executed. The timer interrupt is enabled by going to the **Define > System Parameters** dialog box and checking **Timer Interrupt Interval**.


Define System Parameters

Timer interrupt Interval

☑ Enable/Disable 100 (1 -1000) mS

When enabled, the timer interrupt routine is executed based upon the interval selected (range is 1-1000 milliseconds).

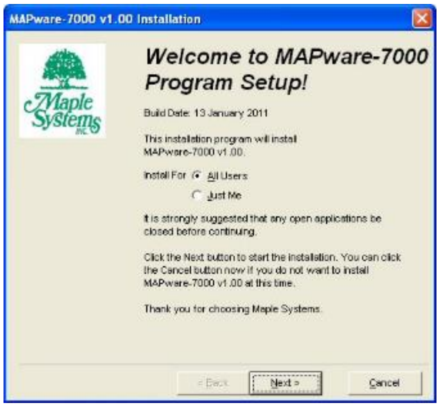
All other operations are suspended when the timer interrupt activates. Use this feature sparingly if you have a continuous operation that is time-critical. Because timer interrupt routines halt all other activities, to minimize its impact on the performance of the controller, design the interrupt routine to be as short as possible and adjust the timer interrupt interval to the maximum setting that still meets the requirements of your application.

Installing MAPware-7000

Use the following procedure to install the MAPware-7000 software from the CD-ROM.

1. Turn on power to your development computer.
2. Make sure no other application programs are running.
3. Insert the MAPware-7000 software CD into the computer's CD-ROM drive.

   Note. if 'Auto Play' mode is not set for your CD-ROM drive, double-click *setup.exe* on the root directory of the CD drive using Windows Explorer.
4. The **Welcome to MAPware-7000** software screen appears.
5. Click **Next** to continue.



Welcome to MAPware-7000 Program Setup!

Build Date: 13 January 2011

This installation program will install MAPware-7000 v1.00.

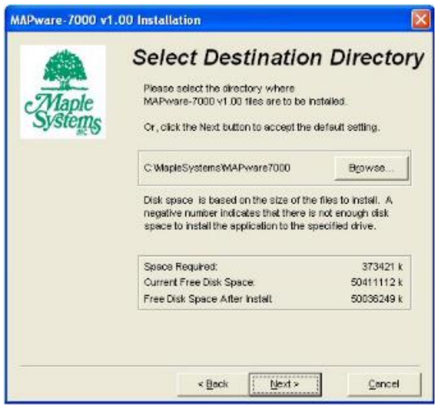
Install For ● All Users
○ Just Me

It is strongly suggested that any open applications be closed before continuing.

Click the Next button to start the installation. You can click the Cancel button now if you do not want to install MAPware-7000 v1.00 at this time.

Thank you for choosing Maple Systems.

6. Select the preferred directory location on your computer to install the MAPware-7000 software (default is *C:\Maple Systems\MAPware-7000*\).
7. Then click **Next**.



Select Destination Directory

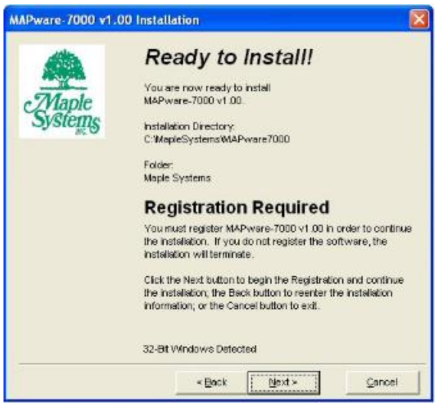
Please select the directory where MAPware-7000 v1.00 files are to be installed.

Or, click the Next button to accept the default setting.

C:\MapleSystems\MAPware7000

Disk space is based on the size of the files to install. A negative number indicates that there is not enough disk space to install the application to the specified drive.

Space Required:	373421 k
Current Free Disk Space:	50411112 k
Free Disk Space After Install:	50036249 k



Ready to Install!

You are now ready to install MAPware-7000 v1.00.

Installation Directory:
C:\MapleSystems\MAPware7000

Folder:
Maple Systems

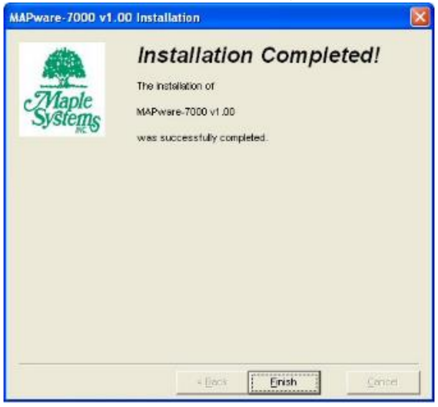
Registration Required

You must register MAPware-7000 v1.00 in order to continue the installation. If you do not register the software, the installation will terminate.

Click the Next button to begin the Registration and continue the installation; the Back button to reenter the installation information; or the Cancel button to exit.

32-Bit Windows Detected

8. Confirm that you are ready to begin installation. Click **Next**.
9. Once installation is complete, click **Close**.



Installation Completed!

The installation of

MAPware-7000 v1.00

was successfully completed.

Creating a Sample Project using Native Ladder

Introduction

This first section will provide step by step instructions for creating a simple test project using the Native Ladder Logic editor. The project will display and manipulate digital inputs and outputs, as well as take a 16 bit analog input and scale it to a human readable engineering value. The value will also be checked against limits to determine if it’s within an acceptable range.

For this example, we will use the following items.

- HMC3102A-M
- HMC3-M0808Y0401T expansion module
- MAPware-7000 software running on a PC
- Micro USB configuration cable (PN 7431-0119)
- Simple test circuit attached to the I/O module to test the inputs and outputs configured in the project.

The instructions presented here should be general enough that they can be adapted for equipment on hand.

The project will demonstrate.

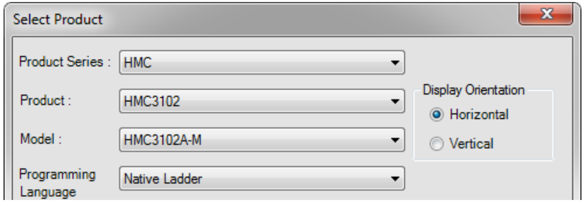
- Configuration of input and output channels
- Writing a simple Native Ladder logic block that controls I/O points
- Scaling an analog input
- Configuring a screen with objects that display HMC data
- Navigating between screens

Create a New Project

Starting a new project in MAPware-7000 is straightforward. After MAPware-7000 has started, it will display the opening screen. This screen displays a list of recent projects and offers the ability to either open an existing project or create a new one.

To create a new project:

1. Click the **New** option under the **Project** category, or select **Project > New** from the menu bar.
2. When the **Select Product** dialog box appears, make the following selections.



Select Product

Product Series : HMC

Product : HMC3102

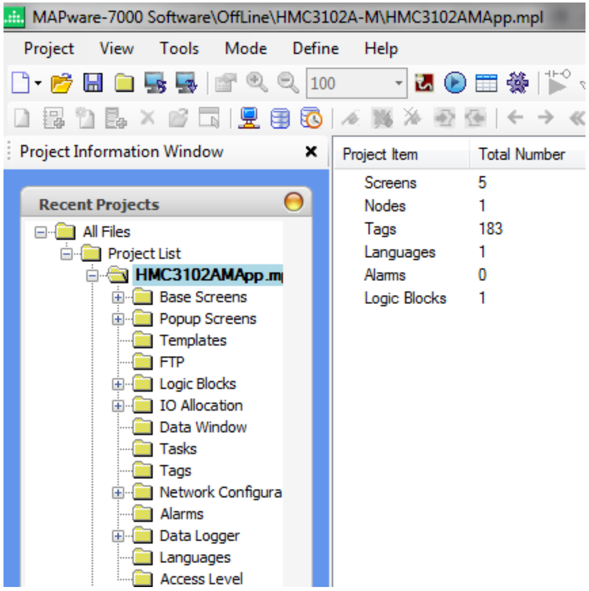
Model : HMC3102A-M

Programming Language: Native Ladder

Display Orientation
● Horizontal
○ Vertical

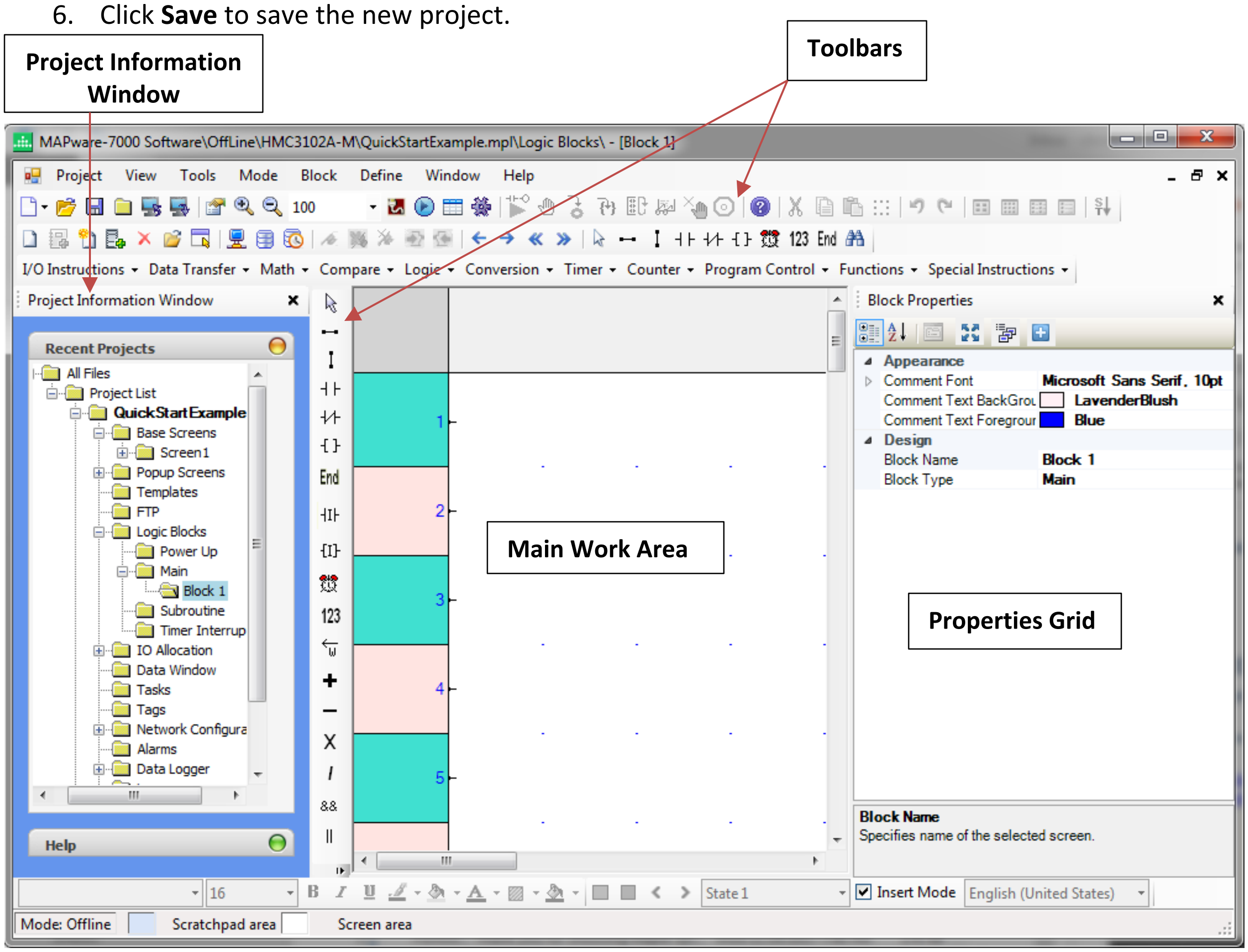
   a. Product Series. HMC
   b. Product. HMC3102
   c. Model. HMC3102A-M
   d. Programming Language. Native Ladder
   e. Display Orientation. Horizontal

3. Click **OK**. At this point, MAPware-7000 will give your project a default name based on the model number specified and open the Project Summary screen.



Project Item	Total Number
Screens	5
Nodes	1
Tags	183
Languages	1
Alarms	0
Logic Blocks	1

4. To save the project with a new name, from the menu bar, select **Project > Save**.
5. Select the directory where you want to store to project and enter a name for the project. We will use *QuickStartExample* for our project name.
6. Click **Save** to save the new project.



Comment Font	Microsoft Sans Serif, 10pt
Comment Text BackGrou	LavenderBlush
Comment Text Foregrour	Blue
Block Name	Block 1
Block Type	Main

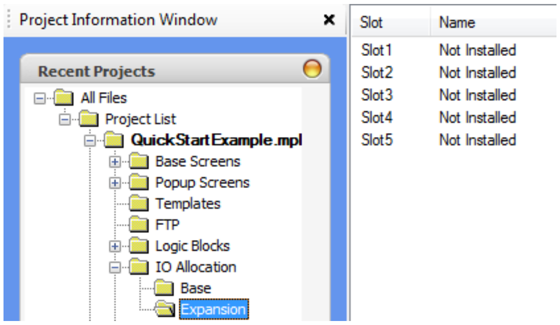
Block Name
Specifies name of the selected screen.

State 1 ☑ Insert Mode English (United States)

Configure the I/O module

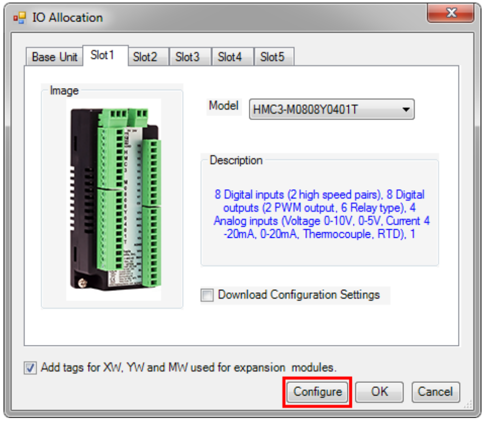
The first thing to set up in this new project is an I/O expansion module. This is done in the **IO Allocation** window.

1. Expand the **IO Allocation** node in the **Project Information Window,** then click the **Expansion** folder.



Slot	Name
Slot1	Not Installed
Slot2	Not Installed
Slot3	Not Installed
Slot4	Not Installed
Slot5	Not Installed

2. The HMC3102A-M has five expansion slots, but we will only be using one of them. This is listed in the Main Workspace window, as shown above. Double click **Slot1** in the list to open the **IO Allocation** window for this slot.
3. In the **IO Allocation** window, select **HMC3-M0808Y0401T** from the **Model** dropdown.
4. Click the **Configure** button at the bottom of this window to configure the channels we will use.



Model HMC3-M0808Y0401T

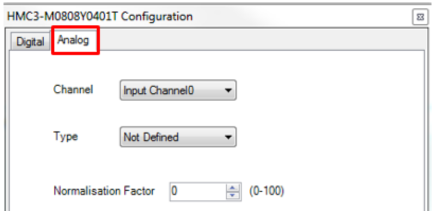
Description

8 Digital inputs (2 high speed pairs), 8 Digital outputs (2 PWM output, 6 Relay type), 4 Analog inputs (Voltage 0-10V, 0-5V, Current 4 -20mA, 0-20mA, Thermocouple, RTD), 1

□ Download Configuration Settings

☑ Add tags for XW, YW and MW used for expansion modules.

5. Click the **Analog** tab.

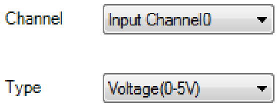


Channel Input Channel0

Type Not Defined

Normalisation Factor 0 (0-100)

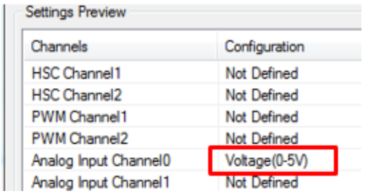
6. We will use **Input Channel 0** with a 0 to 5 V input range. Make sure **Input Channel0** is selected in the **Channel** dropdown menu and select **Voltage(0-5V)** in the **Type** dropdown menu.
Note: You must scroll down in the **Type** dropdown menu to find the **Voltage(0-5V)** option.



Channel Input Channel0

Type Voltage(0-5V)

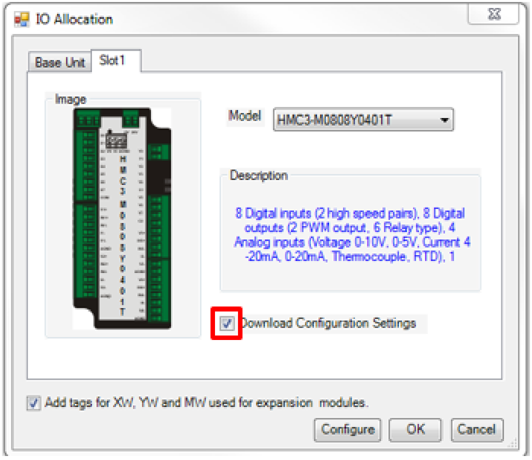
7. Click **Confirm** to save this selection. Note that the selection now appears in the **Settings Preview** section at the bottom of the window.



Settings Preview

Channels	Configuration
HSC Channel1	Not Defined
HSC Channel2	Not Defined
PWM Channel1	Not Defined
PWM Channel2	Not Defined
Analog Input Channel0	Voltage(0-5V)
Analog Input Channel1	Not Defined

8. Click **Close** in the **Configuration** window. In the **IO Allocation** window, check the **Download Configuration Settings** checkbox.



Model: HMC3-M0808Y0401T

Description: 8 Digital inputs (2 high speed pairs), 8 Digital outputs (2 PWM output, 6 Relay type), 4 Analog inputs (Voltage 0-10V, 0-5V, Current 4 -20mA, 0-20mA, Thermocouple, RTD), 1

☑ Download Configuration Settings

☑ Add tags for XW, YW and MW used for expansion modules.

9. Click **OK** in the **IO Allocation** window to complete the allocation.

Doing this adds the selected module to the project, allocates a set of I/O module tags in the tag database, and sets the default values of those tags according to the selected configuration. For example, the settings above will create a tag called **Slot01-CH0_Analog_IP_Type** at address MW0160 and set the initial value of this tag to 6, which is the configuration register setting for the 0 to 5V input range. The raw input value can then be read from the tag named **Slot01-CH0_AnalogIPReg** at address XW0111.

For more information on I/O modules and associated tags, see the appropriate *I/O Module Guide* for your series of product.

Add Tags to the Project

Tags are names assigned to internal memory registers of the HMC, contacts of an expansion module, and any external PLC data registers/coils. Some system tags are predefined when you first begin a project. Other tags are created by the programmer. For example, you must create and assign a tag to every PLC memory address that you wish to read/write. When using the optional I/O Expansion modules, tags are created in order to use them. The **Tag Database** collects and stores all tags for review and editing.

Once a tag is assigned, you can easily link any object (i.e. bit lamp, numeric register, etc.) to the tag. Tags have several advantages.

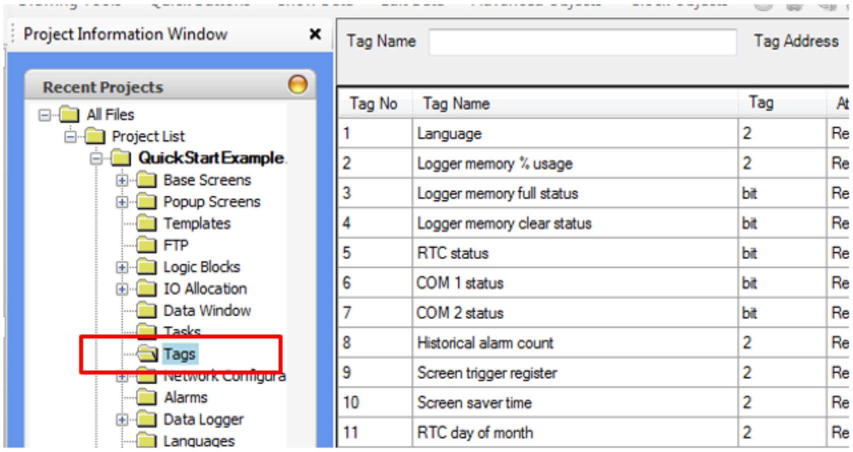
- Tags provide an organized method of tracking all memory addresses used in a project.
- Tags are much more descriptive of functionality than the name of the memory address.
- Tags are easily edited, should a change be required.
- Tags can be exported and imported into other MAPware-7000 projects, regardless of which HMC/MLC unit is selected.

Once defined, the tag name can be used throughout the project to refer to a particular register, without needing to remember its memory address.

