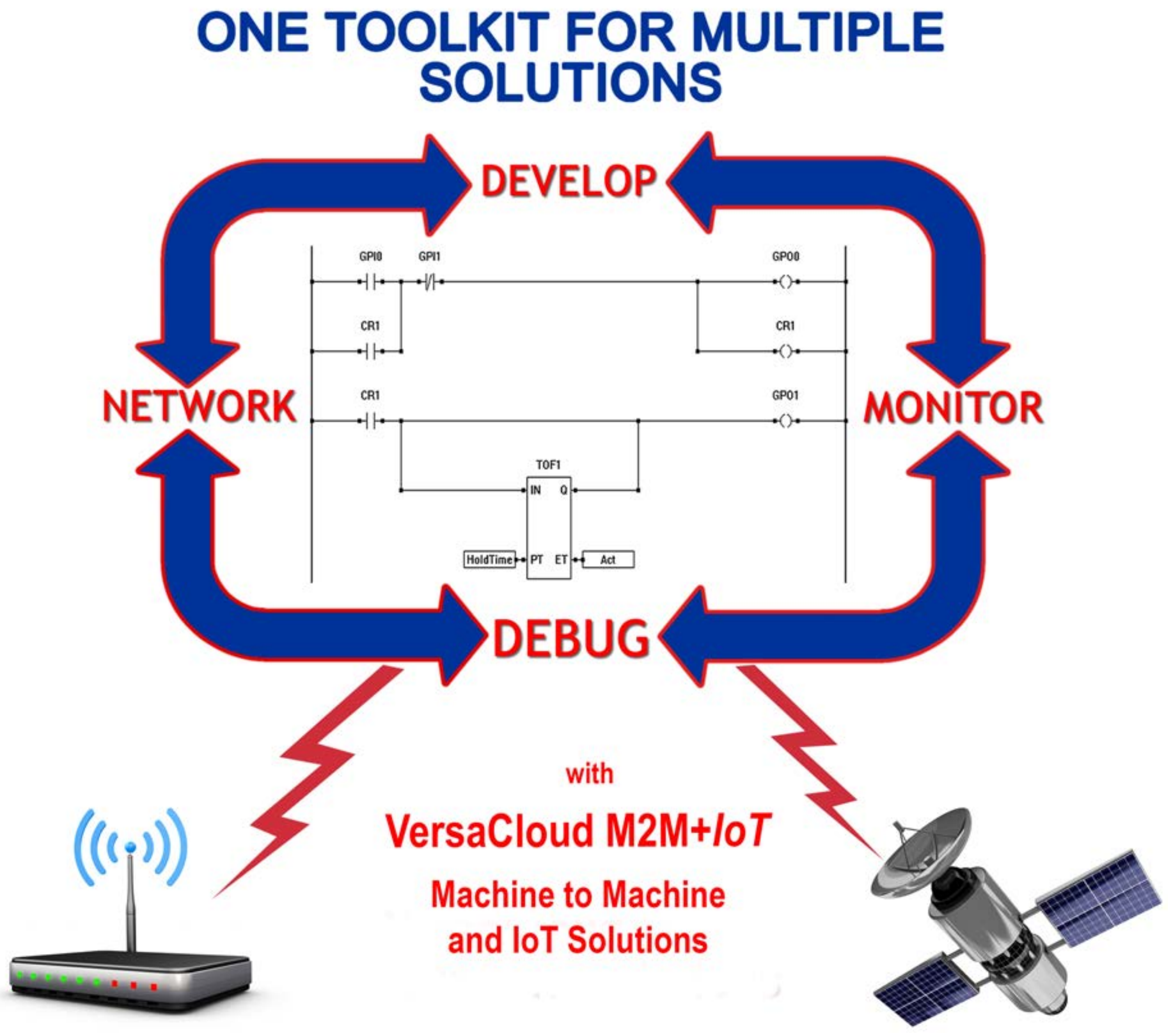
USER MANUAL

*P-Series Standard Edition*

# EZ LADDER

TOOLKIT



**Programming Manual for all P-Series PLC on a Chip™ Based Products**

**Version 1.14 for EZ LADDER Version 1.2.5.8**

# Table of Contents

# CHAPTER 1

## Getting Started

This chapter provides detailed information for getting started using the P-Series EZ LADDER Toolkit. Included in this section are installation instructions, activating EZ LADDER Toolkit and instructions on how information in this manual is presented.

## Chapter Contents

## What’s added or changed in this Manual / Software Version

This manual is for P-Series EZ LADDER Toolkit (EZ LADDER for the **P-Series PLC on a Chip™** based product line). For M-Series based **PLC on a Chip™** products, refer to the M-Series EZ LADDER Toolkit User Manual.

### EZ LADDER Software Changes (V1.2.5.8)

- Added new Azure Root Certificate for MQTT - now allows for more than one root certificate.
- Added timeout for MQTT connect message - Now disconnects on timeout.
- Added optional enable / disable option for TLS with MQT. MQTT TLS memory is only reserved when enabled.
- Changed Structured Text LIMIT function number of inputs require to 3
- Corrected MQTT / SD Card / Watchdog Timer issues (PLCHIP kernel).

The following are changes since the last published manual.

### EZ LADDER Manual Corrections / Additions

- Chapter 28 - Use TLS checkbox details added..

## How to Use this Manual

In this manual, the following conventions are used to distinguish elements of text:

| | |
|---|---|
| **BOLD** | Denotes labeling, commands, and literal portions of syntax that must appear exactly as shown. |
| *Italic* | *Used for variables and placeholders that represent the type of text to be entered by the user.* |
| **SMALL CAPS** | Used to show key sequences or actual buttons, such as OK, where the user clicks the OK button. |

In addition, the following symbols appear periodically appear in the left margin to call the readers attention to specific details in the text:

🚫 Warns the reader of a potential danger or hazard that is associated with certain actions.

💡 Appears when the text contains a tip that is especially helpful.

❗ Indicates that the text contains information to which the reader should pay particularly close attention.

This manual is divided into Chapters that are organized to promote a step by step progression of using the EZ LADDER Toolkit from installation to troubleshooting. Each chapter contains specific information that is relevant to understanding how to use the EZ LADDER Toolkit.

EZ LADDER Toolkit is now available as two separate programs (one for the M-Series PLC on a Chip™ based targets and one for the P-Series PLC on a Chip™ based targets). This manual is for the P-Series based targets and their supported features in relation to programming and EZ LADDER Toolkit. For M-Series based targets, the M-Series EZ LADDER Toolkit User Manual should be used. Only the correct manual for the hardware based (P-Series or M-Series) should be used as there are distinct differences in configuring targets and features and functions supported between the two.

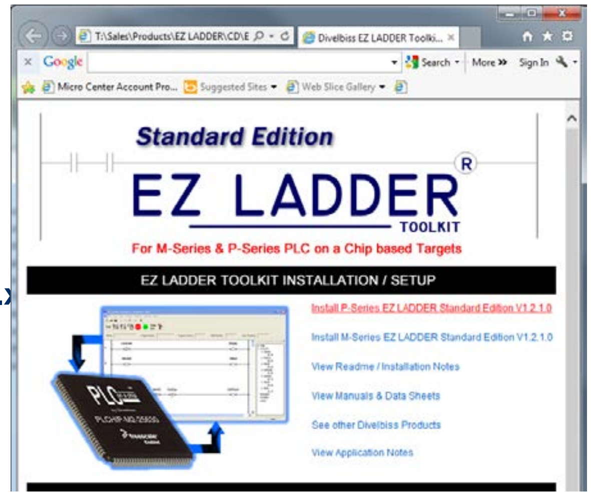
## Installing the EZ LADDER Toolkit from CD

This section of instruction is for installing EZ LADDER Toolkit from CD. If your EZ LADDER Toolkit was provided on USB, refer to the Installing the ***EZ LADDER Toolkit from USB*** section. EZ LADDER Toolkit may also be downloaded and installed, refer to that section of the manual for more details. To install EZ LADDER Toolkit on your computer, follow the following steps. Once EZ LADDER Toolkit is installed, it must be activated before it may be used with actual hardware targets.

EZ LADDER Toolkit supports Windows XP, Vista, Vista 64-bit, Windows 7, Windows 7 64-bit, Windows 8 and Windows 10. Windows 2000 and earlier versions are not supported.

Windows Administrator Rights are required for proper installation. The EZ LADDER directory security is dependent on local / network security settings and may be set for user Read/Execute only. To allow the user to be able to write to this directory, an Administrator must change the permissions accordingly.

Windows Administrator Rights are required to install / register / activate EZ LADDER. Installing EZ LADDER Toolkit without Administrator Permissions will cause EZ LADDER Toolkit not install and not operate correctly.

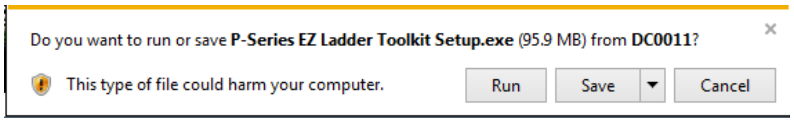

T:\Sales\Products\EZ LADDER\CD\E
Divelbiss EZ LADDER Toolki...
Google
Search
More »
Sign In
Micro Center Account Pro...
Suggested Sites
Web Slice Gallery
Standard Edition
EZ LADDER
TOOLKIT
For M-Series & P-Series PLC on a Chip based Targets
EZ LADDER TOOLKIT INSTALLATION / SETUP
Install P-Series EZ LADDER Standard Edition V1.2.1.0
Install M-Series EZ LADDER Standard Edition V1.2.1.0
View Readme / Installation Notes
View Manuals & Data Sheets
See other Divelbiss Products
View Application Notes

1. Copy the Serial Number printed on the face of the EZ LADDER Toolkit CD to a piece of paper. You will need this serial number during installation.

2. Insert the EZ LADDER Toolkit CD into your CD drive. If you have Active Content Enabled for your CD Drive, a Menu will appear. Click the **INSTALL P-SERIES EZ LADDER STANDARD EDITION V X.X.X.X** the EZ LADDER Toolkit setup.

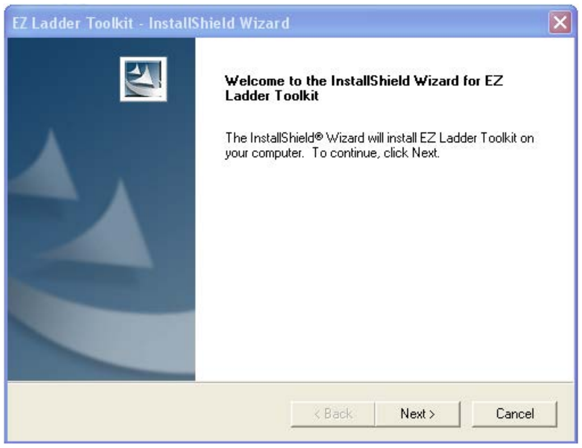
   If this screen does not appear. Click the **START** button and choose **RUN.** Browse to the USB Drive, then double-click **start.exe**.

   You will be prompted with a warning. Select RUN.


Do you want to run or save **P-Series EZ Ladder Toolkit Setup.exe** (95.9 MB) from **DC0011**?

This type of file could harm your computer.

3. The EZ LADDER Toolkit Installation Wizard will open. Click **NEXT.**

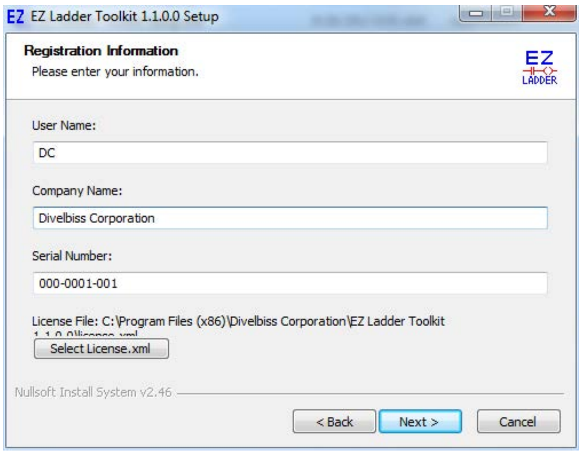

EZ Ladder Toolkit - InstallShield Wizard

**Welcome to the InstallShield Wizard for EZ Ladder Toolkit**

The InstallShield® Wizard will install EZ Ladder Toolkit on your computer. To continue, click Next.

4. Complete the Name, Organization fields and enter the Serial Number. Do not click the SELECT LICENSE XML button. This button should only be used under Divelbiss personnel supervision. Click **NEXT.**

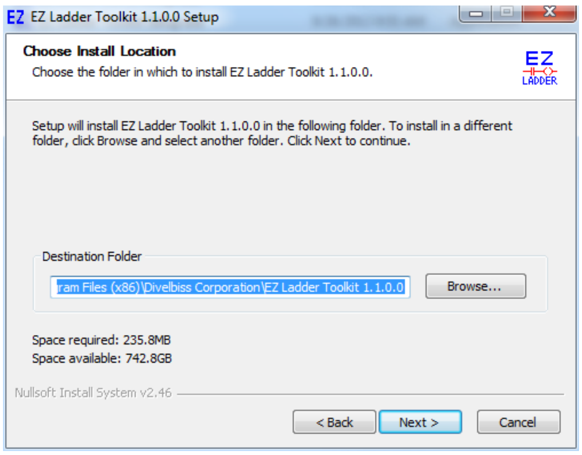
The serial number entered is used during activation. If the serial number is not correct, you will not be able to activate your EZ LADDER Toolkit later.


EZ Ladder Toolkit 1.1.0.0 Setup

**Registration Information**
Please enter your information.

User Name: DC
Company Name: Divelbiss Corporation
Serial Number: 000-0001-001
License File: C:\Program Files (x86)\Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0\license.xml

5. Use the default location for installing the EZ LADDER Toolkit or browse and select a different location. Click **NEXT**.


EZ Ladder Toolkit 1.1.0.0 Setup

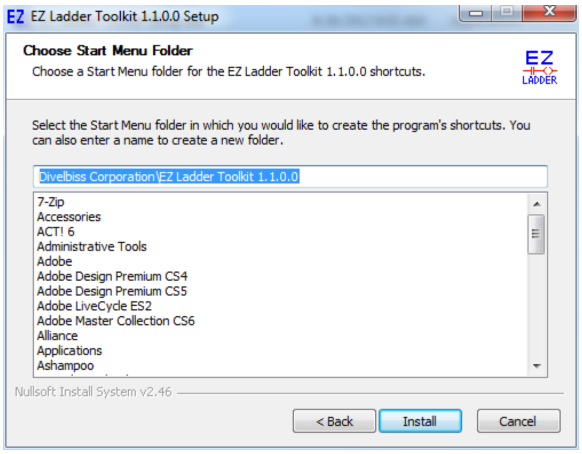
**Choose Install Location**
Choose the folder in which to install EZ Ladder Toolkit 1.1.0.0.

Setup will install EZ Ladder Toolkit 1.1.0.0 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: ram Files (x86)\Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0

Space required: 235.8MB
Space available: 742.8GB

6. All the information is gathered. Click **INSTALL** to install the EZ LADDER Toolkit. The EZ LADDER installer will copy all the required files and create a shortcut.


EZ Ladder Toolkit 1.1.0.0 Setup

**Choose Start Menu Folder**
Choose a Start Menu folder for the EZ Ladder Toolkit 1.1.0.0 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0

7-Zip
Accessories
ACT! 6
Administrative Tools
Adobe
Adobe Design Premium CS4
Adobe Design Premium CS5
Adobe LiveCycle ES2
Adobe Master Collection CS6
Alliance
Applications
Ashampoo

7. When installation is complete, click **FINISH**.

## Installing the EZ LADDER Toolkit from USB

This section of instruction is for installing EZ LADDER Toolkit from USB. If your EZ LADDER Toolkit was provided on CD, refer to the Installing the ***EZ LADDER Toolkit from CD*** section. EZ LADDER Toolkit may also be downloaded and installed, refer to that section of the manual for more details. To install EZ LADDER Toolkit on your computer, follow the following steps. Once EZ LADDER Toolkit is installed, it must be activated before it may be used with actual hardware targets.

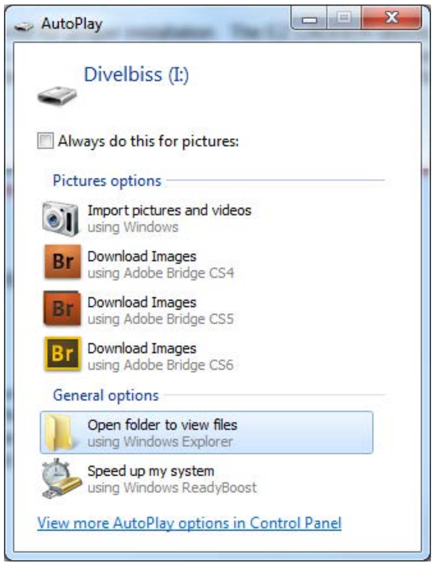
EZ LADDER Toolkit supports Windows XP, Vista, Vista 64-bit, Windows 7, Windows 7 64-bit, Windows 8 and Windows 10. Windows 2000 and earlier versions are not supported.

Windows Administrator Rights are required for proper installation. The EZ LADDER directory security is dependent on local / network security settings and may be set for user Read/Execute only. To allow the user to be able to write to this directory, an Administrator must change the permissions accordingly.

Windows Administrator Rights are required to install / register / activate EZ LADDER. Installing EZ LADDER Toolkit without Administrator Permissions will cause EZ LADDER Toolkit not install and not operate correctly.

1. Insert the EZ LADDER Toolkit USB into a USB port on your computer.

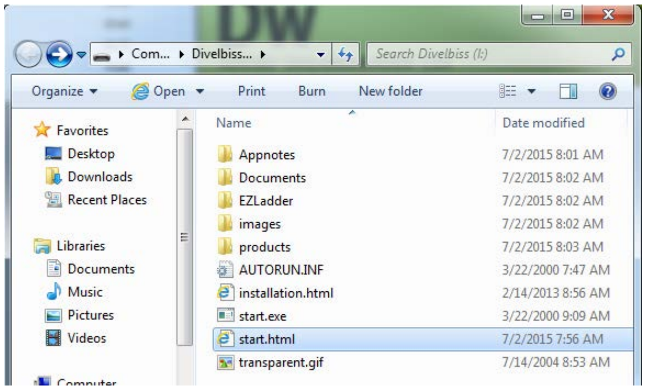
2. An 'Autoplay' dialog should appear. Select the ***Open folder to view files*** option.



Double-click **start.html**

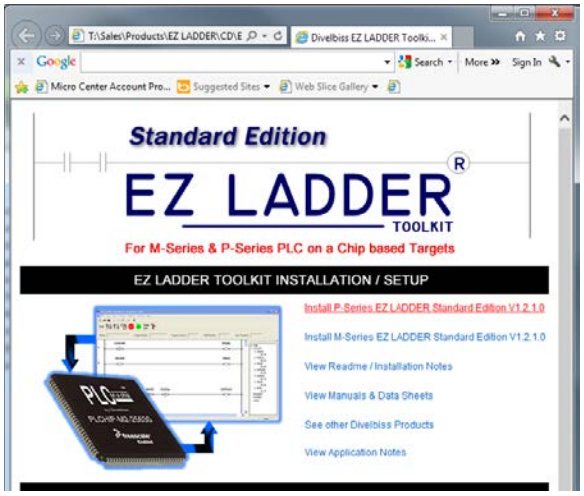
If this screen does not appear. Click the **START** button and choose **RUN.** Browse to the USB Drive, then double-click **start.html**.

You can also choose to skip the menu and browse directly to the EZ LADDER directory , then Vx.x.x.x and find the *P-Series EZ Ladder Toolkit Setup.exe*.

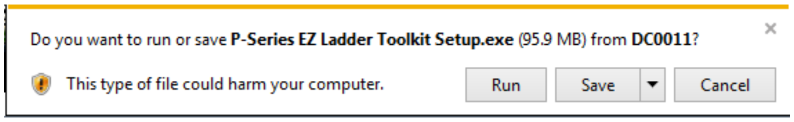


| Name | Date modified |
|---|---|
| Appnotes | 7/2/2015 8:01 AM |
| Documents | 7/2/2015 8:02 AM |
| EZLadder | 7/2/2015 8:02 AM |
| images | 7/2/2015 8:02 AM |
| products | 7/2/2015 8:03 AM |
| AUTORUN.INF | 3/22/2000 7:47 AM |
| installation.html | 2/14/2013 8:56 AM |
| start.exe | 3/22/2000 9:09 AM |
| start.html | 7/2/2015 7:56 AM |
| transparent.gif | 7/14/2004 8:53 AM |

3. The menu will appear. Click the **INSTALL P-SERIES EZ LADDER STANDARD EDITION V X.X.X.X** to run the EZ LADDER Toolkit setup.


T:\Sales\Products\EZ LADDER\CD\E
Divelbiss EZ LADDER Toolki...
Google
Search
More
Sign In
Micro Center Account Pro...
Suggested Sites
Web Slice Gallery
Standard Edition
EZ LADDER
TOOLKIT
For M-Series & P-Series PLC on a Chip based Targets
EZ LADDER TOOLKIT INSTALLATION / SETUP
Install P-Series EZ LADDER Standard Edition V1.2.1.0
Install M-Series EZ LADDER Standard Edition V1.2.1.0
View Readme / Installation Notes
View Manuals & Data Sheets
See other Divelbiss Products
View Application Notes

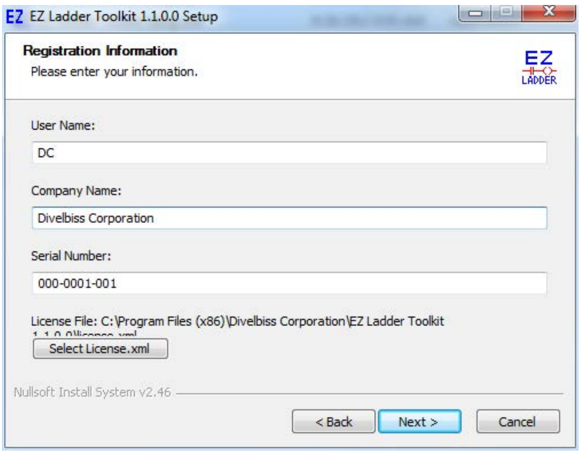
You will be prompted with a warning.
Select RUN.


Do you want to run or save **P-Series EZ Ladder Toolkit Setup.exe** (95.9 MB) from **DC0011**?

This type of file could harm your computer.

4. The EZ LADDER Toolkit Installation Wizard will open. Click **NEXT.**

5. Complete the Name and Organization fields. The Serial Number is automatically read from the USB drive. Do not click the **SELECT LICENSE XML** button. This button should only be used under Divelbiss personnel supervision. Click **NEXT.**


EZ Ladder Toolkit 1.1.0.0 Setup

**Registration Information**

Please enter your information.

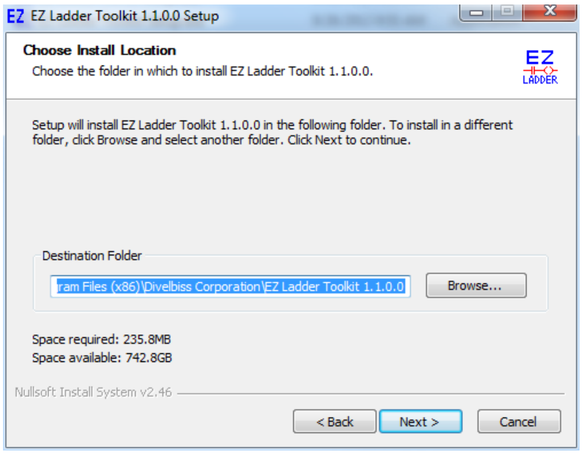
User Name: DC

Company Name: Divelbiss Corporation

Serial Number: 000-0001-001

License File: C:\Program Files (x86)\Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0\license.xml

6. Use the default location for installing the EZ LADDER Toolkit or browse and select a different location. Click **NEXT**.


EZ Ladder Toolkit 1.1.0.0 Setup

**Choose Install Location**

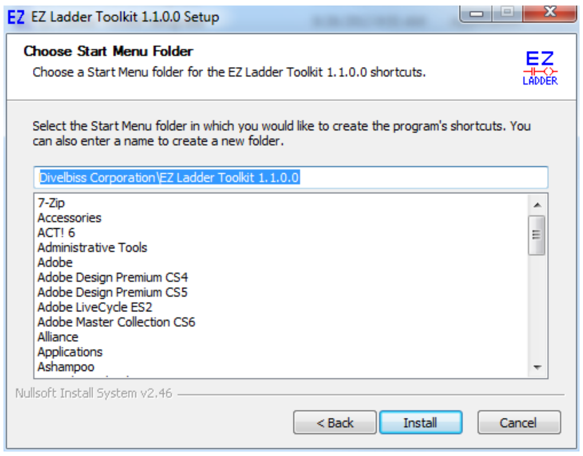
Choose the folder in which to install EZ Ladder Toolkit 1.1.0.0.

Setup will install EZ Ladder Toolkit 1.1.0.0 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: ram Files (x86)\Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0

Space required: 235.8MB
Space available: 742.8GB

7. All the information is gathered. Click **INSTALL** to install the EZ LADDER Toolkit. The EZ LADDER installer will copy all the required files and create a shortcut.



EZ Ladder Toolkit 1.1.0.0 Setup

**Choose Start Menu Folder**

Choose a Start Menu folder for the EZ Ladder Toolkit 1.1.0.0 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0

7-Zip
Accessories
ACT! 6
Administrative Tools
Adobe
Adobe Design Premium CS4
Adobe Design Premium CS5
Adobe LiveCycle ES2
Adobe Master Collection CS6
Alliance
Applications
Ashampoo

Nullsoft Install System v2.46

8. When installation is complete, click **FINISH**.

## Downloading and Installing EZ LADDER Toolkit

This section of instruction is for downloading installing EZ LADDER Toolkit from from www.divelbiss.com. If your EZ LADDER Toolkit was provided on USB, refer to the Installing the ***EZ LADDER Toolkit from USB*** section. If your EZ LADDER Toolkit was provided on CD, refer to the Installing the ***EZ LADDER Toolkit from CD*** section.

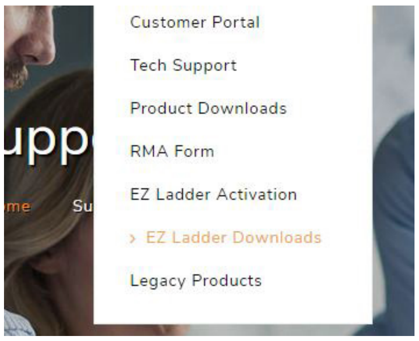
To download and install EZ LADDER Toolkit on your computer, follow the following steps. Once EZ LADDER Toolkit is installed, it must be activated before it may be used with actual hardware targets.

EZ LADDER Toolkit supports Windows XP, Vista, Vista 64-bit, Windows 7, Windows 7 64-bit, Windows 8 and Windows 10. Windows 2000 and earlier versions are not supported.

Windows Administrator Rights are required for proper installation. The EZ LADDER directory security is dependent on local / network security settings and may be set for user Read/Execute only. To allow the user to be able to write to this directory, an Administrator must change the permissions accordingly.

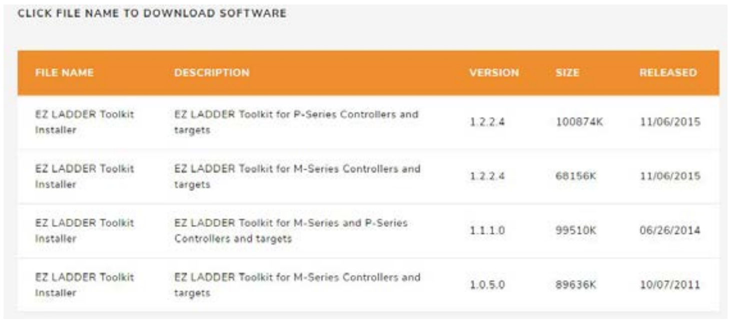
Windows Administrator Rights are required to install / register / activate EZ LADDER. Installing EZ LADDER Toolkit without Administrator Permissions will cause EZ LADDER Toolkit not install and not operate correctly.

1. Go to **https://www.divelbiss.com**. From the menu, select **Support** and from the support drop-down, select **EZ Ladder Downloads.** The EZ LADDER downloads page will open with a list of versions available to download.



2. Choose the proper EZ LADDER version to download and click on the Filename for the version (column on the left). The file should begin the download. Depending on the browser there may be additional dialog boxes. The download may take several minutes.

   The file downloaded is an executable (.exe) file.



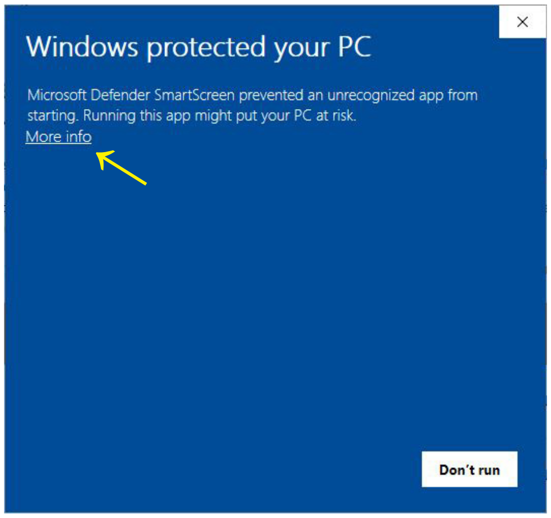
CLICK FILE NAME TO DOWNLOAD SOFTWARE

| FILE NAME | DESCRIPTION | VERSION | SIZE | RELEASED |
|---|---|---|---|---|
| EZ LADDER Toolkit Installer | EZ LADDER Toolkit for P-Series Controllers and targets | 1.2.2.4 | 100874K | 11/06/2015 |
| EZ LADDER Toolkit Installer | EZ LADDER Toolkit for M-Series Controllers and targets | 1.2.2.4 | 68156K | 11/06/2015 |
| EZ LADDER Toolkit Installer | EZ LADDER Toolkit for M-Series and P-Series Controllers and targets | 1.1.1.0 | 99510K | 06/26/2014 |
| EZ LADDER Toolkit Installer | EZ LADDER Toolkit for M-Series Controllers and targets | 1.0.5.0 | 89636K | 10/07/2011 |

An EZ LADDER Toolkit Serial Number is required during the installation process. A CID code is required during the activation process. The Serial Number and CID codes are needed prior to proceeding. Refer to the ***Requesting a Serial Number / CID Code*** section of this chapter for more details.

3. Click (or double-click) on the downloaded file to run the installer. Microsoft Defender will open a warning dialog. This is normal for an executable file.

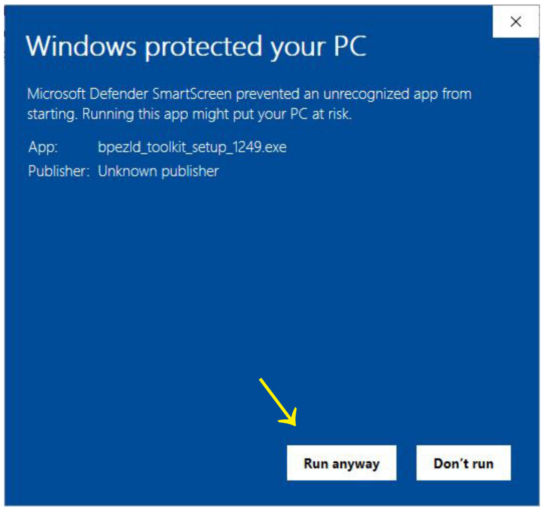
4. Click the **More Info** link in the dialog. The dialog will update with more details on the file and publisher. It will also display a new button - **RUN ANYWAY**.



Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

5. Click the **RUN ANYWAY** button. This will allow the actual installer to run.

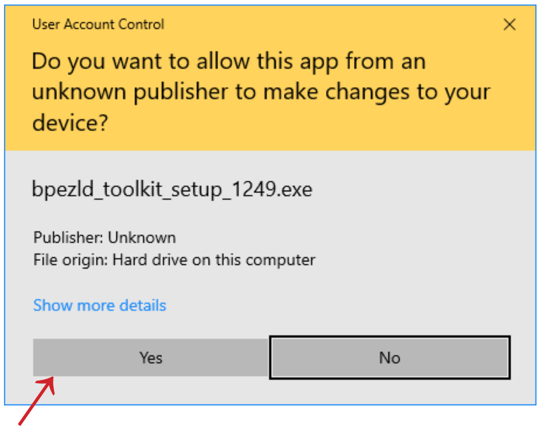


Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

App: bpezld_toolkit_setup_1249.exe
Publisher: Unknown publisher

6. Based on your computer settings, you may see the Window's User Account Control dialog appear. Click **YES** to continue the installation.


User Account Control

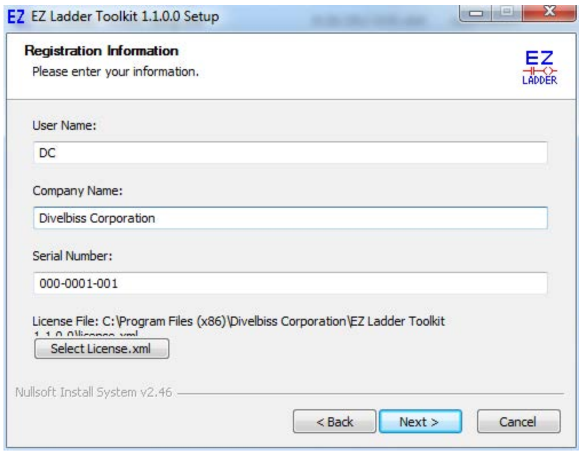
Do you want to allow this app from an unknown publisher to make changes to your device?

bpezld_toolkit_setup_1249.exe

Publisher: Unknown
File origin: Hard drive on this computer

7. The EZ LADDER Toolkit Installation Wizard will open. Click **NEXT.**

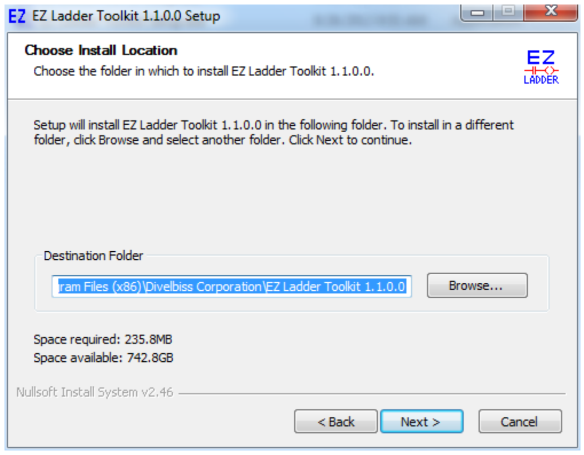
8. Complete the Name, Organization and Serial Number fields. Do not click the **SELECT LICENSE XML** button. This button should only be used under Divelbiss personnel supervision. Click **NEXT.**


EZ Ladder Toolkit 1.1.0.0 Setup

Registration Information
Please enter your information.

User Name: DC
Company Name: Divelbiss Corporation
Serial Number: 000-0001-001
License File: C:\Program Files (x86)\Divelbiss Corporation\EZ Ladder Toolkit

9. Use the default location for installing the EZ LADDER Toolkit or browse and select a different location. Click **NEXT**.


EZ Ladder Toolkit 1.1.0.0 Setup

Choose Install Location
Choose the folder in which to install EZ Ladder Toolkit 1.1.0.0.