In Native Ladder projects, tags are defined to a specific address in the HMC's memory. These addresses are grouped into different address spaces according to how a tag is intended to be used. Examples are; D-registers (general purpose data registers for 16- or 32-bit data), B-registers (single bit internal registers), and S/SW-registers (system tags that control how the HMC hardware functions).

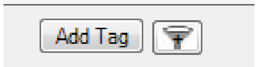
In addition to the tags automatically generated for the I/O card, we will create custom tags to display values on the HMC screen and do calculations within Ladder Logic blocks.

1. Click the **Tags** folder in the **Project Information Window** to open the **Tag Database**.

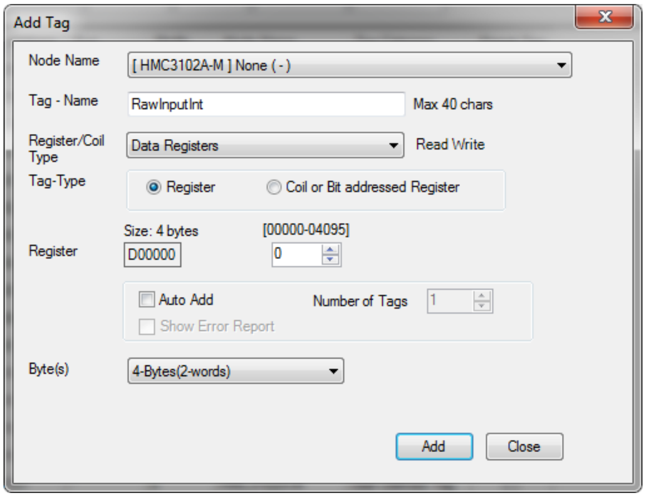


Tag No	Tag Name	Tag	At
1	Language	2	Re
2	Logger memory % usage	2	Re
3	Logger memory full status	bit	Re
4	Logger memory clear status	bit	Re
5	RTC status	bit	Re
6	COM 1 status	bit	Re
7	COM 2 status	bit	Re
8	Historical alarm count	2	Re
9	Screen trigger register	2	Re
10	Screen saver time	2	Re
11	RTC day of month	2	Re

2. To add a tag, right-click in the list of tags and select **Add** from the context menu, or click the **Add Tag** button up top.



3. The **Add Tag** window is displayed. This window is used to configure the parameters for the new tag.



Node Name: [HMC3102A-M] None (-)
Tag - Name: RawInputInt — Max 40 chars
Register/Coil Type: Data Registers — Read Write
Tag-Type: ● Register ○ Coil or Bit addressed Register
Register: D00000 — Size: 4 bytes — [00000-04095] 0
□ Auto Add — Number of Tags 1
□ Show Error Report
Byte(s): 4-Bytes(2-words)

- **Node Name** – Specifies the device in which the tag is located. In this example, there is only one Node, the HMC3102A-M itself. If we had configured an external device in the **Network Configuration** window, such as another PLC, it would be available to select in this dropdown list.
- **Tag-Name** – Enter a descriptive name for the tag.
- **Register/Coil Type** – Pick a memory range to use for the tag.
- **Tag-Type** – Select between a Boolean two state tag (Coil or Bit addressed Register), or a Register tag (i.e. a byte, word or double word tag).
- **Register** – Memory address of the tag.
- **Byte(s)** – Specify the size of the tag (byte, word, or double word).

4. We will add the following tags to the project using the **Add Tag** window.

Name	Register / Coil Type	Register (address)	Byte(s) (length)	Description
RawInputInt	Data Registers	D00000	4-Bytes	Integer register to contain raw analog input
RawInputFloat	Data Registers	D00002	4-Bytes	Floating point format tag version of *RawInputInt*
RawLow	Data Registers	D00004	4-Bytes	Smallest possible value input register can have
RawHigh	Data Registers	D00006	4-Bytes	Largest possible value input register can contain
EngLow	Data Registers	D00008	4-Bytes	Smallest possible value of scaled input
EngHigh	Data Registers	D00010	4-Bytes	Largest possible value of scaled input
ScaledInput	Data Registers	D00012	4-Bytes	Scaled input value.
HighLimit	Data Registers	D00014	4-Bytes	High limit to test *ScaledInput* against
LowLimit	Data Registers	D00016	4-Bytes	Low limit to test *ScaledInput* against
DisableScaling	Internal Coils	B00000	Coil	Disable (ON) / Enable(OFF) Scaling Logic
EnableLimits	Internal Coils	B00001	Coil	Enable (ON) / Disable (OFF) limit logic

   a. The **Node Name** is not modified, as there are no other nodes in our project and all tags are local.
   b. Type *RawInputInt* in the **Tag-Name** field and select **Data Registers** from the **Register/Coil Type** drop down.
   c. This first tag will be at **Register** 0, but for each following tag we will increment the address by 2, as each tag is 4 bytes (32 bits) long, and each **Data Register** is 2 bytes (16 bits) in length.
   d. Select **4-Bytes(2-words)** from the **Byte(s)** dropdown menu.
   e. Click **Add**. The tag is added to the tag database, and the **Add Tag** window remains open.
   f. Change the **Tag-Name** to *RawInputFloat* and increment the **Register** by 2.
   g. Click **Add** again.

5. Continue this process for the rest of the listed tags above. When adding the *DisableScaling* and *EnableLimits* tags, the **Register/Coil Type** must be changed to **Internal Coils**.
6. Once all tags have been added, click the **Close** button. Any typos can be fixed by double-clicking on the tag to open the **Edit Tag** dialog box.

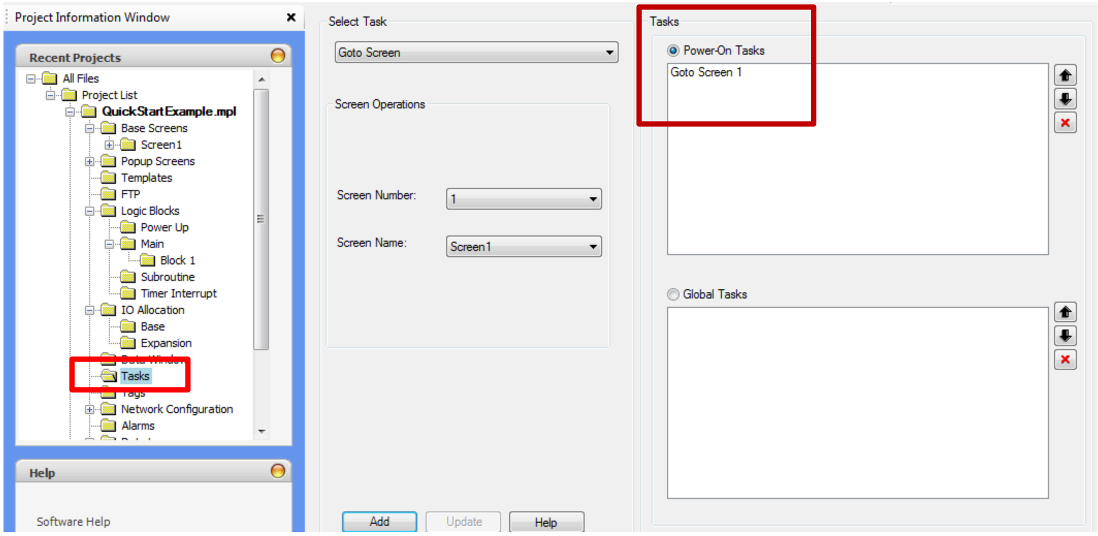
Initializing Tags with Power-On Tasks

To perform the scaling operation in this sample project, we need to initialize the tags used to define the input and output scales. There are two primary places available to perform initializations in MAPware-7000.

- In a **Power-On Task**
- In a **Power Up Logic Block**

We will use a set of **Power-On Tasks**. Tasks are specifically predefined actions taken by the controller, such as writing a value to a tag register, displaying a new screen, turning a bit on/off, etc. The number of tasks is limited only by the total amount of memory available in the controller. Each task has two fundamental components: the action taken when the task activates, and the triggering mechanism that starts the action.

1. Click the **Tasks** folder in the **Project Information Window** to open the task editor.



Select Task

Goto Screen

Screen Operations

Screen Number: 1

Screen Name: Screen1

Tasks

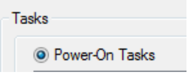
● Power-On Tasks

Goto Screen 1

○ Global Tasks

There are two Tasks sections in this window. **Power-On Tasks** are executed only once when the HMC first powers up. **Global Tasks** are executed continuously while the HMC is running. Notice that there is a default **Power-On Task** that tells the HMC which screen to display first.

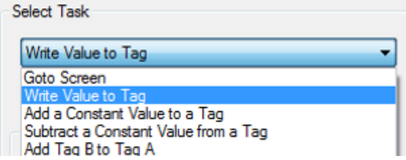
2. Make sure the **Power-On Tasks** radio button is selected.



Tasks

● Power-On Tasks

3. From the **Select Task** dropdown list, select the **Write Value to Tag** task.

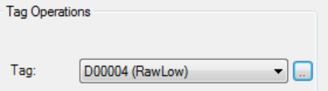


Select Task

Write Value to Tag

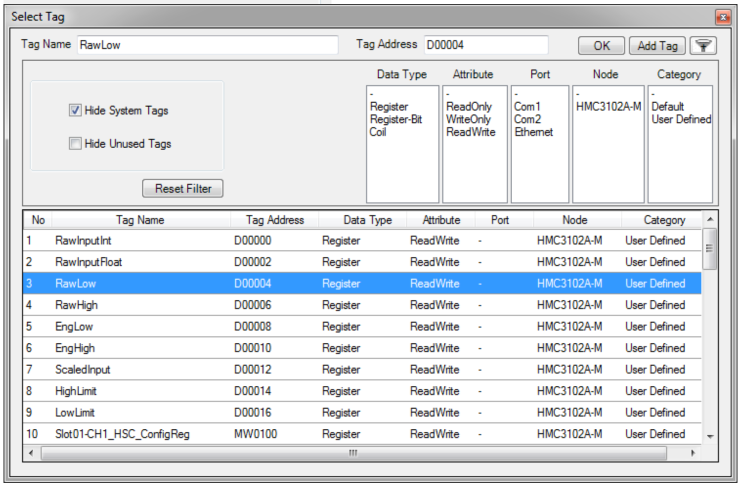
Goto Screen
Write Value to Tag
Add a Constant Value to a Tag
Subtract a Constant Value from a Tag
Add Tag B to Tag A

4. In the **Tag** selection, choose *RawLow* from the dropdown or click the ... button to the right of the dropdown to open the **Select Tag** window.



Tag Operations

Tag: D00004 (RawLow)



Select Tag

Tag Name RawLow

Tag Address D00004

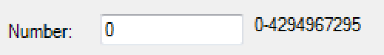
☑ Hide System Tags

□ Hide Unused Tags

Data Type	Attribute	Port	Node	Category
- Register Register-Bit Coil	- ReadOnly WriteOnly ReadWrite	- Com1 Com2 Ethernet	- HMC3102A-M	- Default User Defined

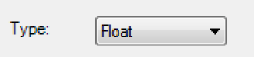
No	Tag Name	Tag Address	Data Type	Attribute	Port	Node	Category
1	RawInputInt	D00000	Register	ReadWrite	-	HMC3102A-M	User Defined
2	RawInputFloat	D00002	Register	ReadWrite	-	HMC3102A-M	User Defined
3	RawLow	D00004	Register	ReadWrite	-	HMC3102A-M	User Defined
4	RawHigh	D00006	Register	ReadWrite	-	HMC3102A-M	User Defined
5	EngLow	D00008	Register	ReadWrite	-	HMC3102A-M	User Defined
6	EngHigh	D00010	Register	ReadWrite	-	HMC3102A-M	User Defined
7	ScaledInput	D00012	Register	ReadWrite	-	HMC3102A-M	User Defined
8	HighLimit	D00014	Register	ReadWrite	-	HMC3102A-M	User Defined
9	LowLimit	D00016	Register	ReadWrite	-	HMC3102A-M	User Defined
10	Slot01-CH1_HSC_ConfigReg	MW0100	Register	ReadWrite	-	HMC3102A-M	User Defined

5. In the **Number** field, enter *0*.



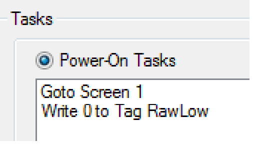
Number: 0 0-4294967295

6. Select **Float** from the **Type** dropdown list.



Type: Float

7. Click the **Add** button. The task now appears in the **Power-On Tasks** list.



Tasks

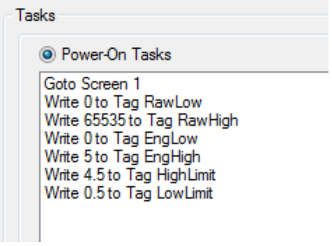
● Power-On Tasks

Goto Screen 1
Write 0 to Tag RawLow

8. Repeat this process to add **Power-On Tasks** that initialize the tags in the table below with the values shown. Make sure to set the **Type** to **Float** for each task.

Tag	Value	Type	Notes
RawLow	0	Float	Minimum analog input value
RawHigh	65535	Float	Maximum analog input value (16 bit resolution)
EngLow	0	Float	Minimum scaled engineering value (0 Volts)
EngHigh	5.0	Float	Maximum scaled engineering value (5 Volts)
HighLimit	4.5	Float	Initial high limit
LowLimit	0.5	Float	Initial low limit

9. When complete, the list of **Power-On Tasks** should look like this.



Tasks

● Power-On Tasks

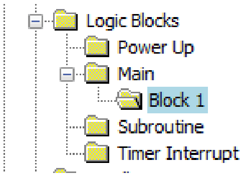
Goto Screen 1
Write 0 to Tag RawLow
Write 65535 to Tag RawHigh
Write 0 to Tag EngLow
Write 5 to Tag EngHigh
Write 4.5 to Tag HighLimit
Write 0.5 to Tag LowLimit

Ladder Logic Blocks

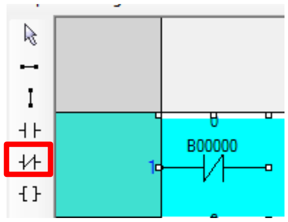
With the tags defined and initialized, it is time to put them to use in a ladder logic program.

Scaling Analog Inputs

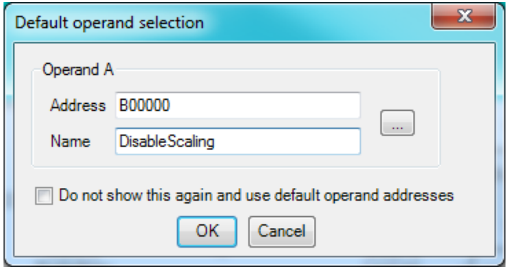
The first rung of the logic block will scale the raw input from Analog Input Channel 0 of the **HMC3-M0808Y0401T** I/O card to an engineering value of 0 to 5 volts. This is done by moving the raw input into a local tag, converting it to a floating point number, and then using the **Scale** instruction to scale the value.



1. Expand the **Logic Blocks** folder in the **Project Information Window**, then expand the **Main** folder.
2. MAPware-7000 creates a default block in the **Main** folder called **Block 1**. Click the folder for this block to open the Ladder Logic editor.
3. Select **NC (Normally Closed) Contact** from the **I/O Instructions** dropdown on the instruction menu, or click the -|/|- icon on the **Common Objects Toolbar**. Then click directly to the left of the connection point for rung 1 to place the **NC Contact**.



4. The **Default operand selection** window will open. Select the *DisableScaling* tag, then click **OK**. The ... button in the popup window will open the **Select Tag** window used in the previous section to help you find the required tag.



Default operand selection

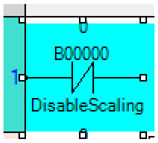
Operand A

Address: B00000

Name: DisableScaling

☐ Do not show this again and use default operand addresses

OK Cancel



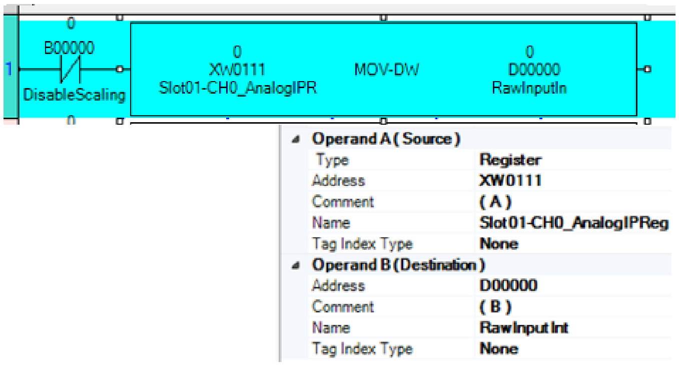
Note: Tag names are truncated by default to 6 characters in the logic editor window. To get the entire tag name to display, select **Tools > Preferences > Project Global Settings** from the menu bar and increase the **Number of tag name characters to be displayed in Instruction** setting to the maximum 20 characters.

This rung of logic can now be turned on and off with the *DisableScaling* tag. It will execute when *DisableScaling* is off, and be skipped if *DisableScaling* is on.

Because our engineering values represent voltages from 0 to 5 volts, we will need them to be floating point numbers. Before scaling, the raw input must be moved into a Data Register and converted to floating point format. To move the input value into a data register, use the **Move DWord** instruction.

For more information on the specific instructions used in the following steps, see the MAPware-7000 Ladder Logic Guide.

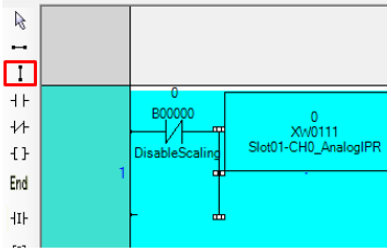
5. Select **Data Transfer > Move DW** from the instruction menu, then click directly to the right of the Normally Open Contact to place the instruction. Select **Slot01-CH0_AnalogIPReg** as **Operand A (Source),** and *RawInputInt* as **Operand B (Destination)**. This will move the 16-bit resolution raw analog input data into the tag we have created to work with the data.



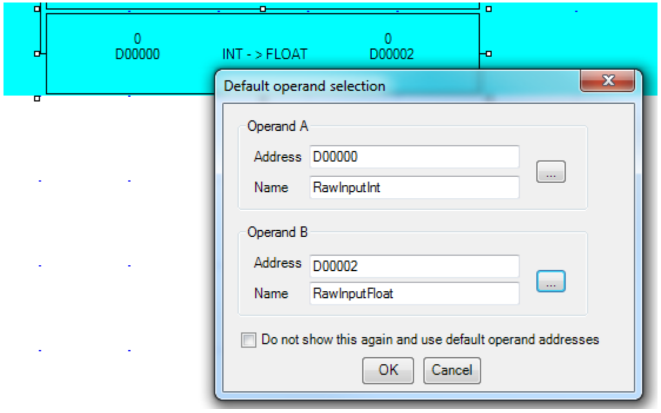
Operand A (Source)	
Type	Register
Address	XW0111
Comment	(A)
Name	Slot01-CH0_AnalogIPReg
Tag Index Type	None
Operand B (Destination)	
Address	D00000
Comment	(B)
Name	RawInput Int
Tag Index Type	None

6. Before placing the next instruction, we need to place a **Vertical Link**, also called a branch, after the contact and before the move instruction. Click the **Vertical Link** icon in the **Common Objects Toolbar**, then click between the **NC Contact** and the **Move DWord** instruction to place

the **Vertical Link**.



7. To the right of this link, place an **Integer to Float** instruction (**Conversion > Integer to Float** from the instruction menu).
8. Configure **Operand A** with the *RawInputInt* tag and **Operand B** with the *RawInputFloat* tag. As the name of the instruction implies, this will convert our raw input from an integer to a float.