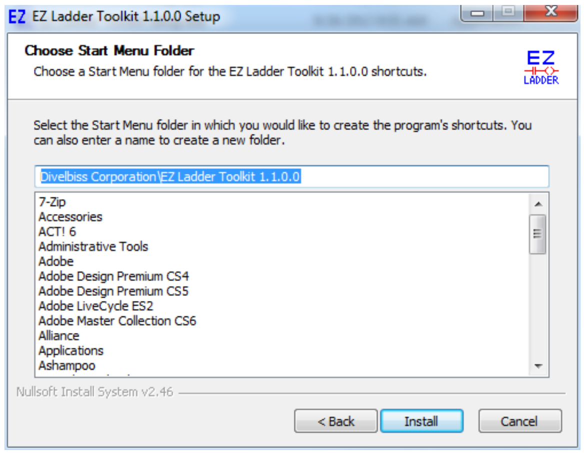
Setup will install EZ Ladder Toolkit 1.1.0.0 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: ram Files (x86)\Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0

Space required: 235.8MB
Space available: 742.8GB

10. All the information is gathered. Click **INSTALL** to install the EZ LADDER Toolkit. The EZ LADDER installer will copy all the required files and create a shortcut.

11. When installation is complete, click **FINISH**.


EZ Ladder Toolkit 1.1.0.0 Setup

Choose Start Menu Folder
Choose a Start Menu folder for the EZ Ladder Toolkit 1.1.0.0 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

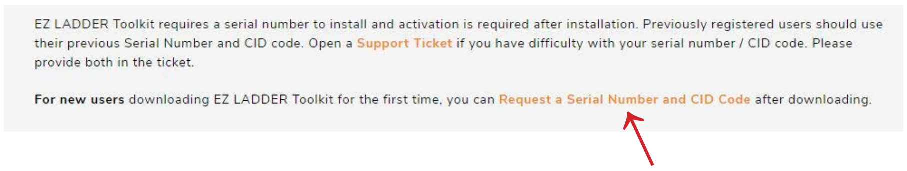
Divelbiss Corporation\EZ Ladder Toolkit 1.1.0.0

7-Zip
Accessories
ACT! 6
Administrative Tools
Adobe
Adobe Design Premium CS4
Adobe Design Premium CS5
Adobe LiveCycle ES2
Adobe Master Collection CS6
Alliance
Applications
Ashampoo

## Requesting a Serial Number / CID Code

This section of instruction is for requesting an EZ LADDER Toolkit Serial Number (required during installation) and CID code (required during activation). For new users downloading EZ LADDER Toolkit (no previous EZ LADDER from CD or USB), a request form is provided on our website.

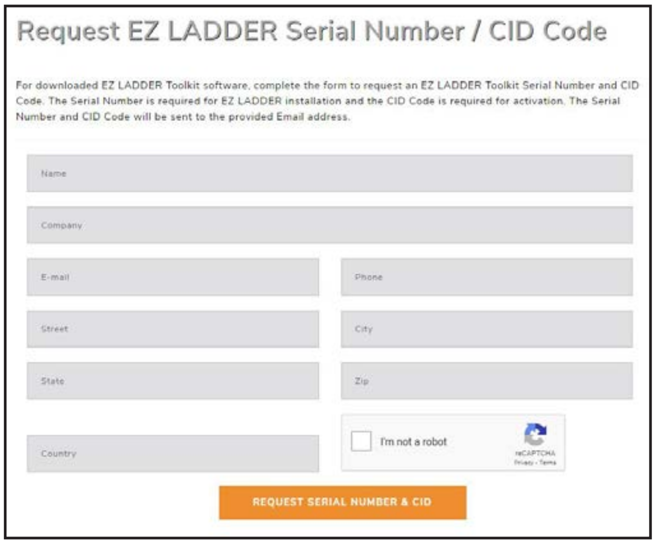
1. Go to **https://www.divelbiss.com**. From the menu, select **Support** and from the support drop-down, select **EZ Ladder Downloads.** There is a link or the page to the request form. You can also go directly to **https://divelbiss.com/EZSN.php**.



EZ LADDER Toolkit requires a serial number to install and activation is required after installation. Previously registered users should use their previous Serial Number and CID code. Open a Support Ticket if you have difficulty with your serial number / CID code. Please provide both in the ticket.

**For new users** downloading EZ LADDER Toolkit for the first time, you can Request a Serial Number and CID Code after downloading.

2. Complete the form fields and submit it.



Request EZ LADDER Serial Number / CID Code

For downloaded EZ LADDER Toolkit software, complete the form to request an EZ LADDER Toolkit Serial Number and CID Code. The Serial Number is required for EZ LADDER installation and the CID Code is required for activation. The Serial Number and CID Code will be sent to the provided Email address.

3. The system will e-mail the Serial Number and CID code to the e-mail address provided during the form submission.

Please record the Serial Number and CID code for future installations / re-installation of your EZ LADDER Toolkit.

## Activating the EZ LADDER Toolkit

Until EZ LADDER Toolkit is activated, it will only operated in DEMO mode which does not allow connecting to actual hardware targets (controllers) or downloading programs.

Now that the EZ LADDER Toolkit is installed, it must be activated to enable all the features. You will need the following to activate your EZ LADDER Toolkit.

1. An internet connection and web browser like Internet Explorer.

2. CD installations: Your EZ LADDER Toolkit CD Case. You will need your CID Code located on the back of the case if you installed from CD.

   If you installed using USB, the CID code is located on a label on the bag that contained the USB Flash drive.

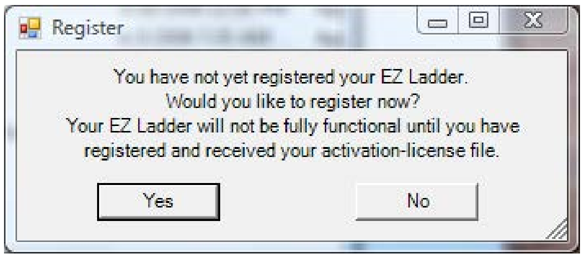
3. EZ LADDER Toolkit installed.

Once activated, EZ LADDER Toolkit is fully functional and will operate with hardware targets. The process of registering and activating is completing the on-line registration form receiving a counter key. This key must be loaded into EZ LADDER Toolkit and when loaded, it will activate your copy of EZ LADDER Toolkit.

If EZ Ladder is not registered, it will prompt you to do so when the application is started.

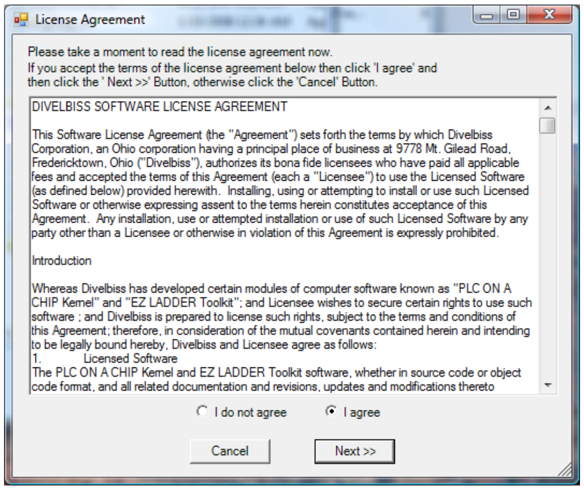
To activate and register your EZ LADDER Toolkit, follow the installation wizard as follows:

1. When prompted to Activate EZ LADDER, click **YES**.



You have not yet registered your EZ Ladder.
Would you like to register now?
Your EZ Ladder will not be fully functional until you have registered and received your activation-license file.

2. You must read and agree to the license agreement to activate the EZ LADDER Toolkit. Click the **I AGREE** box and click **NEXT**.



Please take a moment to read the license agreement now.
If you accept the terms of the license agreement below then click 'I agree' and then click the ' Next >>' Button, otherwise click the 'Cancel' Button.

DIVELBISS SOFTWARE LICENSE AGREEMENT

This Software License Agreement (the "Agreement") sets forth the terms by which Divelbiss Corporation, an Ohio corporation having a principal place of business at 9778 Mt. Gilead Road, Fredericktown, Ohio ("Divelbiss"), authorizes its bona fide licensees who have paid all applicable fees and accepted the terms of this Agreement (each a "Licensee") to use the Licensed Software (as defined below) provided herewith. Installing, using or attempting to install or use such Licensed Software or otherwise expressing assent to the terms herein constitutes acceptance of this Agreement. Any installation, use or attempted installation or use of such Licensed Software by any party other than a Licensee or otherwise in violation of this Agreement is expressly prohibited.

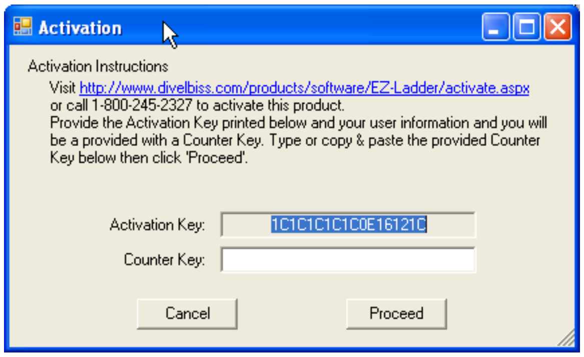
Introduction

Whereas Divelbiss has developed certain modules of computer software known as "PLC ON A CHIP Kernel" and "EZ LADDER Toolkit"; and Licensee wishes to secure certain rights to use such software ; and Divelbiss is prepared to license such rights, subject to the terms and conditions of this Agreement; therefore, in consideration of the mutual covenants contained herein and intending to be legally bound hereby, Divelbiss and Licensee agree as follows:
1. Licensed Software
The PLC ON A CHIP Kernel and EZ LADDER Toolkit software, whether in source code or object code format, and all related documentation and revisions, updates and modifications thereto

I do not agree [✓ I agree]

2. Click the link provided. A web browser window will open to the registration and activation page on Divelbiss.com.

   You will need the Activation key provided by EZ LADDER and your CID Code # located on the back of your EZ LADDER Toolkit CD case (if installed using CD) or on a label on the bag the USB Flash drive was packaged in (if installed using USB).



**Activation**

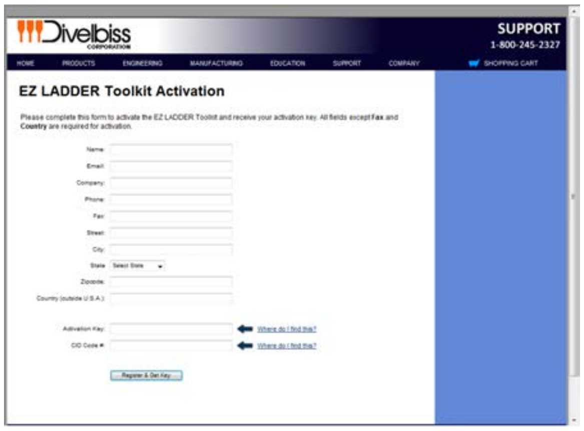
Activation Instructions
Visit http://www.divelbiss.com/products/software/EZ-Ladder/activate.aspx
or call 1-800-245-2327 to activate this product.
Provide the Activation Key printed below and your user information and you will be a provided with a Counter Key. Type or copy & paste the provided Counter Key below then click 'Proceed'.

Activation Key: 1C1C1C1C0E16121C
Counter Key:

If you do not have your CID Code #, you must obtain it prior to continuing the activation. Contact Divelbiss Customer Support.

If you close EZ LADDER prior to completing activation, the original Activation Key cannot be used. A new Activation Key must be used to activate the EZ LADDER Toolkit.

3. Click on the link provided to open the browser and go to the Activation Page. Copy / Paste or type your Activation key into the Activation key form box on the web site Activation and Registration page, if not already pre-loaded (Internet Explorer will preload this for you).



**EZ LADDER Toolkit Activation**

Please complete this form to activate the EZ LADDER Toolkit and receive your activation key. All fields except **Fax** and **Country** are required for activation.

Name:
Email:
Company:
Phone:
Fax:
Street:
City:
State: Select State
Zipcode:
Country (outside U.S.A.):

Activation Key: Where do I find this?
CID Code #: Where do I find this?

4. Complete all other form entries. All information must be completed. The CID Code# is found on the EZ LADDER Toolkit's CD Case (located on back side).

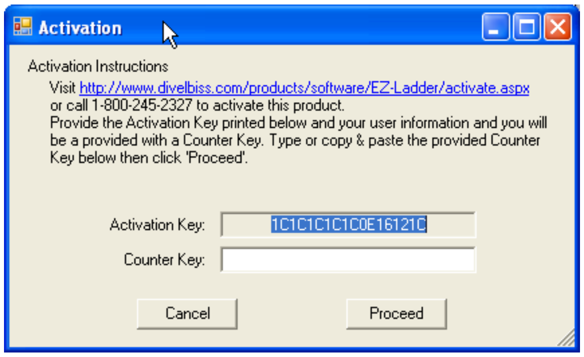
   For USB installations, the CID code should be read automatically and sent to the activation form ( It can be found on a label on the bag the USB Flash drive was packaged in.

5. Click the **REGISTER & GET KEY** button. The Activation key and other information will be confirmed and if valid, a *Counter Key*, *Username* and *Password* will be displayed. Save the *Username* and *Password* as they can be used to download updates to EZ LADDER Toolkit.

If the information is not valid, the registration will fail. Activation can fail due to an incorrect Serial Number, incorrect CID Code#, copying the Activation or Counter key incorrectly or if this serial number has been registered more times than allowed per the license agreement (typically 2 times). Check this information. If you are still unable to activate EZ LADDER Toolkit, Contact Divelbiss Customer Support.

6. Copy / Paste or type the *Counter Key* into the Counter Key form box in the EZ LADDER Toolkit Activation Window. Click **PROCEED**.

7. A Dialog box will appear verifying that EZ LADDER has successfully been activated.



**Activation**

Activation Instructions
Visit http://www.divelbiss.com/products/software/EZ-Ladder/activate.aspx
or call 1-800-245-2327 to activate this product.
Provide the Activation Key printed below and your user information and you will be a provided with a Counter Key. Type or copy & paste the provided Counter Key below then click 'Proceed'.

Activation Key: 1C1C1C1C0E16121C
Counter Key:

## Installing Additional Copies of EZ LADDER Toolkit

The standard EZ LADDER Toolkit license agreement allows the EZ LADDER Toolkit to be installed on up to two computers (usually a PC and a laptop). To install on a second computer, install the EZ LADDER Toolkit and Activate it as was done on the original computer.

If you attempt to activate a serial more than two time (unless you have purchased a site license), the activation will fail as the serial number has been activated the maximum number of allowed times.

If you are re-installing due to a hardware failure or moving computers, Contact Divelbiss Customer Support to allow additional activations.

# CHAPTER 2

## Navigating EZ LADDER Toolkit

This chapter provides detailed information on the basics of navigating and using the EZ LADDER Toolkit's workspace, menus, tool bars and windows.

## Chapter Contents

## EZ LADDER Toolkit Overview

When the EZ LADDER Toolkit is started, it will open in the ***Edit Mode***. This mode is where ladder diagram projects are created, functions are inserted and variables are placed. When actually downloading ladder diagram projects to targets or monitoring ladder diagram operation on hardware targets, it is referred to as ***Run Mode***. The Run Mode is explained in **Chapter 6 - Downloading and Running Projects**.

Figure 2-1 identifies the components that are part of the Edit Mode.

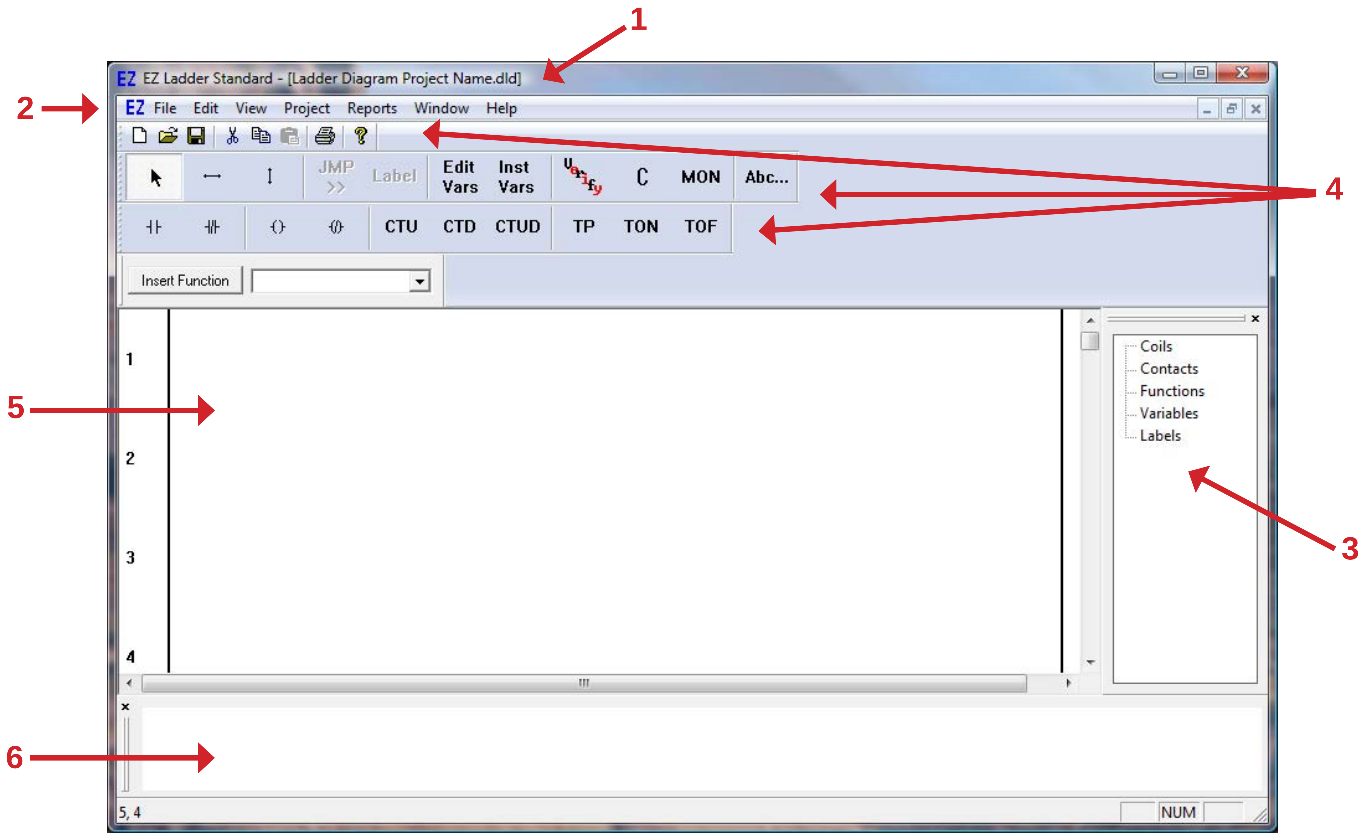


**Figure 2-1**

| | |
|---|---|
| 1. Project Filename: | The name of the currently viewed project will be displayed in this position. |
| 2. Menus: | Drop-down menus for programming features and options. |
| 3. Cross Reference: | Quick Click Cross References for functions, objects and variables. |
| 4. Tool Bars: | Tool bars for placing functions, objects and drop-down function lists. |
| 5. Ladder Workspace: | Area where the ladder diagram is drawn. |
| 6. Output Window: | This is where status messages are displayed when Verifying or Compiling ladder diagram programs. |

# EZ LADDER Toolkit Menus

The EZ LADDER Toolkit has many features and options. Basic commands, features and options are used and controlled through drop down menus. Figure 2-2 shows the standard EZ LADDER Toolkit Menu bar. As with any Windows based application, clicking on a menu heading will cause the drop down menu to open.

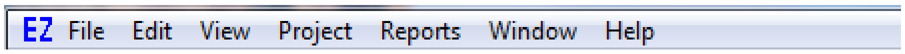


**Figure 2-2**

The menus found in the EZ LADDER Toolkit are: File, Edit, View, Project, Reports, Window and Help. Some of these menus are specific to EZ LADDER Toolkit features while others are part of the basic Windows structure.

## FILE MENU

The FILE Menu includes the standard windows functionality for file control and printing. The FILE Menu items are: New, Open, Close, Save, Save As, Print, Print Preview, Print Setup and Exit. A recently opened file list is also included for quick recall of recently opened ladder diagram projects.

### New
The New menu item is used to create a new, blank EZ LADDER Toolkit Ladder Diagram Project.

### Open
The Open menu item is select and open a previously saved EZ LADDER Toolkit Ladder Diagram Project.

### Close
The Close menu item closes the currently selected EZ LADDER Toolkit Ladder Diagram Project.

### Save
The Save menu item is used to save the currently selected EZ LADDER Toolkit Ladder Diagram Project. If the project has not been saved previously, the Save As dialog is displayed.

### Save As
The Save As menu item is used to save the currently selected EZ LADDER Toolkit Ladder Diagram Project under a new name.

### Print
Opens the Print dialog box for printing the currently selected EZ LADDER Toolkit Ladder Diagram Project with the settings defined in the Print Setup menu.

### Print Preview
Opens a window to view the ladder diagram project as it is to be printed.

### Print Setup
Opens a window to configure print and printer settings.

### Exit
Closes all currently opened ladder diagram projects and closes the EZ LADDER Toolkit application program.

## EDIT MENU

The EDIT Menu includes the standard windows functionality for editing and editing preferences. The EDIT Menu items are: Undo, Redo, Cut, Copy, Paste, Select All, Settings.

### Undo
The Undo will cause the last action performed to be undone.

### Redo
The Redo will cause an action that was undone using the Undo, to be repeated or completed again.

### Cut
The Cut menu is disabled in the EZ LADDER Toolkit. To delete an object or multiple objects, select the object(s) using the selector tool and remove by press the **DELETE** key.

### Copy
The Copy is disabled in the EZ LADDER Toolkit. To copy an object or multiple objects, select the object(s) using the selector tool, right click the mouse and select **COPY**. This will copy all the selected objects to the Windows clipboard.

### Paste
The Paste menu item is disabled in the EZ LADDER Toolkit. To paste an object or multiple objects, position the mouse at the starting point to paste, right click the mouse and select **PASTE**. This will paste the Windows clipboard contents into the ladder diagram project.

When pasting objects and rungs, enough space must be available at the pasting point for the Windows clipboard contents. The paste will not complete unless sufficient space is provided (# of rungs and space on each rung).

### Select All
The Select All menu item is disabled in the EZ LADDER Toolkit.

### Settings
The Settings menu item opens the LD (ladder diagram) settings window. This window allows general setting to be configured such as displaying grid, fonts, etc. Typically, it is recommended to leave the settings at the factory defaults.

## VIEW MENU

The VIEW Menu is used to view currently selected target information and to view or hide tool bars and optional windows.

### Target Information

The target information window provides details of the selected hardware target (selected in the Projects Settings menu) including target name, minimum kernel version required for this version of EZ LADDER Toolkit, Supported Objects and Functions, Analog I/O and Digital I/O. The target information may be printed using the provided **PRINT** button.

### Basic Components

The Basic Components menu item will cause the basic components tool bar to be visible or hidden. This tool bar includes buttons for the direct contact, inverted contact, direct coil, inverted coil, CTU, CTD, CTUD, TP, TON and TOF functions.

### Cross References

The Cross References menu item will cause the Cross Reference Window to be visible or hidden.

### Edit Tools

The Edit Tools menu item will cause the edit tools tool bar to be visible or hidden. This tool bar includes buttons for select, horizontal link, vertical link, Edit Vars, Inst Vars, Verify, Compile (C), MON and text boxes (Abc..).

### Monitor Status

The Monitor Status menu item will cause the monitor status tool bar to be visible or hidden (in monitor mode only). This tool bar is used to view actual target information (name, build, scan time, etc) when monitoring a program in the monitor mode.

### Monitor Tools

The Monitor Tools menu item will cause the monitor tools tool bar to be visible or hidden (monitor mode only). This tool bar is used to select monitoring tools when monitoring a program in the monitor mode.

### Function List

The Function List menu item will cause the drop down function list to be visible or hidden. This tool bar is used to select and insert functions into the ladder diagram project.

### Output

The Output menu item will cause the Output Window to be visible or hidden. This window displays important messages during the Verify and Compile Operations.

During the Compile process, it is important to have this window visible. Information including compile status and errors are displayed here.

### Toolbar
The Toolbar menu item will cause the standard windows functions toolbar to be visible or hidden. This tool bar includes, New, Open, Save, Cut, Print and more.

## PROJECT MENU

The PROJECT Menu is used to view and configure Project target settings including hardware target selections, and installing and configuring optional target features.

### Settings
The Settings menu item opens the Project Settings Dialog. This dialog is used to configure the actual hardware target (controller) and its features. The target is selected from the available list. Depending upon the target selected, additional configuration settings may be required and additional features can be configured from this menu. Refer to **Chapter 4 - Configuring Targets** for detailed information regarding target configurations.

### Bootloader
The Bootloader menu item will open the Bootloader dialog window. This window is used to install or update hardware target kernels and to configure some features such as the SD Card, Ethernet and USB.. The Bootloader menu item is only available in the Run Mode.

### OptiCAN
The OptiCAN menu item will open the OptiCAN Configuration Tool. This tool is used to configure the OptiCAN network. The OptiCAN menu item is only available in the Run Mode and when OptiCAN is enabled and supported.

### File Transfer
The File Transfer menu item will open the File Transfer window (Monitor Mode only). The File Transfer window is used to browse, write to and read files from the SD card (if installed on the actual hardware target).

### WiFi Setup
The WiFi Setup menu item will open the WiFi Setup window (Monitor Mode only). The WiFi Setup window is used to configure the WiFi settings including operation mode, access points, passwords and security. Once configured, the settings are stored on the target device.

## REPORTS MENU

The REPORTS Menu is used to generate, view and print reports that may be helpful when developing a ladder diagram project.

### Variable Definitions
The Variables Definitions report generates a list of all variables present in the ladder diagram project and their specific information including name, I/O Number, Default Value and their description.

### Cross References

The Cross Reference opens the Cross Reference Dialog box. This box is where the criteria for the report is selected. The following are the selectable criteria items: Input, Output, Internal, Function, Unused Variables, Contacts Without Coils, Coils Without Contacts, Drum Sequencer Tables, Retentive Variables and Network Address / Registers. After the required items are selected or deselected, click **OK**. This generates the viewable and printable report.

## WINDOW MENU

The WINDOW Menu is the basic Window's menu for viewing and controlling open application windows. This menu is typically found in every Window's based program. Since this functionality is based on Windows, it will not be described in detail.

## HELP MENU

The HELP Menu is useful to determine software versions and registration information. Currently, there is no active help built-in to the EZ LADDER Toolkit.

### About

Opens the EZ LADDER Toolkit about dialog box. The Toolkit version is displayed at the top of the dialog box. The File Versions tab identifies versions of each of the EZ LADDER Toolkit components. The License Information tab identifies the EZ LADDER Toolkit Serial Number and who it is registered to.

### Splash Screen

Opens the EZ LADDER Toolkit splash screen. This screen is normally viewable for a few seconds when EZ LADDER Toolkit is started.

## EZ LADDER Toolkit Tool Bars and Tool Bar Buttons

The EZ LADDER Toolkit provides tool bars for many common functions for ease of use and to increase efficiency when programming ladder diagram projects. As discussed earlier, many of these tool bars may be either viewed or hidden. EZ LADDER Toolkit defaults these tool bars as viewable.

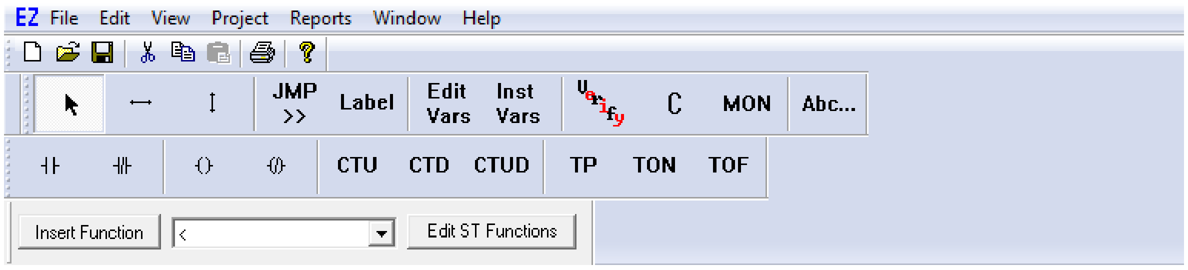


**Figure 2-3**

Each tool bar contains multiple buttons. The following describes the function of each button.

**New** Project. Opens a new blank EZ LADDER Toolkit Project Window.

**Open** Project. Browse and open an existing EZ LADDER Toolkit Project.

**Save** Project. Saves the currently selected EZ LADDER Toolkit Project.

**Cut**. Cuts (Deletes) the selected Items.

**Copy**. Copies the currently selected items to the Window's Clipboard.

**Print** Project. Opens the Print dialog for printing the EZ LADDER Toolkit Project.

**Help**. Opens the Help About dialog.

**Select Tool**. Selects individual or multiple items. Click on item to select or click and drag to select multiple items.

**Horizontal Link**. Used to draw horizontal links between functions, object and variables.

**Vertical Link**. Used to draw vertical links between functions, object and variables.

Edit Vars **Edit Variables**. Opens the Edit Variables Dialog. Variables are created, edited and deleted using this dialog box.

Inst Vars **Insert Variables**. Clicking in the ladder diagram workspace inserts a variable in that location. The inserted variable is selected from a dialog box that opens.

Verify **Verify Program**. Verifies the ladder diagram and elements are complete and do not break any rules. This is automatically done when the **COMPILE** button is clicked.

C **Compile Program**. This does an automatic verify and then compiles the ladder diagram project for the specific hardware target (controller).

MON **Monitor Mode**. This changes the EZ LADDER workspace from the Edit Mode to the Monitor Mode. The Monitor Mode is where ladder diagram projects are downloaded to and monitored on targets.

Abc... **Insert Comment**. This inserts a comment block into the ladder diagram project.

**Direct Contact**. This inserts a Direct Contact (Normally Open Contact) into the ladder diagram project workspace wherever you click.

**Negated Contact**. This inserts a Negated Contact (Normally Closed Contact) into the ladder diagram project workspace wherever you click.

**Direct Coil**. This inserts a Direct Coil (Normally Open Coil) into the ladder diagram project workspace wherever you click. Can only be placed in last column.

**Negated Coil**. This inserts a Negated Coil (Normally Closed Coil) into the ladder diagram project workspace wherever you click. Can only be placed in last column.

**Count Up Function**. This inserts a Up Counter Function into the ladder diagram project workspace wherever you click.

**Count Down Function**. This inserts a Down Counter Function into the ladder diagram project workspace wherever you click.

**Count Up and Down Function**. This inserts an Up and Down Counter Function into the ladder diagram project workspace wherever you click.

**Pulse Timer Function**. This inserts an Pulse Timer Function into the ladder diagram project workspace wherever you click.

**On Timer Function**. This inserts an ON Timer Function into the ladder diagram project workspace wherever you click.

**Off Timer Function**. This inserts an OFF Timer Function into the ladder diagram project workspace wherever you click.

This is used to insert any function (specifically those functions that do not have a quick used tool bar button. Select the function from the drop down menu and click the Insert Function button. This will place the function into the ladder diagram project workspace wherever you click.

**Edit ST Functions**. Opens the EZ LADDER Toolkits Structured Text Function Editor. This window is used for configuring structured text items. For more information on structured text, See **Chapter 26 - Structured Text**.

## Ladder Diagram Workspace

The ladder diagram workspace is the area of the screen where objects and links are placed to create the ladder diagram program. Most objects can be placed at any location in the workspace provided there is actual space available. The DIRECT coil, Negated coil, LATCH coil and UNLATCH coil are the only objects that must be placed in a particular location. They must be located in the last column (next to the right power rail). Any attempt to place one of them in another location will cause an error dialog box to be displayed.

A ladder diagram is created using *rungs*. A rung is a horizontal line of logic. EZ LADDER Toolkit allows the maximum number of rungs to be configured when the target is selected in the **Project Settings** dialog. Figure 2-4 shows the ladder diagram workspace and rungs of horizontal logic.

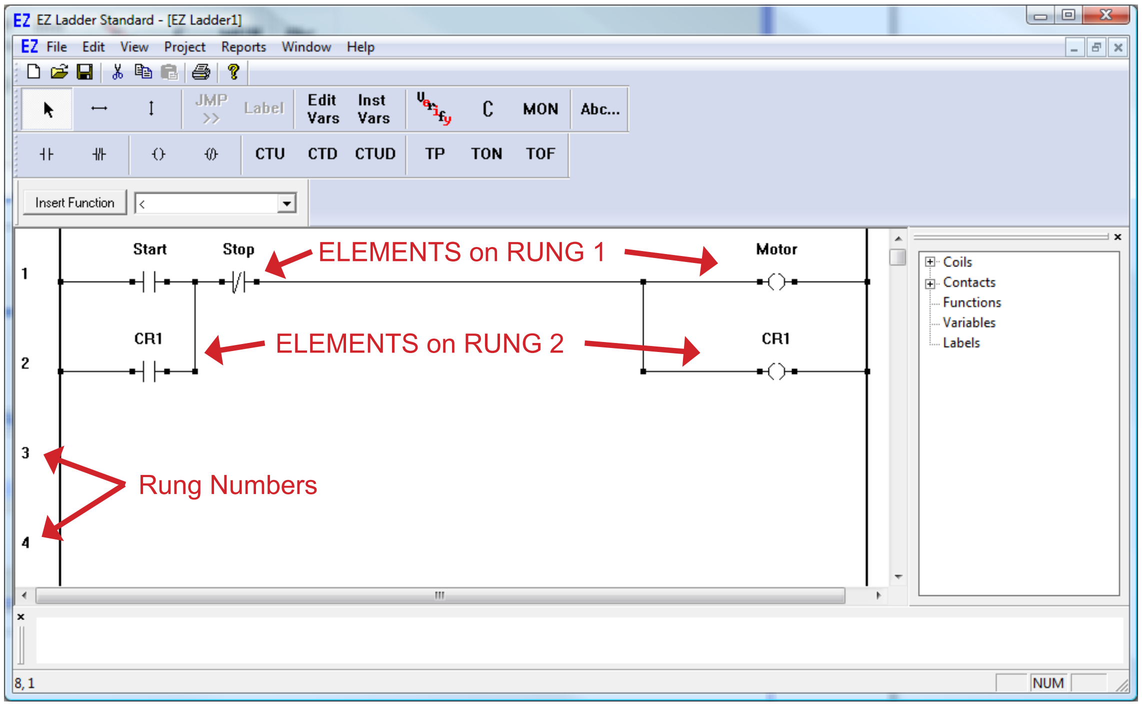


**Figure 2-4**

## Cross Reference Window Pane

EZ LADDER Toolkit provides a real-edit-time Cross Reference Window. This window provides lists of contacts, coils, variables, and functions as well as their location by rung. This quick reference provides an easy method to locate where a contact or other function is located in the ladder diagram program. Figure 2-5 shows the Cross Reference Window. Cross references are updated automatically when objects change.

This window may be used to find objects quickly. Double-click on any of the object rung numbers listed for an object or function and EZ LADDER Toolkit will locate and display that section of the ladder diagram.

The Cross Reference Window may be viewed or hidden by using the **View Cross References** Menu.

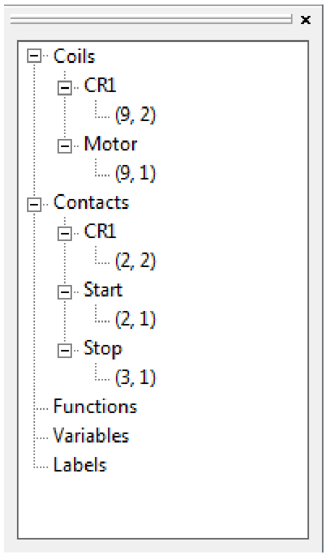
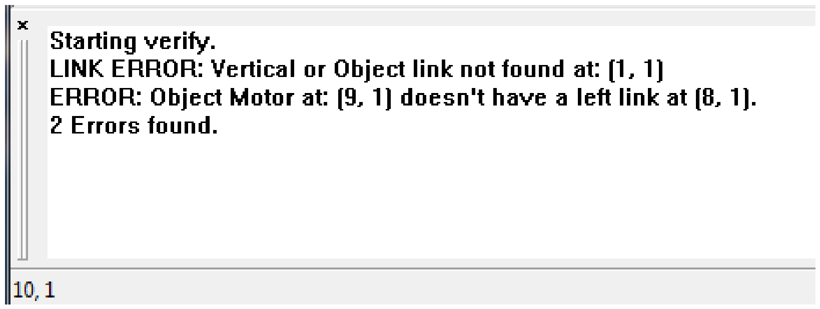


**Figure 2-5**

## Output Window Pane

EZ LADDER Toolkit provides an Output Window pane where error messages are displayed. Typically, error messages are only updated and displayed during a **Verify** operation or **Compile** operation. Figure 2-6 displays an example error identified during a compile process.

When an error message identifies a location, (i.e.: "ERROR: Object Motor at: (9,1) doesn't have a left link at (8,1)), the first number in the location refers to the column in the workspace while the second number refers to the actual rung number where the error occurs (Column, Rung).



```
Starting verify.
LINK ERROR: Vertical or Object link not found at: (1, 1)
ERROR: Object Motor at: (9, 1) doesn't have a left link at (8, 1).
2 Errors found.
```

10, 1

**Figure 2-6**

If the Output Window is not visible and an error is detected during a compilation, the Output Window will be reset to a visible state to announce the error.

## Opening Existing Ladder Diagram Project Files

Existing ladder diagram project files may be opened and manipulated by editing, downloading and saving. To open an existing ladder diagram project (.dld) file, use the File...Open menu.

Password protection is enforced if enabled in the project settings. When enabled, to open the ladder diagram file, you will be required to enter a password that will be compared against the ladder diagram file's credentials list. Only allowed permissions for the password entered will be allowed. If the password entered is not in the list, then the ladder diagram file cannot be opened or viewed. Refer to **Chapter 25 - Password Protection** for password details.

# CHAPTER 3

## Ladder Diagram Basics

This chapter provides detailed information on understanding the origin of ladder diagrams as they relate to original relay logic, basic ladder diagram symbols, power rails, links, types of circuit connections and ladder diagram functionality.

## Chapter Contents

## Relay Logic vs Ladder Diagram

Prior to the invention of the Programmable Logic Controller (PLC), control panels consisted of large numbers of relays, motor starters and other devices, wired to create the required functionality. Today, with the use of PLCs, the same functionality is achieved by drawing the circuit functionality in software; similar to the original relay logic panel wiring diagrams were drawn.

Ladder Diagram is a graphical representation of boolean equations, using contacts (inputs) and coils (outputs). The ladder diagram language allows these features to be viewed in a graphical form by placing graphic symbols into the program workspace similar to a Relay Logic electrical diagram. Both ladder diagram and relay logic diagrams are connected on the left and right sides to power rails.

A comparison of a *hard-wired* relay logic system and a *programmable* system using EZ LADDER Toolkit as the programming platform will show the similarities which make the programming using EZ LADDER Toolkit quick and easy to apply to any application.

Figure 3-1 shows a block diagram on the left and the *hard-wired* relay logic control system on the right. For easy comparison, it is divided into three sections.

**Input Devices**: Includes devices operated manually (i.e.: push buttons) and devices operated automatically (i.e.: limit switches) by the process or machine being controlled.

**Relay Control Logic**: Consists of relays interconnected to energize or de-energize output devices in response to status of input devices, and in accordance with the logic designed into the control circuit.

**Output Devices**: Consists of motor starters, solenoids, etc. which would control the machine or process.

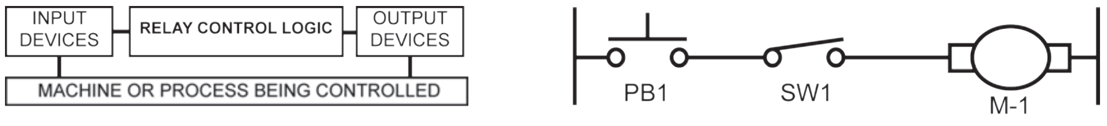


**Figure 3-1**

In place of *hard-wired* relay logic circuitry, EZ LADDER Toolkit applications are programmed using *relay-type symbology*. This symbology brings ease and familiarity to the programming while adding flexibility. Figure 3-2 is the same circuit as shown in Figure 3-1 as it is programmed using the EZ LADDER Toolkit's *relay-type symbology*.

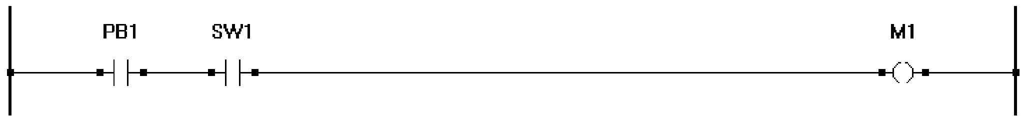


**Figure 3-2**

## Basic Ladder Diagram Symbols

In ladder diagram, all devices are represented by symbols (objects and function blocks). **Appendix A - Function Reference** provides detailed descriptions for all EZ LADDER Toolkit objects and function blocks. This section will give a basic information regarding the most commonly used objects.

### Contacts

Contacts represent two types of devices. The first is real world digital input (devices) such as limit switches, push-button switches, and proximity sensors. The second is that contacts may represent ***internal relays***; also named ***control relays*** (CRs). When acting as a real world input, the ladder diagram object will represent the current state of the real world input it is assigned. When used as an internal control relay, the contact will represent the current state of the control relay's *coil*.

Contacts are represented in the EZ LADDER Toolkit by two different objects: Direct Coil and Negated Contact.

Contacts are always shown in their at-rest or un-powered state.

#### Direct Contact ┤├

Also known as a normally open contact, the direct contact may represent real world inputs or internal relay contacts. As a real world input, when the input is energized (TRUE), it will be represented by a TRUE condition in the ladder diagram; causing power to flow through it to any following objects and function blocks. As a real world input, when the input is de-energized (FALSE), it will be represented by a FALSE condition in the ladder diagram; not allowing power to flow through it to any following objects and function blocks. When used as an internal relay, the state of the contact (TRUE or FALSE) depends upon its internal coil state.

#### Negated Contact ┤/├

Also known as a normally closed contact, the negated contact may represent real world inputs or internal relay contacts. As a real world input, when the input is energized (TRUE), it will be represented by a FALSE condition in the ladder diagram; not allowing power to flow through it to any following objects and function blocks. As a real world input, when the input is de-energized (FALSE), it will be represented by a TRUE condition in the ladder diagram; causing power to flow through it to any following objects and function blocks. When used as an internal relay, the state of the contact (TRUE or FALSE) depends upon its internal coil state and is always opposite of the Direct Contact.

### Coils

Coils represent two types of devices. The first is real world digital output (devices) such as solenoids, valves and motors. The second is that coils may represent ***internal relays***; also named ***control relays*** (CRs). When acting as a real world output, the ladder diagram object will control the current state of the real world output it is assigned. When used as an internal control relay, the coil will control the current state of the control relay's *coil*.

### Direct Coil -( )-
Also known as a normally open coil, the direct coil may represent real world outputs or internal relay coils. As a real world output, when the coil is energized (TRUE), it will be cause the real world output to be TRUE (energized). As a real world output, when the coil is de-energized (FALSE), it will cause the real world output to be FALSE (de-energized). When used as an internal relay, it controls it's contact(s) state.

### Negated Coil -(/)-
Also known as a normally closed coil, the negated coil may represent real world outputs or internal relay coils. As a real world output, when the coil is energized (TRUE), it will cause the real world output to be FALSE (de-energized). As a real world output, when the coil is de-energized (FALSE), it will be cause the real world output to be TRUE (energized). When used as an internal relay, it controls it's contact(s) state and is always opposite of the Direct Coil.

## Power Rails and Links

### Power Rails

As previously discussed, an EZ LADDER Toolkit ladder diagram contains objects (contacts, coils and function blocks). For the ladder diagram to operate correctly, each rung must be complete on each side by connecting it to the **left power rail** and the **right power rail**. This is required because all objects in a rung must have a power source (the left power rail) to provide power to the objects and a common return (right power rail) to complete the circuit.

Power rails run the entire length of an EZ LADDER Toolkit project. Figure 3-3 shows a typical ladder diagram rung and identifies the power rails. Please note, the rung is connected to both the right and left power rails.

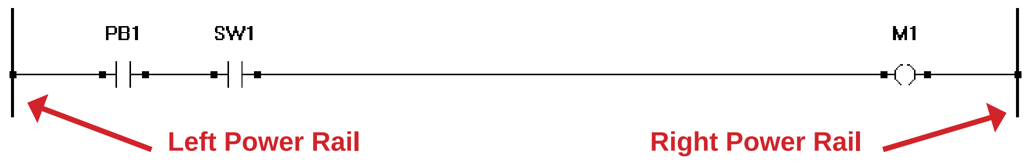


**Figure 3-3**

### Links

As discussed previously, a rung must be complete and connected to both power rails for proper operation. Links (Horizontal and Vertical) are used to connect objects and function blocks to other objects and function blocks as well as to power rails. Horizontal Links connect devices horizontally or on the same rung. Vertical Links connect devices on different rungs. Consider each link like an electrical wire that is needed to connect devices in a circuit. Figure 3-4 identifies the types of links.

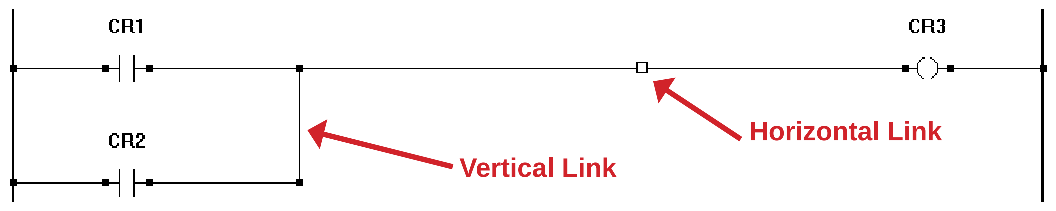


Figure 3-4

## Connection Types

As seen in previous sections, the use of power rails, horizontal and vertical links creates a wide variety of ways to draw ladder diagram circuits. Below are some typical connection types. They are created by using horizontal and vertical links.

### Simple Series Connection

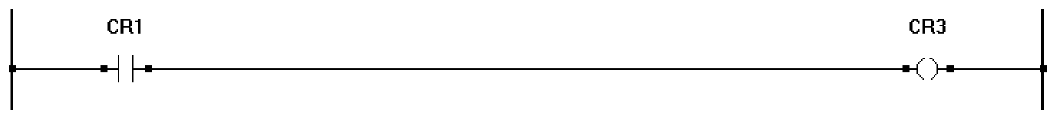


Figure 3-5

### Multiple Device Series Connection

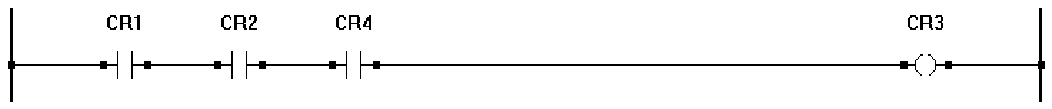


Figure 3-6

### Parallel Connection

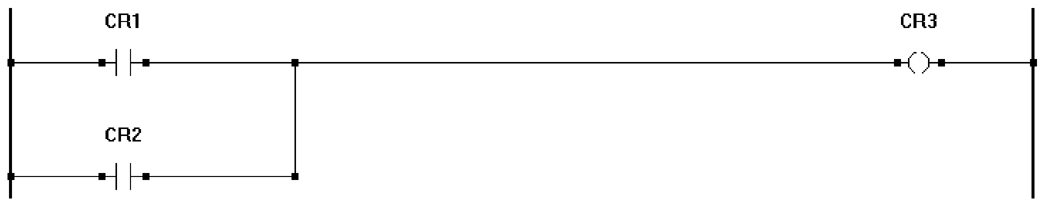


Figure 3-7

## Complex Series / Parallel Connection

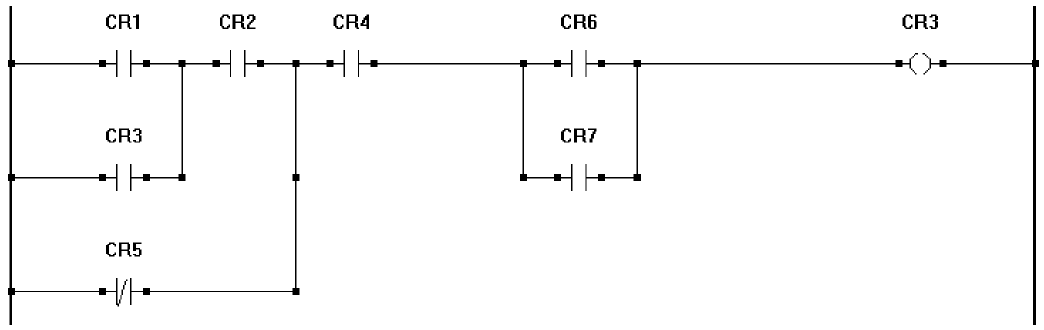


**Figure 3-8**

## Understanding Ladder Diagram Functionality

When a ladder diagram is installed on a PLC or other controller, it will *scan* the program from top to bottom and left to right. A *scan* is similar to reading a page. A page is read from top to bottom reading each line left to right. One complete reading of the program is considered a ***scan***. The larger the ***scan time*** (one complete read cycle), the less often any real world I/O devices are monitored and controlled. ***Scan time*** is an important consideration in the design of a ladder diagram. This scan time may be viewed in the Monitor Mode when running a ladder diagram with a hardware target.

Figure 3-9 diagrams the functionality and order which a ladder diagram functions.

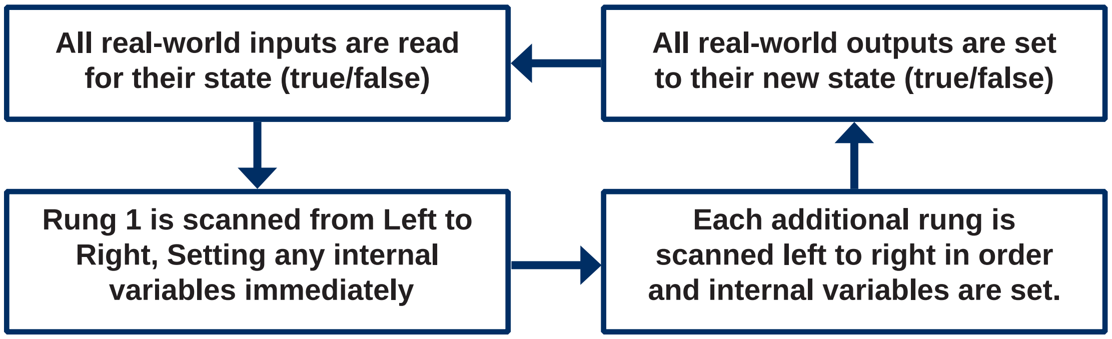

All real-world inputs are read for their state (true/false)
All real-world outputs are set to their new state (true/false)
Rung 1 is scanned from Left to Right, Setting any internal variables immediately
Each additional rung is scanned left to right in order and internal variables are set.

**Figure 3-8**

# CHAPTER 4

## Configuring Targets

This chapter provides basic information and steps required to identify, select and configure hardware targets (actual hardware controllers or PLCs) in the EZ LADDER Toolkit.

## Chapter Contents

## Understanding Targets

A *Target* is the term used in the EZ LADDER Toolkit to describe the actual electronic hardware controller or Programmable Logic Controller (PLC) to which the ladder diagram is specifically written to operate on. Generally, most ladder diagrams can be utilized on different hardware targets with only minor changes to the ladder diagram program itself.

Each hardware target is unique in that each usually have differing number of inputs, outputs, analog I/O and other features. Due to the differences, in each EZ LADDER Toolkit ladder diagram project, the actual target must be identified (selected) and it's optional features (if any) installed and configured properly. Refer to **Chapter 23 - Hardware Targets** for specific target details for each supported target.

It is important to understand that using PLC targets such as, Harsh Environment Controllers, etc typically have some features that require additional configuration after the actual hardware target is selected; while targets such as the PLC on a Chip require you to install and configure each and every I/O point, device and feature you intend to use.

Failure to correctly select, install or configure a feature in the Project Settings may result in the target not operating as anticipated or features and functions not showing available for the target.

For a target to be able to use the compiled ladder diagram program created using the EZ LADDER Toolkit, it must also have its **kernel** installed. The kernel is the hardware target's basic operating system or bios. The kernel is required before any compiled programs can be installed. The kernels for targets are automatically installed on your computer when the EZ LADDER Toolkit is installed. They are typically found in C:\Program Files\Divelbiss\EZ Ladder\Kernel\.

Hardware targets ship from the factory **without a kernel** installed. The kernel must installed prior to downloading the first ladder diagram program. The target is shipped without a kernel to provide greater flexibility in version control.

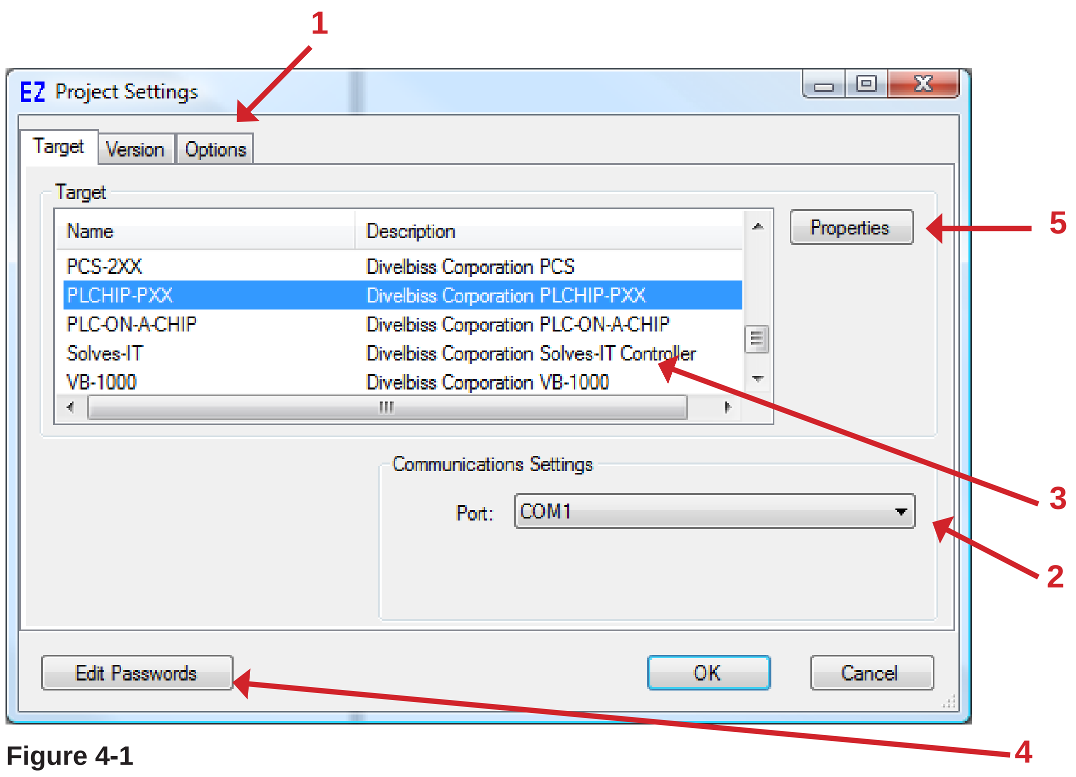
## The Project Settings Window

To create a ladder diagram project in EZ LADDER Toolkit, you must first select the hardware target. If you attempt to place any ladder diagram objects in the workspace prior to selecting a target, the Project Settings Window will open requiring you to select the target automatically. EZ LADDER Toolkit uses the target selection to filter and display only ladder diagram objects and functions supported by the selected target.

Only the actual hardware target that will be used should be selected. Selecting a different target will allow the use of objects and function blocks that may not be supported by the actual hardware target as well as not allow use of objects and function blocks that are supported.

To select the target, either try to place a ladder diagram object in the workspace or use the **Project Menu** and click **Settings.** The Project Settings Window / Dialog box will open. Figure 4-1 is an example of the Project Settings Window and identifies the main components of it.

While all targets are displayed in the Project Settings (M-Series or P-Series), the focus will be on the P-Series products and target configurations.



Target

| Name | Description |
|---|---|
| PCS-2XX | Divelbiss Corporation PCS |
| PLCHIP-PXX | Divelbiss Corporation PLCHIP-PXX |
| PLC-ON-A-CHIP | Divelbiss Corporation PLC-ON-A-CHIP |
| Solves-IT | Divelbiss Corporation Solves-IT Controller |
| VB-1000 | Divelbiss Corporation VB-1000 |

Communications Settings

Port: COM1

**Figure 4-1**

## Target Tab Settings

1. **Project Setting Tabs**: Select the appropriate tab to configure target settings. Figure 4-1 represents the ***TARGET*** tab, Figure 4-2 represents the ***VERSION*** tab, and Figure 4-3 represents the ***OPTIONS*** tab. Clicking on a tab selects the tab for viewing.

2. **Communications Settings**: Select the serial port on the computer that will be used to communicate to the target. These settings are used to connect, download and monitor ladder diagram programs running in EZ LADDER's program run and monitor mode. The baud rate for all hardware targets is automatically set in EZ LADDER and cannot be changed.

3. **Target List**: This is a list of all targets supported by your version of EZ LADDER Toolkit. Click on the target name to select.

   When selecting some targets, an additional dialog may open depending on the hardware target selected.

4. **Edit Passwords**: This button opens Password Setup Dialog box. This box is used to configure a Master Password for the project and additional user / passwords and permissions for others to view, edit, etc. This button will not be visible unless a target is selected.

5. **Properties**: When a target has been selected, the **PROPERTIES** button may appear (target specific). If this button appears, clicking the **PROPERTIES** will allow additional configurations for the selected target. Properties may include: Model Number and installing / removing additional target supported features (Modbus, OptiCAN, Ethernet, etc). This button will not be visible unless a target is selected.

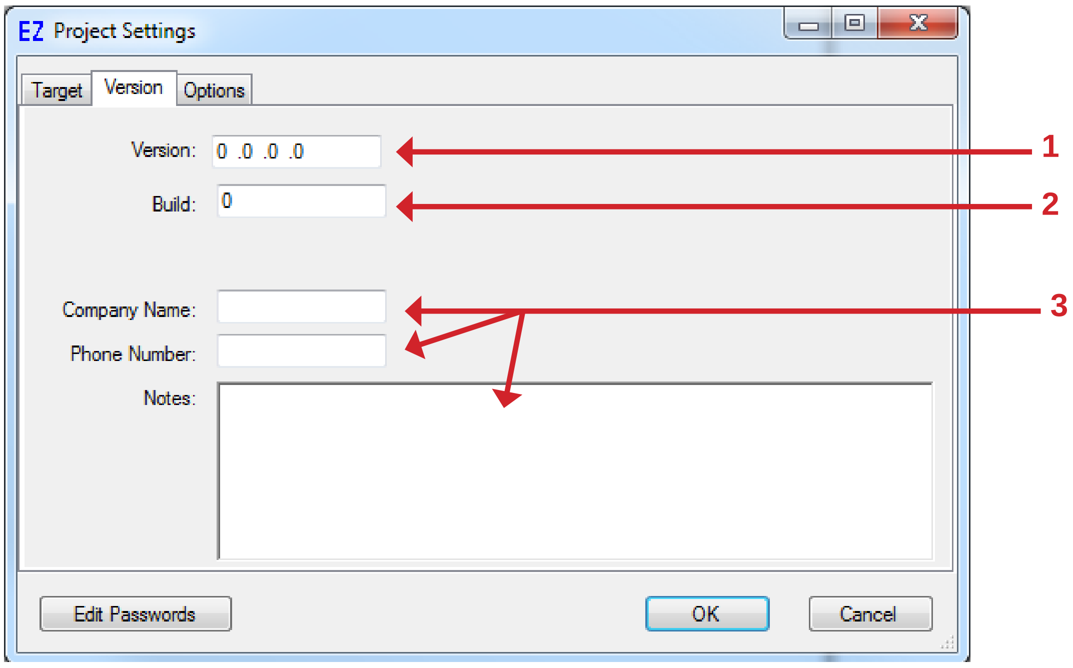
## Version Tab Settings

The Version Tab will display the current build and version of the ladder diagram that is currently active in the EZ LADDER Toolkit. The build and version information is useful when determining if a program version is current. Figure 4-2 shows the Version settings tab of the Project Setting Window. Version setting may be changed in this window, if required.

1. **Version Number**: A version number for the ladder diagram may be entered here. This number will not change automatically. It must be manually adjusted for each revision or release of the ladder diagram project.

2. **Build Number**: The current build number is displayed here. Each time the ladder diagram project is *Compiled*, the build number automatically increments. This number may be over-written in this window if needed.

3. **Company Name**: **Phone Number:** **Notes:** Optional information that can be entered about the company and a section for notes.



Version: 0 .0 .0 .0

Build: 0

Company Name:

Phone Number:

Notes:

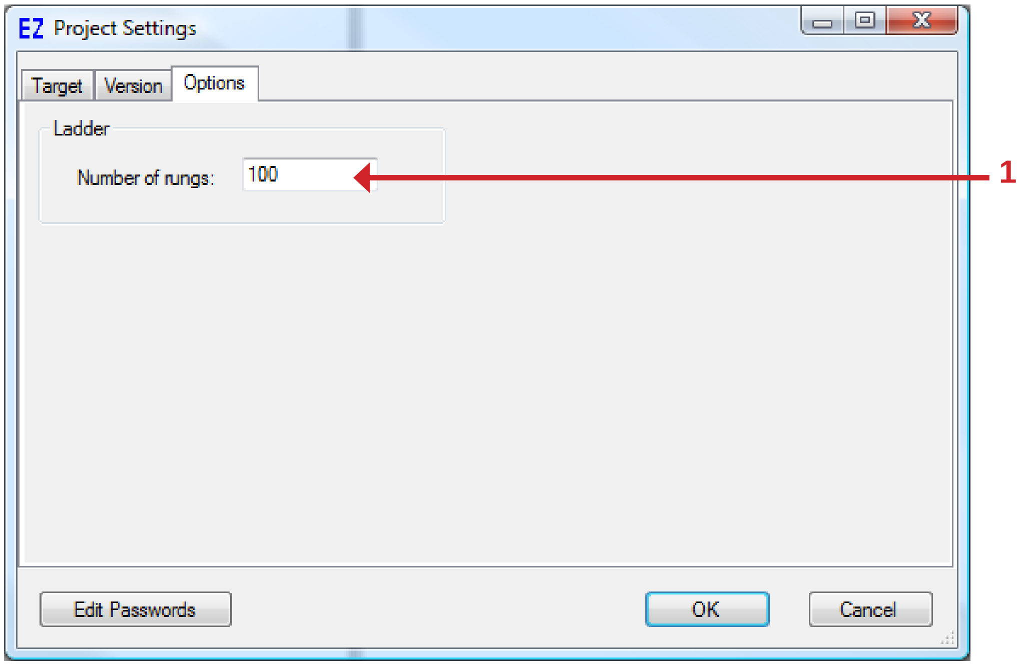
Figure 4-2

## Options Tab Settings

The Options Tab will display the currently selected options and preferences. Some of these options are target specific while others are standard. Figure 4-3 shows the Options settings dialog box.

1. Number of Rungs: This is where the maximum number of rungs in the ladder diagram is specified. The default number is 100 rungs.

Inserting or Deleting rungs in an EZ LADDER Toolkit ladder diagram project will change this number accordingly. This number should be considered a starting number of rungs. If the number of rungs is not sufficient for the program size, return to this dialog and adjust the number of rungs. The number of rungs may be adjusted at any time.


EZ Project Settings

Target | Version | Options

Ladder

Number of rungs: 100

**Figure 4-3**

## Selecting the Hardware Target

As discussed in a previous section, using the dialogs, select the target from the list. If required, select the actual model number, configure any features that you wish to use and click **OK**. **Chapter 23 - Hardware Targets** includes additional information for each supported hardware target.

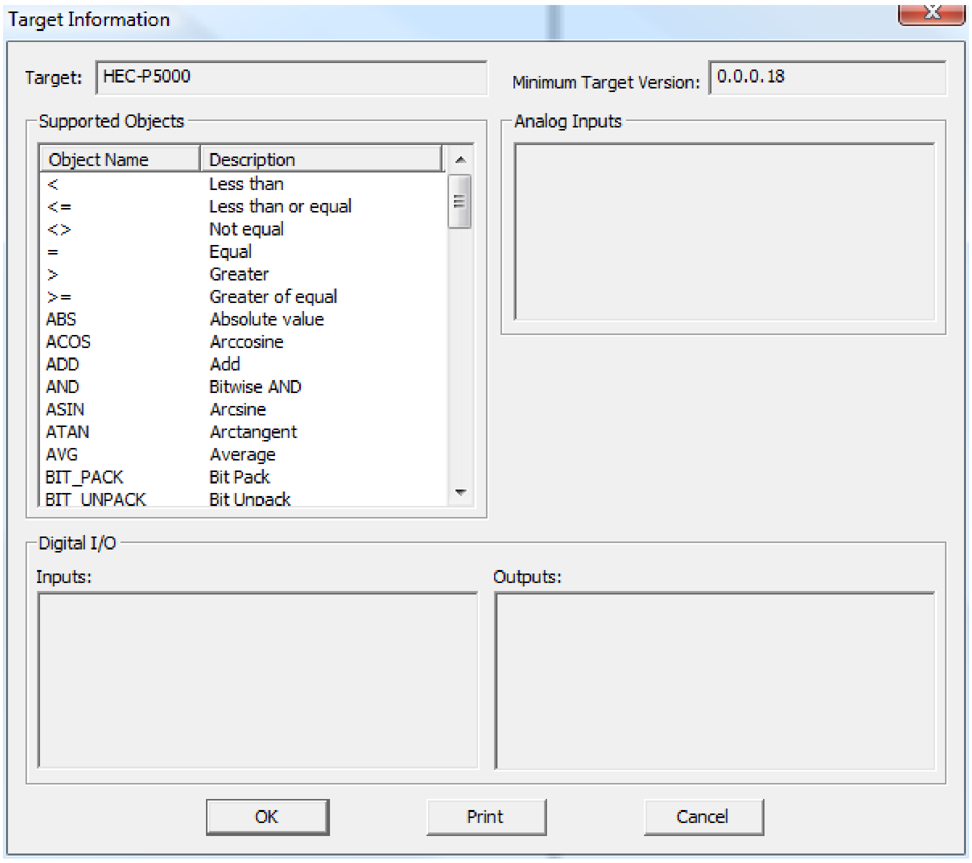
To save the target selection, you must save the ladder diagram project using the **Save** or **Save As** menu items. Hardware Target Selections are for the currently open and active EZ LADDER Toolkit ladder diagram project only. For each new project, you must repeat the hardware target selection and configuration process.

## Viewing Target Information

EZ LADDER Toolkit provides includes a built-in quick reference tool to identify what I/O, Analog and functions are supported by a target. The first step is selecting a hardware target as previously shown. To view the target supported features, from the **View** menu, click **Target Information**. The Target Information window will open as shown in Figure 4-4.

This window identifies: the Target Name, Minimum Target Kernel Version that is needed for this version of EZ LADDER Toolkit, Supported Objects and Function Blocks, Analog Inputs, Digital Inputs and Digital Outputs.

The Target Information is printable using the **PRINT** button.



Target: HEC-P5000

Minimum Target Version: 0.0.0.18

Supported Objects

| Object Name | Description |
|---|---|
| < | Less than |
| <= | Less than or equal |
| <> | Not equal |
| = | Equal |
| > | Greater |
| >= | Greater of equal |
| ABS | Absolute value |
| ACOS | Arccosine |
| ADD | Add |
| AND | Bitwise AND |
| ASIN | Arcsine |
| ATAN | Arctangent |
| AVG | Average |
| BIT_PACK | Bit Pack |
| BIT_UNPACK | Bit Unpack |

**Figure 4-4**

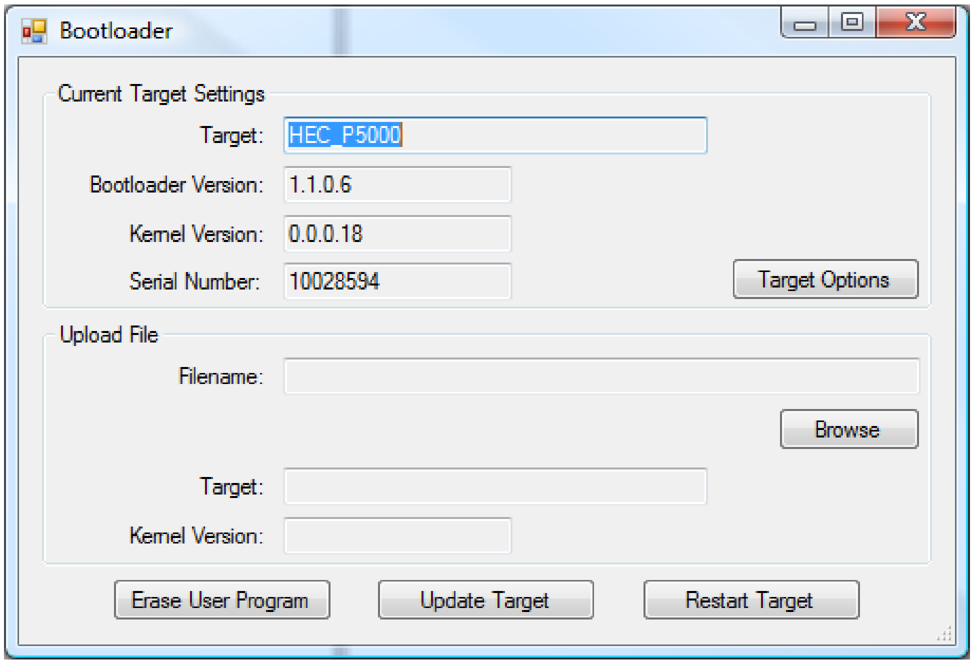
## Updating / Installing Target Kernels

As new hardware targets, functions and features are added to the EZ LADDER Toolkit and new versions of EZ LADDER Toolkit are developed and released, to take advantage of newer features, it will be necessary to update the actual target's kernel with newer version.

These same steps may be taken to install a kernel in a target that is new as all new targets from the factory do not have kernels installed.

EZ LADDER Toolkit provides an easy way to update the kernel on the hardware target.