Default operand selection

Operand A

Address D00000

Name RawInputInt

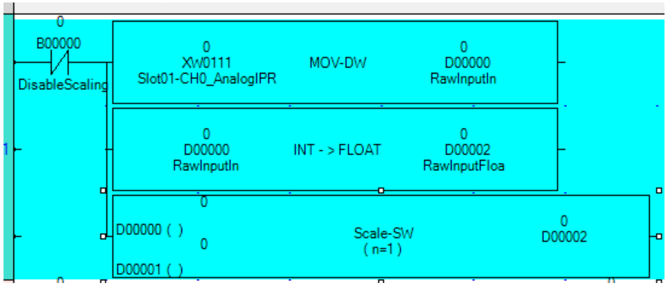
Operand B

Address D00002

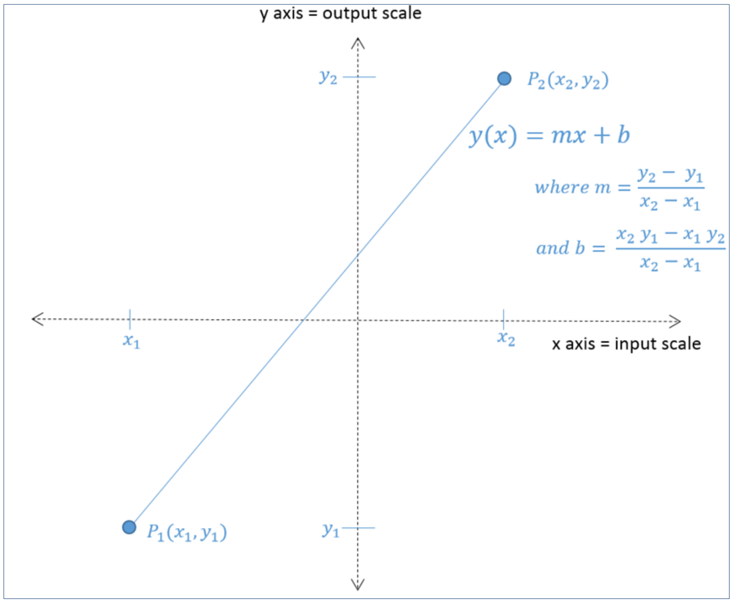
Name RawInputFloat

☐ Do not show this again and use default operand addresses

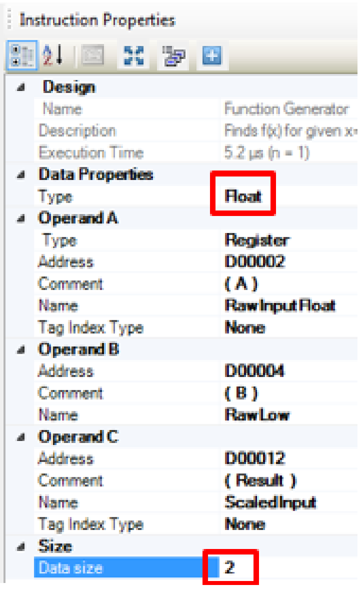
9. Another vertical link is required to add a third instruction below the **Integer to Float** instruction. Again, click the **Vertical Link** icon and then click immediately to the left of the **Integer to Float** instruction.
10. To the right of this vertical link, place a **Scale** instruction (**Functions -> Scale**). Click **OK** to accept the default operand selection.



The **Scale** instruction has three operands. These are used to perform a linear mapping of the input (Operand A), to the output (Operand C). The figure below shows a graphic representation of the **Scale** instruction with two data points, n=2.



The input value x (Operand A) is mapped to an output value y(x) (Operand C) using the relationship shown. m and b, the slope and offset of y(x), are determined by four tags starting at the address specified in Operand B. These correspond, in consecutive order according to tag address, to the parameters. x_1, x_2, y_1, y_2, as shown in the figure.


Instruction Properties

Design	
Name	Function Generator
Description	Finds f(x) for given x
Execution Time	5.2 µs (n = 1)
Data Properties	
Type	Float
Operand A	
Type	Register
Address	D00002
Comment	(A)
Name	RawInputFloat
Tag Index Type	None
Operand B	
Address	D00004
Comment	(B)
Name	RawLow
Operand C	
Address	D00012
Comment	(Result)
Name	ScaledInput
Tag Index Type	None
Size	
Data size	2

11. In the **Instruction Properties** panel, the **Type** property will need to be changed to **Float**. and the **Data size** property to 2 before selecting our tags. The **Data size** specifies the number of points to use in the linear mapping. This appears as *n* in the instruction.
12. **Operand A** is the input, or x in the figure. For this operand use the output of the **Integer to Float** instruction (*RawInputFloat*).
13. **Operand B** is the starting address of a block of four registers. The first of these registers defines the start of the input scale (the x coordinate of P_1 in the figure), the second register defines the end of the input scale (the x coordinate of P_2 in the figure), the third register defines the start of the output scale (the y coordinate of P_1 in the figure), and the fourth register defines the end of the output scale (the y coordinate of P_2 in the figure). Conveniently, we've already defined a block of four consecutive tags for this purpose, starting with the tag. *RawLow* at address D00004.

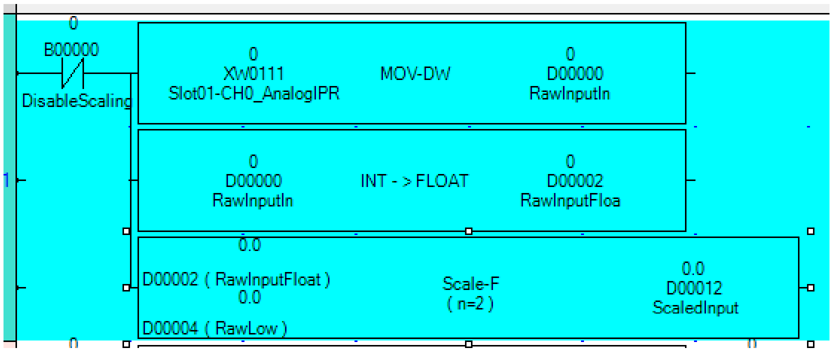
Tag Name	Register	Coordinate
RawLow	D00004	x_1
RawHigh	D00006	x_2
EngLow	D00008	y_1
EngHigh	D00010	y_2

Select *RawLow* as **Operand B**.

14. **Operand C** is the output, or y(x) in the figure. Select tag *ScaledInput* for Operand C.

You must change the instruction type to **Float** before assigning operands, because the instruction's default type is a 2 byte integer. Attempting to select our previously created 4 byte float type tags before changing the instruction's type to float will result in duplicate tags being created.

This completes the setup of the scaling operation. If scaling has not been disabled, this rung will copy the raw analog input into a local tag, convert the data into floating point number, and finally scale it from a raw 16 bit binary number (range from 0 to 65535) to an engineering value (from 0 to 5 volts).

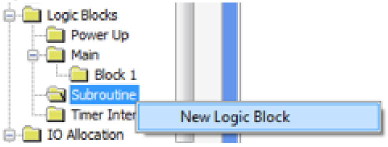


Checking Output Limits

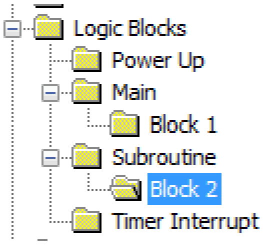
Next we will add the high and low limit logic to control two digital outputs that will indicate when the analog input is either too high or too low. This logic will be enabled or disabled using an enable bit (*EnableLimits*) in an input contact. If the output coil instructions were placed on the same rung as this enable contact, the outputs would be forced OFF whenever the enable bit is OFF. We would instead like the outputs to be free to toggle states even if *EnableLimits* is disabled. To accomplish this, the logic will be placed in a separate subroutine logic block, and the enable contact will be used to control whether or not the block is called (evaluated).

Create a new Subroutine logic block.

1. Right-click the **Subroutine** folder and select **New Logic Block** from the context menu.

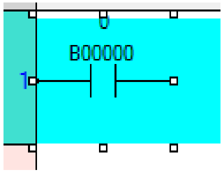


2. **Block 2** is created and a new editor window is opened.

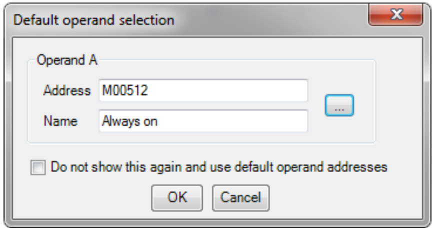


This subroutine will have two lines. One to check the high limit and one to check the low limit. The first rung will be used to check if the scaled input value is above the *high limit*.

3. Click the **NO Contact** icon ┤├ in the common objects toolbar, then click the input to rung 1 in the logic editor to place the contact.



4. Select the **Always on** (**M00512**) system bit. This bit is set on by the processor and will ensure that this rung always executes.



Default operand selection ✕

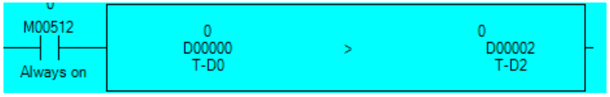
Operand A

Address M00512

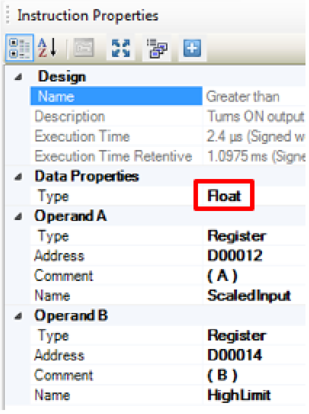
Name Always on

☐ Do not show this again and use default operand addresses

5. Select the **Greater Than** instruction from the Instruction menu (**Compare > Greater Than**) and click to the right of the **NO Contact**. Click **OK** to accept the default operand selection.



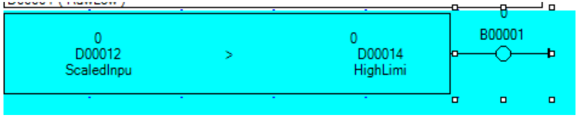
6. The **Type** property for the **Greater Than** instruction will need to be changed to **Float** before assigning operands to ensure that duplicate tags are not created. Do this in the **Instruction Properties** grid.
7. Then select *ScaledInput* for **Operand A**, and *HighLimit* for **Operand B**.



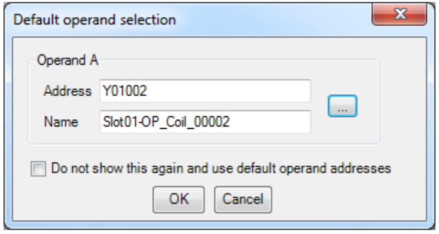
Instruction Properties

Design	
Name	Greater than
Description	Turns ON output
Execution Time	2.4 µs (Signed w
Execution Time Retentive	1.0975 ms (Signe
Data Properties	
Type	**Float**
Operand A	
Type	**Register**
Address	**D00012**
Comment	**(A)**
Name	**ScaledInput**
Operand B	
Type	**Register**
Address	**D00014**
Comment	**(B)**
Name	**HighLimit**

8. If the *ScaledInput* value is above the high limit, the logic will set an output on the I/O card. A standard output coil instruction is used for this purpose. Click the **Output** -( )- icon in the **Common Objects Toolbar**.
9. Click to the right of the **Greater Than** instruction to place the **Output**.



10. Select **Slot01-OP_Coil_00002** (Y01002) as the tag controlled by this output coil and click **OK**.



Default operand selection

Operand A

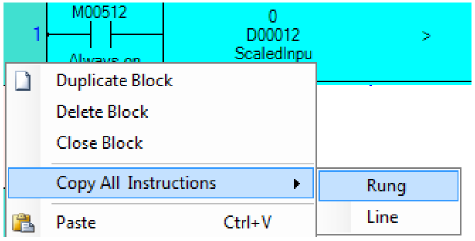
Address: Y01002

Name: Slot01-OP_Coil_00002

☐ Do not show this again and use default operand addresses

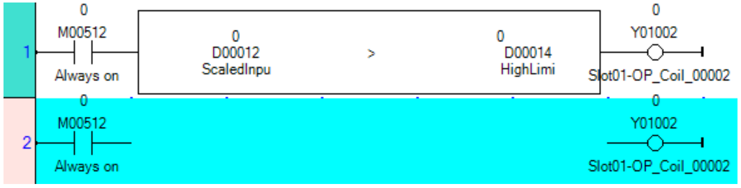
This is the coil controlling the first relay output on the I/O card (terminal Y2).

11. To complete this subroutine, we just need to add the logic to test the lower limit. We will start rung 2 by making a copy of rung 1. Right-click to the left of the input on rung 1. From the context menu select **Copy All Instructions > Rung**.

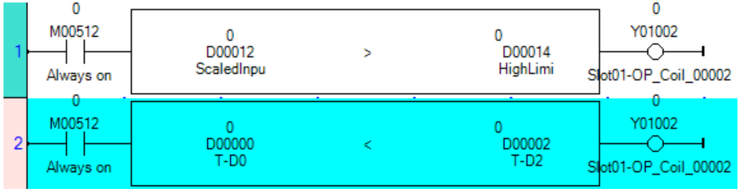


12. Right-click rung 2. From the context menu, select **Paste**. This will put a copy of the instructions from rung 1 on rung 2.

13. Click the second **Greater Than** instruction to select it. Then right-click to delete it or press Delete on your keyboard.

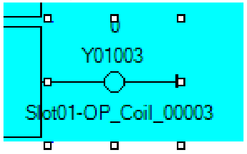


14. Select a **Less Than** instruction (**Compare > Less Than**) to place in the open space and accept the default tags.

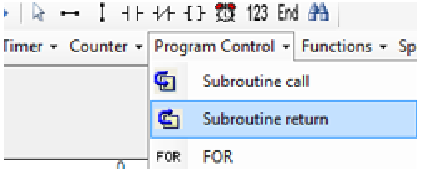


15. In the **Instruction Properties** grid, set the **Type** property to **Float**, **Operand A** to *ScaledInput,* and **Operand B** to *LowLimit*.

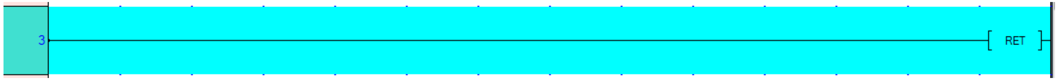
16. Change the **Output** instruction coil address to **Slot01-OP_Coil_00003** (Y01003).



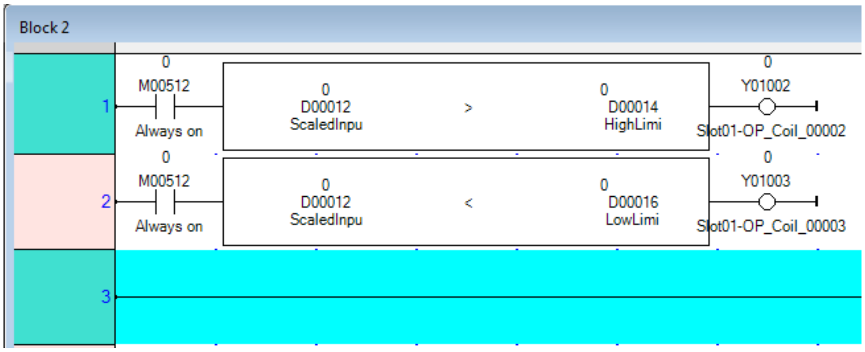
17. Because this logic block is a subroutine, it must end with a **Subroutine Return** instruction. From the instruction menu, select **Program Control > Subroutine return**.



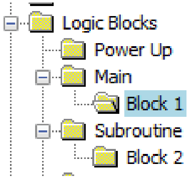
18. Click anywhere on rung 3 to place the instruction.



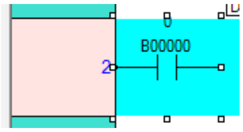
This completes the logic for the subroutine logic block. The completed block should look like this:



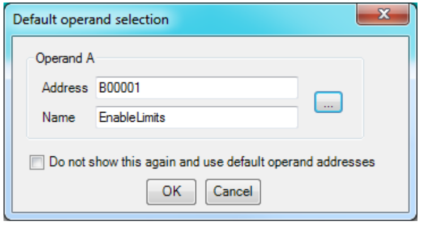
19. Finally, we must add a rung to **Block1** that will call the subroutine. Click the folder for **Block1** in the **Project Information Window** to reopen the Main Block 1 editor.



20. Place an **NO Contact** on rung 2 that can be used to enable/disable the logic rung we are about to create.



21. Select *EnableLimits* to use as the tag for this contact and click **OK** in the tag selection window. This will allow this rung of logic to be turned on and off during runtime.


Default operand selection

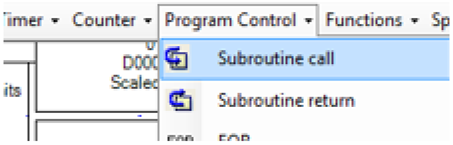
Operand A

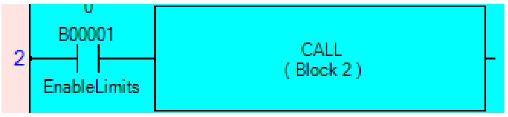
Address B00001

Name EnableLimits

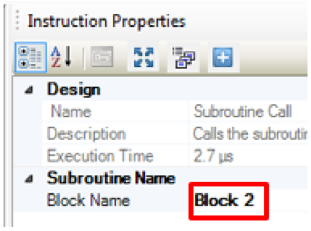
☐ Do not show this again and use default operand addresses

22. Place a **Subroutine call** instruction (**Program Control > Subroutine call**) on the output of this contact.





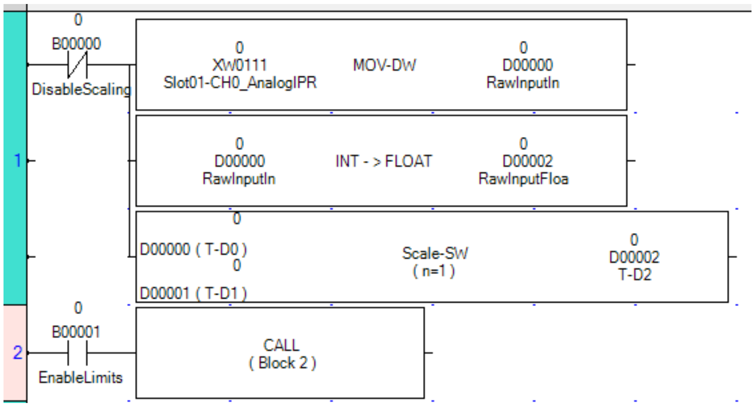
23. Make sure that the **Block Name** property for the subroutine call instruction is set to **Block 2**.



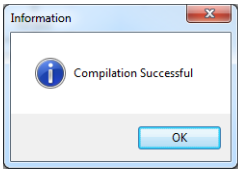
Instruction Properties

Design	
Name	Subroutine Call
Description	Calls the subroutir
Execution Time	2.7 µs
Subroutine Name	
Block Name	Block 2

This completes the logic needed for this sample project. The completed logic block (Block 1) should now look something like this.



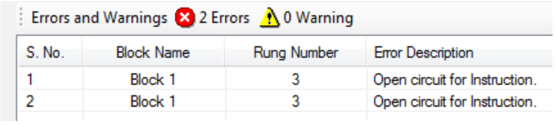
This is a good time to compile and save the project to make sure there are no errors. To do this, either click the compile icon , select **Project > Compile**, or hit F9. A successful compile will show a **Compilation Successful** message.



Information

Compilation Successful

If there are any compile errors, they will be listed in the output window.



Errors and Warnings ⊗ 2 Errors ⚠ 0 Warning

S. No.	Block Name	Rung Number	Error Description
1	Block 1	3	Open circuit for Instruction.
2	Block 1	3	Open circuit for Instruction.

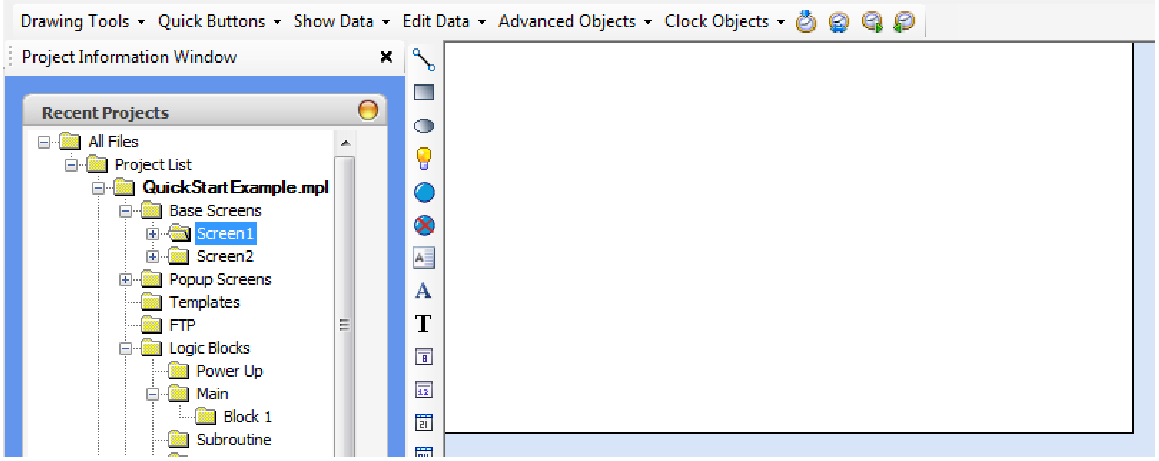
Fix any errors before moving on.