1. Obtain the new kernel for the target (provided by Divelbiss via CD, USB, e-mail or download).
2. Start EZ LADDER Toolkit and open any project that uses the target or create a new project with the actual hardware target selected. This project must have at least one rung of ladder. Compile the program if not already compiled.
3. Verify the Serial Port Settings and connect the target to the computer.
4. Enter the Monitor Mode.
5. From the **Project** menu, select **Bootloader**.
6. EZ LADDER will connect to the target and the Bootloader dialog will open showing the current version of the target's kernel (if any). It will also display the target's bootloader version. See Figure 4-5.


Bootloader

Current Target Settings
Target: HEC_P5000
Bootloader Version: 1.1.0.6
Kernel Version: 0.0.0.18
Serial Number: 10028594

**Figure 4-5**

6. Click the **ERASE USER PROGRAM** button to erase the ladder diagram project on the target (if any).

   It is highly recommended that the user program be erased before upgrading a kernel.

7. Click **BROWSE** and select the kernel file for the hardware target. The dialog will update showing the selected kernel file version in the Upload File section of the Bootloader dialog box.

8. To update or install the new kernel, click **UPDATE TARGET**. A status box will appear indicating the status of the kernel installation. During this, the new kernel is being installed. This may take several minutes.

   When updating or installing a kernel, DO NOT REMOVE the CABLE or the POWER. If interrupted during this process, the target will be corrupted and return to a bootloader mode. You must repeat all the above steps again.

   Only the correct target's kernel may be installed into a target. The target is checked against the kernel automatically and will display an error if the wrong kernel is selected and an update is attempted. If a wrong kernel is somehow loaded, contact Divelbiss Technical Support for help regarding removing incorrect kernels.

## Recovering Communications

EZ LADDER Toolkit provides additional target utilities that may be used correct actual target problems and restore communication from EZ LADDER to the target. Although rare, occasionally, targets get corrupted and communications cannot be established using normal methods. This can be caused by not erasing a ladder diagram prior to upgrading kernels, wrong kernels installed and interruptions during kernel installations.

The bootloader is installed at the factory and can not be updated outside the factory environment.

### When Unable to Connect to the Target

The following steps may be taken if you can verify the connection problems is with the actual hardware target unit specifically (another unit connects with the same setup and program).

1. Start EZ LADDER Toolkit and open any project that uses the target or create a new project with the actual hardware target selected. This project must have at least one rung of ladder. Compile the program.

3. Verify the Serial Port Settings and connect the target to the computer.

4. Enter the Monitor Mode.

5. Press the **F11** key on your computer's keyboard. The dialog box in Figure 4-6 will open.

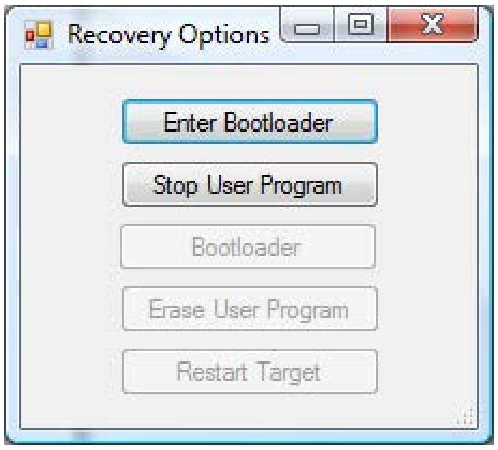


**Figure 4-6**

6. Disconnect power from the hardware target.

7. Click the **ENTER BOOTLOADER** button in the dialog box. A timing dialog box will appear. This is amount of time that is remaining to re-apply power to the hardware target.

8. Apply power to the hardware target. If the time has elapsed, repeat steps 6-8 again. The hardware target will now allow bootloader operations (other buttons are now active).

9. Choose the correct option to try and resolve your target issue.

| | |
|---|---|
| **Bootloader**: | Bootloader will open the bootloader dialog box for updating kernels. |
| **Erase LD Program**: | Erases the ladder diagram project from the hardware target's memory. In the event the program is hanging and preventing a normal connection, this will erase the program to allow a normal connect. |
| **Restart Target**: | Causes the hardware target to reboot. This is required when all other bootloader actions have been completed. Without the restart, the kernel will still not connect normally. |

Using the Restart Target is the same as resetting the power to the hardware target. Both will cause the target to restart and operate normally.

If the incorrect kernel has been installed or there is a corrupt kernel installed, from the bootloader screen, press the F11 key. A dialog will appear asking if you are sure you want to erase the kernel. To erase the kernel and reset the target to a blank target, click the **YES**. If you don't wish to erase the kernel, click **NO**. This is how the kernel may be erased.

When a Kernel or User Program is erased, there is no recovery. Erasing a Kernel / User Program should only be done after verification that these items are available to install / re-install.

# Target Utilities

When communications is present between EZ LADDER and the hardware target, there are additional options / utilities available when connected to the target.

## When Able to Connect to the Target

If you can connect normally to the target, there are only a few additional utilities available in the EZ LADDER Toolkit.

1. If the target is connected to normally, press the **F11** key on your computer's keyboard. The Device Properties dialog box will appear as in Figure 4-7.

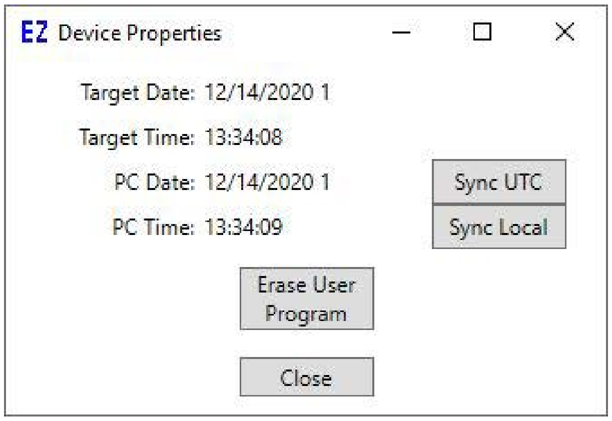


**Figure 4-7**

If the actual hardware target does not support a Real Time Clock, then an error dialog box may appear if you were successfully, connected to the hardware target prior to press the **F11** button. Click **OK** to continue.

From this dialog, you can compare the actual computer time and date to the current time and date set on the hardware target. If you wish to synchronize the time (set the hardware target to the computer time), click the **SYNC UTC** button (for UTC time) or **SYNC LOCAL** (for local date/time). The times should now be synchronized. For more information on date, time and real time clock (including syncing), see **Chapter 12 - Real Time Clock**.

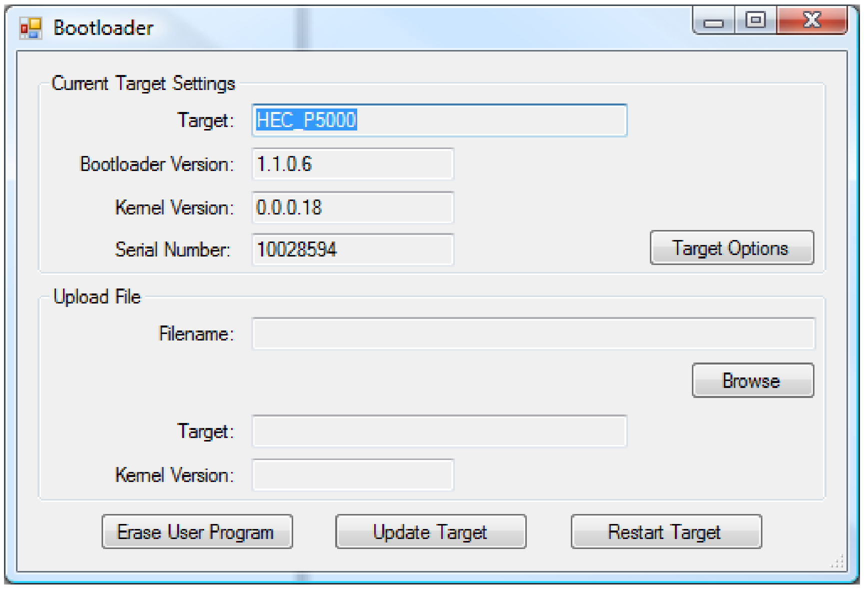
The ladder diagram project can be erased from this dialog by pressing the **ERASE USER PROGRAM** button.

Use caution when deleting the ladder diagram project from the target. There is no Undo. To reload the hardware target, the original ladder diagram project must be opened, compiled and reloaded to the target.

## Bootloader Target Options & Configurations

When communications is present between EZ LADDER and the hardware target, there are additional options available in the Bootloader dialog based on the actual target connected. These options are target dependent and the dialogs associated are used to configure the options.

In the Bootloader dialog, click the TARGET OPTIONS button. See Figure 4-8. The Target Options dialog box will open.



Target: HEC_P5000
Bootloader Version: 1.1.0.6
Kernel Version: 0.0.0.18
Serial Number: 10028594
Filename:
Target:
Kernel Version:

**Figure 4-8**

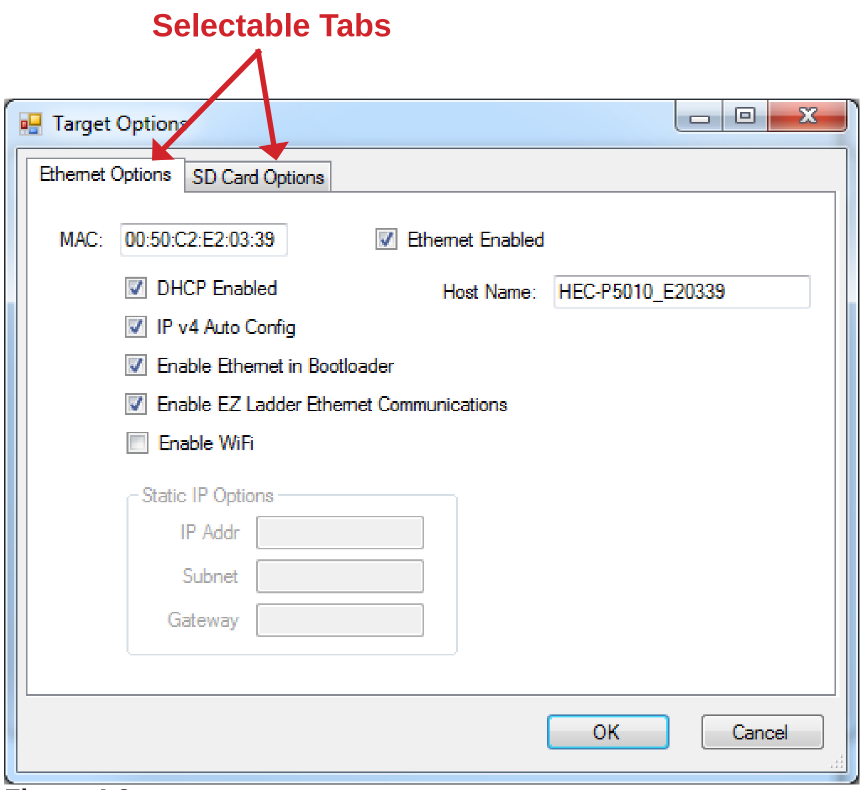
The Target Options dialog has three tabs: Ethernet Options, USB Options and SD Card Options. These items are additional configurations that are required if using any of these features on your target. See Figure 4-9.

For details on the Ethernet settings and port use, see **Chapter 19 - Ethernet**. This dialog displays information regarding the Ethernet configuration including MAC Address, etc and provides user configuration items such as Host Name and DHCP settings.

For details on the SD Card Options settings and use, see **Chapter 20 - SD Card Support**.

When all the configurations required have been completed, click OK to close the Target Options dialog. This will save these options. Click the RESTART TARGET button to restart the target and close the Bootloader dialog.

If the incorrect kernel has been installed or there is a corrupt kernel installed, from the bootloader screen, press the F11 key. A dialog will appear asking if you are sure you want to erase the kernel. To erase the kernel and reset the target to a blank target, click the **YES**. If you don't wish to erase the kernel, click **NO**. This is how the kernel may be erased.



MAC: 00:50:C2:E2:03:39
☑ Ethernet Enabled
☑ DHCP Enabled
Host Name: HEC-P5010_E20339
☑ IP v4 Auto Config
☑ Enable Ethernet in Bootloader
☑ Enable EZ Ladder Ethernet Communications
☐ Enable WiFi

Static IP Options
IP Addr
Subnet
Gateway

Figure 4-9

## Installing Target Devices / Features

For greater flexibility, some target features and devices are not automatically installed on the actual hardware target when the target is selected as previously shown. Devices / Features automatically installed is entirely dependent upon the actual target selected. The P-Series PLC on a Chip™ for example only installs very minimum features while the HEC-P5XXX controller installs some features and devices, but leaves others to the actual user to install and configure.

For devices and features that are not automatically installed, they must be installed manually. The following steps are shown as basics for installing additional devices and features. These are general steps only and some variances will occur based on the target and devices to be installed.

To select the target, either try to place a ladder diagram object in the workspace or use the **Project Menu** and click **Settings.** The Project Settings Window / Dialog box will open. Figure 4-1 is an example of the Project Settings Window and identifies the main components of it.

Select (highlight) the target and click the **PROPERTIES** button. The *XXXX (XXXX represents the target) Target Properties* window will open. Using the *DCPN drop down menu*, select the model number of the target (if available). Refer to Figure 4.10.

This window consists of an Installed Devices pane, DCPN drop down menu and four buttons used to configure devices.

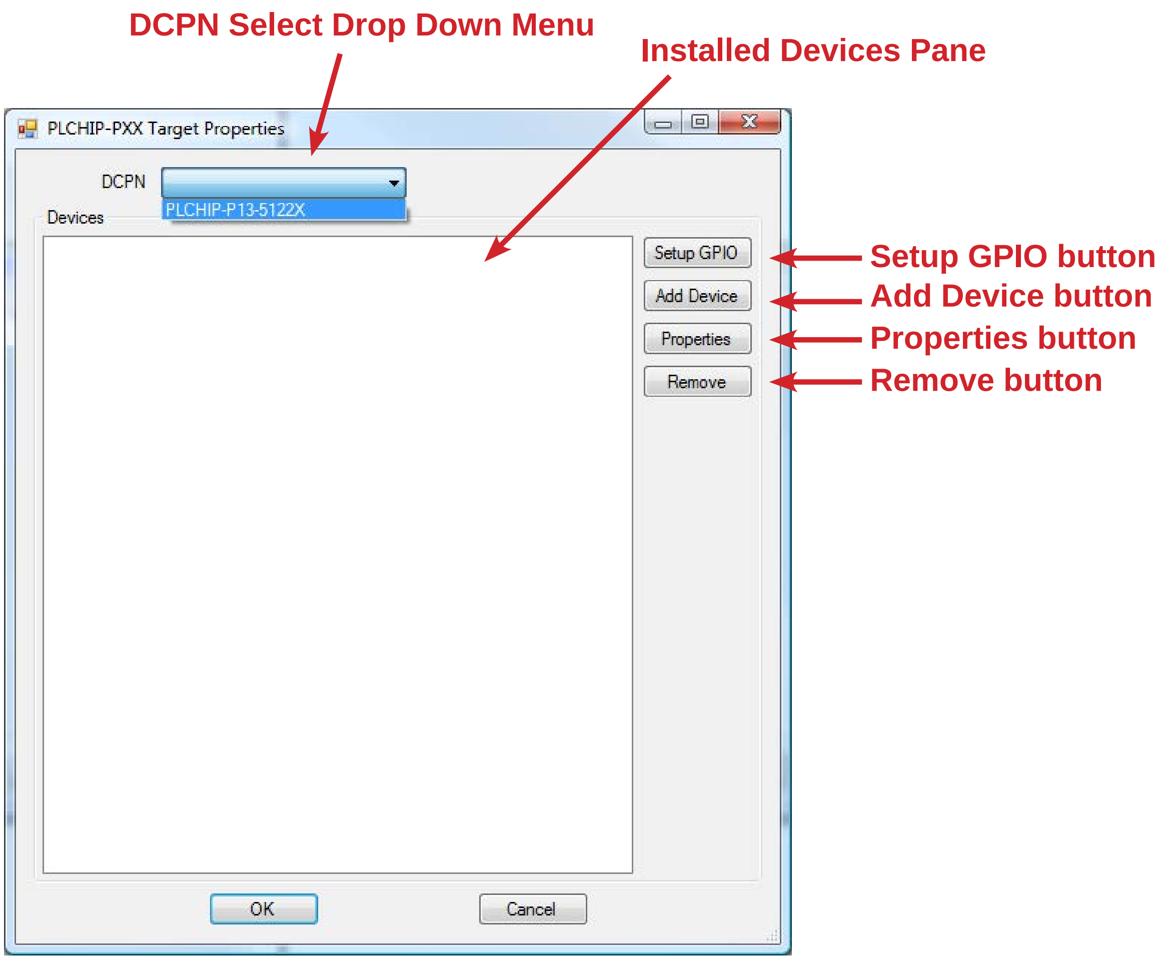


**Figure 4-10**

The Devices Pane shows all the currently installed devices and features for the target. Add new devices are added or devices are removed, this pane will update accordingly.

## Setup GPIO

The **SETUP GPIO** button is used to add / remove General Purpose I/O (GPIO), also known as digital inputs and digital outputs. Any digital I/O to be used in the ladder diagram program must be installed in the target's devices. To configure the digital I/O (GPIO), click the **SETUP GPIO** button. The targets GPIO window / dialog box will open. Refer to Figure 4.11. Many product's GPIO pins are automatically configured when the product is selected from the Project...Settings Menu.

GPIO does not appear in the Devices pane when installed.

To install digital inputs, select *GPIO Pins* (listed by name and actual PLC on a Chip pin) that are required and click the **ADD TO INPUTS** button. These GPIO pins will move from the *GPIO Pins pane* to the *Inputs Pane*. If Inputs were added incorrectly or need to be removed, select them in the *Inputs pane* and click the **REMOVE FROM INPUT(S)** button.

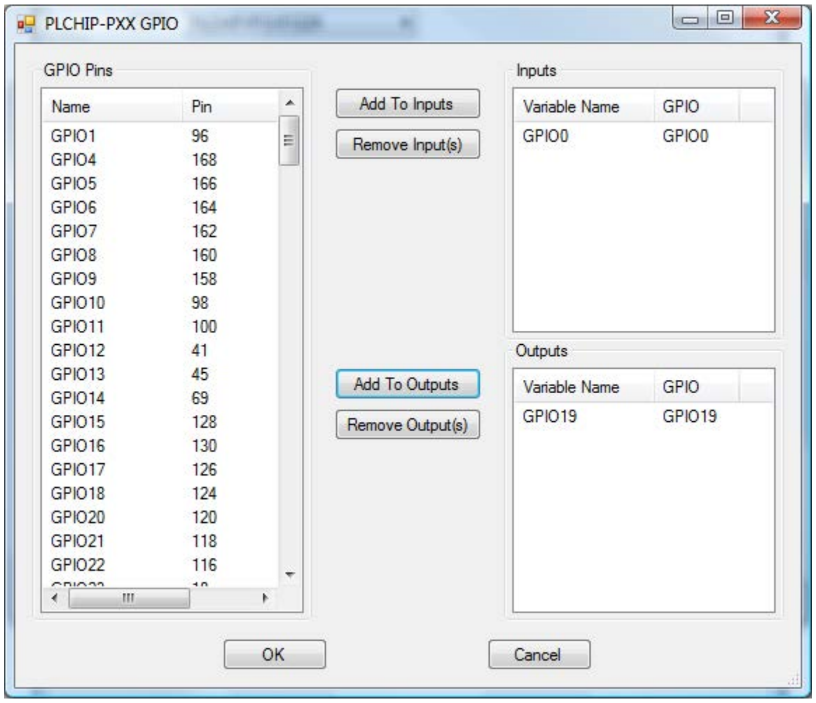
For targets that automatically configure the GPIO settings, it is not possible to change any of the GPIO assignments.

To install digital outputs, select *GPIO Pins* (listed by name and actual PLC on a Chip pin) that are required and click the **ADD TO OUTPUTS** button. These GPIO pins will move from the *GPIO Pins pane* to the *Outputs Pane*. If Outputs were added incorrectly or need to be removed, select them in the *Outputs pane* and click the **REMOVE FROM OUTPUT(S)** button.

When all the GPIO have been installed, click the **OK** button to save the settings and close the dialog / window.

GPIO pins are designed to be configured as Inputs or as Outputs, but cannot be both simultaneously. Based on the Target selected, there may be no GPIO configuration available.

When PWM (Pulse Width Modulator) is configured, the GPIO pins associated with it are not available as GPIO as they are used as PWM. GPIO pins can only be used as either Outputs or PWM, not both simultaneously in the same program.



GPIO Pins

| Name | Pin |
|---|---|
| GPIO1 | 96 |
| GPIO4 | 168 |
| GPIO5 | 166 |
| GPIO6 | 164 |
| GPIO7 | 162 |
| GPIO8 | 160 |
| GPIO9 | 158 |
| GPIO10 | 98 |
| GPIO11 | 100 |
| GPIO12 | 41 |
| GPIO13 | 45 |
| GPIO14 | 69 |
| GPIO15 | 128 |
| GPIO16 | 130 |
| GPIO17 | 126 |
| GPIO18 | 124 |
| GPIO20 | 120 |
| GPIO21 | 118 |
| GPIO22 | 116 |

Inputs

| Variable Name | GPIO |
|---|---|
| GPIO0 | GPIO0 |

Outputs

| Variable Name | GPIO |
|---|---|
| GPIO19 | GPIO19 |

**Figure 4-11**

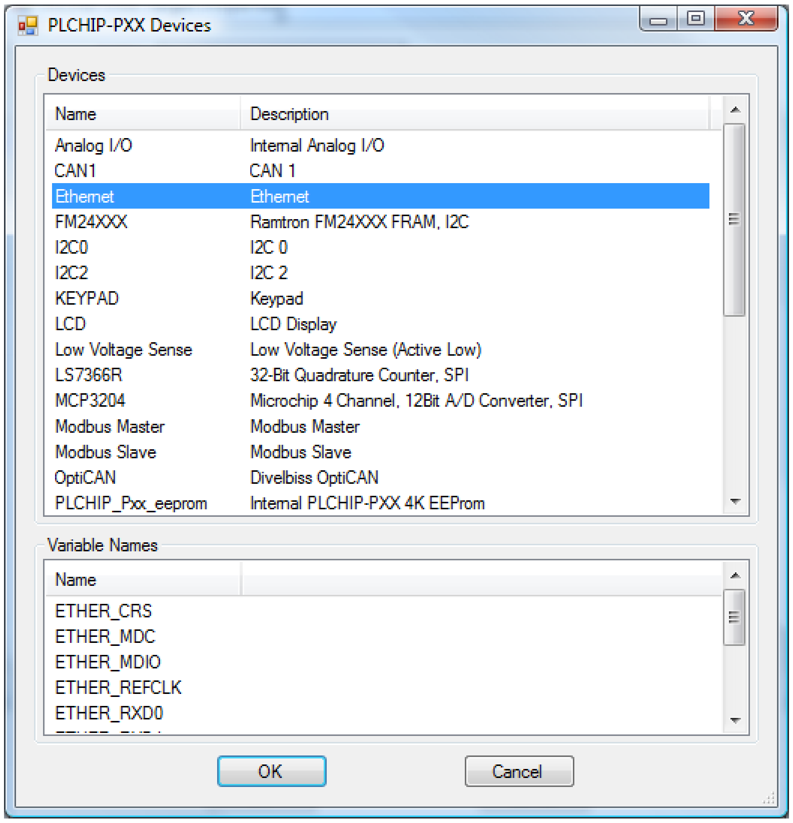
## Adding Devices

The **ADD DEVICE** button is used to add / remove devices such as serial ports, Ethernet, Quadrature Inputs, etc. Any device to be used in the ladder diagram program must be installed in the target's devices. To add ad device, click the **ADD DEVICE** button. The targets Devices window / dialog box will open. Refer to Figure 4.12.

All supported devices are shown for the target.

The window / dialog is divided into two panes, *Devices* and *Variable Names*. As devices are selected, the associated variables that will be installed with them are shown in the *Variable Names pane*. Select a device from the *Devices pane* by clicking on the Name (highlight) and click **OK** to install it.

Depending upon the device, additional dialog boxes may appear with additional configuration items that are required for the device before it can operate. Select the appropriate information for these dialog boxes. Additional support is given in later chapters that are specifically written for some devices such as Ethernet, etc.



| Name | Description |
|---|---|
| Analog I/O | Internal Analog I/O |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |

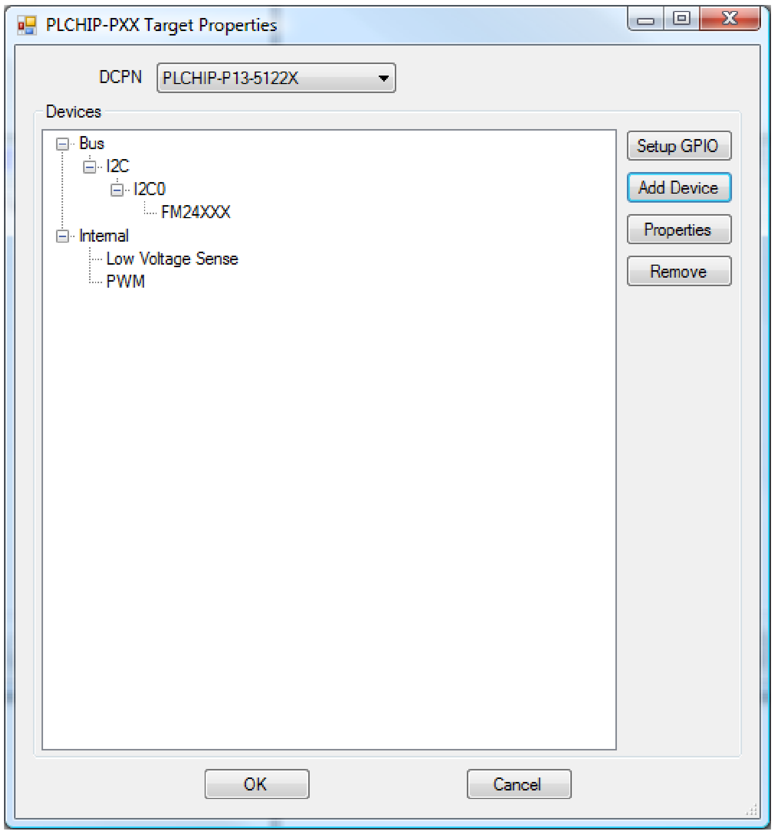
| Name |
|---|
| ETHER_CRS |
| ETHER_MDC |
| ETHER_MDIO |
| ETHER_REFCLK |
| ETHER_RXD0 |

**Figure 4-12**

Some device may be required to be installed prior other devices. For example, before Modbus Slave can be installed, ether the Ethernet or a Serial Port (UART) must be installed previously as when installing the Modbus Slave, the port must be selected. This is only an example, other situations exist similar to this. Other examples may be adding the I2C port before a FM24XXX FRAM, etc.

When PWM (Pulse Width Modulator) is configured, the GPIO pins associated with it are not available as GPIO as they are used as PWM. GPIO pins can only be used as either Outputs or PWM, not both simultaneously in the same program.

Figure 4-13 is the Target Properties Window with some devices installed. When all the devices required have been configured, click **OK** to save the settings and close the window. Be sure to save the ladder diagram program after making target changes.


PLCHIP-PXX Target Properties

DCPN: PLCHIP-P13-5122X

Devices
- Bus
  - I2C
    - I2C0
      - FM24XXX
- Internal
  - Low Voltage Sense
  - PWM

**Figure 4-13**

# CHAPTER 5

## Creating Ladder Diagram Projects

This chapter provides basic information and understanding to create ladder diagram projects using EZ LADDER Toolkit including variables, variable types, inserting variables, inserting objects and functions, bit addressable variables, drawing links, inserting and deleting rungs, saving ladder diagram projects and verifying and compiling ladder diagram projects.

## Chapter Contents

# Creating Ladder Diagram Projects

When EZ LADDER Toolkit is opened, it will automatically create a *new* blank project and it's corresponding workspace as shown in Figure 5-1. A new project may be created at any time by choosing **New** from the **File** menu.

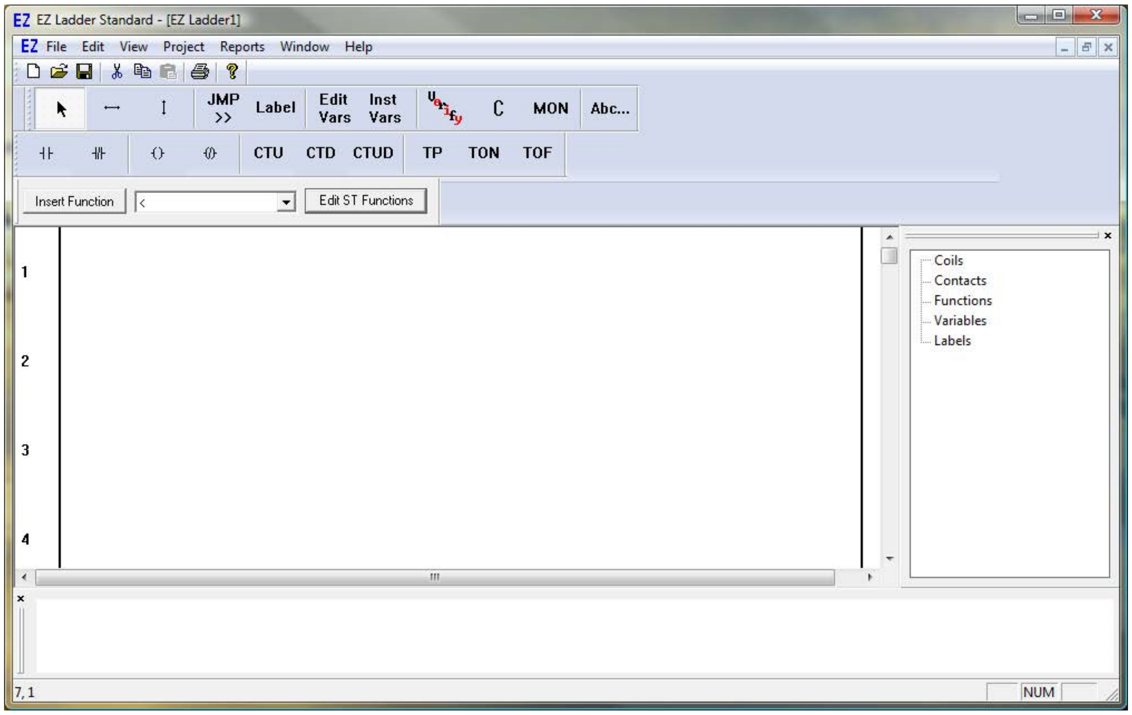


**Figure 5-1**

Before adding any objects, functions or variables to the new workspace, the Target must be selected and configured according to the target needs and your program requirements. Select your target by choosing **Settings** from the **Project** menu. See **Chapter 4 - Configuring Targets** and **Chapter 23 - Hardware Targets** for details.

When configuring your target, it is recommended to only install and configure features that are intended to be used. Installing unused features may degrade target performance.

## Understanding Objects & Variables

Ladder diagram projects in EZ LADDER Toolkit are comprised of a combination of objects, function blocks, variables and links. It is important to understand the basic relation of these items. These items will be covered first generally, then in more detail as this chapter progresses.

Nearly all ladder diagram objects and function blocks rely on variables, either as a definition for the object or addition to the function block to provide required data to function properly. A variable is a placeholder that represents values that *can* change. A variable can represent any number depending upon its type.

Variables are an important part of understanding how EZ LADDER uses functions and objects. Some objects, such as Direct Contacts or Direct Coils are actually defined as variables themselves while other function blocks such as TON will require variables created, inserted and connected, using links, to the function block itself to provide set points and other functional requirements.

Variables in the EZ LADDER Toolkit are global, meaning that each variable must be uniquely named and can be changed or used anywhere in the ladder diagram project.

Using function blocks, variables can *pass* data (copy or move) to other variables, functions and objects. Figure 5-2 illustrates a simple ladder diagram project that contains objects that are variables and inserted variables linked to function blocks.

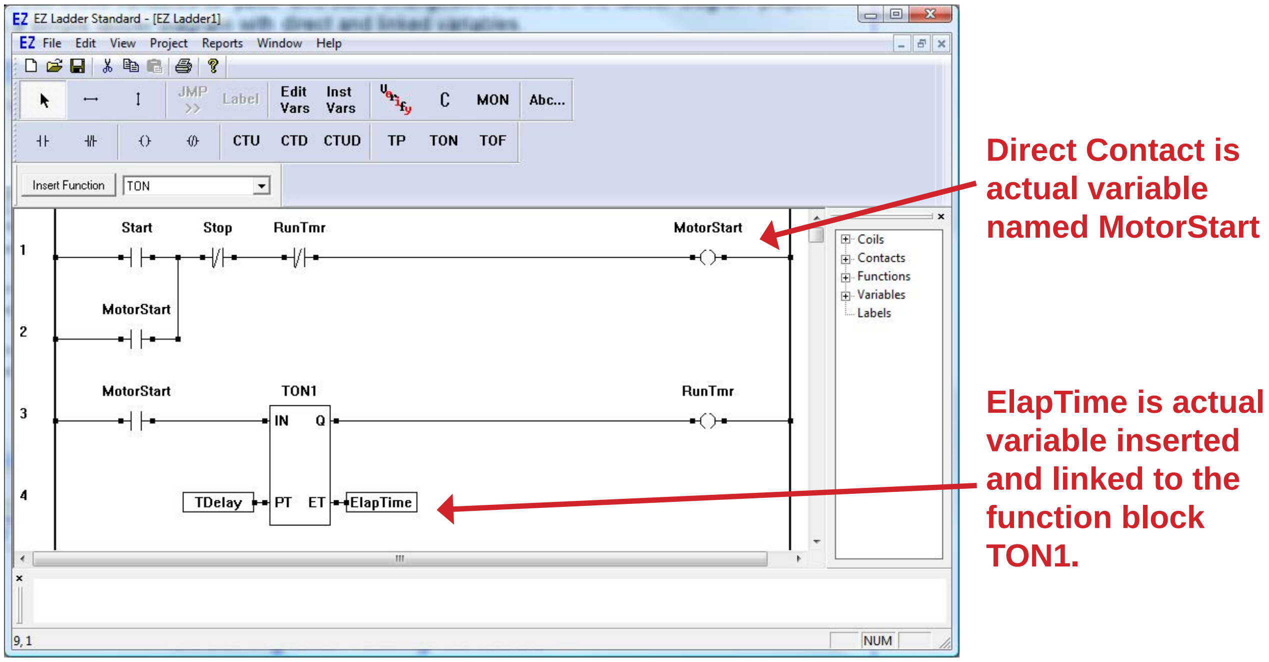


**Figure 5-2**

Figure 5-2 identifies the two typical ways variables are used in an EZ LADDER Toolkit ladder diagram project. As shown, the On-Delay timer function block identified as TON1 uses two unique variables (one for the set point - PT and one for the elapsed time - ET). All contacts and coils are actually variables themselves and as they are created, they must be either assigned to an existing variable or a new variable created must be created (declared) for them.

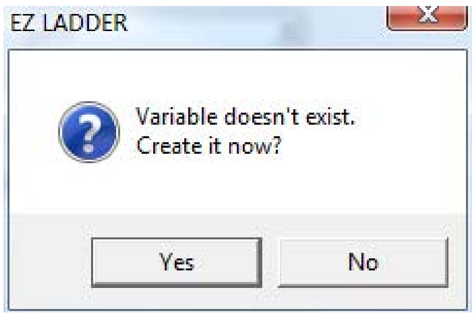
## Creating and Placing Variables

Placing and creating variables can be done several ways. Inserting some objects automatically require the selection or creation of a new variable when being inserted (an automatic dialog box), while function blocks typically require you to insert any needed variables and link them without being prompted to do so.

We will identify how to create and assign variables using two methods, although variable creation is basically the same for all methods.

### Placing Contacts and Coil Type Objects

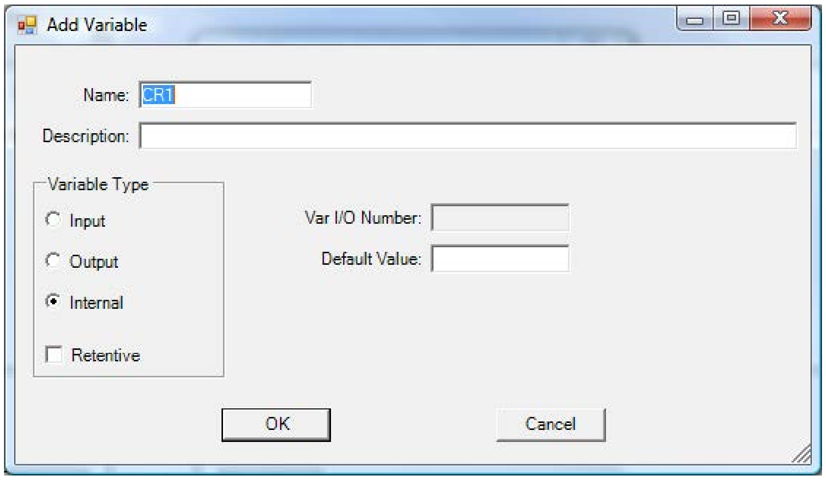
To place a contact, from the tool bar, select the Direct Contact and locate a point in the workspace to place the item. Clicking that location will place the object. When placing certain objects (coils and contacts), a ***Contact Properties*** dialog box will appear. You can choose a variable that already exists from the drop-down list or type in a new name. For this example, we will type in a new name and click **OK**. If you had selected a name that already exists, the object placement would be completed. Since we have chosen a new variable, the dialog in Figure 5-3 will appear.


EZ LADDER

Variable doesn't exist.
Create it now?

**Figure 5-3**

Click **YES** to create the new variable. The Add Variable dialog box will open automatically with the variable name you entered already in the Name field. See Figure 5-4. For now click **OK** to create the variable. We will cover the details of variable attributes later in this chapter. You have now successfully created a contact with a new variable. Repeat the same as needed for new contacts or coils.


Add Variable

Name: CR1

Description:

Variable Type
- ( ) Input
- ( ) Output
- [✓ Internal]
- [ ] Retentive

Var I/O Number:

Default Value:

**Figure 5-4**

## Placing a Linked Variable

To place a variable that is linked to a function block, from the tool bar, select the Insert Variables button (Inst Vars). In the ladder diagram workspace, click in an open area and the **Variables** window will open. This window contains tabs at the top for all variable types supported.

When inserting a variable next to a function block input, only the variable types supported by the function block will be displayed as tabs in the **Variables** window.

Select the appropriate type for the variable you are needing to insert and click the **ADD** button. The Add Variable dialog box will open automatically. Enter a variable name in the Name field. See Figure 5-4. For now click **OK** to create the variable. We will cover the details of variable attributes later in this chapter. You have now placed a linkable variable.

If the variable you need to insert already exists, select it from the list and click **OK** to insert it.

Variables names must always begin with a letter and cannot contain spaces. Trying to begin variables with numbers or using spaces will result in a error message being displayed.

Variables may be created at any time without inserting or placing them in the ladder diagram workspace. To create a variable without placing it, from the tool bar, select the Edit Variables button (Edit Vars). The Variables window will open as shown previously. Use the **ADD** button to create variables as needed.

When function blocks are used with variables, as previously covered, only supported variable types are allowed. Typically, most function blocks will lock the types of variables linked to it's outputs as the same type linked to it's inputs. When changing variable types that are an input or an output to a function block, delete the variables and function block. Then insert the function block and new variables to remove all the variable type associations that previously existed.

## Variable Types

There are four variable types supported in the ladder diagram of EZ LADDER Toolkit. They are: Boolean (BOOL), REAL, INTEGER and TIMER. Each type of variable exists for specific purposes and each has pros and cons depending upon the ladder diagram project needs.

Examples of Variables:

| | |
|---|---|
| Boolean: | 0 or 1, False or True, Off or On |
| Real: | 234.56, 192.345 |
| Integer: | 1, 525, 1034 |
| Timer: | Days, Hours, Minutes, Seconds, Milliseconds |

### Boolean Variables

Boolean variables are based on only being in one of two states, typically either true or false (1 or 0, On or Off). Boolean variables are most commonly used for contacts and coils, but also may be used with function blocks as individual bits. Boolean variables are 1 byte in size.

### Real Variables

Real variables are based on numbers that use floating point math (use decimal points). Real variables can range from $-1.7x10^{38}$ to $1.7x10^{38}$. Real variables are typically used for calculations and with functions where decimal point accuracy is required. Real variables used with function blocks result in a slower ***Scan Time***. Real variables are 4 bytes in size.

### Integer Variables

Integer variables are based on whole numbers (no decimal points) Integers can be ranged from -2147483648 to 2147483647. Integers are used when decimal points are not required. Integer result in a faster ***Scan Time*** than real variables. Integer variables Default Value can be entered in Hexadecimal format. Integer variables are 4 bytes in size.

### Timer Variables

Timer variables are used in conjunction with timer function blocks to provide input set points and output elapsed time. Timer variables consist of milliseconds, seconds, minutes, hours and days. Timer variables are 4 bytes in size.

## Variable Attributes

For each variable type, specific attributes will apply. For most variables, these attributes are common. While some attributes are optional such as description, others are required prior to creating the variable. When creating a new variable, it is ideal to set it's attributes with as much detail as possible.

### Integer, Real and Boolean Variable Attributes

When adding new integer, real or boolean variables, refer to Figure 5-5 for the Add Variable dialog box. The following are fields (attributes) for variables. Some must be completed while others are optional.

1. **Name**: The variable *name* is entered in this field. This name will be used to identify this variable and will be the name viewed in the workspace and any cross reference and reports. All names must begin with a letter and cannot contain any spaces. A unique name is require for each variable.

2. **Description**: This is where a text based description may be entered for more clarification and details regarding this variable. Descriptions appear in reports and in many dialog boxes. This attribute is optional.

3. **Variable Type**: The variable type is selected in this box. The choices are:

| | |
|---|---|
| Input: | Select Input if the variable will actually represent a real world digital input on the target. Selecting this option will require that physical address of the input to be entered in the Var I/O Number field. |
| Output: | Select Output if the variable will actually represent a real world digital output on the target. Selecting this option will require that physical address of the output to be entered in the Var I/O Number field. |
| Internal: | Select Internal if the variable has no real world connection but is to be used internal in the ladder diagram project only. |
| Retentive: | This check box is used to identify retentive variables (variables that will store their current value on power loss and reload it automatically when power is restored). This feature must be supported on the hardware target. |

4. **Var I/O Number**: This is where the physical address of real world I/O points is entered. This field is only used if the Variable Type is either Input or Output. Some targets' Var I/O number is hard coded and cannot be changed.

5. **Default Value**: This is where default variable values are set. An internal variable will be equal to this value unless it has been altered by the ladder diagram. This is used to preset values in the ladder diagram for comparisons as well as other uses. This field is optional.

6. **Address Register**: When the ladder diagram project is configured to use register based communications such as Modbus or OptiCAN, the register assignment for the variable is configured in this field. If left blank, there is no assigned register. This field only appears if a feature that will use a register is installed or supported on the hardware target and Projects Settings. This field is optional. Clicking the **EDIT** button opens an additional dialog box with all the available networks and makes assigning registers easier.

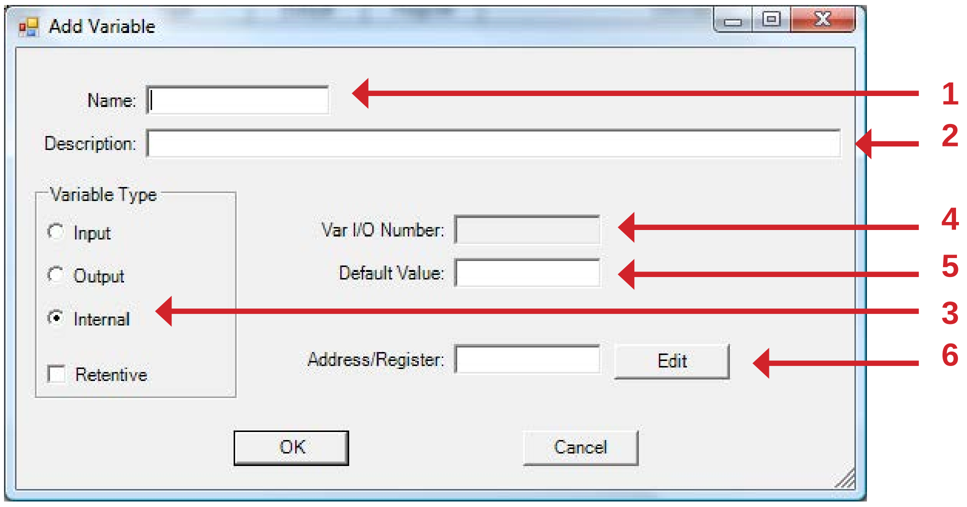


**Figure 5-5**

## Timer Variable Attributes

When adding new timer variables, refer to Figure 5-6 for the Add Variable dialog box. The following are fields (attributes) for timer variables. Some must be completed while others are optional. Typically, time durations are entered as the unit of measure closes to the set point.

Larger times may be entered into fields provided that the total timer value does not exceed 24 days. For example, 1000 ms may be entered and will be considered 1 second when the program executes. However, if 750 hours is entered, the time is greater than 24 days and the timer will malfunction.

1. **Name**: The variable *name* is entered in this field. This name will be used to identify this variable and will be the name viewed in the workspace and any cross reference and reports. All names must begin with a letter and cannot contain any spaces. A unique name is require for each variable.

2. **Description**: This is where a text based description may be entered for more clarification and details regarding this variable. Descriptions appear in reports and in many dialog boxes. This attribute is optional.

3. **Days**: The time duration in number of days.

4. **Hours**: The time duration in hours.

5. **Minutes**: The time duration is minutes.

6. **Seconds**: The time duration in seconds.

7. **Milliseconds** The time duration in milliseconds. The millisecond resolution is target specific and is shown in parenthesis.

8. **Retentive**: This check box is used to identify retentive variables (variables that will store their current value on power loss and reload it automatically when power is restored). This feature must be supported on the hardware target.

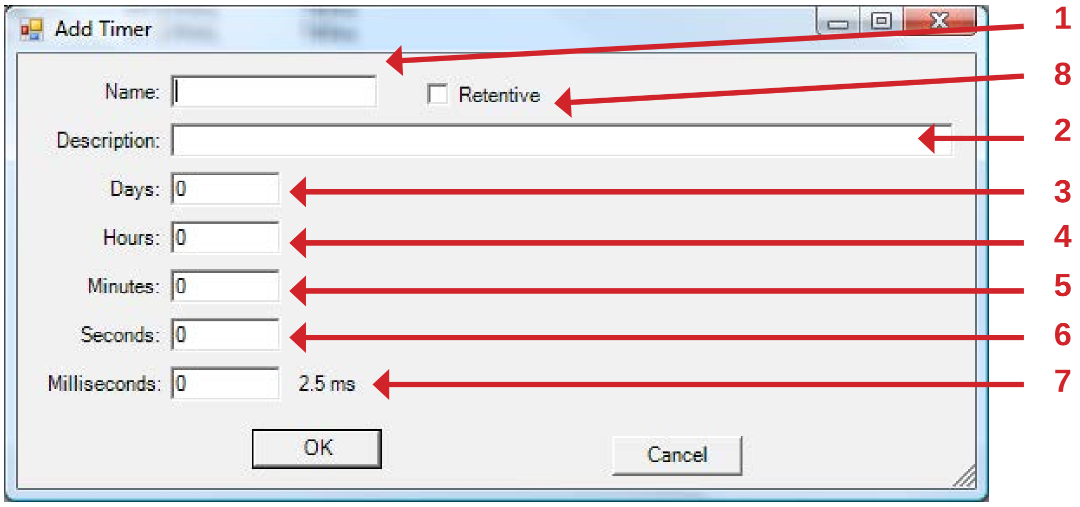


Figure 5-6

## Keeping Variable Values on Power Loss

In the event of a power loss to the target, EZ LADDER Toolkit is designed to allow ladder diagram variables to be stored and then be reloaded when power is restored. This is called the Retentive feature and variables must be configured as retentive as well as the hardware target must support this feature. See **Chapter 7 - Retentive Variables** for more details on the retentive feature and **Chapter 23 - Hardware Targets** for which targets support retentive variables.

## Placing Objects and Drawing Links

To place an object in an EZ LADDER Toolkit project, select the object or function block from the tool bar or select the object or function block from the tool bar drop down menu. Position the pointer in the ladder diagram workspace where the object is to be inserted and left-click. This *places* the object at that point. As you add objects, variables may need created. See earlier in this chapter for how to create variables.

Figure 5-7 illustrates the placement of a Direct Contact and Direct Coil. Please refer to **Appendix A - Function Reference**.

The last placed object stays selected until a different object or button in the tool bar is chosen. This feature allows an object be placed multiple times without the need of re-selecting the object.

To place an object or function, there must be enough space in the ladder diagram workspace at the point of insertion. If there is insufficient space, an error message will display.

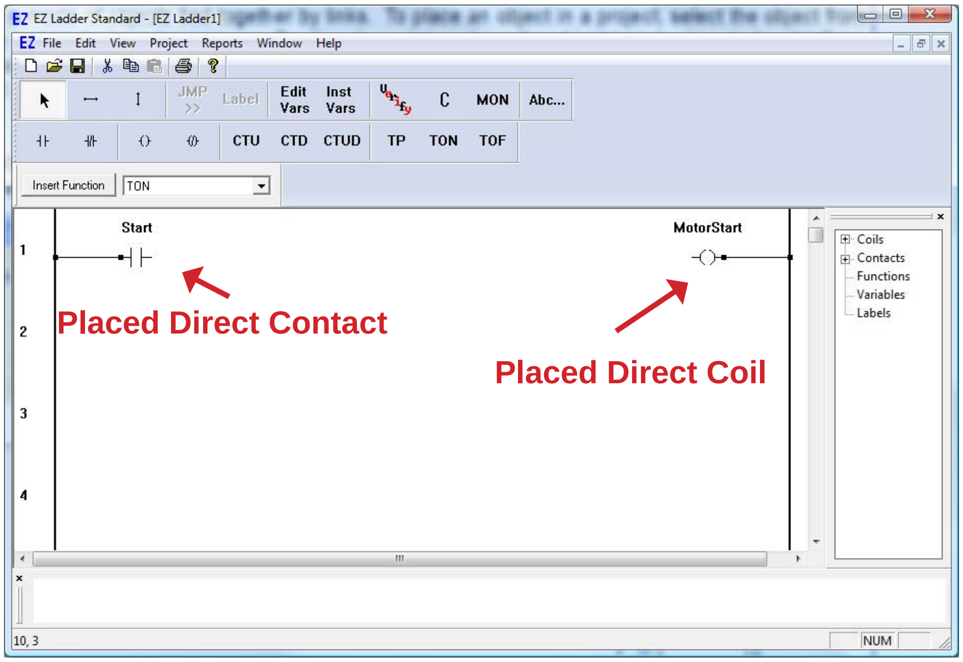


**Figure 5-7**

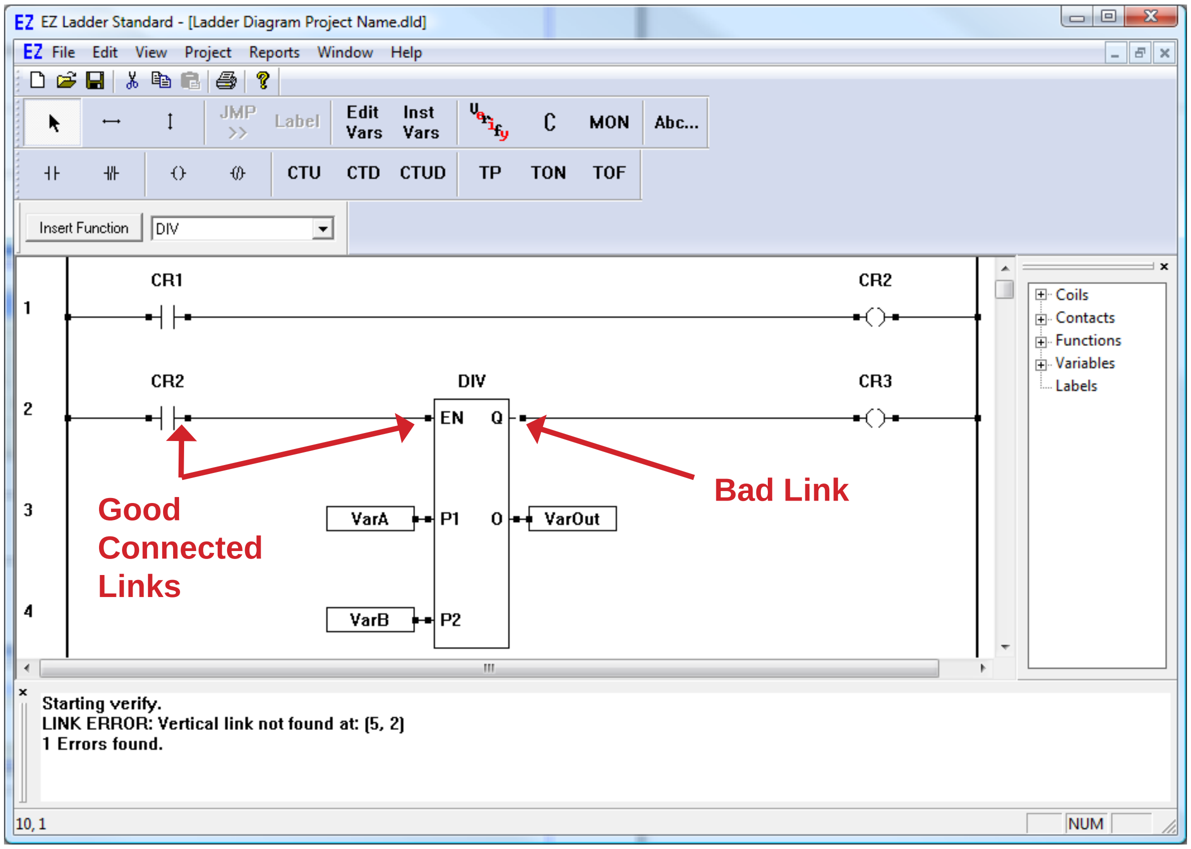
Refer to Figure 5-7, note when placing objects near the left or right power rails, links are automatically drawn to the power rails. This also applies when variables are inserted next to functions; the links are automatically drawn from the inserted variable to the function.

To finish the circuit shown in Figure 5-7, it will be necessary to draw a horizontal link between the contact and coil on rung 1. Select the Horizontal Link Tool from the tool bar. Refer to **Chapter 2 - Navigating EZ LADDER Toolkit** for details on tool bars and buttons.

To draw the link, click and hold the click at the location where the link will start, at the right side of the contact on rung 1. Holding the mouse button down (clicked), drag the pointer to the left side of the coil on rung 1. When the link is drawn connecting both objects, release the mouse button to complete the link.

If a vertical links are required (as in parallel circuits), select the Vertical Link Tool from the tool bar. Using the same method, click and drag until a vertical link is created.

Horizontal and Vertical links snap to grid locations and can only be started and stopped at one of these locations. Take care when connecting links to objects and function blocks that the link actually connects to the block and not just near it. If a link does not connect, then an error will occur when Verifying or Compiling the ladder diagram project. Figure 5-8 shows a connected link and a link that is not connected.



```
Starting verify.
LINK ERROR: Vertical link not found at: (5, 2)
1 Errors found.
```

Figure 5-8

When connecting two function blocks in series, where a variable output of the first needs to be connected to the second function blocks variable input, you must insert a variable between them. Failure to place a variable between function blocks (variable inputs and outputs only) will result in the ladder diagram project Compiling successfully, but it will not operate as intended.

## Using Copy and Paste

EZ LADDER Toolkit, being a Windows based application, allows the copy and paste functions inherit to Windows with certain limitations imposed. It is possible to copy any single or combination of objects, function blocks, variables and links to the Windows Clipboard.

To Copy object(s), choose the Select Tool from the tool bar. To choose a single object, left click on the object to select it.  To select multiple objects, click and drag across the objects.  Objects may be selected by holding the **CTRL** key while clicking on them.  With the items selected, using the **Edit Menu**, choose **Copy** or right-click and choose **Copy**.  The objects are now copies to the Windows Clipboard.

Unlike a standard windows application, objects on the clipboard cannot be pasted using **CTRL-V** or by using the **Edit Menu's Paste** feature. To paste an object or multiple objects in EZ LADDER Toolkit, use the Select Tool from the tool bar and hover the point at the location to paste (if pasting multiple objects, this would be the top, left of the objects that will be pasted). Right-click and choose **Paste**. The objects will be pasted.

When pasting objects or rungs, there must be enough room to paste the copied section (horizontally and vertically) or an error will occur. When pasting rungs, move the pointer near the left power rail as an pasting point.

## Inserting and Deleting Rungs

During development of a ladder diagram project, it often becomes necessary to insert new rungs between existing rungs or to delete rungs that will not be required.

### Inserting Rungs

To insert a new rung in EZ LADDER Toolkit, position the pointer where the insertion needs to occur (typically near the left power rail). Right-click and choose **Insert Rung**. A rung will insert at this location. All later rungs will be moved accordingly and all cross references will update with the new rung numbers.

### Deleting Rungs

To delete a rung, position the pointer on the rung to be deleted. Right-click and choose **Delete Rung**. The selected rung will be deleted. Only empty rungs may be deleted.

## Saving EZ LADDER Toolkit Projects

Saving an open ladder diagram project can be done two ways. Click the Save button on the tool bar or use the **File Menu**, and choose **Save**. If the project has not been previously saved, a dialog box will appear to enter name and save the project. The **Save As** selection in the **File Menu** always provides a dialog box for naming the project.

## Verifying and Compiling Ladder Diagrams

After a ladder diagram has been created, it must be *verified* and *compiled* prior to downloading it to an actual hardware target. This process checks the ladder diagram for adherence to all EZ LADDER Toolkit and target rules and then creates a file that will be downloaded to the hardware target. This file, while maintaining the functionality of the ladder diagram actually has no graphical representation and is generally not recognizable or viewable.

### To Verify a Project

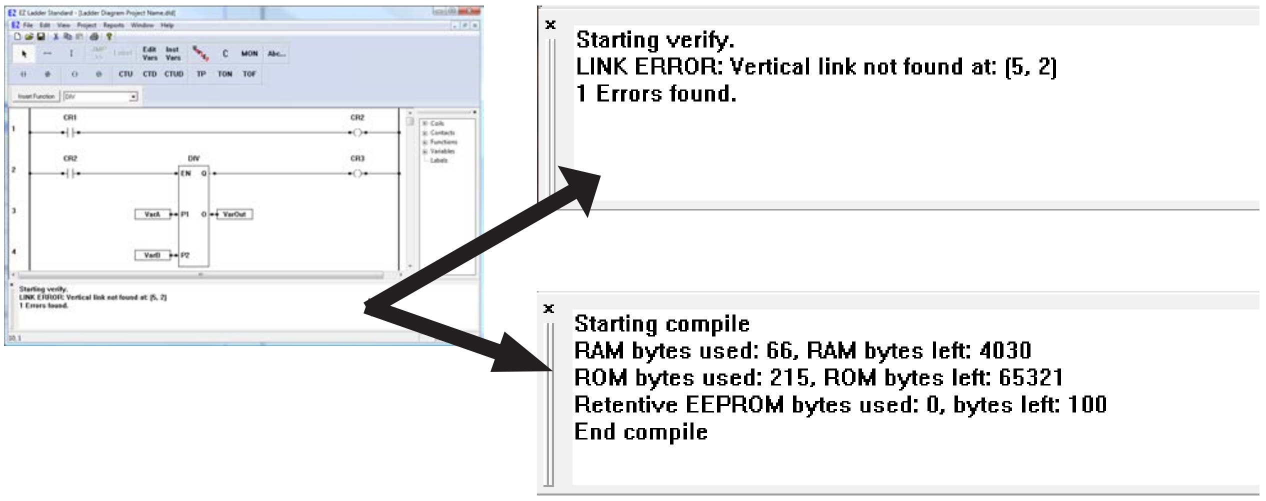
The verification process will check the ladder diagram for completeness and common rules, verifying there are not broken links, etc. To verify the ladder diagram project, on the tool bar, click the Verify button. In the Output Window at the bottom, a message will be displayed with the status and results of the verification process.

### To Compile a Project

The compilation process involves two actions. The first is an automatic verification is done and if no problems are detected, the ladder diagram is then compiled (converted into machine language code for downloading to the hardware target). To compile a ladder diagram project, on the tool bar, click the Compile button.

All EZ LADDER Toolkit Projects must be compiled prior to downloading them to a hardware target. Once a program has been compiled, it does not need to be compiled again unless the actual ladder diagram project has changed since it was last compiled.

Any errors encountered during the compilation process must be corrected before the compilation will successfully complete and provide operational compiled code. See **Chapter 22 - Troubleshooting** for common error messages. Figure 5-9 illustrates two Output Window messages for the same ladder diagram project. The first identifies errors during the compile process while the second illustrates a successful compile.


```
Starting verify.
LINK ERROR: Vertical link not found at: (5, 2)
1 Errors found.
```

```
Starting compile
RAM bytes used: 66, RAM bytes left: 4030
ROM bytes used: 215, ROM bytes left: 65321
Retentive EEPROM bytes used: 0, bytes left: 100
End compile
```

Figure 5-9

## Bit Addressable Variables

As covered earlier in this chapter, variables are an important part of an EZ LADDER Toolkit project. While most projects will use variables as described earlier, the EZ LADDER Toolkit also provides a feature to use integer variables and then actually control the individual bits that make up the entire integer variable (number, total of 32 bits per integer). This feature is called Bit Addressable Variables.

Any integer may be used as a bit addressable variable. As a bit addressable variable, each variable has 32 individual bits that are numbered 0-31 and each bit represents the *binary* bit of the total integer variable number. To understand bit addressable variables, you must have a basic understanding of the binary numbering system where numbers are created using ones and zeros in specific placeholder bits that represent an actual number.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bit Number (0-31): | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Placeholder: | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Binary Bits: | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

$\longrightarrow$

Add the placeholder numbers of bits with 1's only

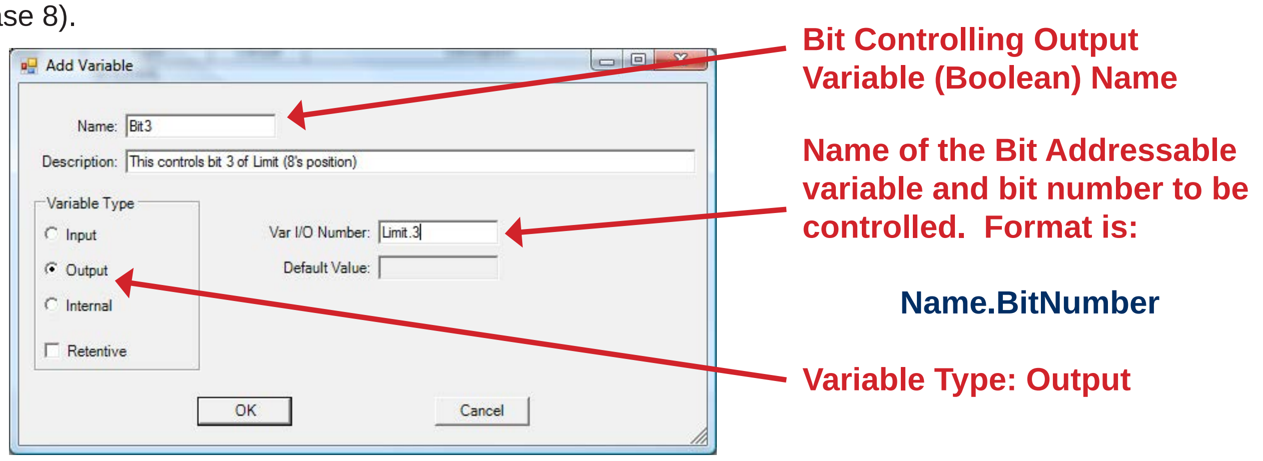
8 + 4 + 1 = 13

The integer variable value would be 13.

### Setting the bit of a Variable

To set the bit of an integer variable, identify or create the variable. In addition to the variable that will be bit addressable (the one you just identified), additional variables will be required to write to the bits of the original bit addressable variable (one for each bit that you intend to use).

These additional variables will be (boolean) output variables, representing a boolean 0 or 1 for the actual bit. In Figure 5-10, the Add Variable dialog box shows the creation of one of the actual bit controlling boolean variables. These bit controlling variables are always set as Output and the Var I/O Number is the variable name of the bit addressable variable and the bit number to control separated by a period. In Figure 5-10, the bit addressable variable is named *Limit* and the bit shown being controlled is 3 or the placeholder for the number 8 in integer form and the variable that is controlling the bit is named *Bit3*. Therefore, if *Bit3* is true then bit 3 of the variable *Limit* would be true, changing the value of the variable Limit by its placeholder value (in this case 8).


Add Variable
Variable Type
OK
Cancel
Bit Controlling Output Variable (Boolean) Name
Name of the Bit Addressable variable and bit number to be controlled. Format is:
Name.BitNumber
Variable Type: Output

Name: Bit3

Description: This controls bit 3 of Limit (8's position)

( ) Input [✓ Output] ( ) Internal

☐ Retentive

Var I/O Number: Limit.3

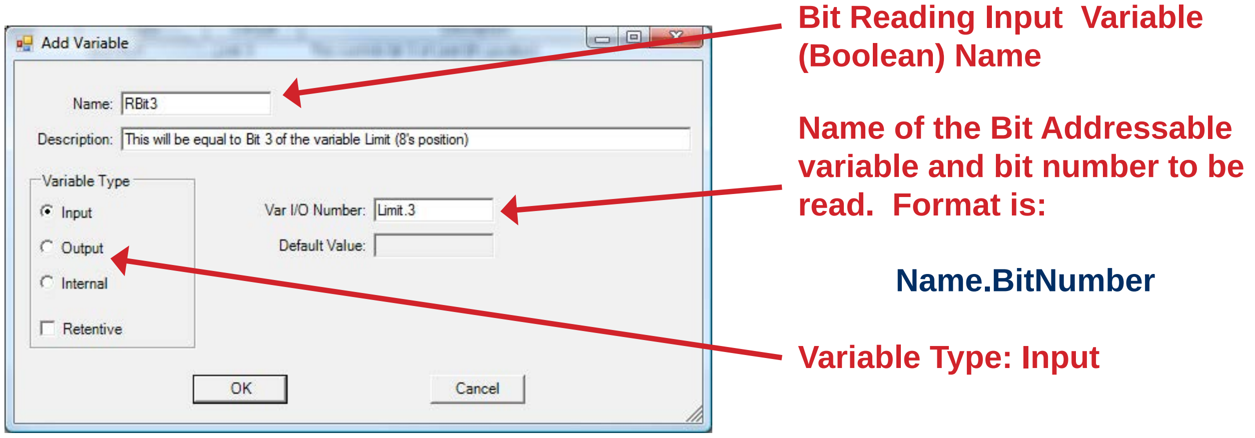
Default Value:

Figure 5-10

## Reading the bit of a Variable

To read the bit of an integer variable, identify or create the variable. In addition to the variable that will be bit addressable (the one you just identified), additional variables will be required to read to the bits of the original bit addressable variable (one for each bit that you intend to use).

These additional variables will be (boolean) input variables, representing a boolean 0 or 1 for the actual bit. In Figure 5-11, the Add Variable dialog box shows the creation of one of the actual bit reading boolean variables. These bit reading variables are always set as Input and the Var I/O Number is the variable name of the bit addressable variable and the bit number to read separated by a period. In Figure 5-11, the bit addressable variable is named *Limit* and the bit shown being read is 3 or the placeholder for the number 8 in integer form and the variable that is reading and storing the bit is named *RBit3*. Therefore *RBit3* will be equal to the actual binary status (0 or 1) of bit 3 of the *Limit* variable.


Add Variable

Name: RBit3

Description: This will be equal to Bit 3 of the variable Limit (8's position)

Variable Type: [✓ Input] Output Internal

Retentive: ☐

Var I/O Number: Limit.3

Default Value:

OK
Cancel
Bit Reading Input Variable (Boolean) Name
Name of the Bit Addressable variable and bit number to be read. Format is:
Name.BitNumber
Variable Type: Input

Figure 5-11

# CHAPTER 6

## Downloading and Running Projects

This chapter provides basic information needed to connect to hardware targets, download ladder diagram projects and use real-time EZ LADDER Toolkit features.

## Chapter Contents

## Switching Modes in EZ LADDER Toolkit

EZ LADDER Toolkit is generally has two modes of operation. Up to this point, most of the time we have been using the ***Edit Mode***. The Edit Mode is used to open and close projects, configure targets, create ladder diagram projects, verify and compile them. The ***Monitor Mode*** is used to connect to hardware targets, download ladder diagram projects, monitor their power flow in real-time and to work with target utilities. Typically switching modes is done often during ladder diagram project development.

### Switching to Monitor Mode

To switch to monitor mode, on the tool bar, click the **MONITOR** button. Refer to **Chapter 2 - Navigation EZ LADDER Toolkit** for tool bar and buttons. EZ LADDER Toolkit will switch from the Edit Mode to the Monitor Mode. While the ladder diagram workspace will appear similar, some tool bars and buttons will change adding functionality for features only needed in Monitor Mode. Figure 6-1 shows EZ LADDER Toolkit in the Monitor Mode.