Create Screen Objects

To see what the logic is doing in real time, we will create HMI screens to display information. The first screen will be used to display the digital I/O points. It will show the current state of input channels X0 and X1 and control the outputs Y4 and Y5

1. In the **Project Information Window**, click to expand the **Base Screens** folder. By default, **Screen1** is already present.

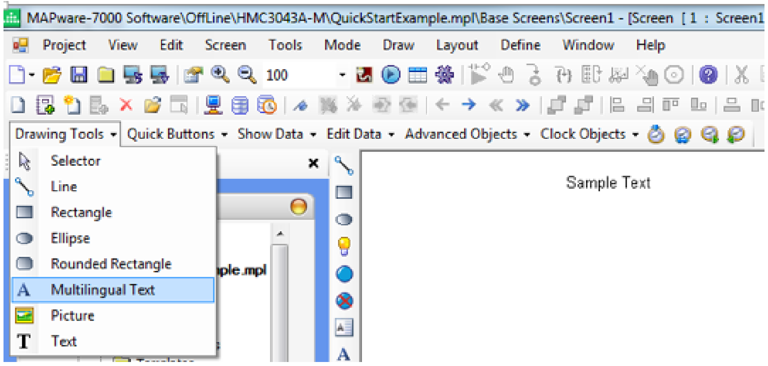
2. Click the **Screen1** subfolder to display the work area for Screen1.



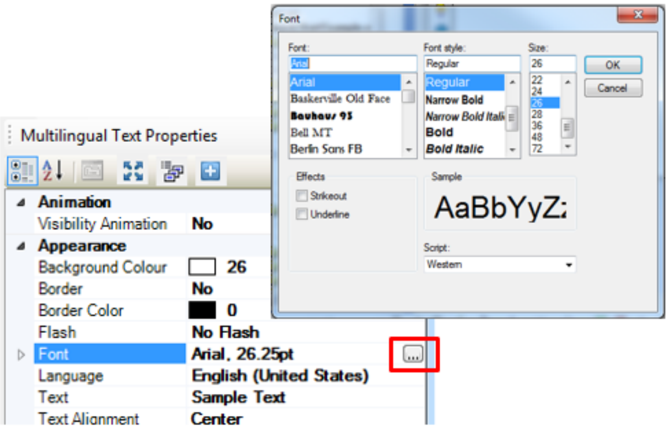
Labels

Before adding data display objects to the screen, we will add labels so that it is clear what data the screen is displaying. There are two text objects in MAPware-7000 – **Multilingual Text** and **Text**. **Multilingual Text** uses Windows fonts and supports the Languages feature, which allows you to select up to nine languages. The **Text** object has more animation features, but fewer font options. We will use **Multilingual Text** objects for our labels.

1. Select **Drawing Tools > Multilingual Text** from the **Draw** menu, then click in the work area to place the **Multilingual Text** object.

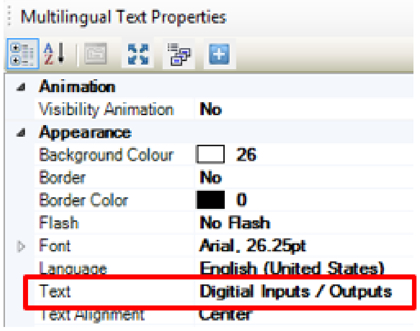


2. This text will provide a title for the window. In the **Properties** grid, select the **Font** property and then click the ellipsis button to the right. Change the font size to 48 point.



Visibility Animation: No
Background Colour: 26
Border: No
Border Color: 0
Flash: No Flash
Font: Arial, 26.25pt
Language: English (United States)
Text: Sample Text
Text Alignment: Center

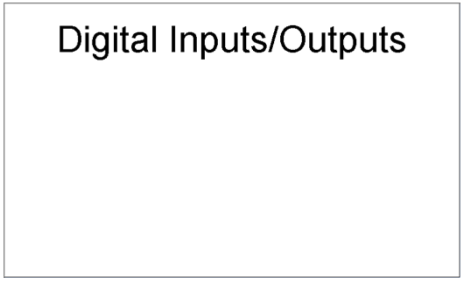
3. Change the **Text** property from Sample Text to *Digital Inputs/Outputs*.



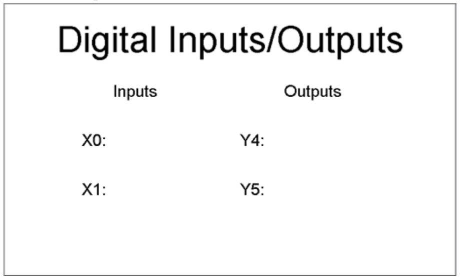
Multilingual Text Properties

Animation	
Visibility Animation	No
Appearance	
Background Colour	□ 26
Border	No
Border Color	■ 0
Flash	No Flash
Font	Arial, 26.25pt
Language	English (United States)
Text	Digitial Inputs / Outputs
Text Alignment	Center

4. Use the position boxes to expand the text object as needed to display the entire text. Use the **Center along Screen's Horizontal** toolbar icon to center the title on the screen.



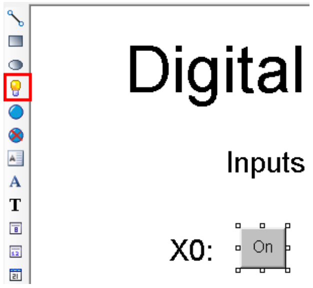
5. Create a copy of this **Multilingual Text** object to use as a label for the input section. Right-click the object and select **Copy** from the context menu. Then right-click elsewhere on the screen and select **Paste** from the context menu. Click and drag the copy to a new location on the screen.
6. Change the **Font Size** property to 22 point and the **Text** property to *Inputs*.
7. Continue to make copies to create the following labels as shown [*Inputs*, *Outputs*, *X0:*, *X1:*, *Y4:*, and *Y5:*].



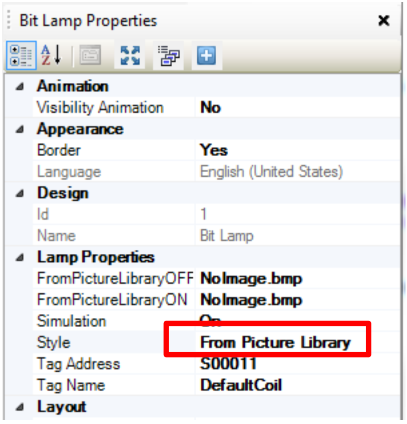
Bit Lamps

With the labels in place, we will now add bit lamps to monitor the state of the two inputs.

1. Click the **Bit Lamp** icon , then click the screen to the right of the label for *X0:*.



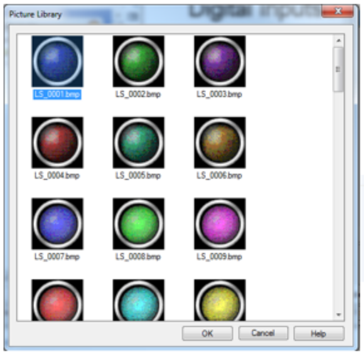
2. In the properties grid for this **Bit Lamp**, change the **Style** property to **From Picture Library**.

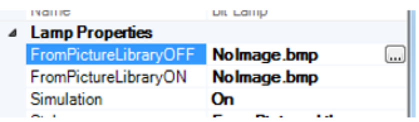


Bit Lamp Properties

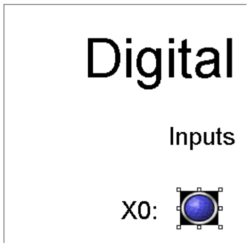
Animation	
Visibility Animation	No
Appearance	
Border	Yes
Language	English (United States)
Design	
Id	1
Name	Bit Lamp
Lamp Properties	
FromPictureLibraryOFF	NoImage.bmp
FromPictureLibraryON	NoImage.bmp
Simulation	On
Style	From Picture Library
Tag Address	S00011
Tag Name	DefaultCoil
Layout	

3. Click the field for the **FromPictureLibraryOFF** property, then click the ellipsis button ⋯ to open the library browser. Select any of the lamps, then click **OK**.



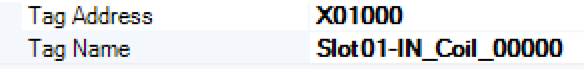


Lamp Properties	
FromPictureLibraryOFF	NoImage.bmp ⋯
FromPictureLibraryON	NoImage.bmp
Simulation	On



Note: This will automatically select the corresponding ON image for the **FromPictureLibraryON** property.

4. In the **Tag Name** property for the **Bit Lamp** object, select **Slot01-IN_Coil_00000** as the tag to use.



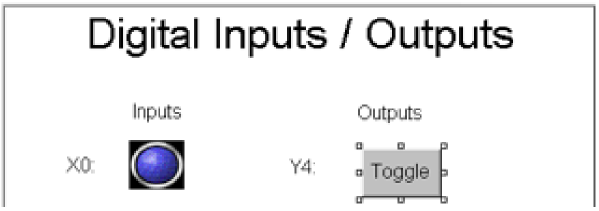
Tag Address	X01000
Tag Name	Slot01-IN_Coil_00000

5. Make a copy of this bit lamp and place it to the right of the *X1:* label.
6. Change the **Tag Name** property of the new **Bit Lamp** object to **Slot01-IN_Coil_00001**.

Toggle Bit Objects

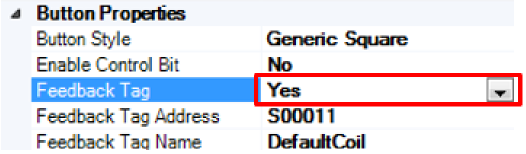
Next, we will add Toggle Bit objects to control two outputs.

1. From the Draw toolbar select **Advanced Objects > Buttons > Bit Action > Toggle Bit**. Click to the right of the label for output Y4 to place the **Toggle Bit** button.



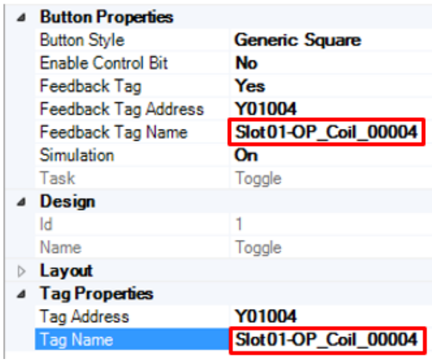
2. In the properties grid for this **Toggle Bit** object, change the **Feedback Tag** property to **Yes**. This allows a tag to control the appearance of the **Toggle Bit** object on the screen. It can be set to the

same tag that the object is controlling, or to a different tag.



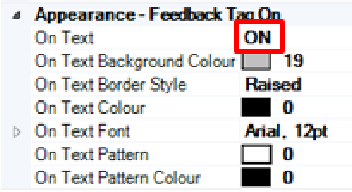
Button Properties	
Button Style	Generic Square
Enable Control Bit	No
Feedback Tag	Yes
Feedback Tag Address	S00011
Feedback Tag Name	DefaultCoil

3. Select **Slot01-OP_Coil_00004** for both the **Feedback Tag Name** property and the **Tag Name** property.



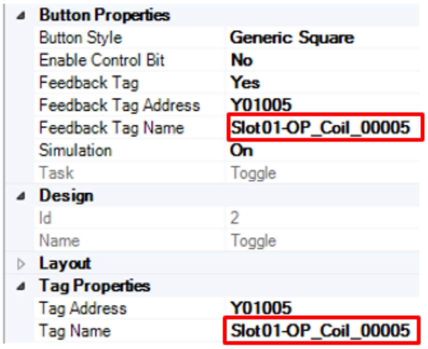
Button Properties	
Button Style	Generic Square
Enable Control Bit	No
Feedback Tag	Yes
Feedback Tag Address	Y01004
Feedback Tag Name	Slot01-OP_Coil_00004
Simulation	On
Task	Toggle
Design	
Id	1
Name	Toggle
Layout	
Tag Properties	
Tag Address	Y01004
Tag Name	Slot01-OP_Coil_00004

4. In the **Appearance - Feedback Tag On** section, change the **On Text** property to *ON*.



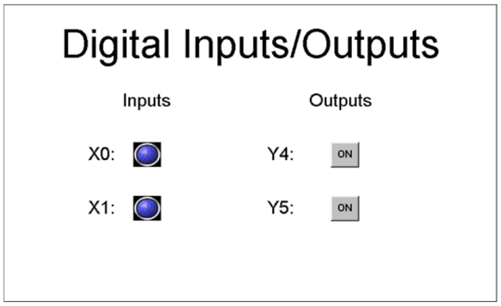
Appearance - Feedback Tag On	
On Text	ON
On Text Background Colour	▨ 19
On Text Border Style	Raised
On Text Colour	■ 0
On Text Font	Arial, 12pt
On Text Pattern	□ 0
On Text Pattern Colour	■ 0

5. Make a copy of the **Toggle Bit** object and place it to the right of the *Y5:* label.
6. In the properties for the new **Toggle Bit** object change both the **Feedback Tag Name** and **Tag Name** properties to **Slot01-OP_Coil_00005**.



Button Properties	
Button Style	Generic Square
Enable Control Bit	No
Feedback Tag	Yes
Feedback Tag Address	Y01005
Feedback Tag Name	Slot01-OP_Coil_00005
Simulation	On
Task	Toggle
Design	
Id	2
Name	Toggle
Layout	
Tag Properties	
Tag Address	Y01005
Tag Name	Slot01-OP_Coil_00005

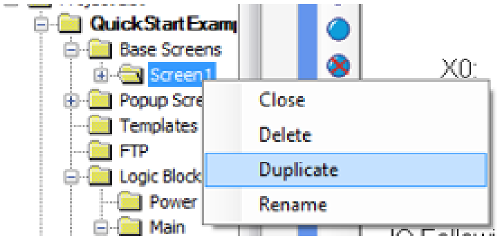
7. Use the alignment toolbar options to neaten up the screen.



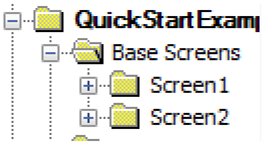
Duplicating a Screen

To save time setting up the next screen, we will duplicate this first screen and use it as a starting point for the second.

1. Right-click the **Screen1** folder in the Project Information Window and select **Duplicate** from the context menu.

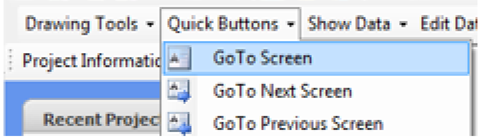


2. You should now see two screens listed in the project tree.

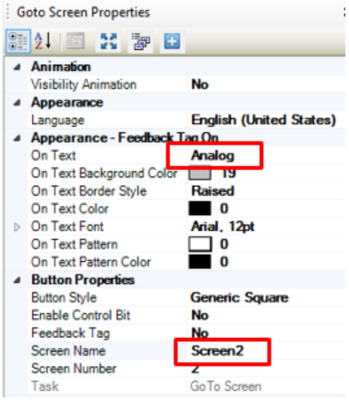


   The editor screen has changed to the new **Screen2,** even though it looks exactly the same as **Screen1**, because it is a duplicate.

3. Before moving on to **Screen2**, we need to add a navigation button to Screen1 that opens **Screen2**. Click the **Screen1** folder to switch back to the **Screen1** editor window.
4. From the **Draw** toolbar, select **Quick Buttons > GoTo Screen**.

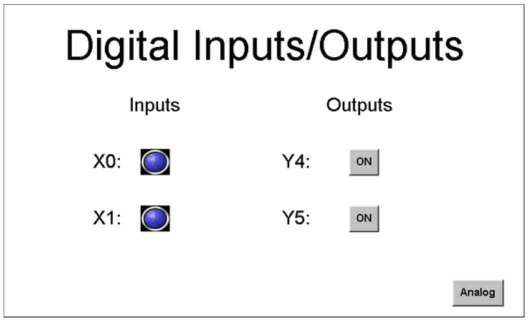


5. Click in the lower right of the screen to place the navigation button.
6. In the Property Grid for the **GoTo Screen** button, change the **On Text** property to **Analog**, and the **Screen Name** property to **Screen 2**.



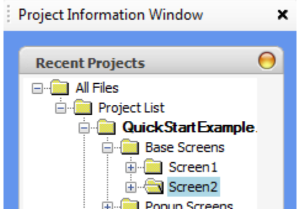
Goto Screen Properties

Animation	
Visibility Animation	No
Appearance	
Language	English (United States)
Appearance - Feedback Tag On	
On Text	Analog
On Text Background Color	19
On Text Border Style	Raised
On Text Color	0
On Text Font	Arial, 12pt
On Text Pattern	0
On Text Pattern Color	0
Button Properties	
Button Style	Generic Square
Enable Control Bit	No
Feedback Tag	No
Screen Name	Screen2
Screen Number	2
Task	GoTo Screen



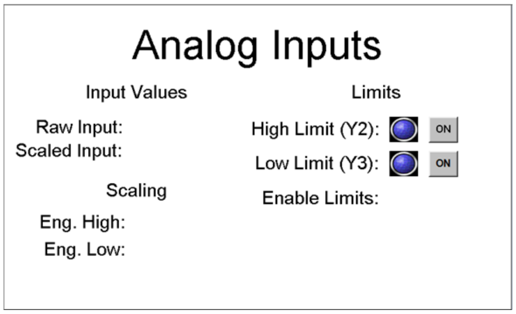
This completes **Screen1**. **Screen2** will be used to monitor analog input channel 0, along with the scaling and limits functionality created earlier in the logic blocks.

7. Click the **Screen2** folder in the **Project Information Window**.



Screen2, looking exactly the same as **Screen1** without the **GoTo Screen** button, will appear in the work area.

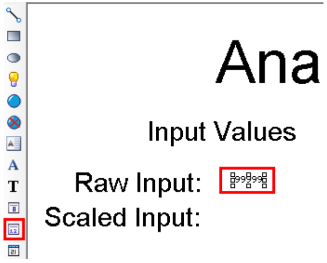
8. To edit the text of an existing label, click the label to open its property grid. Then enter the desired text in the **Text** property. Edit the title of **Screen2** so that it reads *Analog Inputs*.
9. Change the *Inputs* label to *Input Values*, the *X0:* label to *Raw Input:*, the *X1:* label to *Scaled Input:*, the *Outputs* label to *Limits*, the *Y4:* label to *High Limit (Y2):*, and the *Y5:* label to *Low Limit (Y3):*. Adjust the size of the text objects as needed to display the full text.
10. Copy and paste to create four more new labels. Set the text to *Scaling*, *Eng. High:*, *Eng. Low:*, and *Enable Limits:*.
11. Move around your labels, **Bit Lamps**, and **Toggle Bit** objects to the approximate locations shown below.



Numeric Displays

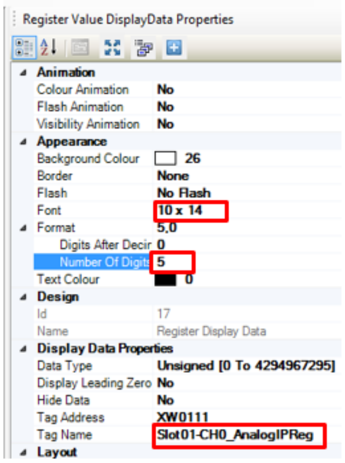
Next we will configure **Numeric Display** objects to show the raw input value and the scaled input value.

1. From the **Common Objects** toolbar, click the **Numeric Display** icon , then click to the right of the *Raw Input:* label.



2. In the properties grid for this **Numeric Display** object, set the **Font** property to **10 x 14** and the **Tag Name** property to **Slot01-Ch0_AnalogIPReg**. In the **Format** section, change the **Number Of**

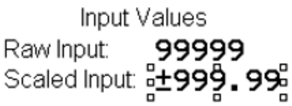
Digits to **5**.

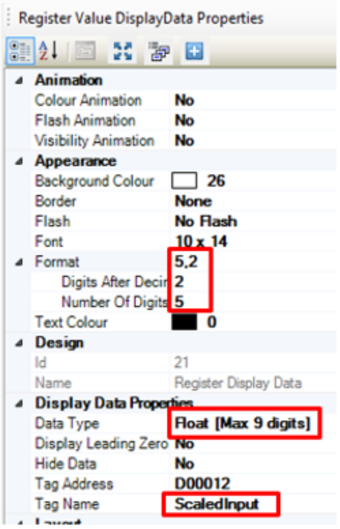


Register Value DisplayData Properties

Animation	
Colour Animation	No
Flash Animation	No
Visibility Animation	No
Appearance	
Background Colour	□ 26
Border	None
Flash	No Flash
Font	10 x 14
Format	5,0
Digits After Decir	0
Number Of Digits	5
Text Colour	■ 0
Design	
Id	17
Name	Register Display Data
Display Data Properties	
Data Type	Unsigned [0 To 4294967295]
Display Leading Zero	No
Hide Data	No
Tag Address	XW0111
Tag Name	Slot01-CH0_AnalogIPReg
Layout	

3. Create a copy of this **Numeric Display** object and paste it directly below to use as the *Scaled Input:* display.
4. In the properties grid for the *Scaled Input:* numeric display, change the **Format > Digits After Decimal** property to **2,** the **Data Type** property to **Float**, and the **Tag Name** property to *ScaledInput*.





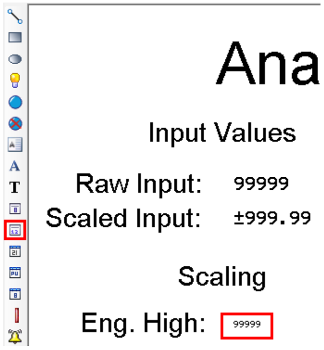
Register Value DisplayData Properties

Animation	
Colour Animation	No
Flash Animation	No
Visibility Animation	No
Appearance	
Background Colour	□ 26
Border	None
Flash	No Flash
Font	10 x 14
Format	5,2
Digits After Decir	2
Number Of Digits	5
Text Colour	■ 0
Design	
Id	21
Name	Register Display Data
Display Data Properties	
Data Type	Float [Max 9 digits]
Display Leading Zero	No
Hide Data	No
Tag Address	D00012
Tag Name	ScaledInput

Numeric Entry

Next we will place two **Numeric Entry** objects next to the *Eng. High:* and *Eng. Low:* labels. The **Numeric Display** object simply displays a register value. The **Numeric Entry** object displays a register value and allows the user to modify the register value. These two **Numeric Entry** objects will allow the user to modify the engineering scale during runtime.

1. From the **Common Objects** toolbar, click the **Numeric Entry** icon, then click to the right of the *Eng. High:* label.



Ana

Input Values

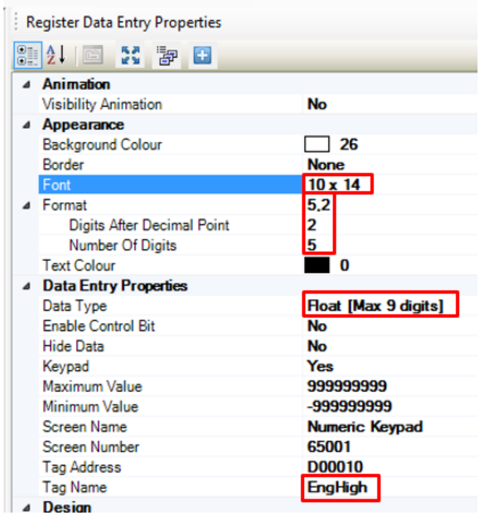
Raw Input: 99999

Scaled Input: ±999.99

Scaling

Eng. High: 99999

2. In the properties grid for the **Numeric Entry** object, set the **Font** property to **10 x 14,** the **Tag Name** to *EngHigh*, the **Data Type** to **Float**, the **Format > Number Of Digits** to **5**, and the **Format > Digits After Decimal Point** to **2**.



Register Data Entry Properties

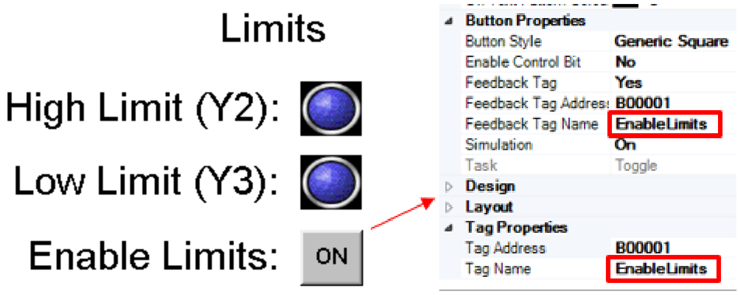
Animation	
Visibility Animation	No
Appearance	
Background Colour	□ 26
Border	None
Font	10 x 14
Format	5.2
Digits After Decimal Point	2
Number Of Digits	5
Text Colour	■ 0
Data Entry Properties	
Data Type	Float [Max 9 digits]
Enable Control Bit	No
Hide Data	No
Keypad	Yes
Maximum Value	999999999
Minimum Value	-999999999
Screen Name	Numeric Keypad
Screen Number	65001
Tag Address	D00010
Tag Name	EngHigh
Design	

3. Create a copy of this Numeric Input object and paste it directly below the original to create a **Numeric Input** for *EngLow:*. Change the **Tag Name** property from *EngHigh* to *EngLow*.

The last section of **Screen2** will display the limit indicators controlled by the ladder logic program. The existing **Bit Lamp** and **Toggle Bit** objects will be modified for this section. Two **Numeric Entry** objects will be added, allowing the limits to be adjusted.

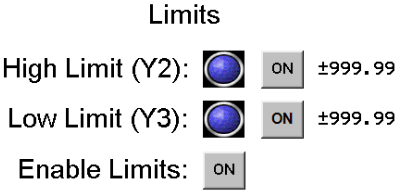
4. Change the **Tag Name** property for the *HighLimit* bit lamp from **Slot01-IN_Coil_00000** to **Slot01-OP_Coil_00002**. Change the **Tag Name** and **Feedback Tag Name** properties for the *HighLimit* toggle switch from **Slot01-OP_Coil_00004** to **Slot01-OP_Coil_00002** as well.
5. Change the **Tag Name** property in the *LowLimit* bit lamp from **Slot01-IN_Coil_00001** to **Slot01-OP_Coil_00003**. Change the **Tag Name** and **Feedback Tag Name** properties for the *LowLimit* toggle switch from **Slot01-OP_Coil_00005** to **Slot01-OP_Coil_00003**.
6. Copy and paste one of the toggle bit objects and place it next to the *Enable Limits:* label.

7. Change both the **Feedback Tag Name** and **Tag Name** properties in the *Enable Limits:* toggle bit object to *EnableLimits*.

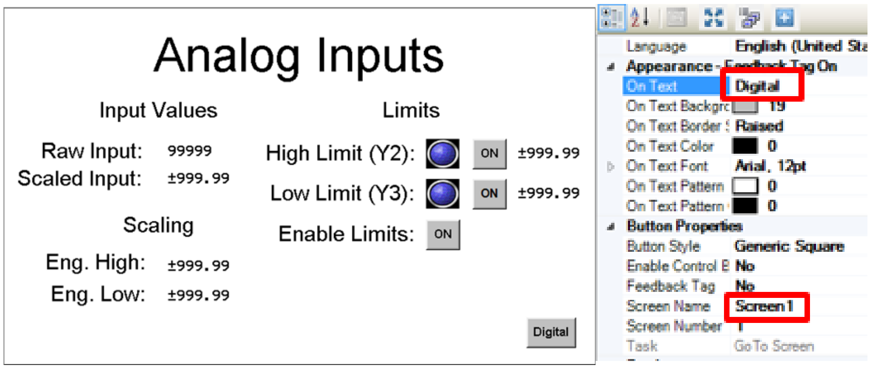


Button Properties	
Button Style	Generic Square
Enable Control Bit	No
Feedback Tag	Yes
Feedback Tag Addres	B00001
Feedback Tag Name	EnableLimits
Simulation	On
Task	Toggle
Design	
Layout	
Tag Properties	
Tag Address	B00001
Tag Name	EnableLimits

8. Create a copy of the *Eng. High:* **Numeric Entry** object and paste it to the right of the *High Limit (Y2):* **Toggle Button** object. Change the **Tag Name** property in this new **Numeric Entry** object from *EngHigh* to *HighLimit*. Repeat to create a *LowLimit* **Numeric Entry** object.



9. To complete the second screen, add a **GoTo Screen** button allowing the operator to navigate back to **Screen1**. Select **Quick Buttons > GoTo Screen** from the **Draw** toolbar. Click in the lower right corner of the screen to place the button.
10. In the property grid for the new button, set the **On Text** property to **Digital** and the **Screen Name** property to **Screen1**.



Language	English (United Sta
Appearance - Feedback Tag On	
On Text	Digital
On Text Backgrc	19
On Text Border S	Raised
On Text Color	0
On Text Font	Arial, 12pt
On Text Pattern	0
On Text Pattern	0
Button Properties	
Button Style	Generic Square
Enable Control B	No
Feedback Tag	No
Screen Name	Screen1
Screen Number	1
Task	GoTo Screen

This completes the set up for the second screen. **Save** and **Compile** to check for errors. The project is now complete and ready to be downloaded to the HMC hardware.

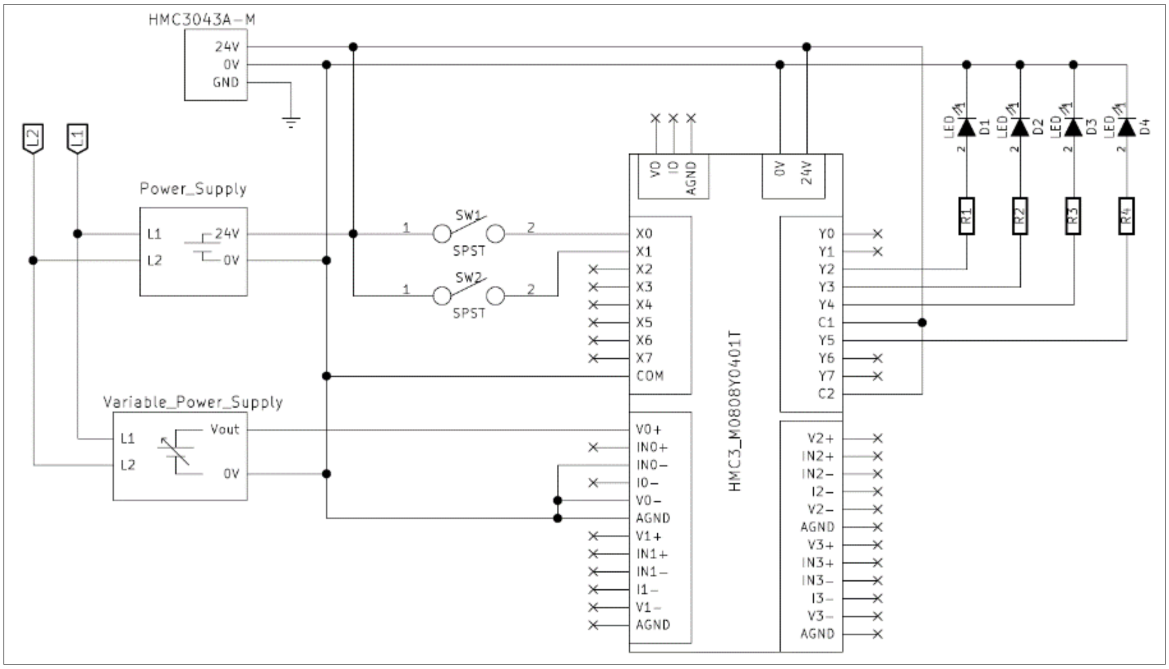
Testing the Project

Test Hardware Setup

Install the HMC3-M0808Y0401T expansion module into the expansion slot of the HMC3102A-M. Connect the HMC3102A-M to a 24VDC power supply. Connect the 0 and 24VDC connectors on the HMC3-M0808Y0401T to the 24VDC power supply.

See the *HMC3000 I/O Guide* and appropriate *Quick Start Guide* for more detail on how to install the expansion module.

A simple test circuit to control the inputs and view the outputs is shown below.



If indicator LEDs or lamps are not available, the continuity function on a digital multi-Meter can be used to test the state of the relay outputs. For outputs Y2, Y3, and Y4, test continuity between C1 and the output pin. For output Y5 test continuity between C2 and Y5. Do not connect 24V to C1 and C2 in this case.

Downloading the Project

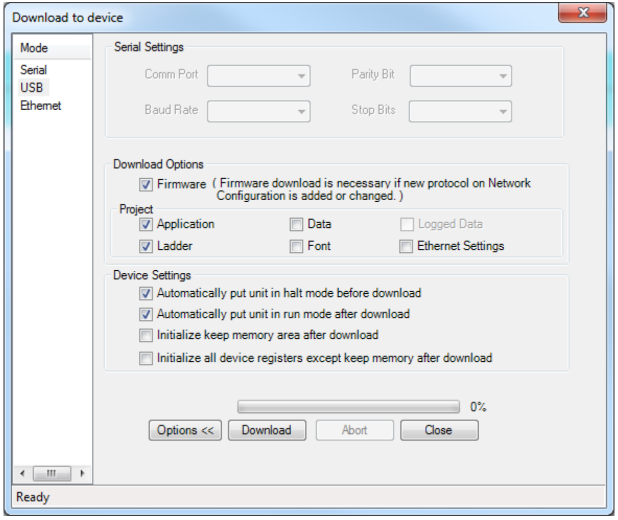
This step assumes that you are using a USB download cable. You may also download to the HMC via Ethernet, but additional steps are required to ensure that your computer and HMC are on the same IP Subnet.

See the *MAPware-7000 Programming Manual* for more detail on downloading via Ethernet.

To download the project.

1. Connect a Micro USB download cable (PN. 7431-0019) between your programming computer and the HMC USB Slave Port.

2. Click **Project > Transfer > Download** to display the **Download to device** dialog box.



Download to device

Mode: Serial, USB, Ethernet

Serial Settings: Comm Port, Parity Bit, Baud Rate, Stop Bits

Download Options

☑ Firmware (Firmware download is necessary if new protocol on Network Configuration is added or changed.)

Project

☑ Application ☐ Data ☐ Logged Data

☑ Ladder ☐ Font ☐ Ethernet Settings

Device Settings

☑ Automatically put unit in halt mode before download

☑ Automatically put unit in run mode after download

☐ Initialize keep memory area after download

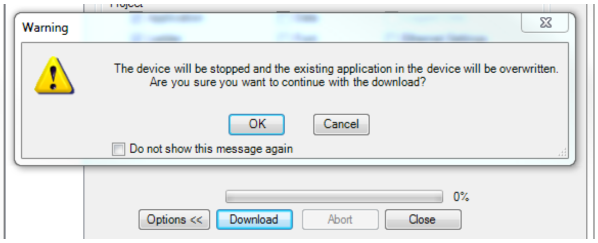
☐ Initialize all device registers except keep memory after download

0%

Ready

   a. Under **Download Options**, check **Firmware**.
   b. Under **Project**, check **Application** and **Ladder**.
   c. Under **Device Settings**, check **Automatically put unit in halt mode** and **Automatically put unit in run mode**.

3. Click the **Download** button.



Warning

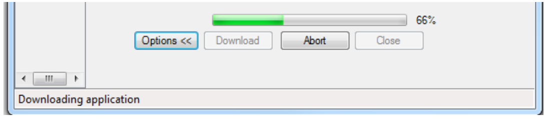
The device will be stopped and the existing application in the device will be overwritten. Are you sure you want to continue with the download?

☐ Do not show this message again

0%

Note: You can check the **Do not show this message again** box to hide this warning message on future downloads during the current session.

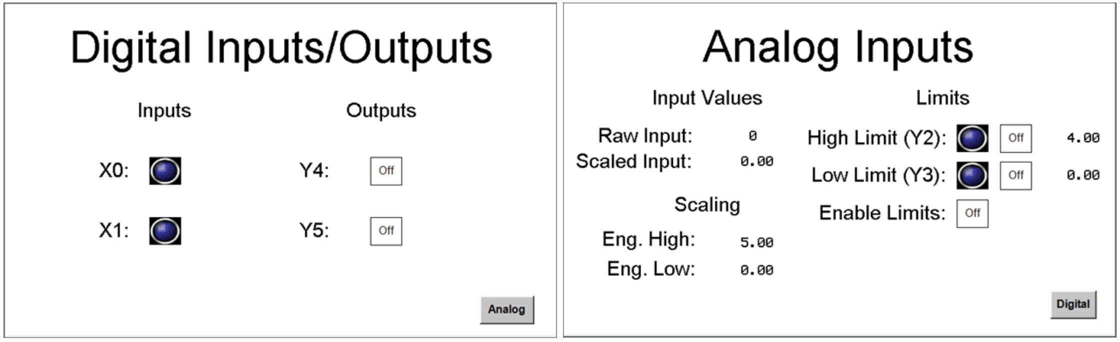
4. Click **OK**. The file will begin downloading.



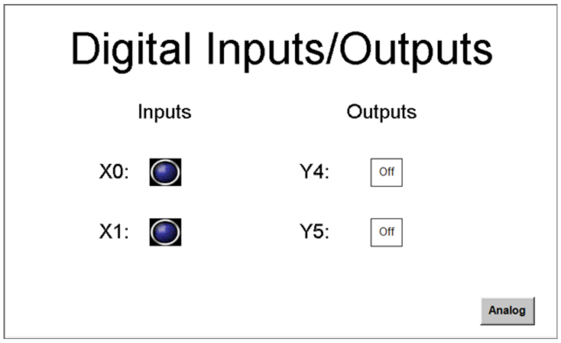
66%

Downloading application

5. When complete, the HMC3102A-M will reinitialize and display the application.



Running the Application



Digital Inputs/Outputs

Inputs

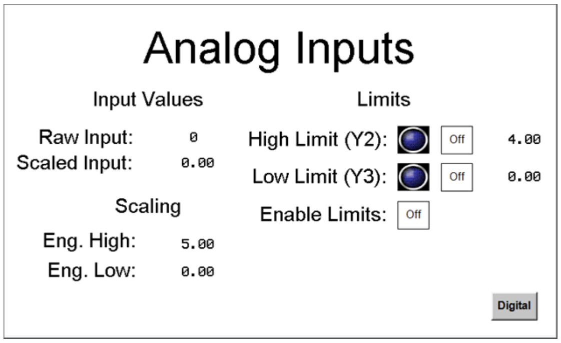
X0: ●

X1: ●

Outputs

Y4: Off

Y5: Off



Analog Inputs

Input Values

Raw Input: 0

Scaled Input: 0.00

Scaling

Eng. High: 5.00

Eng. Low: 0.00

Limits

High Limit (Y2): ● Off 4.00

Low Limit (Y3): ● Off 0.00

Enable Limits: Off

Application test procedure:

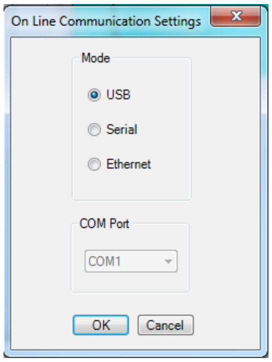
1. Toggle the output toggle bit buttons. Verify the corresponding outputs come on.
2. The bit lamp indicators should change state when the corresponding input switch is closed.
3. On the analog screen, vary the input voltage and watch the raw and scaled values change.
4. Enable the High/Low limits and vary the input voltage to see Y2 and Y3 toggle on and off at each limit.
5. Verify the toggle switches for Y2 & Y3 work when the *Enable Limits* toggle is off, and do nothing when the *Enable Limits* toggle is on.
6. Touch the High and Low limit numeric entry objects to change the values on the fly and verify that Y2 and Y3 toggle on and off at the new limits.
7. Do the same for the Eng. High and Eng. Low to change the output scale of the scaling instruction.

Online Monitoring

The HMI screen provides a window into what is happening in the HMC; however, for a complicated project with many logic blocks, it is often not enough to debug the logic. Online Monitoring allows the programmer to view the logic in real time, and modify data directly in logic blocks. With the project running in the HMC and the USB download cable still attached, MAPware-7000 can be used to monitor logic block execution in real time.

To begin an online monitoring session:

1. Open the editor for **Block 1**
2. Select **Tools > Preferences > Online Communications Mode** to select the communication method used for online monitoring.



On Line Communication Settings

Mode

● USB

○ Serial

○ Ethernet

COM Port

COM1

Note: For Ethernet mode, the IP address of the device must be entered in this window. The IP address here should match what is configured on the **Ethernet** tab of the **Project Properties** dialog. The address of the device is only updated when the **Ethernet Settings** box is checked in the **Download** window. The current IP address is shown on the HMI screen when the device is booting up. Here we assume USB is selected.