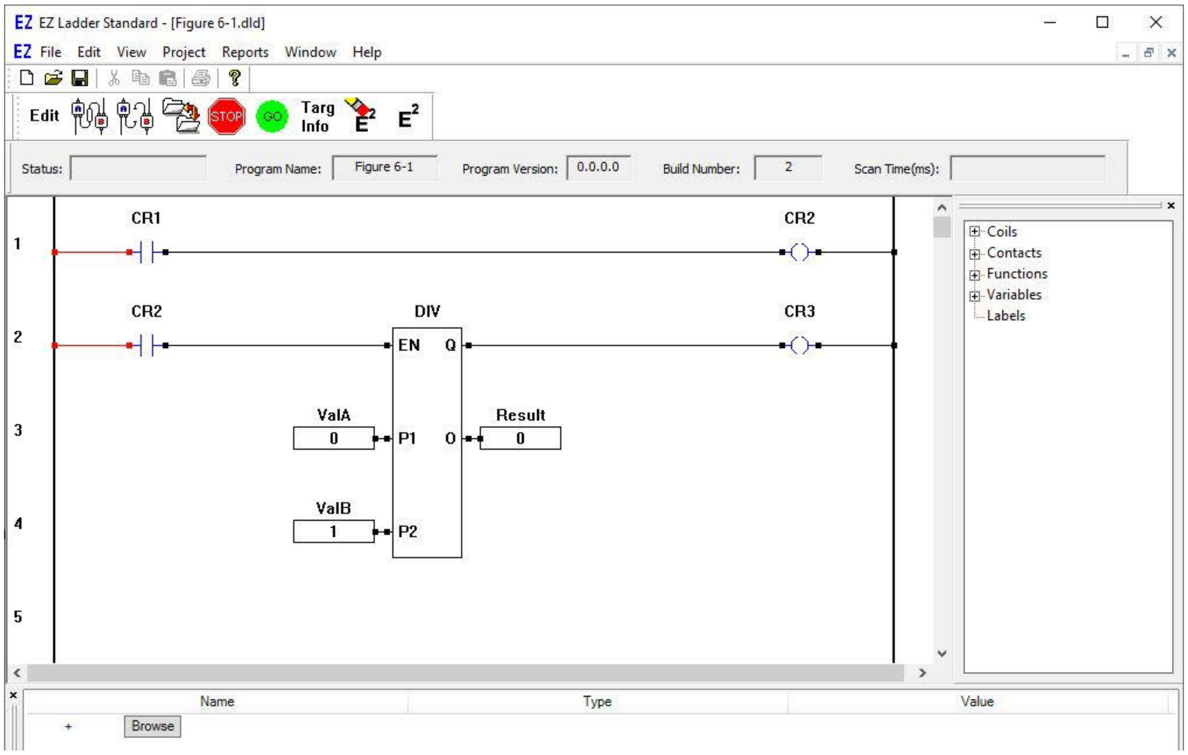


**Figure 6-1**

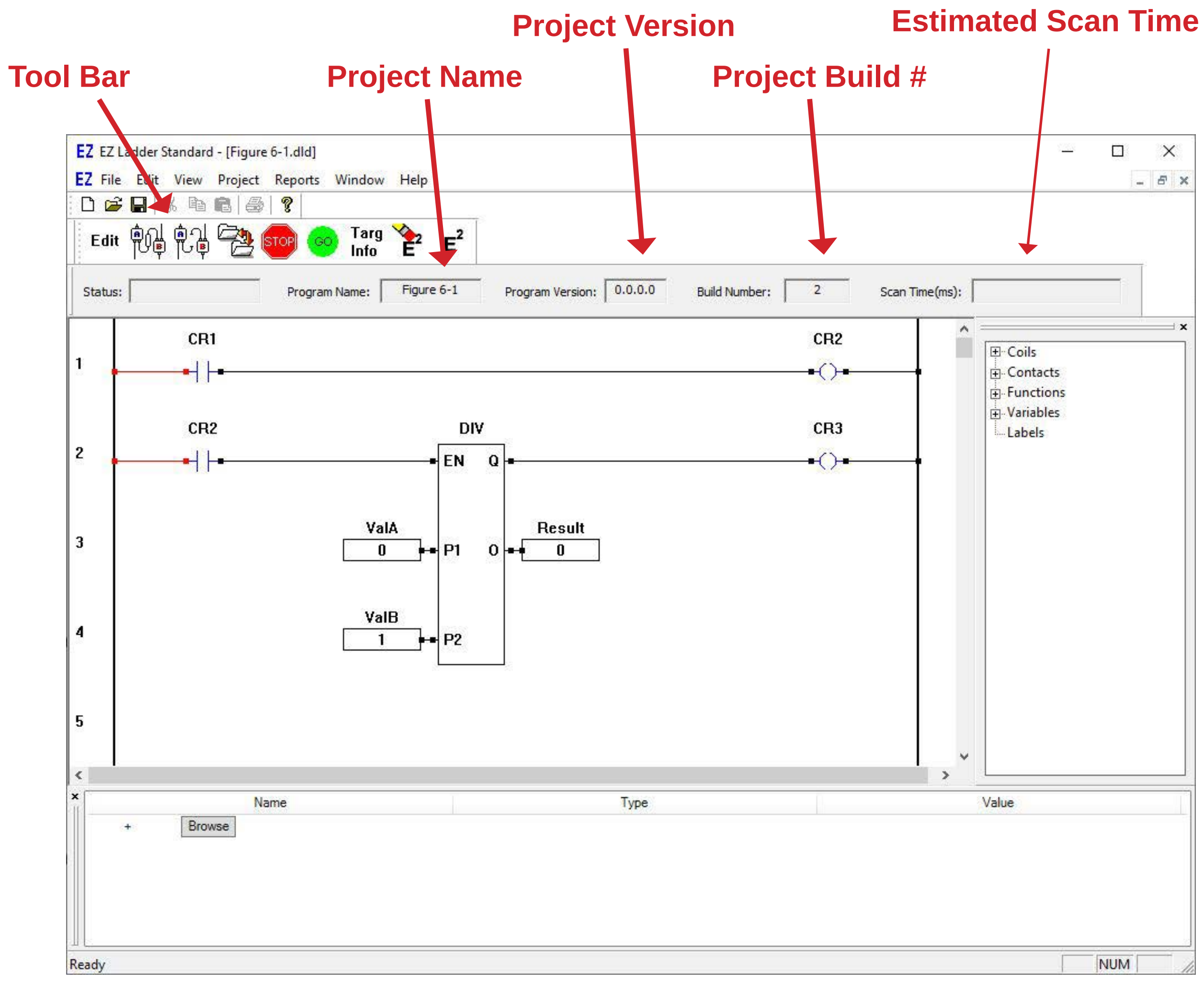
In addition to the tool bar changes, the Output Window is not available in the Monitor Mode as the program should be compiled in the Edit Mode prior to switching to the Monitor Mode.

## Switching to Edit Mode

When in the Monitor Mode, to switch back to the Edit Mode, on the tool bar, click the Edit button. EZ LADDER Toolkit will switch from Monitor Mode to the Edit Mode. All Edit Modes standard tool bars, menus and windows will reappear.

## Monitor Mode Overview

While the Monitor Mode generally looks similar to the Edit Mode, the tool bars, menus and windows can differ greatly. Refer to Figure 6-2 for identification status fields.



Status: Program Name: Figure 6-1 Program Version: 0.0.0.0 Build Number: 2 Scan Time(ms):

ValA 0, Result 0, ValB 1

Figure 6-2

The following descriptions are for buttons found on the Monitor Mode Tool Bar.

Edit **Edit Mode.** Switches EZ LADDER Toolkit to the Edit Mode.

**Connect**. Connects EZ LADDER Toolkit to the hardware target's Programming Port.

**Disconnect**. Disconnects EZ LADDER Toolkit from the hardware target..

`

**Download**. Transfers the compiled ladder diagram project to the hardware target and saves the program in memory and starts executing the program. The program will remain until over written by a new downloaded program.

**Stop**. Stops execution of the ladder diagram project on the hardware target.

**Go**. Starts execution of the ladder diagram project on the hardware target.

**Target Information**. Opens the a target information dialog that identifies the actual target version connected to EZ LADDER Toolkit and the current Target's Name or Model Number.

**EEPROM Erase**. This erases the EEPROM on the hardware target. The target must support EEPROM storage for this feature to function.

**EEPROM View/Set.** This opens a window that allows viewing of the FRAM (if installed) and EEPROM (if installed). Based on other target features, it may be used to write data to the EEPROM (not FRAM).

**There is no UNDO when erasing or writing to the EEPROM. Once the EEPROM has been erased or over-written, all previous contents are lost. Take care in erasing / writing to the EEPROM as to not lose valuable data.**

## Connecting to a Target

To download a ladder diagram project to a hardware target, it must first be connected to in the Monitor Mode. To successfully connect to a target, the Serial Port Settings in the Project Settings Window must match your computers setup, the appropriate programming cable must be connected from the computer's serial port to the hardware target's programming port and the hardware target must be turned on.

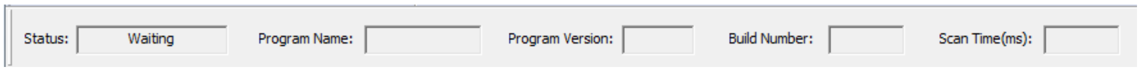
In addition to the serial port, the target may be connected to (for programming) using Ethernet or Wi-Fi provided the actual hardware targets support Ethernet or Wi-Fi. As with the serial ports, the actual communications port (COM x port, Ethernet IP address or Wi-fi IP address) must be selected in the Project Settings window. Before the Ethernet or Wi-Fi may be used for programming, it must be configured on the actual hardware target using the serial port in the Bootloader. See **Chapter 4 - Configuring Targets**.

To connect to target, click the **CONNECT** button located on the tool bar. If an error occurs, check the Serial Port Settings, cable and target. Also see **Chapter 22 - Troubleshooting**.

When connecting to a target, the background of the workspace may change with watermark text identifying conditions such as when a different program is running than is open in EZ LADDER Toolkit.

## When Target has no Project Loaded

If the target does not have a previously loaded ladder diagram project, then no dialog boxes will open when the Connect button is clicked. The Status window typically will change to *Waiting* to identify that the connection is complete and the hardware target is waiting for a ladder diagram project to be downloaded. Figure 6-3 illustrates the status as described.

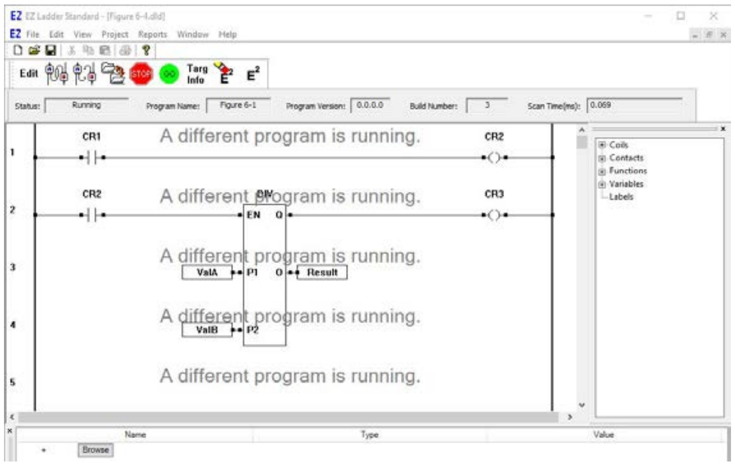


Status: Waiting Program Name: Program Version: Build Number: Scan Time(ms):

**Figure 6-3**

## When Target has Different Project Loaded

If the ladder diagram project name of the project open in EZ LADDER Toolkit does not match the name of the ladder diagram project that is loaded on the target, workspace background will change as shown in Figure 6-4. This warning can be caused because the projects differ or the project open in EZ LADDER Toolkit was renamed or saved with a different name using the Save As since it was loaded on the target. Downloading the project will clear this watermark text shown.



Status: Running Program Name: Figure 6-1 Program Version: 0.0.0.0 Build Number: 3 Scan Time(ms): 0.069

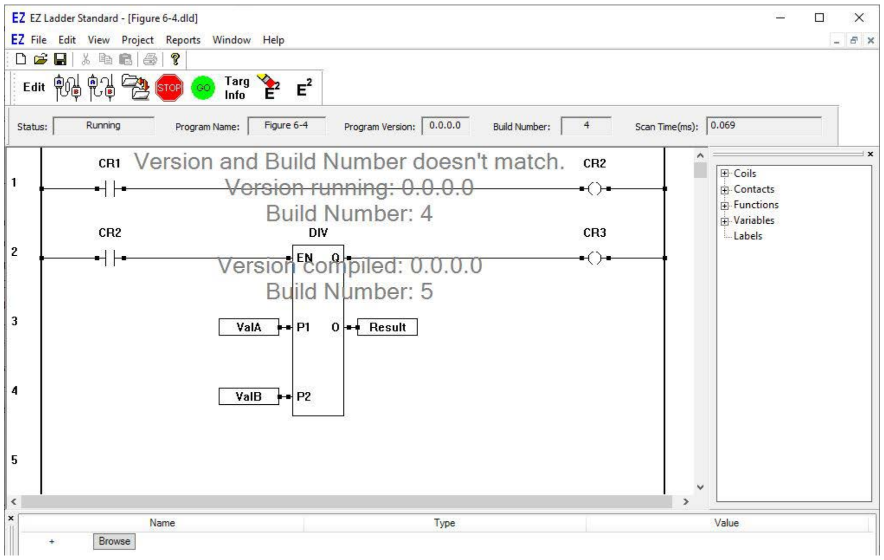
A different program is running.

**Figure 6-4**

## When Target has the Same Project

If the ladder diagram project name of the project open in EZ LADDER Toolkit does match the name of the ladder diagram project that is loaded on the target, two results can occur. If the build number (that automatic number that increments each time a project is compiled, See **Chapter 4 - Configuring Targets**), is the same as the build number of the project loaded on the target, no dialog boxes are displayed. The Status, Program Name, Program Version, Build Number and Scan Time are updated. Now ladder diagram project can be viewed in real-time.

If the two build numbers differ, then the warning water mark test in Figure 6-5 is displayed. This text serves as a warning that the two build numbers do not match. While this is usually caused by the ladder diagram project being compiled again since it was downloaded, it also requires that you must download the new build of the ladder diagram project to view it in real-time.



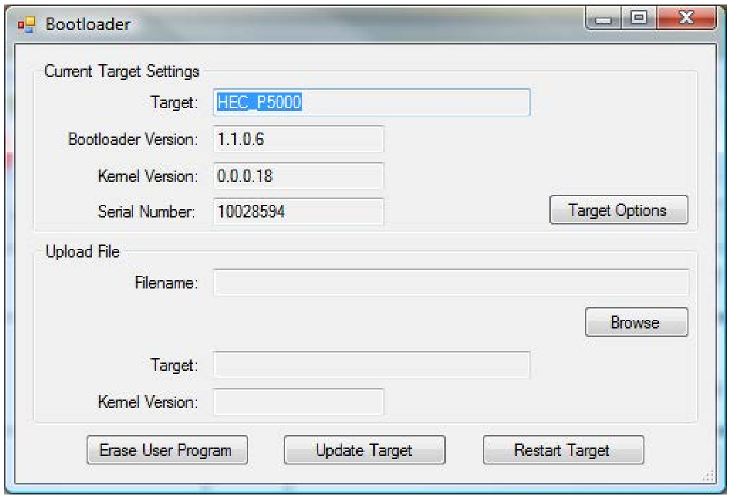
Status: Running  Program Name: Figure 6-4  Program Version: 0.0.0.0  Build Number: 4  Scan Time(ms): 0.069

Version and Build Number doesn't match.
Version running: 0.0.0.0
Build Number: 4
Version compiled: 0.0.0.0
Build Number: 5

Figure 6-5

## Connecting for the First Time to a New Target

To connect to a target, the target must have a kernel installed. As hardware targets are shipped from the factory without kernels, the kernel must be loaded prior to being able to connect and download projects. When trying to connect to a new target for the first time (if it is configured correctly and successful), the Bootloader Window automatically is displayed. From this window, the kernel can be selected and installed on the hardware target. Figure 6-6 illustrates the Bootloader window. Refer to **Chapter 4 - Configuring Targets** for details on installing and upgrading the hardware target kernel.



Target: HEC_P5000
Bootloader Version: 1.1.0.6
Kernel Version: 0.0.0.18
Serial Number: 10028594
Filename:
Target:
Kernel Version:

Figure 6-6

## Downloading Ladder Diagram Projects to Targets

When connected to a hardware target, click the **DOWNLOAD** button located on the tool bar. A dialog box will temporarily be displayed showing the status of the ladder diagram's download to the target and the Status field will indicate *Downloading*.

Upon completion of the download, the Status field will update and indicate *Running*. The target has now been programmed with the ladder diagram project. A download action causes the project to download, for the project to be saved in the target's non-volatile memory and then it is given a execute command to begin running on the target.

The project is now executing on the hardware target. The status of contacts, coils, function blocks, variables and power flow may be viewed in real-time.

Disconnecting from the target or changing to Edit Mode does not stop the target from operating as it can only be stopped by removing power or the use of the Stop button in the Monitor Mode.

It is important that all ladder diagram projects be archived for safe keeping. There is no method to recover a ladder diagram project from the target. The actual ladder diagram file must be available for editing and future downloads.

## Real-Time Features

When connected to a hardware target with an executing program, there are additional real-time monitoring features available in the EZ LADDER Toolkit. These features include Power Flow indication, Scan Time, Starting and Stopping program execution, hover boxes and the ability to change variable values.

### Power Flow Indication

Monitoring a project in real-time provides the ability to watch the state of contacts, coils, function blocks and variables. See Figure 6-7. Contacts and Coils are actually represented in their current state (On / Off) by color. Blue represents the contact or coil in it's rest state (un-powered state) while Red represents a powered or flow condition. As real world and internal objects change during program execution, they are represented in color accordingly and the flow of power can be viewed (Power Flow) from the left power rail to the right power rail).

Although contacts and coils change colors based on their actual state, some links may change color, but most links and all function blocks remain the standard black and white color.

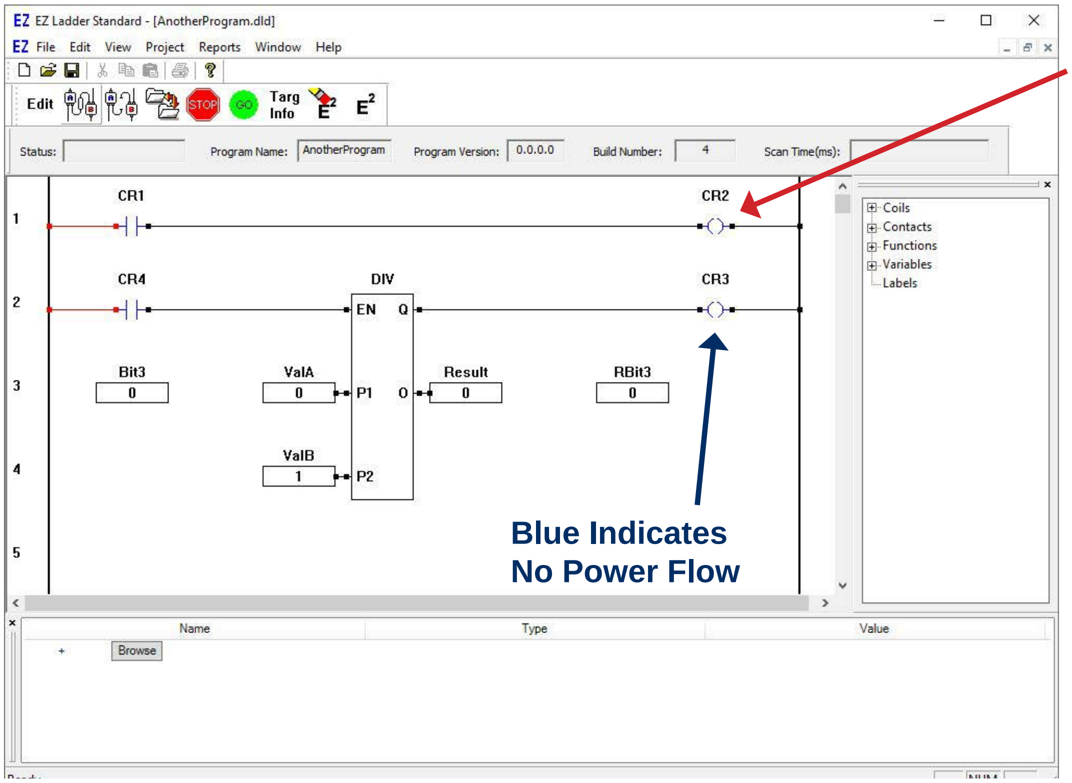


**Figure 6-7**

## Scan Time

Scan time is calculated in real-time, updated and displayed in the Scan Time Field. The scan time is always represented in milliseconds. The scan time resolution is target specific. For more information on scan time, please see **Chapter 3 - Ladder Diagram Basics**.

## Starting and Stopping Program Execution

The program on the target can be stopped and started again using the EZ LADDER Toolkit when in Monitor Mode and connected to the target.

To Stop a program from executing on the target, on the tool bar, click the Stop button. This can be useful when troubleshooting and diagnosing ladder diagrams that do not operate as expected.

To Start a program executing on the target, on the tool bar, click the Go button. This can be useful when troubleshooting and diagnosing ladder diagrams that do not operate as expected.

## Hover Boxes

Another useful feature that can be utilized in real-time monitoring is the use of hover boxes. When the mouse pointer is hovered over an object, a hover box will appear that provides additional information in regards to the function or object including it's name and current status. Figure 6-8 shows a typical hover box. The mouse pointer is located over the contact CR2. Notice the hover box is now shown and identifies the contact by name, type and it's current state or value.

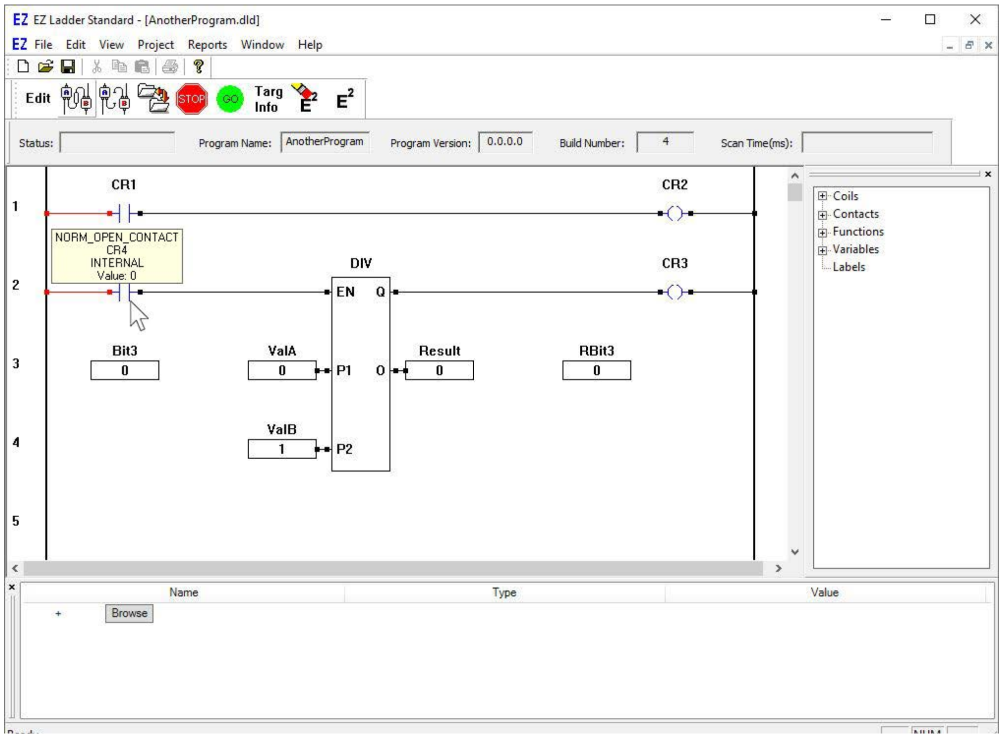


Figure 6-8

## Changing Variable Values

EZ LADDER Toolkit provides an option for changing the value of a variable while the ladder project is executing. Double-click on the object and an dialog box appears with the current state or variable value. This box is changeable and the value may be changed. Change the value as needed and click **OK**. The changes take place immediately. The change does not affect the actual ladder diagram (in the Edit mode), only the executing program. This is helpful for adjusting timer and counter values in real time during debugging.

Changing a contact variable (boolean) does not always have the desired effect. For example: If the value of an internal coil (that is connected to a real world input) is changed using the dialog box, the actual value will change only until the next scan and then will revert to its real world status. Since all I/O status is re-evaluated each scan, the contacts and coils are updated and will override variable changes. Actual real world inputs cannot be changed at all.

Changing any variable value in real-time does not change the ladder diagram project. Changes that wish to be kept must be manually changed in the project in the Edit Mode. Additionally, any variable changes on the target are lost if the target is stopped, started or power is reset to it.

# CHAPTER 7

## Retentive Variables & EEPROM Memory

This chapter provides basic information to understand what Retentive variables are, when to use and how to use them including their limitations. EEPROM installation and use is also provided.

## Chapter Contents

## What is a Retentive Variable

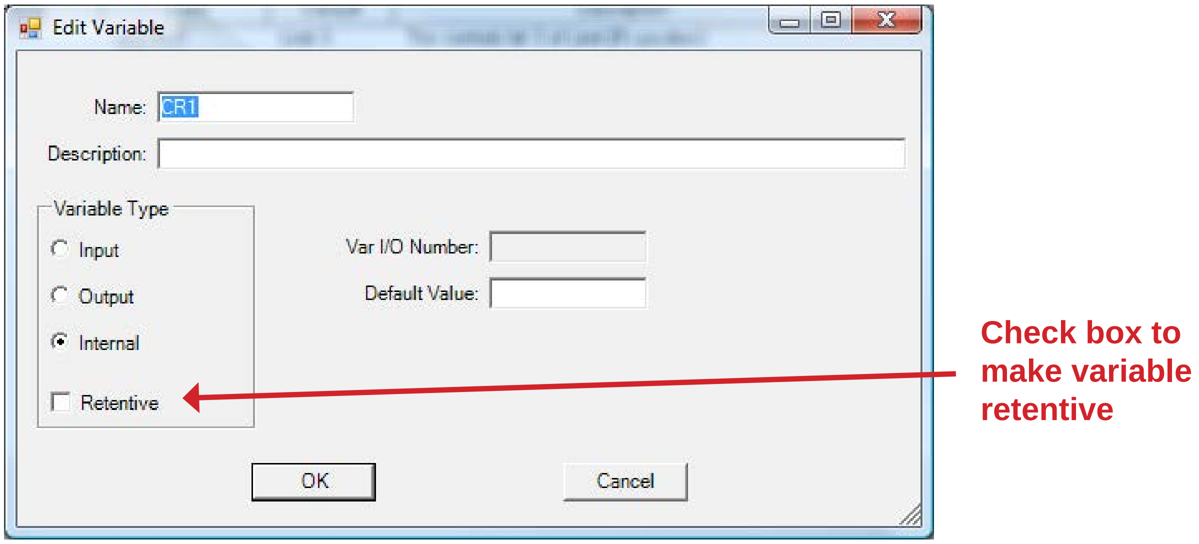
A Retentive variable is a variable that's value is automatically stored in non-volatile memory in the event of a power interruption on the hardware target. When power is restored, retentive variable values are automatically read from the non-volatile memory and re-loaded into their original variable.

Retentive variables are used often to recover from a power interruption and continue the process that is being controlled without initializing the process or wasting materials.

The hardware target must support Retentive Variables for this feature to work. Adding retentive variables to a ladder diagram project alone does not guarantee retentive functionality. The actual hardware target must support and be configured to used retentive memory.

## How to Make a Variable Retentive

For a variable to be retentive, it must be identified as retentive. To identify a variable as retentive in the Edit Mode, click the Edit Vars button located on the tool bar. Select the variable that is to be retentive. Click the **EDIT** button. The Edit Variable dialog will appear as in Figure 7-1.


Edit Variable

Name: CR1

Description:

Variable Type

- Input
- Output
- Internal [✓]

Retentive [ ]

Var I/O Number:

Default Value:

OK Cancel

**Figure 7-1**

To make the variable retentive, click the Retentive check box and click **OK**. The variable is now retentive and will be stored in the event of a power interruption provided the actual target supports the retentive feature. The same check box is present when creating a new variable.

The target must be configured in the Project Settings to use retentive memory before any variables may be set as retentive. The Retentive check box will not appear until the retentive project settings are configured.

## Retentive Variable / Memory Limitations

While retentive variables and functionality can be a useful tool when creating ladder diagram projects. There are limitations to when retentive variable usage.

As was discussed previously, retentive variables are stored in non-volatile memory (memory that retains data without power) and that retentive variable functionality is target dependent. The actual target must have non-volatile memory capability and the capability to detect a loss of power before power drops below the operating range for the target. In other words, the target must be able to sense the loss of power early enough to provide the time needed to write the retentive variables to the non-volatile memory while the target's input power is still sufficient for proper operation.

If designing a custom product using the P-Series PLC on a Chip$^{TM}$ Integrated Circuit or Module, this functionality is based on the use of the active low LOW_VOLT_SENSE pin input (Integrated Circuit Pin: 130). This pin must be connected to a circuit that can detect the loss of power early enough to allow the completion of the Retentive memory write
cycle.

For Retentive memory, the P-Series PLC on a Chip$^{TM}$ generally requires an external FRAM technology device. This device connects to the PLC on a Chip using its I$^2$C ports. Please refer to the PLC on a Chip P-Series Data Sheets for details on the retentive memory requirements
and the FRAM connections.

The internal (P-Series PLC on a Chip) EEPROM memory is a configurable option for selecting the location to store retentive memory. Generally, the on-board chip EEPROM should not be used for retentive memory storage as the on-board EEPROM memory is too slow to allow sufficient time
to complete the retentive write cycle.

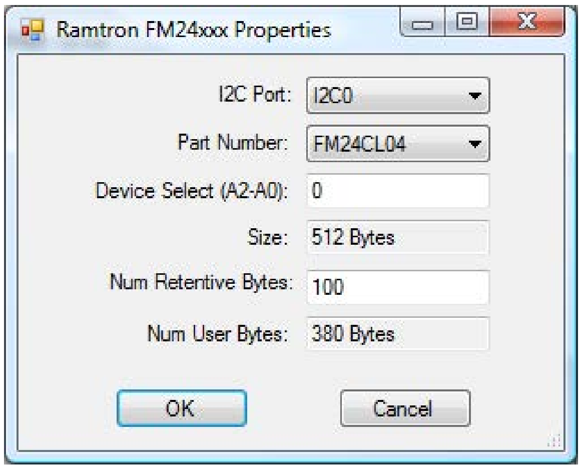
Some targets may be able to use the EEPROM for retentive memory for some number of variables. The actual amount that would write correctly would need to be determined by testing. Custom (customized) targets may be designed to use this on-board EEPROM provided the power is held long enough and the loss of power is detected early enough to complete the write cycle for the number of bytes needed as retentive.

The PLC on a Chip / EZ LADDER Toolkit supports multiple FRAM devices that in turn provide different size memory. Connected FRAM device memory is split into Retentive and EEPROM storage. The amount of this FRAM specified for Retentive is configured in the hardware target's Project Settings. Regardless of the memory size for Retentive, the hardware must support a loss of power detection and have sufficient power supply to allow enough time for the Retentive Memory to write successfully.

## Configuring Retentive Memory in the Project Settings

To configure the Retentive Memory, use the **Project Menu** and click **Settings.** The Project Settings Window / Dialog box will open. Click the Properites button. See **Chapter 4 - Configuring Targets** for more details on target configuration menus. The *Target Properties* window will open. Located in the Devices pane, the Low Voltage Sense should be listed under Internal and under the Bus, the I$^2$C, I$^2$C port and FM24XXX should be listed. Both these items are required for retentive memory to operate correctly. If these devices are not present, they must be installed. See **Chapter 4 - Configuring Targets.**

Click on the FM24XXX (highlight) and click the **PROPERTIES** button. The Ramtron FM2xxx Properties dialog will open. This dialog is used to configure the Retentive Memory. See Figure 7-2.



I2C Port: I2C0
Part Number: FM24CL04
Device Select (A2-A0): 0
Size: 512 Bytes
Num Retentive Bytes: 100
Num User Bytes: 380 Bytes

**Figure 7-2**

When configuring for PLC on a Chip™, select the I2C port from the I2C Port drop-down menu and the FRAM part number from the Part Number drop-down. The Device Select should be 0 and the Size is based on the actual FRAM part number.

If the target automatically configures the FRAM device, these settings are preset.

The number of bytes on the device is divided into Number of Retentive Bytes (Num Retentive Bytes) and the Number of User Bytes (Num User Bytes). The Num Retentive Bytes are where retentive variables are stored on power loss. The Num User Bytes are used for the program to store information using the EEPROM_WRITE and EEPROM_Read function blocks.

To set the amount of retentive memory, in the Num Retentive Bytes box, enter the desired number of bytes that should be retentive, up to the maximum allowed for the device. The Num User Bytes is automatically recalculated and updated based on the value you enter for Num Retentive Bytes.

When the Retentive memory has been configured, click **OK** to close the Ramtron FM24xxx Properties box. Click **OK** to close the Target Properties window and click **OK** to close the Project Settings Window. Be sure to save the ladder diagram project (program) after making any changes to the Target Settings.

To configure the on-chip EEPROM memory to be used as retentive memory, refer to the ***Installing EEPROM Functionality*** heading in this chapter of the manual.

## Selecting the Retentive Memory Type

The hardware target can only use one of the retentive memory types (EEPROM or FRAM). As such, it must be configured to identity the memory type. From within the Project Settings, inside the targetXXXX's Properties window, select the **Low Voltage Sense** (under internal) and click the **PROPERTIES** button. See Figures 7-3 and 7-4.
Using the drop-down menu, select either the FM24XXX for the FRAM device or the PLCHIP_Pxx_eeprom device for the retentive storage memory to use.

For retentive storage, only one memory type may be selected for all retentive variables to be stored.

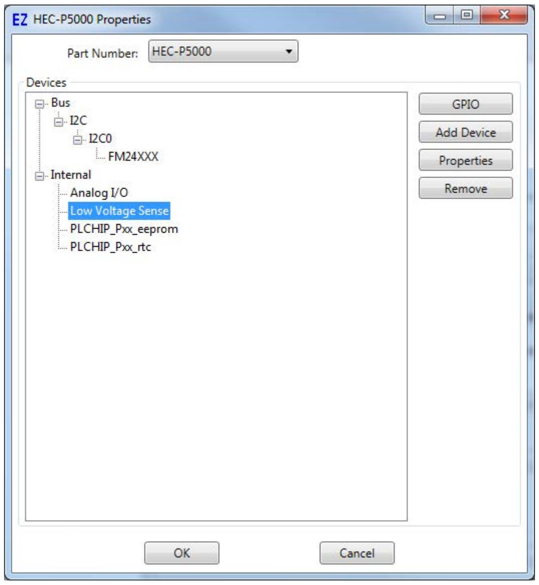


Figure 7-3

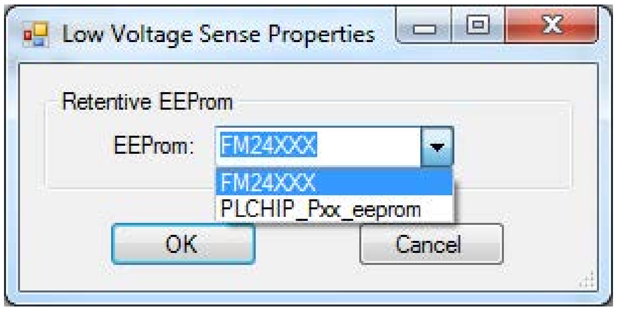


Figure 7-4

Click **OK** the number of times necessary to store your selection and close any open dialog windows. Be sure to save the program.

## EEPROM Memory Overview

EEPROM memory is a non-volatile memory (meaning its values are kept in the event of a power loss) that may be used to store data from variables. The data may be stored and retrieved as needed. The EEPROM memory is ideal for storing operational parameters of a program that don't change regularly but need the ability to change.

EEPROM memory is not suited for storing values or data that changes rapidly and must be stored at each change. EEPROM technology provides a limited number of write cycles to an EEPROM location before it will fail. This number of writes before failure is large (from hundreds of thousands to millions) and does not pose any issues for items that change occasionally; however, if a process were to try and write once per second, the number of writes would exceed the life of the EEPROM much faster. Retentive Variables and memory is better suited for rapid and repeated writes.

## Installing EEPROM Functionality

For P-Series based PLC on a Chip targets, the EEPROM functionality may or may not be automatically installed based on the actual hardware target. If the target automatically installs the EEPROM functionality, then no other configuration is required.