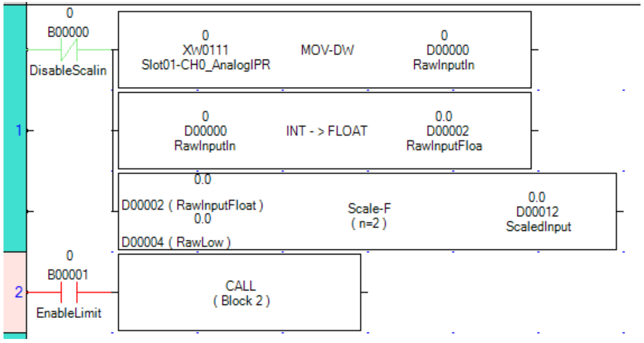
There are three options for initiating an online monitoring session.

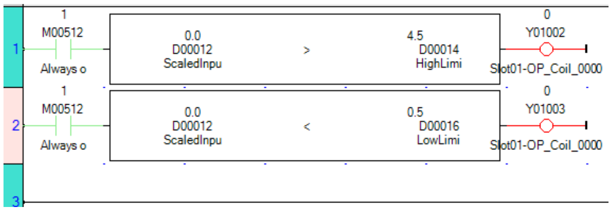
- **With Download** – The currently open project will be downloaded to the device before the online monitoring session begins.
- **Without Upload** – The online session will begin with the project that is open and with the project that is in the device. This assumes that the currently open project is the same as the project on the device.
- **With Upload** – The currently open project will be closed. The project on the device is uploaded and opened, before the online session begins. This is the only option available when no project is opened.

These options are available by selecting **Mode > Go Online** from the menu. Note. clicking the online icon is equivalent to selecting **Without Upload** form the **Go Online** menu option.

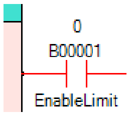
3. Because we already have our project open, select **Mode > Online > Without Upload** from the menu bar.

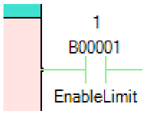
The logic block will be shown with the current values above the operands.



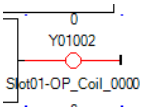


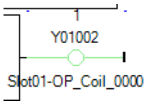
Contacts and coils are color coded according to their current state. Contacts are red when open (off), and green when closed (on).



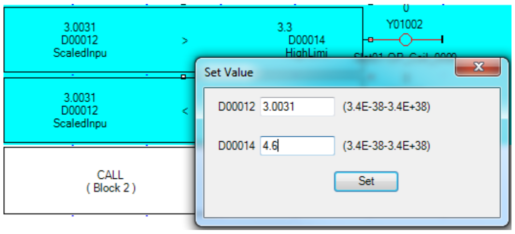


Output coils are green when energized (on), and red when not energized (off).





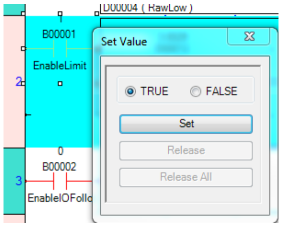
Values can be changed from within the logic block editor. Double click the contact or operand to be changed, then enter the state or value as needed.



Set Value

D00012 3.0031 (3.4E-38-3.4E+38)

D00014 4.6 (3.4E-38-3.4E+38)



Set Value

● TRUE ○ FALSE

For more information on using online monitoring or debugger mode, refer the *MAPware-7000 Ladder Logic Guide*.

Review

Before moving on to IEC 61131-3 programming, let’s review what has been learned in this section. Consider what has been accomplished:

- A new project was created and configured for an HMC model, including setting up and configuring an expansion module.
- The tag database was used to create new tags using internal HMC memory as well as I/O module registers.
- Two screens were created using data display, data entry, and navigation objects.
- The project showed how to create a simple ladder logic program using Logic Blocks, including a subroutine block.
- The project was saved, compiled, and downloaded into the HMC.
- Online monitoring mode was used to view the application as it executed.

The next section of this guide will explore the IEC 61131-3 programming environment.

Creating a Sample Project using IEC 61131-3

Introduction

This section guides you through the steps needed to create and run a simple IEC 61131-3 project. It will use a common engineering task, mapping a value from one scale to another, to demonstrate how the features of the IEC editor can be used to develop solutions. The sample project demonstrates:

- Use of the **Structured Text (ST)**, **Ladder Diagram (LD)** and **Function Block Diagram (FBD)** editors
- How to define a **User Defined Function Block (UDFB)**
- The difference between a **Function Block** and a **Function Block Instance**
- Passing an instance of a **Function Block** as a parameter to another **Function Block**

Like the previous Native Ladder Sample Project, this sample project will map an input value linearly to produce an output value on a different scale, only this time, we will create the logic that does the scaling. We will scale values from Celsius to Fahrenheit and from a raw input value to an engineering value (i.e. from a 12 bit raw input to a voltage). The solution should be general enough so that the Function Blocks created can be re-used for any linear scaling operation. We also want the calculation to be as efficient as possible. The task of scaling a number can be broken down into two parts.

1. Calculate the slope and offset
2. For a given input calculate the output

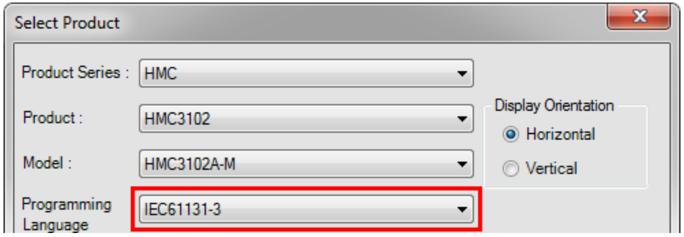
We could do both of these operations in one function block, but that would mean that every time the value is scaled, the slope and offset are recalculated. Instead we will create two function blocks; one to calculate the slope and offset given maximum and minimum values, and one to actually do the scaling as the input changes.

Creating **User Defined Function Blocks** to accomplish the task eliminates the need for multiple logic blocks that do the same thing, making the project easier to maintain. Edits can be made in one place and take effect throughout the project.

Create a New Project

We will use the same HMC3102A-M that we used in the previous Native Ladder sample project. Once a project has been created in either Native Ladder or IEC 61131-3, it cannot be converted to use the other programming language, so we will create a new project for this example.

1. To create the project, select **Project > New**. The **Select Product** window is displayed.
2. Select the **Product Series**, **Product**, and **Model**.
3. Leave the **Display Orientation** as **Horizontal**.
4. Native Ladder is the default **Programming Language** and for this sample project must be changed. Click the drop down to select **IEC61131-3**.



Select Product

Product Series : HMC

Product : HMC3102

Model : HMC3102A-M

Programming Language: IEC61131-3

Display Orientation

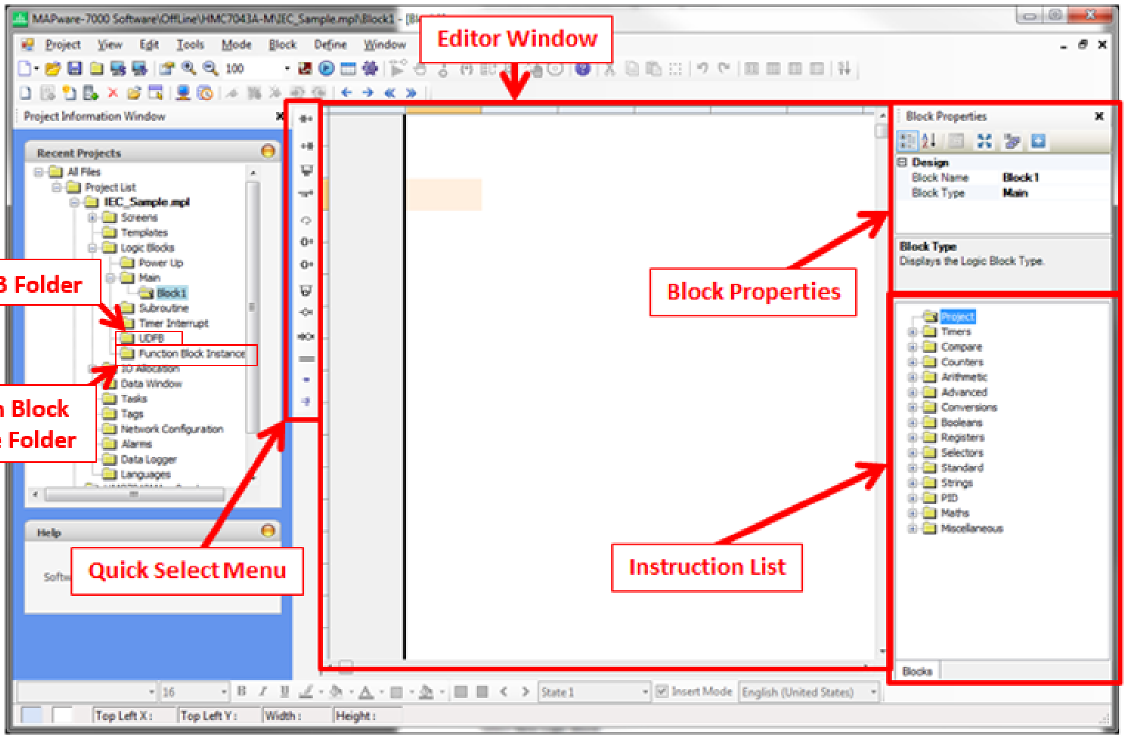
● Horizontal

○ Vertical

5. Click **OK**. A new project is created with a default name.
6. Select **Project > Save** to save the project with a unique name.

The Editor Window

Let’s take a quick tour of some of the aspects of MAPware-7000 that are unique to an IEC 61131-3 project. Clicking the folder for **Block1** in the **Project Information Window** displays the editor for that block. The default logic block uses the Ladder Diagram editor, so the Ladder Diagram editor is displayed when the **Block1** folder is selected.



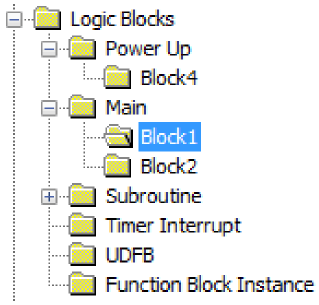
- **Block Properties** – This area displays the execution type and name of the logic block.
- **Instruction List** – Lists all the available instructions for the project. Instructions are categorized according to functionality. Expand the node for a given category to see individual instructions. To add a particular instruction, click and drag it into the editor window. Subroutines and UDFBs (Function Blocks that users create) will appear under the Project folder of the Instruction List once they are defined. Information on how to configure and use a given function block is available in the help file. To access the help file entry for any of the blocks in the Instruction List, simply double-click the instruction.
- **Editor Window** – This is where the logic is defined. It displays the graphical or text representation of the logic program.
- **Quick Select Menu** – The options available on this menu depend on the editor in use but provide quick access to common program elements. Click the location in the editor where you want to place the element then click the element in the quick select menu to place it in the editor.
- **UDFB Folder** – This folder contains the definitions for User Defined Function Blocks further described below. To add or edit a UDFB, click the block name in this folder. To use a UDFB in another logic block, select it in the Project folder of the Instruction List.
- **Function Block Instance Folder** – This folder contains a list of all the Function Block *Instances* in the project further described below. Each block can have multiple instances, and each instance

will have its own private set of data to work with. Thus, one type of function block can be utilized for multiple purposes in the project.

IEC Specific Logic Blocks and Execution Style

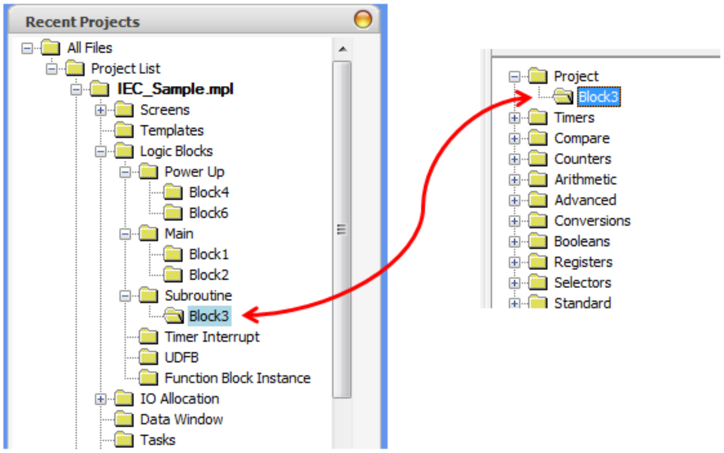
In addition to the Power Up, Main, Subroutine, and Timer Interrupt execution styles available in the Native Ladder Editor, the IEC programming mode also allows you to create User Defined Function Blocks and Function Block Instances.

The current **Execution Style** of a logic block is indicated by the block's location within the **Logic Blocks** folder of the project tree.



Subroutines

In IEC mode, when a block is created as a **Subroutine** block, it will appear in the **Instruction List** under the **Project** folder as a function block that can be used within logic blocks.



User Defined Function Block (UDFB)

A User Defined Function Block (**UDFB**) operates similarly to a subroutine. It is a logic procedure defined by the user that can be executed as a component in another block. The **UDFB** also allows the user to define input and output parameters, making the block easy to reuse throughout the project. Once defined, the **UDFB** is also available to select from the **Instruction List** under the **Project** folder. We will create two UDFBs in this sample project.

Function Block Instances

One of the major advantages of the IEC61131-3 editor is the ability to modularize and reuse functionality through the use of Function Blocks. Once the logic in a UDFB or Subroutine is defined, it can be used to create Function Block Instances.

- A Function Block Instance contains all of the logic defined in the Function Block as well as its own set of data to operate on.

- The Function Block can be thought of as a cookie cutter and the Function Block Instance is the cookie that the cutter creates.
- Multiple instances of the same block can be defined and each will have its *own set of data to work with*. The same block can be used for multiple purposes.
- Function Block Instances can be passed as parameters to other Function Blocks.
- Instances can be defined using the built in Function Blocks or User Defined Function Blocks.

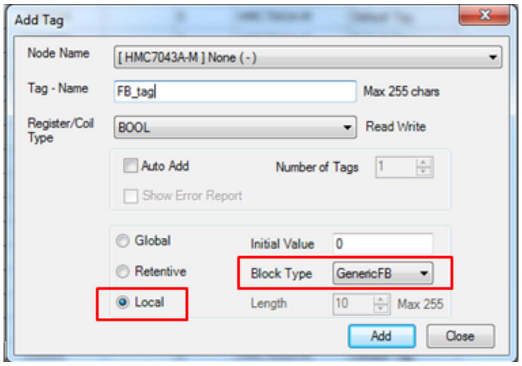
The **Function Block Instance** folder, in the **Project Information Window**, is used to create Function Block Instances and contains a list of all the instances in the project. We will create multiple Function Block Instances in our sample project.

Add Tags to the Project

Tags are handled somewhat differently in IEC 61131-3 projects vs. Native Ladder projects. In IEC mode projects, tags are not assigned explicit addresses. Instead they are given a name and a type, and MAPware-7000 is responsible for allocating and tracking a memory address for the tag.

Tag Scope

In addition to Global scope tags, tags can be associated with a Function Block Instance. UDFBs can have input, output, and internal tags. When an instance of the UDFB or built in function block is added to the project, it will have a copy of each of the tags. Internal tags can be defined from the Tag Database by setting the **Scope** to **Local** and the **Block Type** to the Function Block Instance the tag is associated with.



Add Tag

Node Name: [HMC7043A-M] None (-)

Tag - Name: FB_tag (Max 255 chars)

Register/Coil Type: BOOL (Read Write)

□ Auto Add   Number of Tags: 1

□ Show Error Report

○ Global   Initial Value: 0

○ Retentive   Block Type: GenericFB

● Local   Length: 10 (Max 255)

Input and output tags are defined by right-clicking the UDFB's folder in the project tree and selecting the **Edit Parameters** option. Once defined, Function Block Tags will appear in the Tag Database as <Function Block Name>\<Tag Name>. For example, *GenericFB\FB_tag* is a tag internal to the *GenericFB* function block named *FB_tag*.

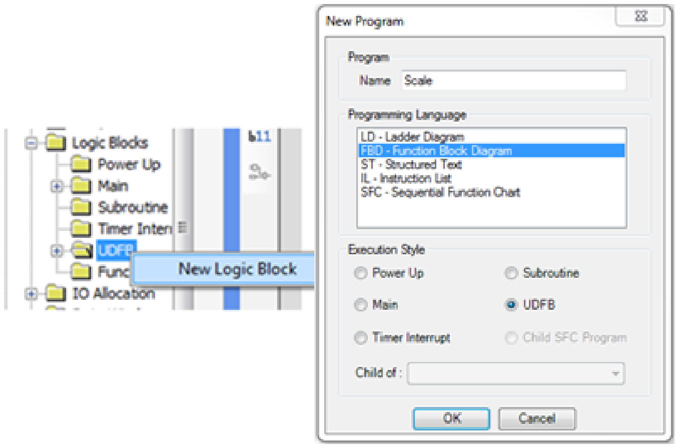
For more information on creating and using tags, refer to the *MAPware-7000 Programming Manual*.

1. Open the **Tag Database** by clicking the **Tags** folder in the **Project Information Window**.
2. Use the **Add Tag** window to create the following tags.

Tag Name	Type	Scope	Description
Temp1C	INT	Global	Simulated Temp input 1 [C]
Temp2C	INT	Global	Simulated Temp input 2 [C]
Temp1F	REAL	Global	Scaled Temp output 1 [F]
Temp2F	REAL	Global	Scaled Temp output 2 [F]
RawInput	INT	Global	Simulated IO card input
Voltage	REAL	Global	Input scaled to a voltage value

Logic Blocks

Create a User Defined Function Block (UDFB)

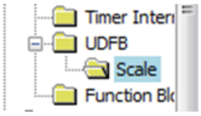


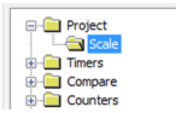
Name Scale

○ Power Up ○ Subroutine
○ Main ● UDFB
○ Timer Interrupt ○ Child SFC Program

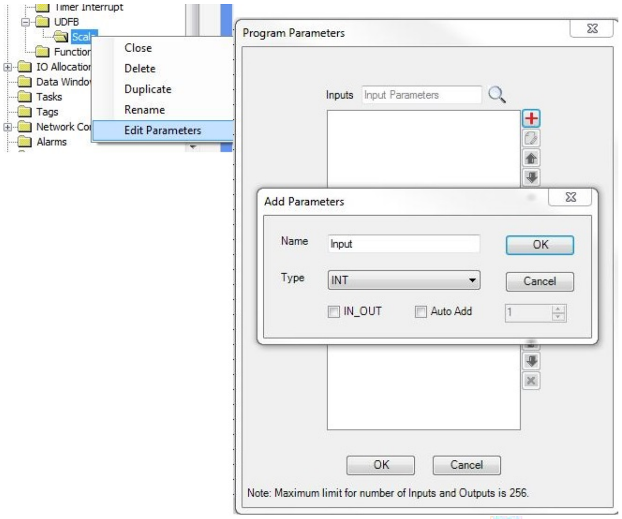
Next we want to create our own User Defined Function Block (UDFB). This block performs the scale operation. We will use the Function Block Diagram (FBD) editor to define this block.

1. Expand the **Logic Blocks** folder in the **Project Information Window**
2. Right-click the **UDFB** subfolder and select **New Logic Block**
3. In the **New Program** window name the function *Scale* and select **FBD – Function Block Diagram** as the **Programming Language** and **UDFB** as the **Execution Style**
4. Click **OK** to create the Logic Block
5. A new block called *Scale* appears in the **UDFB** folder of the **Project Information Window**, and a new instruction called *Scale* appears in the **Project** folder of the **Instruction List**.





6. Before creating the logic for this block we need to define the inputs and outputs so that other logic blocks can pass data to it. Right-click the *Scale* folder in the Project Information Window and select **Edit Parameters**.



Name Input

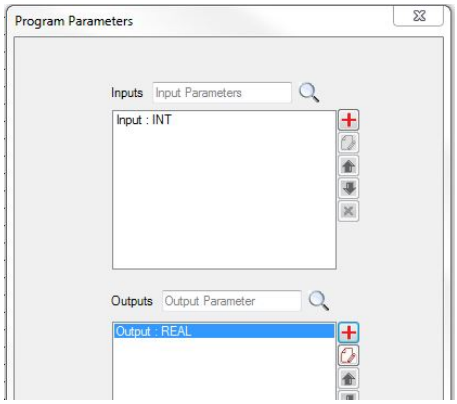
Type INT

□ IN_OUT □ Auto Add 1

Note: Maximum limit for number of Inputs and Outputs is 256.

7. In the **Program Parameters** window, click + ‘Add Input Parameter’ under **Inputs**.

8. The **Add Parameters** pop-up window allows you to specify a name and data type for a new input parameter. This block contains only one input parameter imaginatively named *Input*. Set the type to **INT** then click **OK**.
9. Next click ⊞ 'Add Output Parameter' under **Outputs** to create the output. It is named *Output* and has type **REAL**.
10. Once complete, the **Program Parameters** window should appear as shown below. Click **OK** to create the Parameters.



Program Parameters

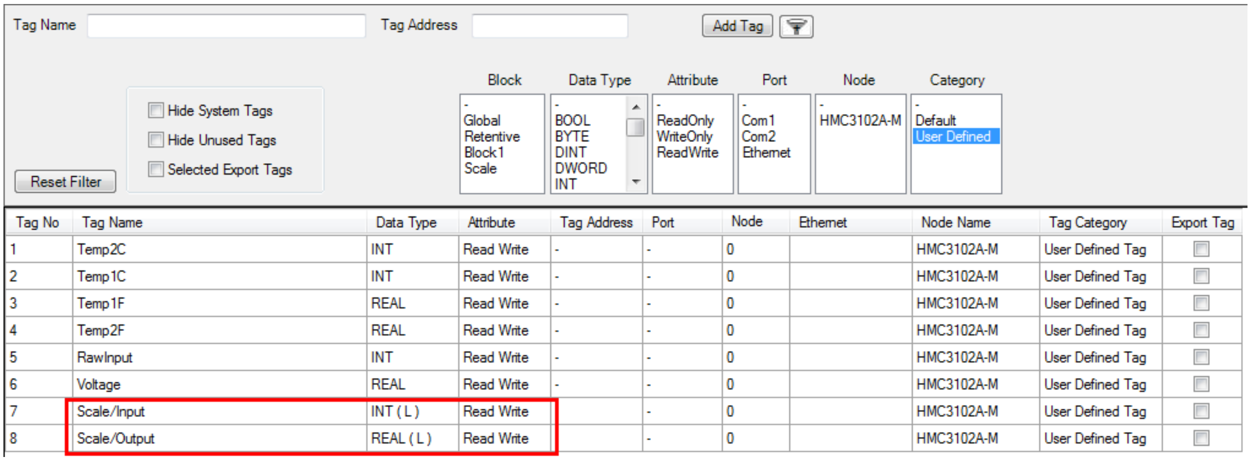
Inputs: Input Parameters

Input : INT

Outputs: Output Parameter

Output : REAL

Both tags should now appear in the tag database with the Function Block/Local Tag format we talked about earlier.



Tag Name

Tag Address

Add Tag

□ Hide System Tags
□ Hide Unused Tags
□ Selected Export Tags

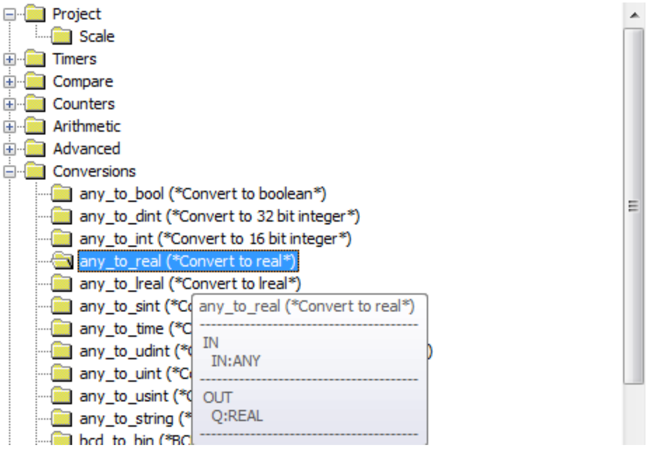
Reset Filter

Block	Data Type	Attribute	Port	Node	Category
- Global Retentive Block1 Scale	- BOOL BYTE DINT DWORD INT	- ReadOnly WriteOnly ReadWrite	- Com1 Com2 Ethernet	- HMC3102A-M	- Default User Defined

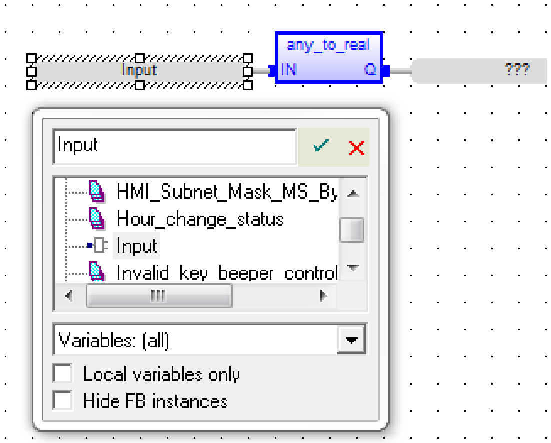
Tag No	Tag Name	Data Type	Attribute	Tag Address	Port	Node	Ethernet	Node Name	Tag Category	Export Tag
1	Temp2C	INT	Read Write	-	-	0		HMC3102A-M	User Defined Tag	□
2	Temp1C	INT	Read Write	-	-	0		HMC3102A-M	User Defined Tag	□
3	Temp1F	REAL	Read Write	-	-	0		HMC3102A-M	User Defined Tag	□
4	Temp2F	REAL	Read Write	-	-	0		HMC3102A-M	User Defined Tag	□
5	RawInput	INT	Read Write	-	-	0		HMC3102A-M	User Defined Tag	□
6	Voltage	REAL	Read Write	-	-	0		HMC3102A-M	User Defined Tag	□
7	Scale/Input	INT (L)	Read Write		-	0		HMC3102A-M	User Defined Tag	□
8	Scale/Output	REAL (L)	Read Write		-	0		HMC3102A-M	User Defined Tag	□

Logic can now be entered into the block to define its functionality. The first thing we want the *Scale* logic block to do is convert the input to floating point (real) format.

11. In the **Instruction List** at the bottom right of the editor window, expand the **Conversions** folder and locate the **any_to_real** instruction. Click the instruction and drag it into the editor window.


Project
Scale
Timers
Compare
Counters
Arithmetic
Advanced
Conversions
any_to_bool (*Convert to boolean*)
any_to_dint (*Convert to 32 bit integer*)
any_to_int (*Convert to 16 bit integer*)
any_to_real (*Convert to real*)
any_to_lreal (*Convert to lreal*)
any_to_sint (*C
any_to_time (*C
any_to_udint (*C
any_to_uint (*C
any_to_usint (*C
any_to_string (*
bcd_to_bin (*BC
any_to_real (*Convert to real*)
IN
IN:ANY
OUT
Q:REAL

12. The input to this instruction will be the UDFB's Input parameter. Double-click the instruction's input area and type *Input* into the pop up box.

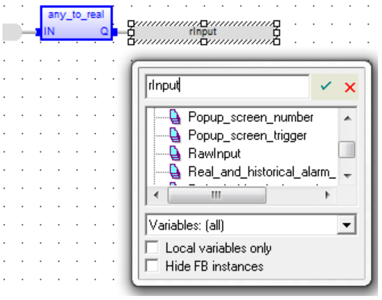


Input

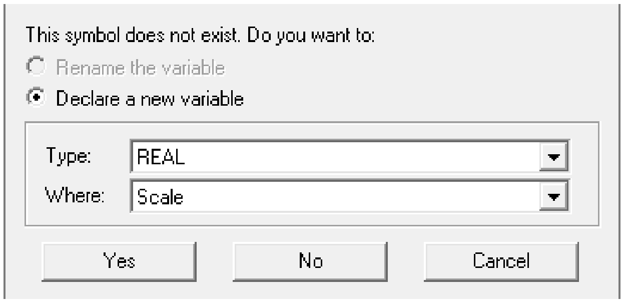
Note: Notice that because *Input* is an input parameter for this UDFB, it shows up in the context list with an input icon ▪⊏.

13. For the output, we want to define a new tag that will be local to the *Scale* function block. Double-click the output area of the **any_to_real** instruction and type *rInput* into the pop up and hit enter.

14. Because a tag with this name does not exist yet, MAPware-7000 will display a dialog box that can be used to define the tag.



rInput


This symbol does not exist. Do you want to:

○ Rename the variable

● Declare a new variable

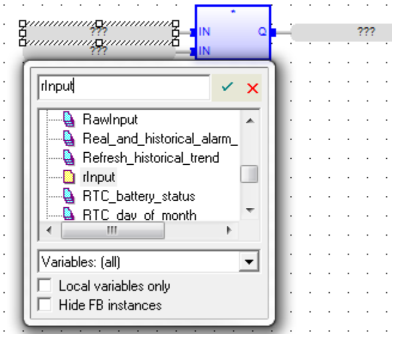
Type: REAL

Where: Scale

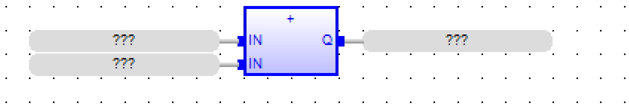
Set the type to **REAL** and **Where** to *Scale*. This sets the scope of the variable to the Scale function block.

15. Click **Yes** to create the new variable.

16. Now that the input is in the correct format, all we have to do is multiply by the slope and add the offset. Expand the **Arithmetic** folder in the **Instruction List** and drag a *** (*Multiply*)** instruction to the editor window.
17. The first input here will be the *rInput* tag created in the last step. Double-click in the first input and type *rInput* into the pop up window. Notice that *rInput* is now in the list as a local variable.

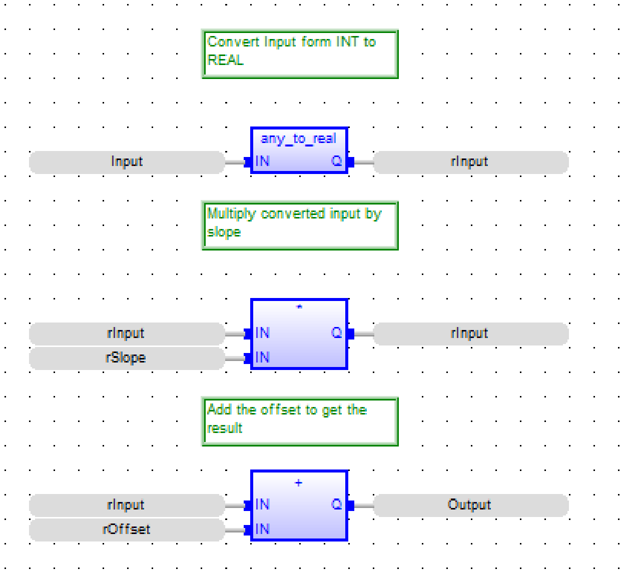


18. The second input will be a local variable called *rSlope*. This tag doesn't exist yet, so click the second input and type *rSlope* to create the new tag.
19. Set **Type** to **REAL** and **Where** to *Scale* using the context window as described above.
20. We will write the result of the multiplication back to the *rInput* variable. Select *rInput* for the output (**Q**) of the instruction.
21. The last step is to add the offset. Drag an **+ (*Addition*)** instruction from the **Instruction List** into the editor window.



22. The first input for this instruction will again be the *rInput* variable. Double-click the first input and select *rInput* from the tag list.
23. The second input will be a new local variable called *rOffset*. Type *rOffset* in the box for the second input and create a new tag. Make sure the **Type** is **REAL** and **Where** is set to *Scale*.
24. Finally, the output will be the previously created output parameter of the function block. Select *Output* from the menu for the output (**Q**) tag. Notice that it will have an output icon (▫) in the tag list.

25. Here is what the *Scale* function block should look like when complete (some comments were added for clarity).


Convert Input form INT to REAL
Input
any_to_real
IN
Q
rInput
Multiply converted input by slope
rInput
rSlope
*
IN
IN
Q
rInput
Add the offset to get the result
rInput
rOffset
+
IN
IN
Q
Output

This is a good time to compile (**Project > Compile**) and save the project to make sure there are no errors before moving on.

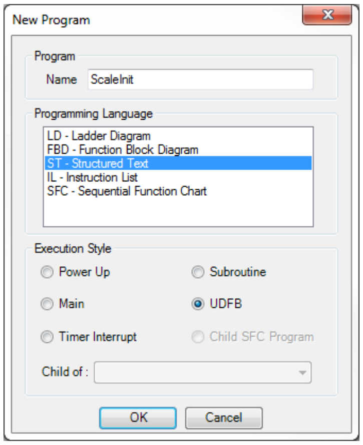
Create a Second UDFB to Initialize Instances of the Scale Function Block

Before we can use the Scale function block, we have to initialize the offset and slope parameters. These are calculated from maximum and minimum values for the input value and the scaled value. We will create a new **UDFB** to do this calculation and initialize the function block instances. This new block will use the **ST - Structured Text** editor.

Structured Text is a text based programming language in which program instructions are entered as discrete statements in a text source file. This programming method is similar to other high level programming languages such as C or Visual Basic.

The basic unit of a structured text program is a statement. A statement is an instruction for the processor to perform some set of actions. Structured Text programs are simply lists of statements.

1. Right-click the **UDFB** folder in the project information window and select **New Logic Block**.
2. Name this block *ScaleInit*, and select **ST - Structured Text** for the **Programming Language**.
3. Click **OK** to create the block.


New Program

Program

Name: ScaleInit

Programming Language: ST - Structured Text

Execution Style

○ Power Up ○ Subroutine

○ Main ● UDFB

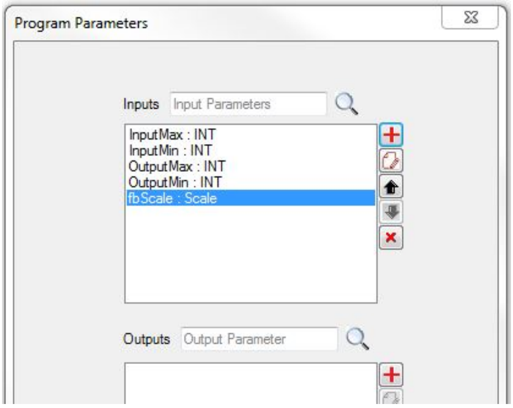
○ Timer Interrupt ○ Child SFC Program

Child of :

4. Next, we will define the input parameters for the *ScaleInit* function block. Right-click the block's folder in the **Project Information Window** and select **Edit Parameters** from the context menu to open the **Program Parameters** window.
5. This block will have no outputs. Enter the following inputs.

Parameter Name	Type	Description
InputMax	INT	Maximum value for the input
InputMin	INT	Minimum input value
OutputMax	INT	Maximum value of output
OutputMin	INT	Minimum value of input
fbScale	Scale	Function block instance to be initialized

Note: The last parameter is of type *Scale*. This means that a *Scale* function block instance will be passed into the function block. The function block can then operate on the data specific to that function block instance. The input parameters should look something like this when finished.



InputMax : INT
InputMin : INT
OutputMax : INT
OutputMin : INT
fbScale : Scale

Now we can start entering code for the *ScaleInit* function block. This block will contain several local variables to perform the calculation. In the **Structured Text Editor**, when enter is pressed at the end of a statement or line of code, the editor will validate all of the parameters in that line. If it finds parameters that don't exist, a dialog box will pop up allowing new variables to be defined, just as in the Function Block Diagram editor. Below is a table of all the local variables to be defined.

Parameter Name	Type	Scope	Description
r*InputMax*	REAL	ScaleInit	Floating point version of *InputMax*
r*InputMin*	REAL	ScaleInit	Floating point version of *InputMin*
r*OutputMax*	REAL	ScaleInit	Floating point version of *OutputMax*
r*OutputMin*	REAL	ScaleInit	Floating point version of *OutputMin*

Below is the code to be entered into the *ScaleInit* function block. The first four lines use the **any_to_real** function to convert our input tags into real numbers. Then next line calculates the slope from the given range, and places it into the internal *rSlope* variable of the *fbScale* function block instance. The last line calculates the offset and places it in the internal *rOffset* tag of *fbScale*.

```
// Convert inputs to floating point numbers
rInputMax := any_to_real (InputMax);
rInputMin := any_to_real (InputMin);
rOutputMax := any_to_real (OutputMax);
rOutputMin := any_to_real (OutputMin);

// Calculate slope
fbScale.rSlope := ( rOutputMax - rOutputMin ) / ( rInputMax - rInputMin );

// Calculate offset
fbScale.rOffset := rOutputMax - ( fbScale.rSlope * rInputMax);
```

Note: The green lines beginning with "//" are comments and provide context, but are optional and have no effect on the code.

6. Copy the first statement above, `rInputMax := any_to_real (InputMax);`, and paste it into the **Structured Text Editor**. Hit the **Enter** button on your keyboard.
7. A popup window to define r*InputMax* appears. Set the **Type** to **REAL** and **Where** to *ScaleInit*. Click **Yes**.
8. Copy the rest of the code into the **Structured Text Editor** one line/statement at a time, pressing **Enter** after each line to define the local tags. Each tag should be of type **REAL** and local to *ScaleInit*.
9. After all the code has been entered, compile (**Project > Compile**) and save the project to verify that there are no errors, and fix any typos as needed.

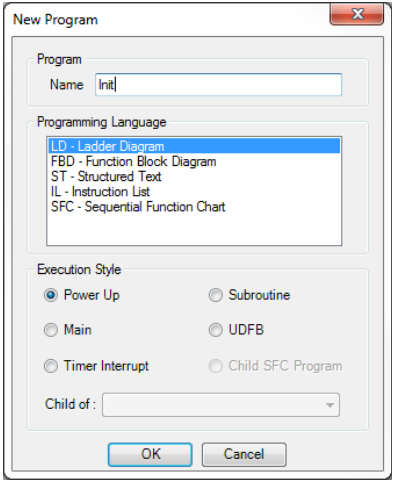
Note: If you copy and paste the code in its entirety from here into the editor, tags will not be created for the internal variables because the code was not entered one statement at a time. You must manually enter each tag into the database in this case.

For more information on Structured Text commands and formatting, see the *MAPware-7000 IEC 61131 Programming Guide*.

Use the UDFBs in Ladder Diagram Blocks

Now that the function blocks have been created, they can be put to use in other parts of the project. We want to use the *ScaleInit* function block in a power up routine to initialize two *Scale* function block instances.

1. Right-click the **Power Up** folder and select **New Logic Block**. This block will be called *Init* and will be a **LD - Ladder Diagram** block.



New Program

Program

Name: Init

Programming Language

- LD - Ladder Diagram
- FBD - Function Block Diagram
- ST - Structured Text
- IL - Instruction List
- SFC - Sequential Function Chart

Execution Style

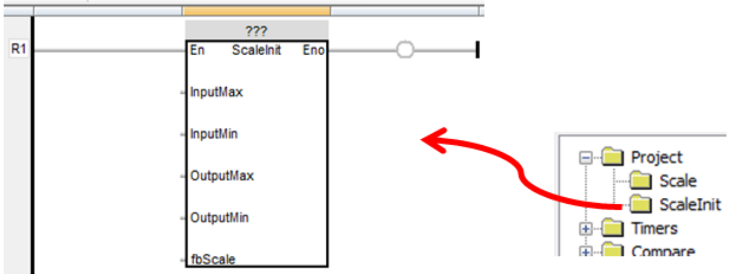
● Power Up ○ Subroutine

○ Main ○ UDFB

○ Timer Interrupt ○ Child SFC Program

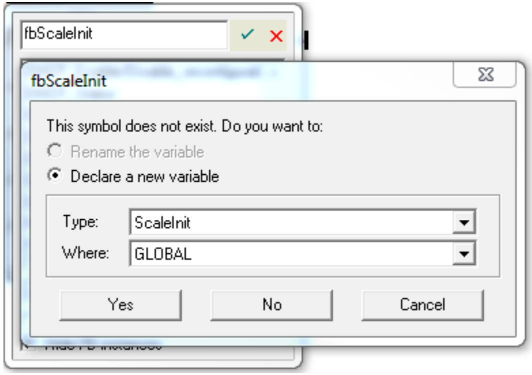
Child of :

2. Expand the **Project** node in the **Instruction List**. The *Scale* and *ScaleInit* function blocks should appear there.
3. Click the *ScaleInit* instruction and drag it to the top rung of the **Ladder Diagram**.



Notice the '???' at the top of the block. This indicates the function block instance has not been selected.

4. Double-click the question marks and enter *fbScaleInit* in the tag selection window. Click the Accept icon ✓ and declare the new variable as type *ScaleInit* and **Global** scope.



fbScaleInit ✓ ✕

fbScaleInit

This symbol does not exist. Do you want to:

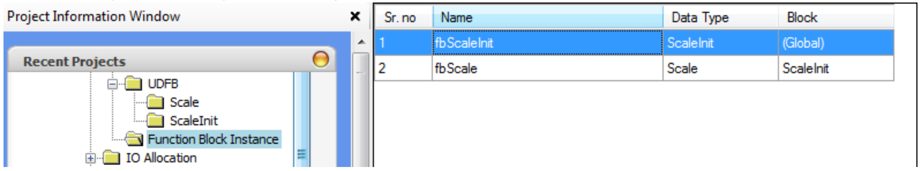
○ Rename the variable

● Declare a new variable

Type: ScaleInit

Where: GLOBAL

This defines a new instance of the *ScaleInit* Function Block called *fbScaleInit*. This instance will now be listed in the **Function Block Instance** folder in the **Project Information Window** along with the previously created *fbScale* instance of *Scale* defined earlier, local to *ScaleInit*.