If the EEPROM_READ and EEPROM_WRITE function blocks are in the function block drop down list, then the EEPROM functionality is installed and may be used.

If the EEPROM functionality is not installed and it is required for a ladder diagram project, it must be installed before the EEPROM may be utilized.

To install the EEPROM device (some target knowledge is assumed, refer to the hardware target's user manual) follow these basic steps. A VB-2000 is used as the example hardware target. All others will install similarly.

1. In EZ LADDER, from the File Menu at the top, click **PROJECT** then **SETTINGS**. This will open the Project Settings Window. Select the target from the choices (the **VB-2000** is used in this example).

2. Click the **PROPERTIES** button to the right side of the window. The target (VB-2000) Properties Window will open. Make sure the proper model (Part Number) is selected in the drop-down menu. Refer to Figure 7-3

3. Click the **ADD DEVICE** button. The *PLCHIP-PXX Devices* window will open. Locate the PLCHIP_Pxx_eeprom in the Devices pane of this window. Refer to Figure 7-4.

4. Click the ***PLCHIP_Pxx_eeprom*** device (highlight) and click **OK**.

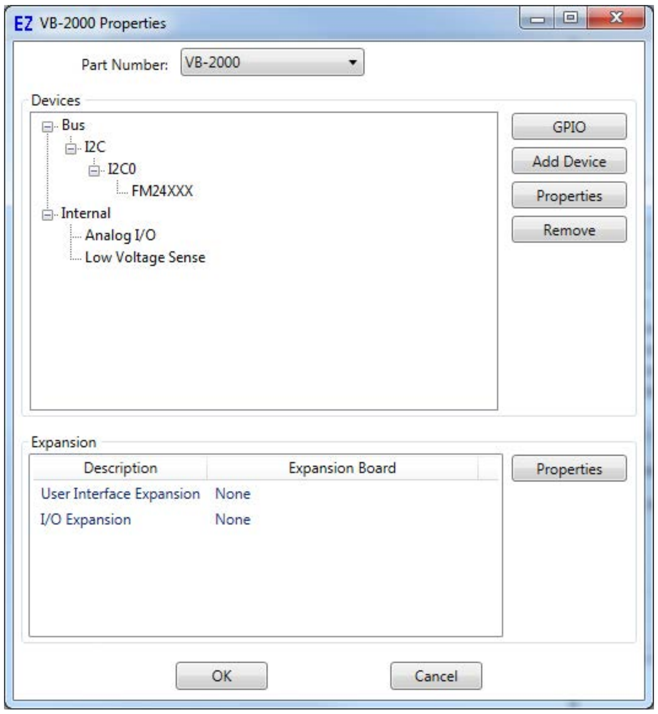
5. The PLCHIP_Pxx_eeprom Properties window will open. This window is used to configure the number of EEPROM bytes to be blocked for retentive memory use only. See the Retentive Memory / Memory ‘ Limitations part of this manual section for details on how and when EEPROM may be used for retentive. Refer to Figure 7-5. Leave the Num Retentive Bytes set to zero unless you are planning to use the EEPROM for retentive memory storage.

6. The PLCHIP_Pxx_eeprom is now listed as an internal device in the Devices Pane. Click **OK** to close the *VB-2XXX Properties*.

7. Click **OK** to close the *Project Settings* Window.

8. Save your ladder diagram using the menu **FILE** and **SAVE** or **SAVE AS** to save the current settings in your program.

The EEPROM functionality is now installed in the ladder diagram program. The EEPROM_WRITE and EE-PROM_READ function blocks should now be available in the function block drop down list.


VB-2000 Properties

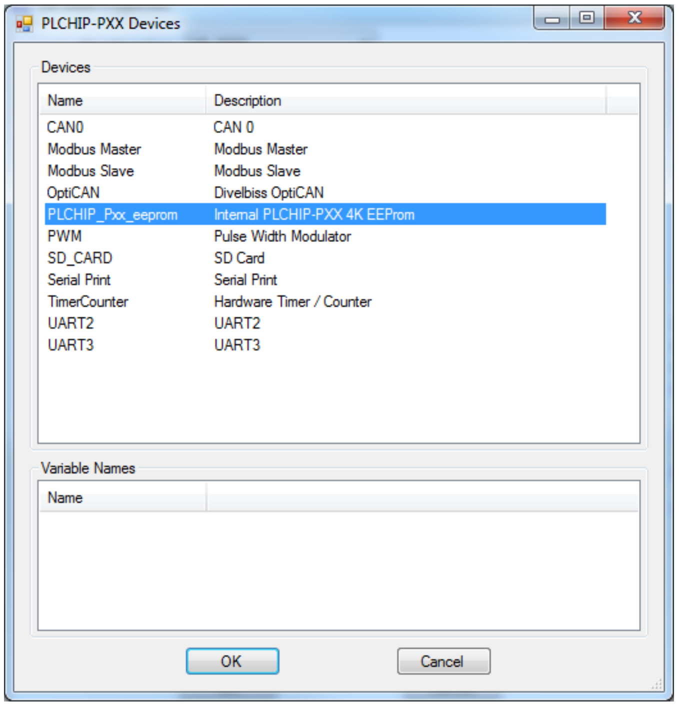
Part Number: VB-2000

Devices
- Bus
  - I2C
    - I2C0
      - FM24XXX
- Internal
  - Analog I/O
  - Low Voltage Sense

Expansion

| Description | Expansion Board |
|---|---|
| User Interface Expansion | None |
| I/O Expansion | None |

Figure 7-5


PLCHIP-PXX Devices

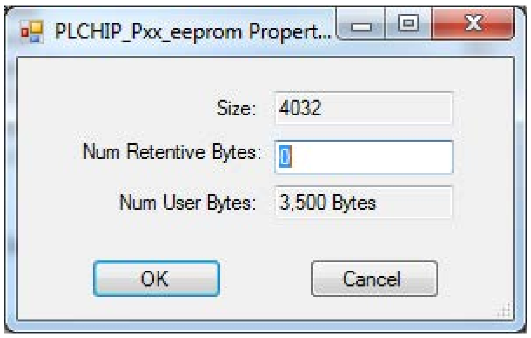
Devices

| Name | Description |
|---|---|
| CAN0 | CAN 0 |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |
| UART3 | UART3 |

Variable Names

Name

Figure 7-6


PLCHIP_Pxx_eeprom Propert...

Size: 4032

Num Retentive Bytes: 0

Num User Bytes: 3,500 Bytes

Figure 7-7

## Using EEPROM Memory

EEPROM memory is accessed by using the EEPROM_WRITE and EEPROM_READ function blocks. The EEPROM_WRITE and EEPROM_READ function blocks use variables to set the EEPROM address.

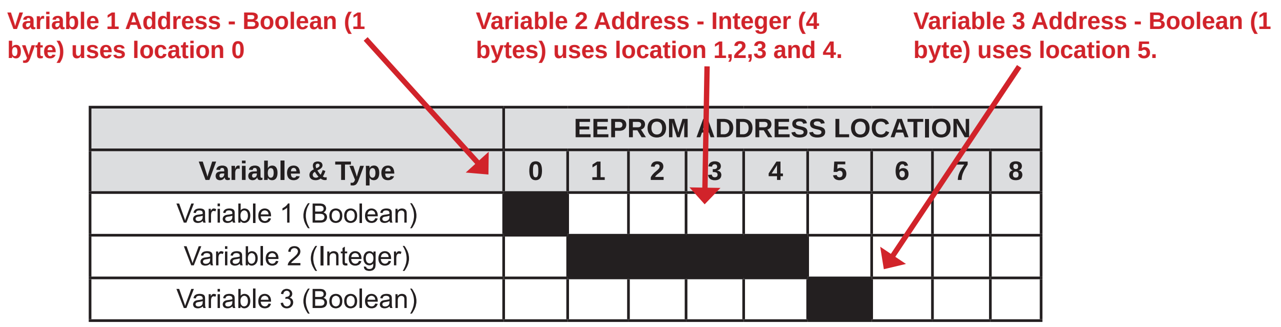
To write and read values from the EEPROM, you must understand that the EEPROM memory is basically a bank of memory and the variable values may be stored into this bank. The EEPROM bank is organized by per byte and each variable type has a specific number of bytes that it will require. Boolean variables fill one byte while all other variable types fill four bytes of EEPROM.

EEPROM based on its design does have a limited life based on number of write cycles. This number is generally large, EEPROM is not recommended for storage where values are changing and stored often (seconds or minutes). It would be appropriate for configuration items stored randomly and items stored at slow rates.

The address provides the location where to store the variable or from where to read the data into a variable from. The actual address is the first byte location of the EEPROM memory. Each EEPROM address is absolute and is one byte in size. To correctly store and read variables (of the same or different type), they must be mapped based on the starting byte location (address) and the number of bytes to store or read for the variable type.

When writing a boolean to address 0, the actual variable will use addresses 0 (one byte). Should you write an integer variable into address 0, then it would use addresses 0-3 (4 bytes). A memory map should be created and used to assign variable types and addresses prior to coding to ensure that variable size and types are accounted for.

You must use the same address for writing and reading a variable for correct operation. If the addresses are not the same and/or you have overwritten some bytes of where a value is stored, the data read will be corrupted.


Variable 1 Address - Boolean (1 byte) uses location 0
Variable 2 Address - Integer (4 bytes) uses location 1,2,3 and 4.
Variable 3 Address - Boolean (1 byte) uses location 5.

| | EEPROM ADDRESS LOCATION | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Variable & Type** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| Variable 1 (Boolean) | ■ | | | | | | | | |
| Variable 2 (Integer) | | ■ | ■ | ■ | ■ | | | | |
| Variable 3 (Boolean) | | | | | | ■ | | | |

Refer to **Appendix A - Function Reference** for details on using the EEPROM_READ and EEPROM_WRITE function blocks.

## FRAM used as EEPROM

When configuring Retentive Memory (see earlier this Chapter), the NumUser Bytes are automatically calculated based on the number of retentive bytes configured. The NumUser Bytes are memory locations in FRAM that may be used to store data just as in the EEPROM using the EEPROM_WRITE and EEPROM_READ function blocks.

The FRAM memory locations, as they are not based on EEPROM technology do not have the write cycle limitations. FRAM locations will not fail after repeated write cycles; therefore, they are appropriate to use in any situation.

When using the EEPROM_READ or EEPROM_WRITE function block, ensure you select the correct device from the drop down menu (the menu is available when placing the function blocks in the ladder diagram). The FMXXX devices are FRAM devices while the PLCHIP_Pxx_eeprom is the actual PLC on a Chip EEPROM.

# CHAPTER 8

## Pulse Width Modulation

This chapter provides basic information to understand what Pulse Width Modulation is and how it is used as feature in the EZ LADDER Toolkit and hardware target.

## Chapter Contents

## What is Pulse Width Modulation

Pulse Width Modulation, also referred to as PWM is a term common to the industrial controls and electronics industries. Essentially PWM is generally an output that can be controlled in such a manner that will cause a device connected to have varying operation. Consider a light dimmer, changing the knob changes the light intensity; this is how a PWM output can affect a load such as a light.

Pulse-width modulation (PWM) is used for controlling the amplitude of digital signals in order to control devices and applications requiring power or electricity. It essentially controls the amount of power, in the perspective of the voltage component, that is given to a device by cycling (known as the ***frequency***) the on-and-off phases of a digital signal quickly and varying the width of the “on” phase or ***duty cycle***. To the device, this would appear as a steady power input with an average voltage value, which is the result of the percentage of the on time. The ***duty cycle*** is expressed as the percentage of being fully (100%) on. The frequency is express in hertz (hz). Figure 8-1 illustrates an example PWM output waveform.

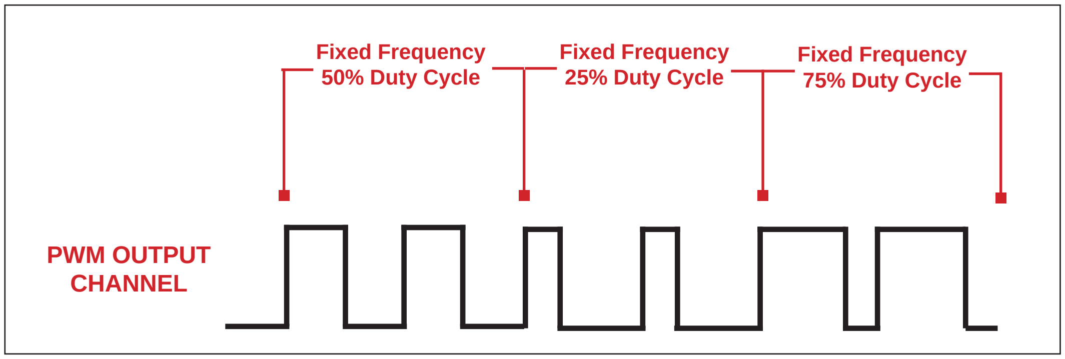


**Figure 8-1**

## PWM Output Basics

Pulse Width Modulation (PWM) Outputs for P-Series targets are easy implement and utilize using the EZ LADDER Toolkit. PWM channels are 32 bit resolution and up to 12 channels are available based on the actual hardware target.

PWM functionality, supported types and number of channels is always target dependent. While EZ LADDER Toolkit provides the basic programmability, the hardware target must support PWM and the selected configuration for the PWM outputs to operate correctly.

PWM output frequency is dependent upon the actual hardware target and the resolution configured. Changing the resolution (or hardware target) may change the acceptable range of the PWM outputs.

## Configuring PWM in Project Settings

As with most EZ LADDER Toolkit hardware supported features, the PWM channels and functionality must installed and configured before it may be used in a ladder diagram project. The PWM channels are configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The Target Properties window will open. If PWM is already listed in the *Devices pane*, you can select it (highlight) and click the **PROPERTIES** button to make additional configurations to the PWM. If the PWM is not listed in the *Devices pane*, it must be installed and configured. Figure 8-2 shows no installed devices.

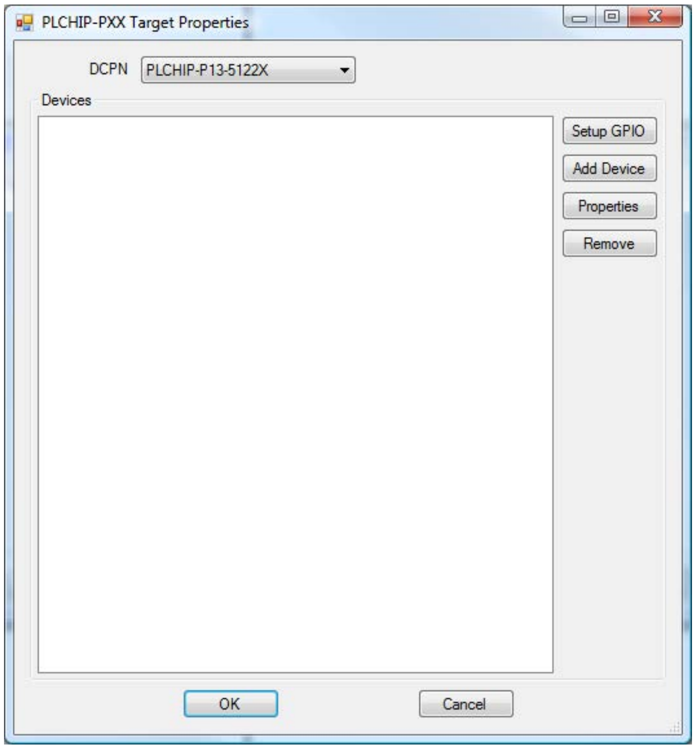
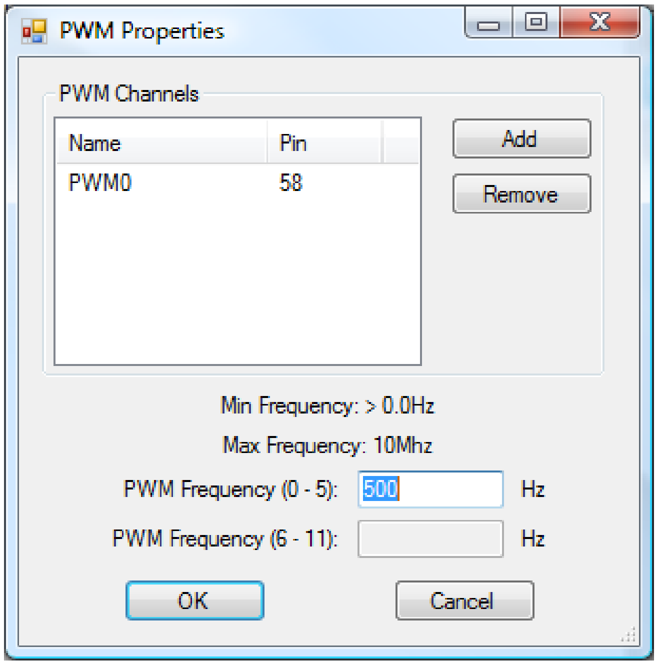


**Figure 8-2**

To install the PWM, click the **ADD DEVICE** button. The *Devices window* will open. Choose **PWM** from the devices (click to highlight) and then click **OK**. The *PWM Properties* window will open. See Figure 8-3.

Click the **ADD** button, the *Add PWM Channels* dialog will open. Select the channel(s) and click **OK** to add them.

There are up to 12 channels available based on the target. Channels 0-5 operate on a base PWM frequency and channels 6-11 operate on a different base PWM frequency. These frequencies are > 0 Hz and up to 10 MHz. Enter the base frequency desired for the channels that have been added. Click **OK** to close the PWM Properties window, click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project (program).


PWM Properties

PWM Channels

| Name | Pin |
|---|---|
| PWM0 | 58 |

Min Frequency: > 0.0Hz

Max Frequency: 10Mhz

PWM Frequency (0 - 5): 500 Hz

PWM Frequency (6 - 11): Hz

**Figure 8-3**

The actual PWM base frequency will be application dependent. The PWM frequency can be changed in the ladder diagram program using the PWM_FREQ function block.

## Controlling PWM in the Ladder Diagram Project

With PWM channels configured in the Project Settings, it is simple to control the actual PWM channels in the ladder diagram project.

### Enabling a PWM Channel

To control a PWM output, specifically when it is enabled, disabled and it's duty cycle, the ***PWM*** function block is used.  This function block has two inputs (EN for Enable and DC for Duty Cycle) and also has one output (Q). When the PWM function block is enabled (the EN input is true), the PWM channel is active and operating at the frequency defined in the project settings and the Duty Cycle (variable connected to DC of the PWM function block).  When the EN input is false, the PWM channel output is disabled.  Figure 8-4 illustrates the PWM function block in a sample circuit.

When placing the PWM function block, a new dialog is opened to select the PWM channel and the polarity of the PWM Channel.

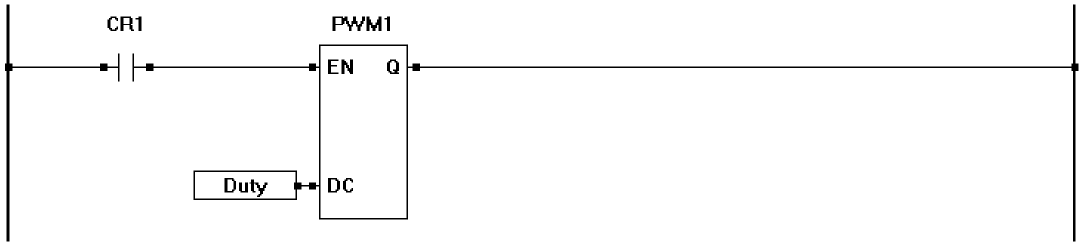


Figure 8-4

## Controlling the PWM Channel Duty Cycle

The PWM output's duty cycle is controlled the PWM function block for that channel. Changing the value of the variable connected to the DC input of the PWM function block immediately changes the duty cycle accordingly. This gives a PWM output the ability to change duty cycle in real-time in response to control parameter changes.

## Changing the PWM Frequency

In addition to an adjustable duty cycle, the PWM clock frequencies (CLK A / CLK B) can be changed in the ladder diagram project by use of the PWM_FREQ function block. The PWM_FREQ function block has two inputs (EN for enable and F for frequency) and one output (Q). When the EN is true, the PWM channel frequency is changed to the value of the variable connected to the F input of the function block. Figure 8-4 illustrates a sample circuit using PWM_FREQ.

When using the PWM_FREQ to change the frequency, the actual CLK A or CLK B frequency is changed. This affects all channels that use that specific CLK signal. For example, if PWM channel 0 uses CLK A and PWM channel 2 uses CLK A, then adjusting the frequency using PWM_FREQ to CLK A affects all the PWM channels that use CLK A, in this case 0 and 2 respectively.

When placing the PWM_FREQ function block, a new dialog is opened to select the PWM channel Clock.

The PWM clock frequency will be equal to the value of the F input of the PWM_FREQ function block, thus allowing real-time frequency changes.

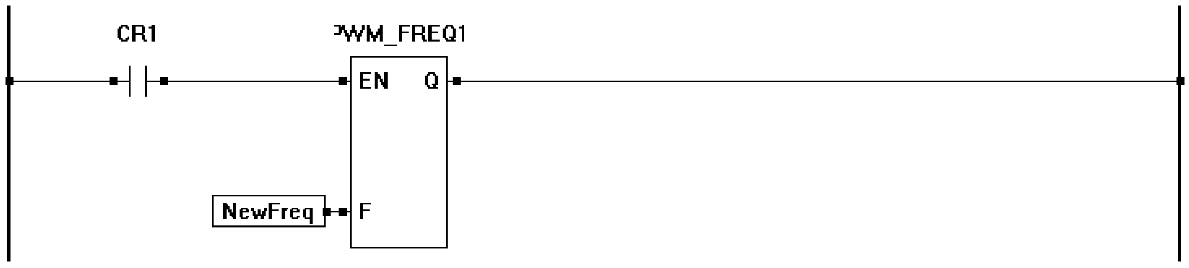


Figure 8-5

# CHAPTER 9

## LCD Display Support

This chapter provides basic information to understand how to install, configure and use a Character or Graphics LCD Display with the EZ LADDER Toolkit

## Chapter Contents

## Character (CHR) LCD Display Functionality

EZ LADDER Toolkit provides the ability to display text and variables using it's built-in LCD support. EZ LADDER Toolkit supports LCD displays that meet the HD44780 interface specification. In addition to the specification, the displays must have 1 to 4 rows and 8-40 columns.

LCD support is based on actual hardware target specifications. PLC on a Chip™ Integrated Circuits and Modules support LCD display functionality. For PLCs and controllers, refer to the supported features. See **Chapter 23 - Hardware Targets**.

EZ LADDER Toolkit supports only one character LCD display or one graphics LCD displayin a ladder diagram project.

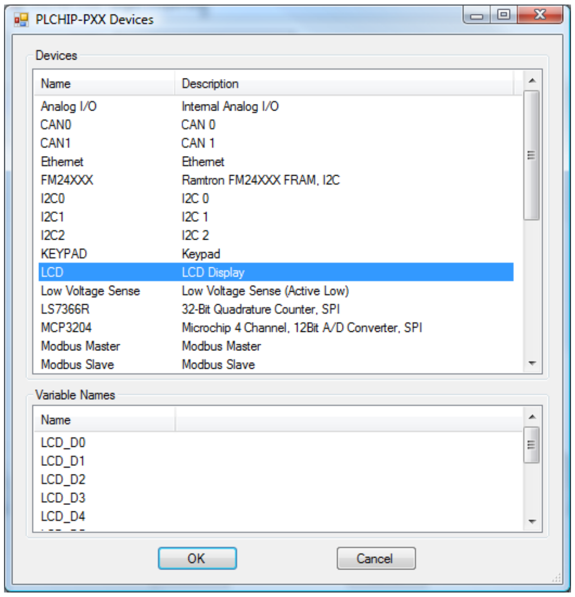
## Configuring the CHR LCD Display in the Project Settings

To be able to use an LCD display in an EZ LADDER Toolkit ladder diagram project, the LCD display must first be installed and configured. The LCD display is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN) , select the actual part number (if not already done). In the Device section of the Devices Pane, if the LCD is installed, it will be displayed. To install the device, click the **ADD DEVICE** button. The *Target Devices* window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find LCD. Figure 9-1 shows the Target Devices window.

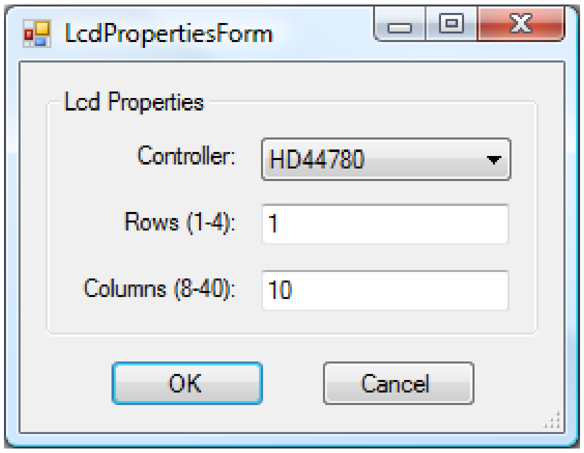


| Name | Description |
|---|---|
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |

| Name |
|---|
| LCD_D0 |
| LCD_D1 |
| LCD_D2 |
| LCD_D3 |
| LCD_D4 |

**Figure 9-1**

Click **LCD** and click **OK**. The LCDpropertiesForm dialog will open. using this dialog set the number of Rows (1-4) and the number of Columns (8-40). Click OK when the LCD properties are entered to close the dialog and return to the Target Properties Window. You will now see the LCD listed in the Devices pane. See Figure 9-2 for setting the rows and columns.



Controller: HD44780
Rows (1-4): 1
Columns (8-40): 10

**Figure 9-2**

Click the **OK** button to close the *Target Properties* window and click **OK** to close the *Project Settings* window. Be sure to save the ladder diagram project (program).

The LCD display can now be utilized from the ladder diagram project.

## Displaying Messages on the CHR LCD Display

With the LCD display configured in the ladder diagram project, it is now possible to use the EZ LADDER Toolkit's function blocks to display messages and variables. Two basic function blocks that are used control the display are: **LCD_CLEAR** and **LCD_PRINT**. The LCD can also be printed to and controlled using structured text.

### Clearing the Display

To clear the LCD display (blank all rows and columns), the LCD_CLEAR function block is used. The LCD_CLEAR will clear the display when it senses a rising edge on it's enable input (EN). Figure 9-3 shows an example program using the LCD_CLEAR function block.

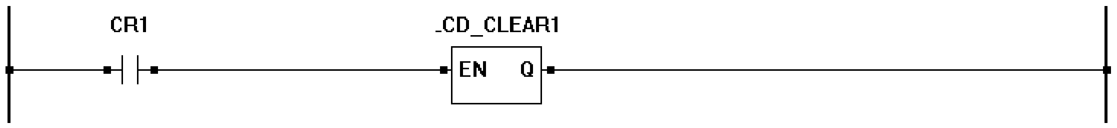
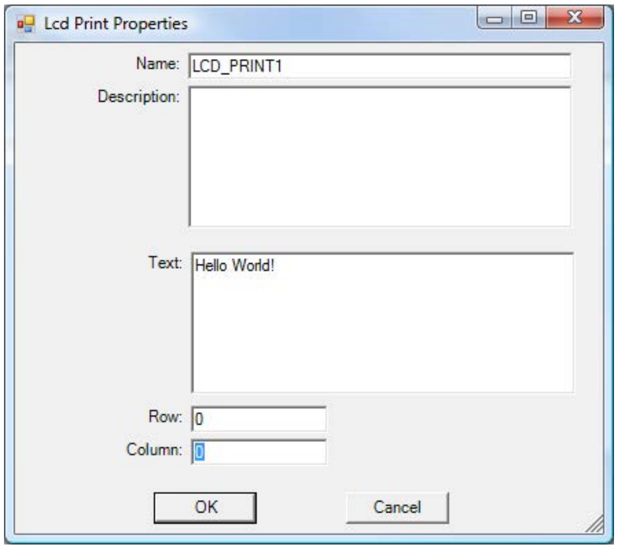


**Figure 9-3**

## Writing to the Display

To write messages to the LCD Display, the **LCD_PRINT** function block is used. Using the **LCD_PRINT** function block is a two step process. When placing the function block, a new LCD Print Properties dialog box will open. See Figure 9-4. The *Text* field is where the message is typed that will be displayed. The *Row* field is the row of the display where the text will be displayed. The Column field is the column where the text will begin displaying.


Lcd Print Properties

Name: LCD_PRINT1

Description:

Text: Hello World!

Row: 0

Column: 0

**Figure 9-4**

The first row and first column are always zero (0) and are limited by the actual hardware target display size. If text in a row is more than can be displayed on the LCD, it will be truncated. It does not automatically wrap to the next line. Each row of the display must be written to individually with separate LCD_PRINT function blocks.

When all the information is entered, clicking **OK** will cause the function block to be placed in the ladder diagram project. Figure 9-5 is a sample of a complete LCD_PRINT circuit.

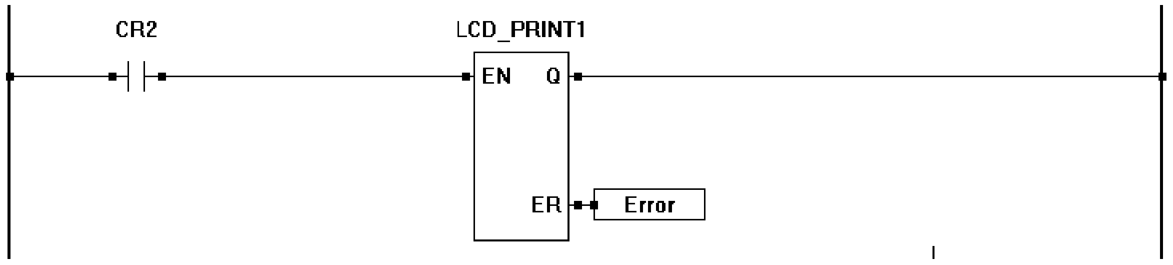
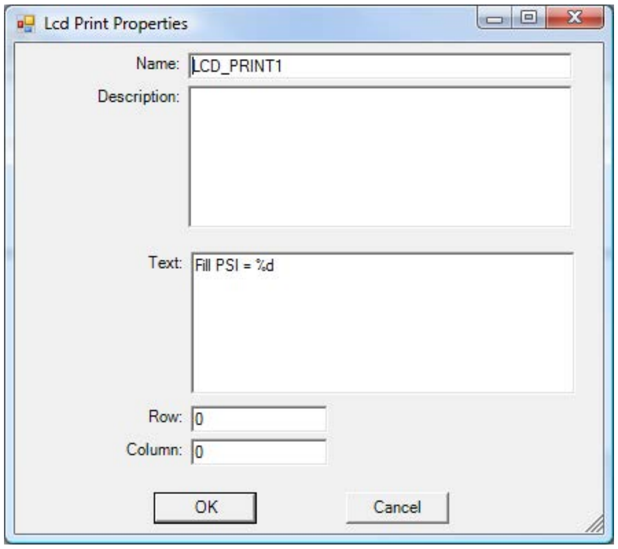


**Figure 9-5**

## Writing Variables to the Display

In addition to printing static text, it is often desirable to be able to print variables to the display. This is helpful in displaying process parameters and menu items. To write a variable to the LCD display, the same LCD_PRINT function block is still used. As in the simple text printing, the text is entered into the *Text* field.

In addition to the text, *control characters* may be inserted that represent variables and how to format the variable text. For a full listing of what control characters and formatting is supported, please see the LCD_ PRINT function block in **Appendix A - Function Reference.** Figure 9-6 illustrates a sample text dialog with control characters.



Name: LCD_PRINT1
Description:
Text: Fill PSI = %d
Row: 0
Column: 0

**Figure 9-6**

When an LCD_PRINT function is inserted to display variables, a new variable input is added to the function block automatically for each variable that will be displayed.

Figure 9-7 represents a sample ladder diagram project using a LCD_PRINT function block with a variable input that will be displayed.

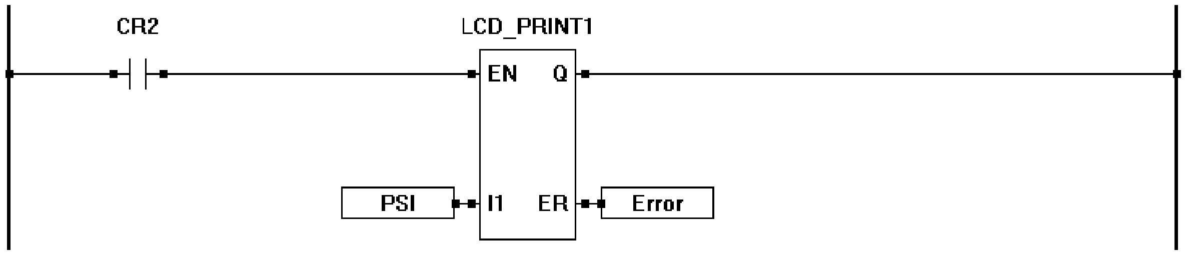


**Figure 9-7**

The LCD_PRINT function block is rising edge sensitive. Therefore, it will only display one time as the ENable input goes high. The text will appear normally, but the variable will not appear to update or change as the ladder diagram is executing.

To overcome the rising edge issue when displaying variable, create TON timer circuit as shown in Figure 9-8 and use the timer contact to act as a refresh for the ENable input on the LCD_PRINT function block. The refresh timer should be adjusted to your display preferences. in Figure 9-8, CR1 will toggle on and off based on the Timer function TON, giving the result of the LCD_PRINT seeing a rising edge at that timing rate.

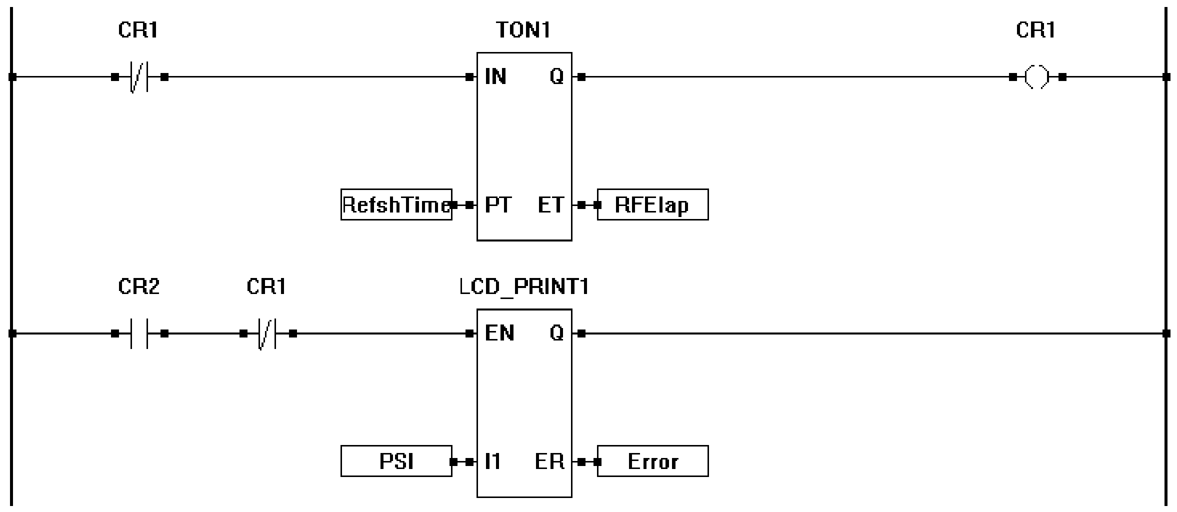


**Figure 9-8**

For more detail on all EZ LADDER Toolkit Function Blocks and objects, refer to **Appendix A - Function Reference**.