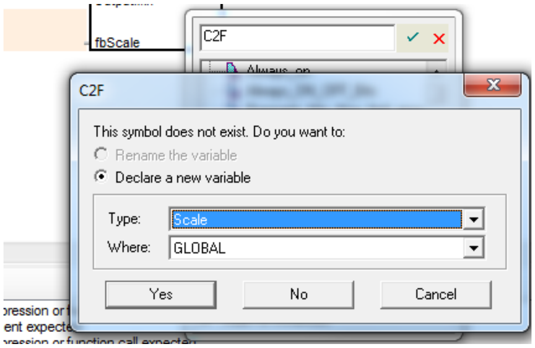


Sr. no	Name	Data Type	Block
1	fbScaleInit	ScaleInit	(Global)
2	fbScale	Scale	ScaleInit

5. Next we will specify the input parameters. For this sample, we will use literal number values for the maxes and minimums. Double click to the left of each of the first four inputs and type in the following values.

Input Parameter Name	Value	Description
InputMax	100	Celsius input maximum
InputMin	0	Celsius input minimum
OutputMax	212	Fahrenheit equivalent to 100°C (Max °F)
OutputMin	32	Fahrenheit equivalent to 0°C (Min °F)

6. The last parameter is the *Scale* function block instance to be initialized. In this case it will be a new instance called *C2F* (Celsius to Fahrenheit). Double-click the *fbScale* input and type *C2F* into the selection box.
7. Set the **Type** to *Scale* and **Where** to **GLOBAL** in the popup window and click **Yes** to create the *C2F* instance.


C2F

This symbol does not exist. Do you want to:

○ Rename the variable

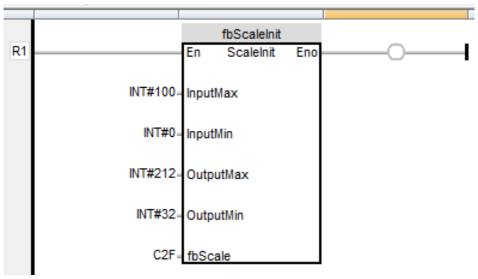
● Declare a new variable

Type: Scale

Where: GLOBAL

The *C2F* instance is created, added to the **Function Block Instance** Folder and selected as the *fbScale* input parameter.

This is what the instruction should look like now.



Next we want to initialize a different Scale function block instance called *Input2Volts*. This instance will be used to convert from a raw input to a voltage. The process is the same as above, but with a different function block instance as the input parameter and with different maximum and minimum values.

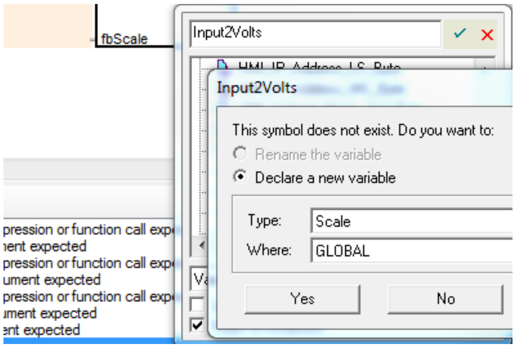
8. We will use the same instance of the *ScaleInit* function block. Drag a new *ScaleInit* instruction to rung 2 from the **Instruction List**.
9. Click the '???' above the block and again enter *fbScaleInit* in the popup window. This time, the *fbScaleInit* instance will appear in the variable list since it was already defined.

10. Enter the following values for the input parameters.

Input Parameter Name	Value	Description
InputMax	65535	16 bit analog maximum input
InputMin	0	16 bit analog minimum input
OutputMax	5	0-5V maximum
OutputMin	0	0-5V minimum

This sets up the *Input2Volts* function block to convert from a 16 bit input to a 0 to 5 V output.

11. Enter *Input2Volts* for the *fbScale* input.
12. The *Input2Volts* function block instance will need to be defined. Set the **Type** to *Scale* and **Where** to **GLOBAL** as before.


fbScale
Input2Volts
pression or function call exp
ent expected
pression or function call exp
ument expected
pression or function call exp
ment expected
ent expected

Input2Volts

This symbol does not exist. Do you want to:

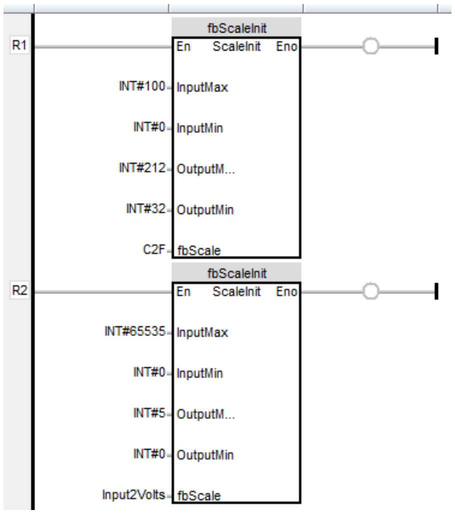
○ Rename the variable

● Declare a new variable

Type: Scale

Where: GLOBAL

This will add a new *Scale* function block instance called *Input2Volts* to the **Function Block Instance** folder. Here is what the *Init* function block should look like when complete:

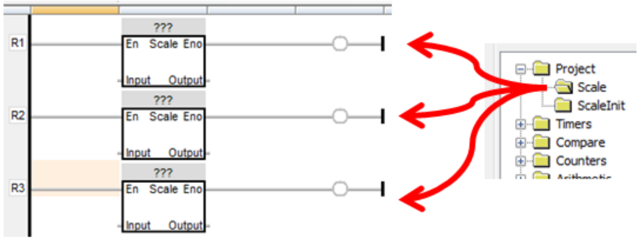


Define the Main Routine Block 1

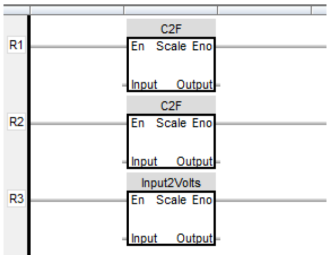
The final piece of logic in the sample project uses *C2F* and *Input2Volts* instances to continuously convert our simulated inputs to the desired output values as the inputs change over time. We can use the ladder diagram block automatically created by MAPware-7000 to do this.

1. Click the **Block1** folder under **Logic Blocks** / **Main** in the **Project Information Window** to edit the block.

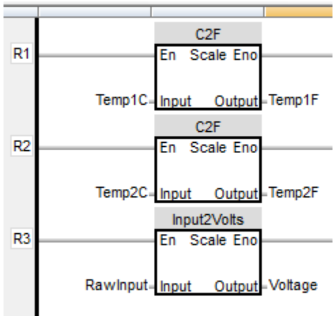
2. Click and drag three *Scale* blocks from the **Instruction List** into the editor.



3. Next click the '???' in each block to specify the function block instance to be used. The first two will use *C2F* and the last one will use *Input2Volts*.



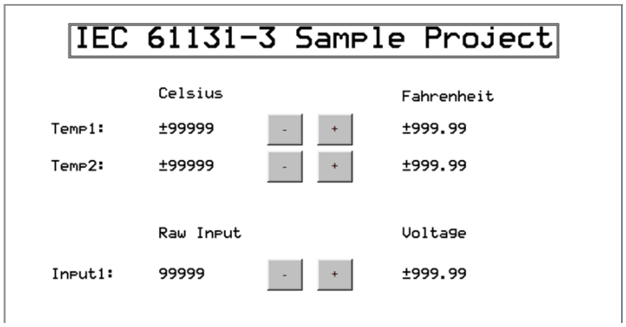
4. Specify the input and output parameters for each instruction. We will use the tags created at the beginning of this section.
   - Rung 1 converts *Temp1C* to *Temp1F*
   - Rung 2 converts *Temp2C* to *Temp2F*
   - Rung 3 converts *RawInput* to *Voltage*.



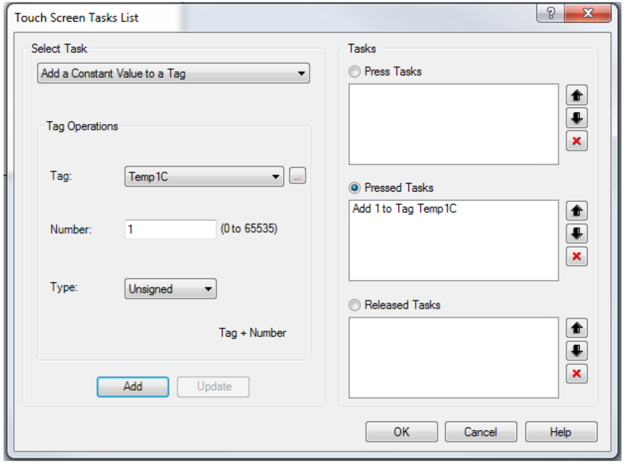
5. That's it for the logic. Check that the project compile (**Project > Compile**) and save the project to verify that there are no errors. Fix any typos as needed.

Create Screen Objects

We will create a very simple screen to control the inputs and observe how the outputs change. This section will not go into great detail on creating the objects needed, for they were covered in the previous Native Ladder section. The screen should eventually look something like this:



1. Navigate to **Screen1** under the **Base Screens** folder in the **Project Information Window**.
2. Create the **Text** object labels shown above [*Celsius*, *Fahrenheit*, *Raw Input*, *Voltage*, *Temp1:*, *Temp2:*, and *Input1:*].
3. Create three **Numeric Entry** objects (**Draw > Input Objects > Data Entry > Numeric Entry**), to display and write to the three inputs; *Temp1C, Temp2C* and *RawInput*.
4. Change the **Font** for these objects to **10 x 14** to make them easier to read, and select the appropriate tag in the **Tag Name** property.
5. For the *Temp1C* and *Temp2C* objects set the **Data Type** format to **Signed [-32768 To 32767]** to allow negative temperatures to be displayed correctly.
6. Next we will use **Multi-Task Single-state** buttons (**Draw > Buttons > Multi-Task Single-state**) to create the increment and decrement buttons that control the input tags. Place a button next to the first **Numeric Entry** object.
7. For the increment button, change the **On Text** property to "+".
8. Click the **Tasks** property, then on the ... button to configure a task for the button.
9. Before selecting the task, click the **Pressed Task** radio button. This means that the value will increment *while* the button is pressed.
10. Select the **Add a Constant Value to Tag** task from the **Select Task** list. Select *Temp1C* for the **Tag** and enter 1 for the **Number** to add.
11. Click the **Add** button. The task should appear under **Pressed Tasks**.


Touch Screen Tasks List

Select Task: Add a Constant Value to a Tag

Tag Operations

Tag: Temp1C

Number: 1 (0 to 65535)

Type: Unsigned

Tag + Number

Tasks

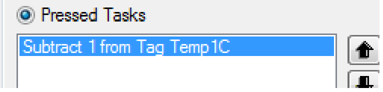
○ Press Tasks

● Pressed Tasks

Add 1 to Tag Temp1C

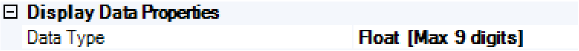
○ Released Tasks

12. Click **OK** to save the task.
13. Make a copy of the increment button to use as the starting point for the decrement button.
14. Change the **On Text** to "-".
15. Change the **Task** to a **Subtract a Constant Value from Tag** task that subtracts 1 form *Temp1C*.


● Pressed Tasks

Subtract 1 from Tag Temp1C

16. Once you have increment and decrement buttons controlling *Temp1C,* make copies of them to control *Temp2C and RawInput*. You can update the task with the correct tag by clicking the task in the **Pressed Tasks** list. When changing the tag, make sure the number is re-entered, and don't forget to click **Update** and **OK** to save the changes. For *RawInput,* you may want to change the add/subtract numbers to 10 or 100 so that the value changes faster.
17. Finally add three **Numeric Display** objects to display *Temp1F, Temp2F* and *Voltage*. These are floating point numbers so, after you select the tag, change the **Data Type** property to **Float [Max 9 digits]**.


⊟ Display Data Properties

Data Type: Float [Max 9 digits]

18. Compile and save the project one last time. It is ready to download and run on the HMC hardware.

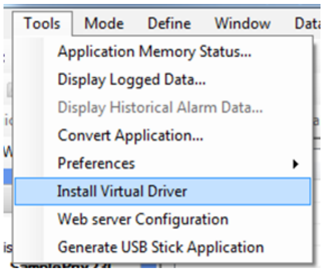
Testing the Project

Download the project to the HMC3102A-M using the same procedure described before. Then use the increment and decrement buttons we created to change the inputs and verify that the outputs change to the correct scaled value.

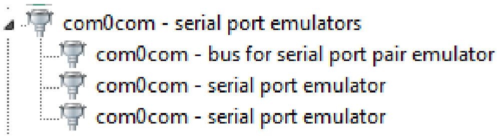
Online Monitoring

Installing the Virtual Com Port driver

In IEC 61131-3 mode, MAPware-7000 requires an additional communication driver, the virtual com port driver, be installed before starting an Online Monitoring session using over USB. To install the driver go to **Tools > Install Virtual Driver**.



This driver will add several ports in the development PC’s Device Manager.

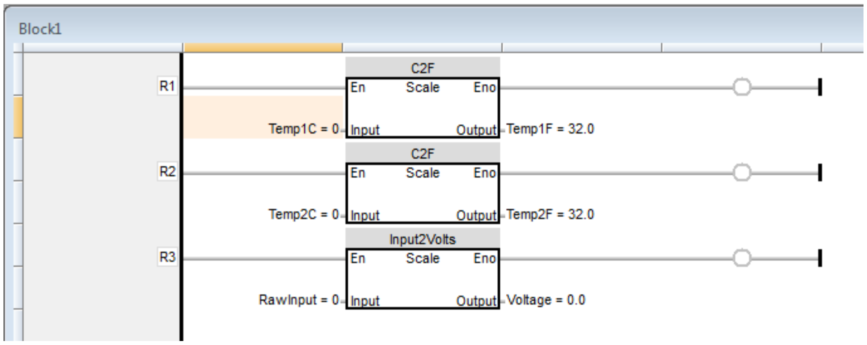

com0com - serial port emulators
com0com - bus for serial port pair emulator
com0com - serial port emulator
com0com - serial port emulator

Going Online

With the drivers in place, the HMC/MLC can be monitored with MAPware-7000 while it is executing a program. Select the USB mode from **Tools > Preferences > Online Communication Mode** and then click the icon to go online.

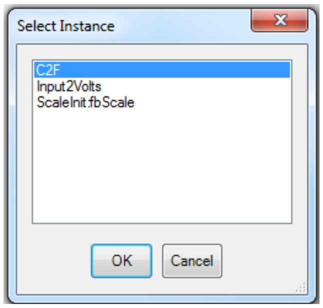
Selecting Logic Blocks and Function Block Instances

Once the online session begins, any open logic blocks will have variables loaded with their real time values.

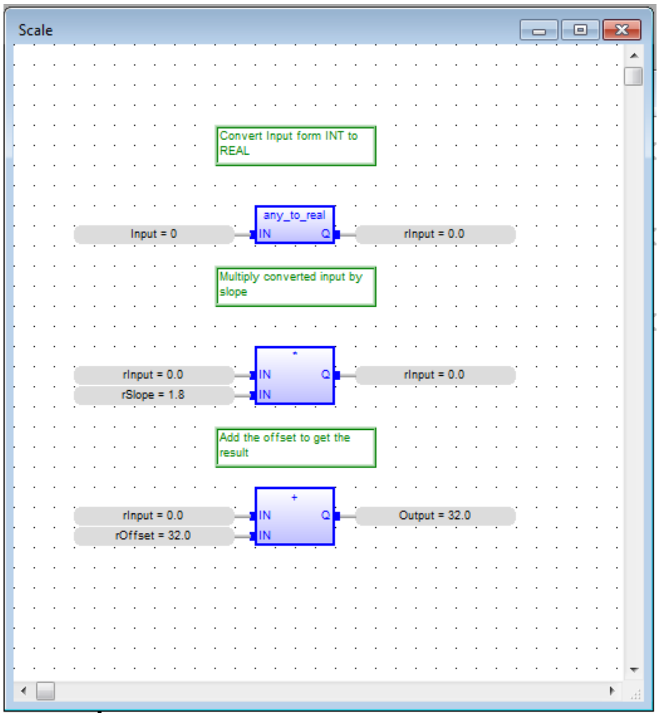


Logic blocks can be opened for monitoring by selecting them in the project tree.

When opening a Function Block that has multiple instances, MAPware-7000 needs to know which instance to open. Thus when a UDFB that has multiple instances is selected, a popup window will appear listing the available instances of that function block to monitor.

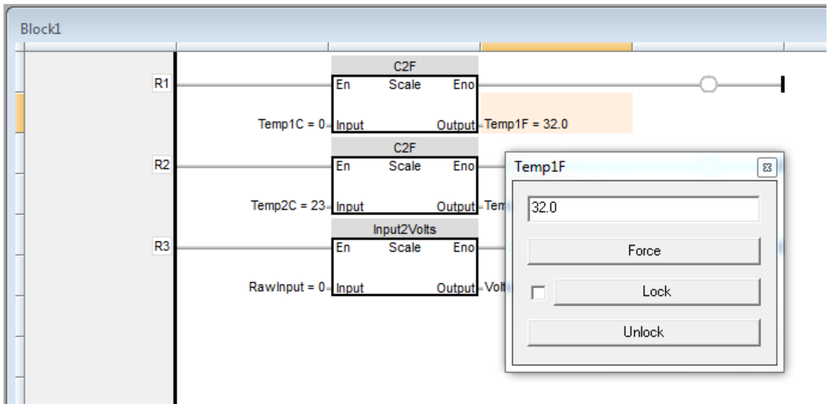


Once the selection is made the function block will be loaded with the real time data for the selected instance.

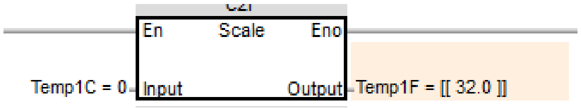


Manipulating Data

To change the value of a parameter simply double-click the parameter.



The popup window allows you to force and or Lock / Unlock the value of the parameter. When a parameter is forced it is written to once. If some other logic writes to the parameter after it is forced the forced value will be overwritten. If the value is locked, MAPware will prevent any other logic in the block from overwriting the forced value. Parameters that are locked will appear in double square brackets.



Note: The lock only prevents logic blocks from writing to the value. The value can still be changed by a task or by entering data in a numeric object on an HMI screen.

Review

Let’s review some important points about how this sample project is structured that will allow you to take full advantage of IEC 61131-3 features. Although not all of the features available in MAPware-7000 or the HMC Series have been covered, we have taken our first steps in using this software and becoming familiar with device operation.

First, notice how we used multiple instances of a User Defined Function Block. We are scaling three inputs, but have defined our scaling logic in only one place, the *Scale* UDFB. If something needs to be changed, it only needs to be changed in one place, and it isn’t necessary to hunt through the project to make sure the logic is updated everywhere it is used. We reused a single instance of the function block to scale two different inputs and used a separate instance to scale another input on a totally different scale. We could have many more channels and many more scales, but still only need one UDFB. If there was a radically different scaling operation that needed to be done, such as a lookup table, then we would need to create a different UDFB.

Next, note that three different editors were used, and it is possible to call the logic created in one editor from logic created in another. The different languages each have their own strengths. Ladder Diagram provides a clear graphical representation of the logic flow, Function Block Diagram is great for combining operations in a simple to read structure, and Structured Text can be used for more involved operations that might look quite complicated in one of the graphical editors. By combining logic from the different editors we are able to take advantage of the strengths of each.

Finally, note that we were able to separate the logic for initializing the scale block (UDFB-ScaleInt) and actually performing the scale operation (UDFB-Scale) into two separate operations. We did this by passing a UDFB instance as a parameter to another UDFB. This keeps the scaling operation, which occurs frequently while the project is running, as simple and quick as possible. Breaking complex operations into simple building blocks is another way to make a project more maintainable.

Your Industrial Control Solutions Source

www.maplesystems.com

AW10101057 Rev. 00