## Graphics (GFX) LCD Display Functionality

EZ LADDER Toolkit provides the ability to display graphics, multiple sized text and variables using it's built-in Grpahics LCD support. EZ LADDER Toolkit supports specific graphics LCD displays.

The following graphics LCD displays are supported:

Crystal Fontz part number: CFAG12864A-*xxxx*
New Haven part number: NHD-12864AZ-*xxxx*
*xxxx are optional items in the part number for specifying features.*

Graphics LCD support is based on actual hardware target specifications. PLC on a Chip™ Integrated Circuits and Modules support LCD display functionality. For PLCs and controllers, refer to the supported features. See **Chapter 23 - Hardware Targets**.

EZ LADDER Toolkit supports only one character LCD display or one graphics LCD displayin a ladder diagram project.

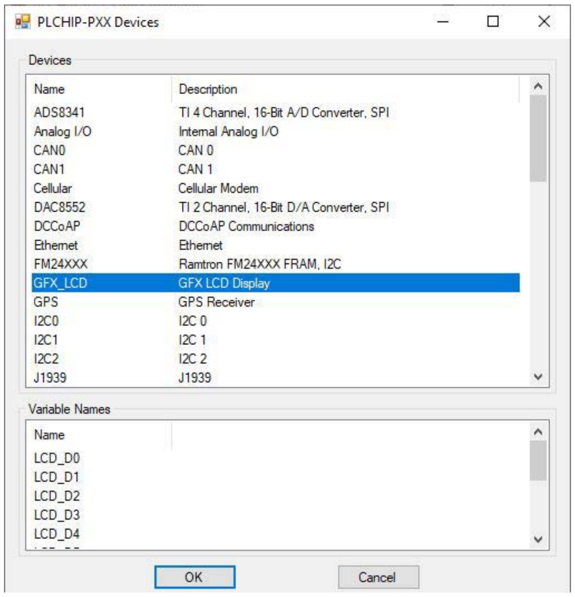
## Configuring the GFX LCD Display in the Project Settings

To be able to use a graphics LCD display in an EZ LADDER Toolkit ladder diagram project, the graphics LCD display must first be installed and configured. The graphics LCD display is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN) , select the actual part number (if not already done). In the Device section of the Devices Pane, if the GFX LCD is installed, it will be displayed. To install the device, click the **ADD DEVICE** button. The *Target Devices* window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find **GFX LCD**. Figure 9-9 shows the Target Devices window.



| Name | Description |
|---|---|
| ADS8341 | TI 4 Channel, 16-Bit A/D Converter, SPI |
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Cellular | Cellular Modem |
| DAC8552 | TI 2 Channel, 16-Bit D/A Converter, SPI |
| DCCoAP | DCCoAP Communications |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| GFX_LCD | GFX LCD Display |
| GPS | GPS Receiver |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| J1939 | J1939 |

| Name |
|---|
| LCD_D0 |
| LCD_D1 |
| LCD_D2 |
| LCD_D3 |
| LCD_D4 |

**Figure 9-9**

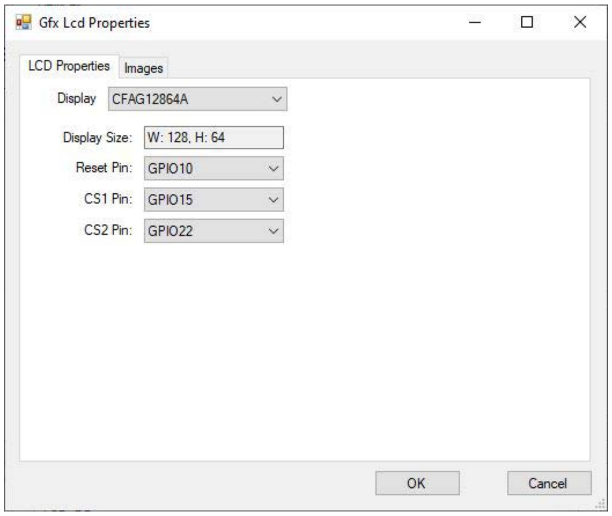
Click **GFX LCD** and click **OK**. The *Gfx Lcd Properties* dialog will open. Using the **Display** drop-down, select the graphics display being used from the supported list. Additional fields will now display that are required configuration items before the GFX LCD display can be added to the project. Refer to Figure 9-10.

**Display Size**: Automatically updated with the size of the display (W for width, H for height).

**Reset Pin**: Select the PLC on a Chip GPIO pin connected to the graphics display's reset pin. For controllers with built-in graphics support, this featue is automatically populated or limited in choices.

**CS1 Pin:** Select the PLC on a Chip GPIO pin connected to the graphics display's CS1 (chip select 1) pin. For controllers with built-in graphics support, this featue is automatically populated or limited in choices.

**CS2 Pin:** Select the PLC on a Chip GPIO pin connected to the graphics display's CS2 (chip select 2) pin. For controllers with built-in graphics support, this featue is automatically populated or limited in choices.



Display: CFAG12864A
Display Size: W: 128, H: 64
Reset Pin: GPIO10
CS1 Pin: GPIO15
CS2 Pin: GPIO22

**Figure 9-10**

Click OK when the GFX LCD properties are entered to close the dialog and return to the *Target Properties* Window. You will now see the GFX LCD listed in the Devices pane.

Click the **OK** button to close the *Target Properties* window and click **OK** to close the *Project Settings* window. Be sure to save the ladder diagram project (program).

The GFX LCD display can now be utilized from the ladder diagram project using **Structured Text**.

Due to the graphical nature of the GFX LCD display, the ability to size fonts and display images, the GFX LCD display can only be controlled and printed to using Structured Text using Target specific functions in the structured text editor.

## Displaying Items on the GFX LCD Display

Due to the graphical nature of the GFX LCD display, the ability to size fonts and display images, the GFX LCD display can only be controlled and printed to using Structured Text using Target specific functions in the structured text editor. Each function provides control over some printing aspect.

### Graphics LCD Display Layout

To control printing graphics and text to the display, the first step is identifying the graphics' display layout. When the display was selected in the Project Settings, the Display Size provided a W (width) and H (height) value. See Figure 9-10. These are the maximum number of pixels supported in each of the directions and always print from 0 to the (maximum -1) location in each direction. Location 0,0 is always the top left-hand corner of the GFX display. The locations is specified by width (x), height (y). Refer to Figure 9-11.

Printing locations are specified using coordinates (x,y). X for the horizontal and y for the vertical. The top left-hand corner is always location 0,0.

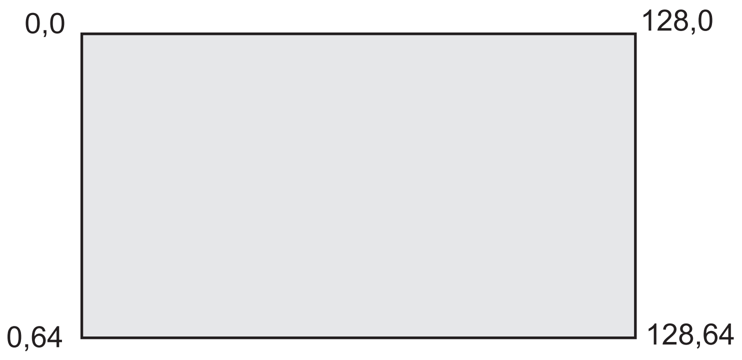


**Figure 9-11**

## Displaying Overview

The GFX display has the ability to display different types of images and text based on the structured text command used. These include lines, rectangles, images and text (including font size changes). Lines, rectangles and text are generated and displayed by the structured text commands while images are rendered based on provided image files.

The following structured text commands are used to control / print to the graphics LCD display.

**EZ_LcdClear**
This function erases / clears everything from the graphics LCD display.

**EZ_LcdDrawImage**
This function draws an image on the graphics LCD display (from stored image files).

**EZ_LcdDrawLine**
This function draws an line on the graphics LCD display.

**EZ_LcdDrawRectangle**
This function draws an un-filled rectangle on the graphics LCD display.

**EZ_DrawRectangleFilled**
This function draws a filled rectangle on the graphics LCD display.

**EZ_LcdSetFont**
This function is used to set the font on the graphics LCD display. The only supported font is the de fault font at this time (6x8).

**EZ_LcdSetFontSize**
This function is used to set the font size (scale) on the graphics LCD display.

**EZ_LcdSetPixel**
This function is used to turn on/off a pixel on the graphics LCD display.

**EZ_LcdWriteString**
This function is used to display text on the graphics LCD display.

For details on the structured text functions syntax and options, refer to **Appendix B - Target Specific ST Function Reference**. For more information on using structured text, refer to **Chapter 26 - Structured Text**.

## Displaying Images

The GFX display has the ability to display pre-drawn images. These images must be stored in the same directory as the ladder diagram project (.dld file). The images is drawn using the **EZ_LcdDrawImage** structured text function. These images are listed in the ***Variables*** tab of the Structured Text editor. Refer to Figure 9-12.

The images to be drawn must be stored in the same directory as the ladder diagram project (.dld file). When in the correct location, the names will appear in the ***Variables*** tab of the Structured Text editor.

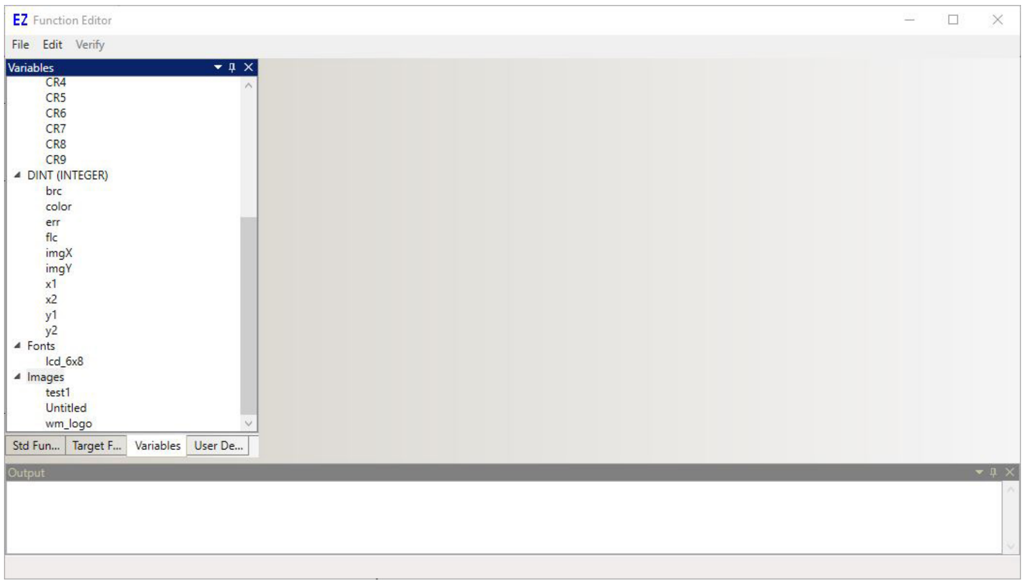


**Figure 9-12**

Images may be cropped when displayed if there is not enough room to draw the entire image based on image size and placement location.

## Fonts & Font Scaling

The GFX display has the ability to display scaled font sizes (and future releases may allow for different fonts). The font size  is drawn using the **EZ_LcdSetFontSize** structured text function. Fonts are listed in the ***Variables*** tab of the Structured Text editor. The lcd_6x8 font is set by default and cannot be changed.

# CHAPTER 10

## Keypad Support

This chapter provides basic information to understand how to install, configure and use the Keypad feature in the EZ LADDER Toolkit.

## Chapter Contents

## Keypad Functionality

EZ LADDER Toolkit provides the ability for the addition of keypad functionality. EZ LADDER Toolkit supports a basic 4 row, 5 column keypad matrix. This keypad matrix includes the numbers 0-9, Enter, Clear, Up, Down, +/-, Decimal Point, and F1-F4 (programmable function keys). Using this keypad matrix and the built-in EZ LADDER functions, menus and user interactions may be programmed into a ladder diagram project.

Keypad support is based on actual hardware target specifications. PLC on a Chip™ Integrated Circuits and Modules support Keypad functionality. For PLCs and controllers, refer to the supported features, see **Chapter 23 - Hardware Targets**.

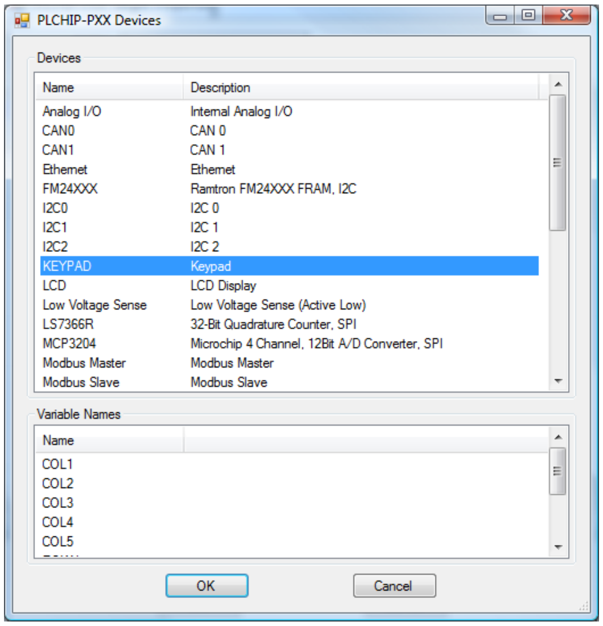
## Configuring the Keypad in the Project Settings

To be able to use an keypad in an EZ LADDER Toolkit ladder diagram project, the keypad must first be installed and configured. The keypad is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the keypad were installed, it would be listed in the *Devices pane* under the Devices section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find Keypad. Figure 10-1 shows the Target's Devices window.


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |

Variable Names

| Name |
|---|
| COL1 |
| COL2 |
| COL3 |
| COL4 |
| COL5 |

**Figure 10-1**

Click *Keypad* and click **OK**. The Devices window will close and return to the Target Properties window. The Device section will now list the Keypad as an installed device. Refer to Figure 10-2.

No additional configuration is required to begin using the Keypad. Click **OK** close the Target Properties window and click **OK** again to close the Project Settings window. Use the File Menu and Save the ladder diagram project. The keypad matrix can now be utilized from the ladder diagram project

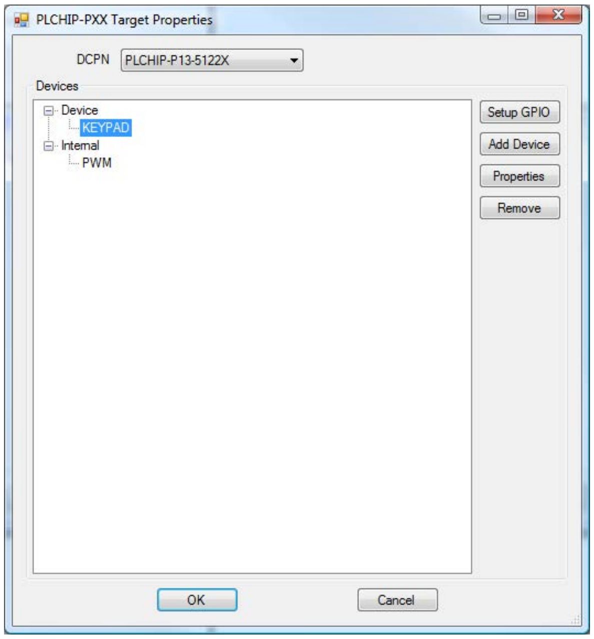


**Figure 10-2**

## Getting Data from the Keypad

With the keypad configured in the ladder diagram project, it is now possible to use the EZ LADDER Toolkit's function blocks and objects to input user data and set points. The keypad can be read using two methods. The two methods are: Integer and Real Variable entry using the Keypad Function block and the second is identifying discrete key presses using contacts.

### Real and Integer Inputs using the Keypad Function Block(s)

There are two Keypad function blocks provided for use in the ladder diagram program, **Keypad** and **Keypad2**.

### Using the KEYPAD Function

To read data (integer or real) from the keypad using the **KEYPAD** function block, select the KEYPAD function block from the drop-down menu and place it in the ladder diagram at the desired location. The Keypad block will be inserted into the ladder diagram.

Each keypad function block has three inputs and three outputs. As with all function blocks, the EN (enable) will enable the keypad function block or disable it. The MI and MA inputs are used to identify Minimum and Maximum allowed entries respectively. The Q Output is true when the function is enabled. The KB output will maintain the contents of the keypad buffer while KO is the actual value that was entered on the keypad (and **ENTER** pressed). Figure 10-3 represents a typical keypad function in a ladder diagram project.

The Keypad function block can be used to input real or integer variables. When connecting a variable, the type connected will limit all number inputs and outputs to the selected type (all integer or all real).

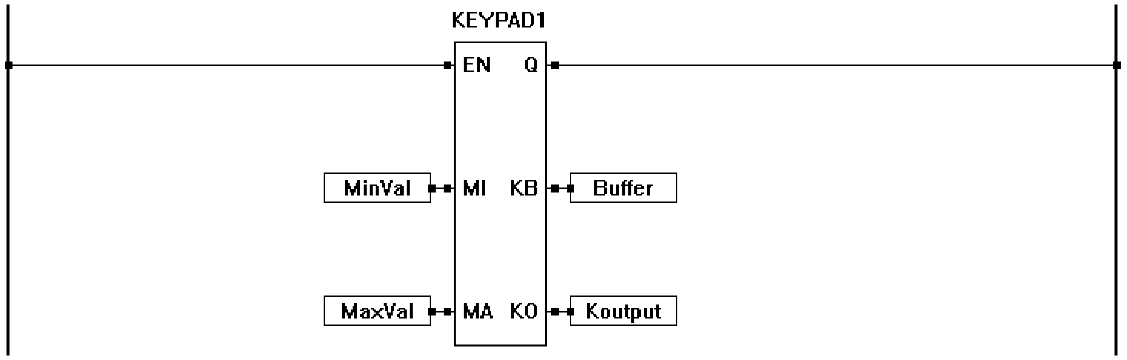


**Figure 10-3**

## Using the KEYPAD2 Function

The **KEYPAD2** function block provides additional features over the KEYPAD function block. These features allow menus and Discrete key press menu items to be combined, allowing for a more powerful and easier to implement menu.

To read data (integer or real) from the keypad using the **KEYPAD2** function block, select the KEYPAD2 function block from the drop-down menu and place it in the ladder diagram at the desired location. The KEYPAD2 block will be inserted into the ladder diagram.

Each KEYPAD2 function block has three inputs and six outputs. As with all function blocks, the EN (enable) will enable the KEYPAD2 function block or disable it. The MI and MA inputs are used to identify *Minimum* and *Maximum* allowed entries respectively.

The Q Output is true when the function is enabled. The KB output will maintain the contents of the keypad buffer while KO is the actual value that was entered on the keypad (and **ENTER** pressed). The M output is a boolean that is set to true when any number (0-9), + or . is pressed. Pressing the Clear or Enter will reset the M output to false. The KP output is a boolean output that is true only for the single scan that a key was pressed. The KY output is an integer output of the actual ASCII value of the key that was pressed.

Figure 10-4 shows the ASCII output for the key press detected.

Figure 10-5 represents a typical KEYPAD2 function in a ladder diagram project.

| KEY | ASCII Value | KEY | ASCII Value |
|---|---|---|---|
| 0 | 48 | F1 | 65 |
| 1 | 49 | F2 | 66 |
| 2 | 50 | F3 | 67 |
| 3 | 51 | F4 | 68 |
| 4 | 52 | UP | 69 |
| 5 | 53 | DOWN | 70 |
| 6 | 54 | ENTER | 13 |
| 7 | 55 | CLEAR | 8 |
| 8 | 56 | Decimal Point (.) | 46 |
| 9 | 57 | + / - | 45 |

**Figure 10-4**

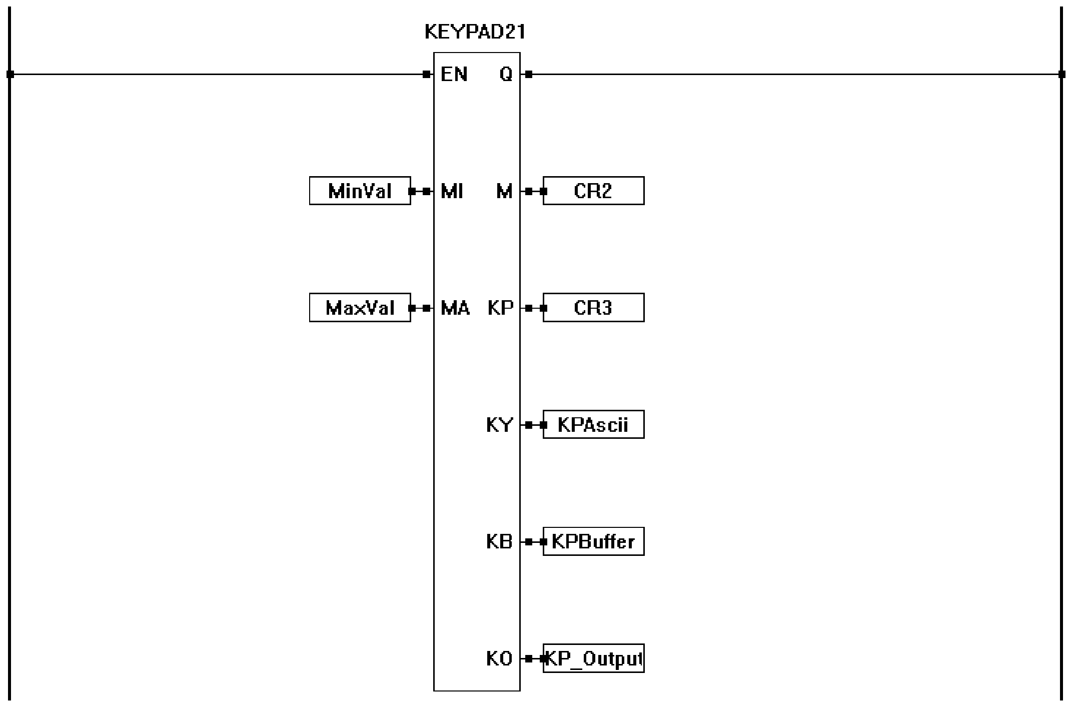


**Figure 10-5**

## Reading Discrete Key Presses using Contacts

Before being able to read any key presses using the Discrete key method, the ladder diagram must have at least one KEYPAD (or KEYPAD2) Function Block installed and in use. Any discrete keys will not operate unless one KEYPAD (or KEYPAD2) Function Block is installed in the ladder diagram project.

In addition to reading complete values from the keypad, it is possible to read individual keys to determine if they are pressed. Each key has a predefined address that can be used as an input (boolean type variable that is classified as an input). Create a contact as a new variable, and in the ***Var I/O Number*** field, enter the address of the specific key desired. When the key is pressed, the contact will be true.

The following addresses are used to read discrete keypad buttons.

| I/O Assignment | Button Description | I/O Assignment | Button Description |
|---|---|---|---|
| KB_0 | Numeric 0 | KB_CLEAR | Clear Button |
| KB_1 | Numeric 1 | KB_DP | Decimal Point Button |
| KB_2 | Numeric 2 | KB_+- | + / - Button |
| KB_3 | Numeric 3 | KB_F1 | F1 Button |
| KB_4 | Numeric 4 | KB_F2 | F2 Button |
| KB_5 | Numeric 5 | KB_F3 | F3 Button |
| KB_6 | Numeric 6 | KB_F4 | F4 Button |
| KB_7 | Numeric 7 | KB_UP | Up Button |
| KB_8 | Numeric 8 | KB_DOWN | Down Button |
| KB_9 | Numeric 9 | KB_ENTER | Enter Button |

For more detail on all EZ LADDER Toolkit Function Blocks and objects, refer to **Appendix A - Function Reference**.

# CHAPTER 11

## UARTS and Serial Ports

This chapter provides basic information to understand how to install, configure and use the UARTS, Serial Ports and to enable the Serial Printing feature in the EZ LADDER Toolkit.

## Chapter Contents

# UARTS & Serial Ports

EZ LADDER Toolkit provides the software interface to bring P-Series hardware target's communication ports to life. These serial ports (name UARTS in EZ LADDER) may be used to communicate Modbus Master or Slave, Print Serially to external devices and even transmit or receive data using Structured Text. For more information about structured text, see **Chapter 26 - Structured Text**.

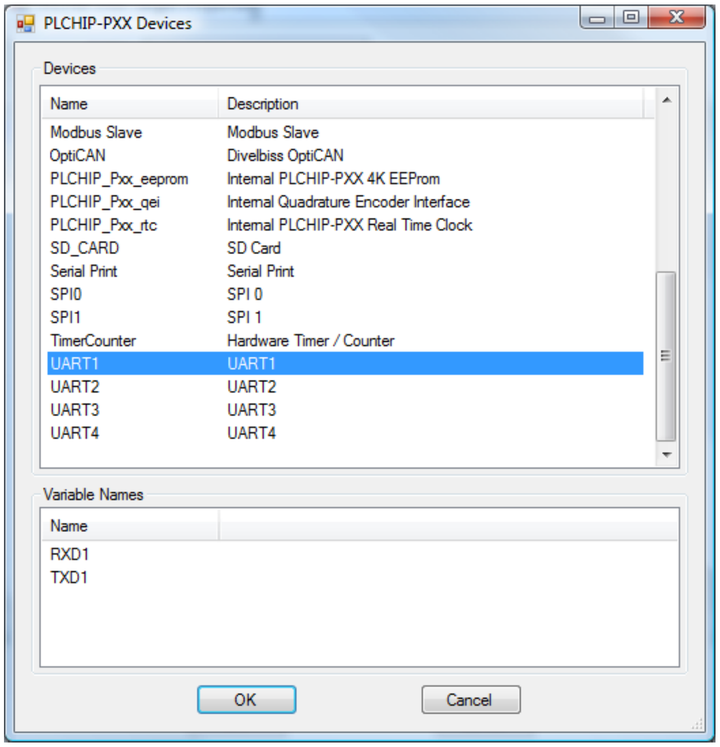
The availability of UARTs (Serial Ports) is entirely dependent upon the actual target. For more information regarding hardware target support and features, see **Chapter 23 - Hardware Targets.**

Prior to using the UARTs in EZ LADDER Toolkit, the UART(s) must be installed and configured in the target. The UARTs are configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any UARTs were installed, they would be listed in the *Devices pane* under the Bus/Uart section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find the UARTS (UART1 - UART4). Figure 11-1 shows the Target's Devices window.

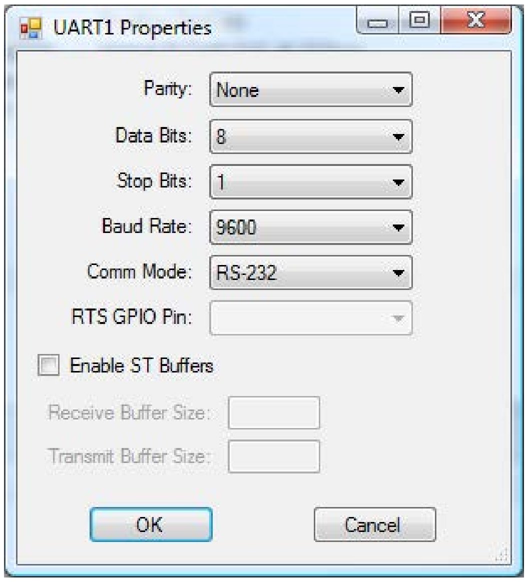


| Name | Description |
|---|---|
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART1 | UART1 |
| UART2 | UART2 |
| UART3 | UART3 |
| UART4 | UART4 |

| Name |
|---|
| RXD1 |
| TXD1 |

**Figure 11-1**

Select the UART required and click **OK**. The UARTx Properties dialog box will open. The parameters for the UART are set in this dialog. See Figure 11-2.



UART1 Properties

Parity: None

Data Bits: 8

Stop Bits: 1

Baud Rate: 9600

Comm Mode: RS-232

RTS GPIO Pin:

Enable ST Buffers

Receive Buffer Size:

Transmit Buffer Size:

OK Cancel

**Figure 11-2**

Configure the UART by setting each of the parameters to your hardware and applications needs.

| | |
|---|---|
| **Parity**: | Parity. Select from None, Even and Odd. |
| **Data Bits**: | Number of Data Bits. Select from 5, 6, 7 or 8. |
| **Stop Bits**: | Number of Stop Bits. Select from 1 or 2. |
| **Baud Rate**: | Baud Rate. Select from 9600, 19200, 28400, 57600 and 115200 |
| **Comm Mode**: | Communications Mode. Select from RS232, RS422 or RS485. Select the type of communication interface. Communications Mode is subject to the actual hardware features available. |
| **RTS GPIO Pin**: | Request to Send GPIO. For RS485, a GPIO (output) is required to hand the transmit / receive pin on the RS485 transceiver. The connected GPIO to the transceiver should be specified here. |
| **Enable ST Buffers**: | Check box to enable the Transmit / Receive buffers for use with Structured Text. This must be enabled when using structured text UART functions. |
| **Receive Buffer Size**: | Receive Buffer Size for Structured Text. Enter the number of bytes to buffer. |
| **Transmit Buffer Size**: | Transmit Buffer Size for Structured Text. Enter the number of bytes to buffer. |

When configured, click **OK** to close the UARTx Properties dialog and return to the Target Properties window.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window.

The UART is now installed. The UART supports the use of Modbus Master, Modbus Slave, Serial Printing and Structured Text.

The UART baud rate may be changed in the ladder diagram program by using the UART_SET_PROPERTY function block.

For Modbus Master / Slave, see **Chapter 13 - Modbus Networking**.
For Structured Text, see **Chapter 26 - Structured Text**.

## Serial Print Functionality

EZ LADDER Toolkit provides the ability to serially print text and variables to other devices using a serial port. This feature can be useful to send data to data loggers, displays and other devices. The serial print feature utilizes a standard serial port (UART) that may be configured as RS232, RS422 or RS485 and can operate with multiple baud rates.

Serial Printing support is based on actual hardware target specifications. PLC on a Chip™ Integrated Circuits and Modules support Serial Printing functionality as well as do some standard Divelbiss PLCs and Controllers. For PLCs and controllers, refer to the supported features. See **Chapter 23 - Hardware Targets**.

## Installing / Configuring the Serial Print Device

As with most features, the Serial Print feature must be installed and configured in the EZ LADDER Toolkit before it may be used.

The Serial Print is configured using the *Project Settings*. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the part number / model number if not selected.

Verify the UART to be used for serial printing is installed. The UART to be used must be installed prior to installing the Serial Printing device / feature. See Figure 11-3. If not installed, please see the UARTS & Serial Ports section of this Chapter.

Click the **ADD DEVICE** button. The ***Targets Devices*** window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find Serial Print. Figure 11-4 shows the Device Properties window.

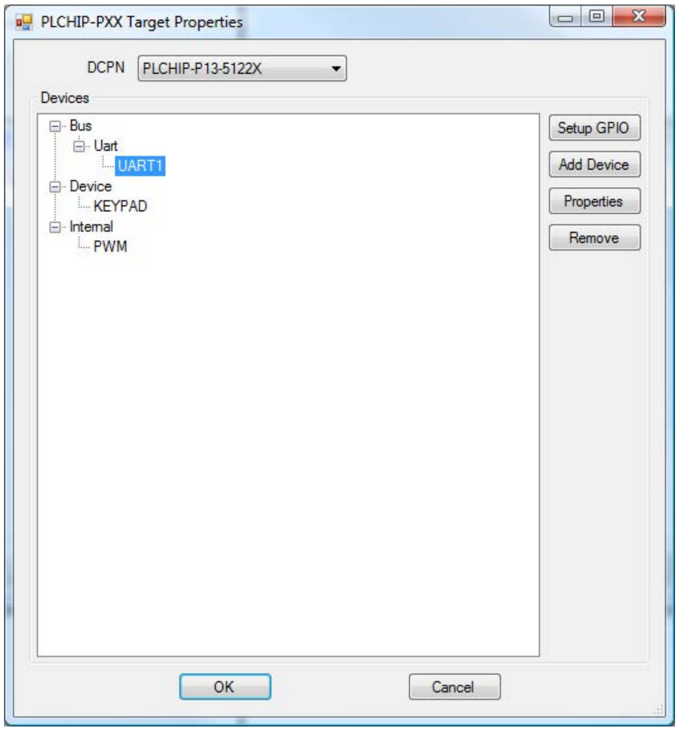
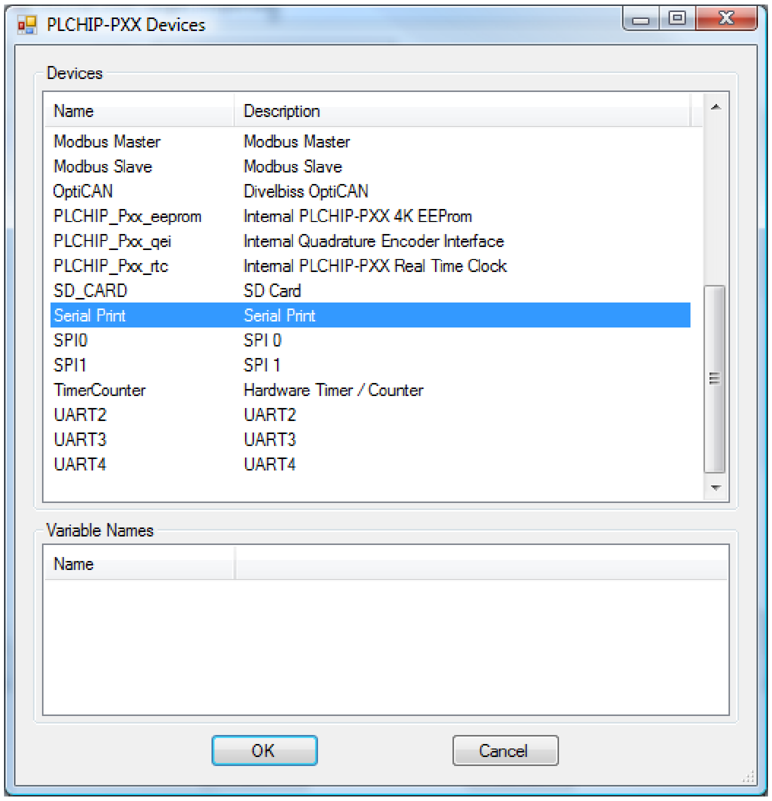


Figure 11-3


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |
| UART3 | UART3 |
| UART4 | UART4 |

Figure 11-4

Click *Serial Print* and click **OK**. The Serial Print Properties window will open. This window is used to configure the Serial Print properties. See Figure 11-5.

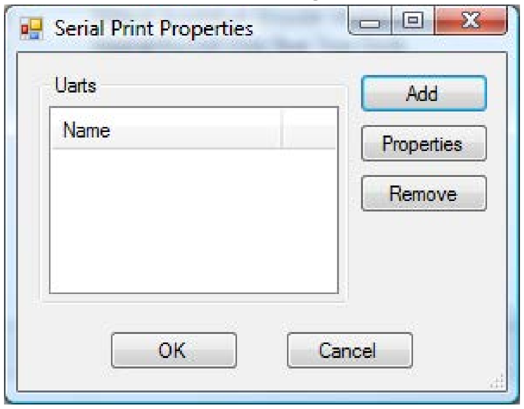
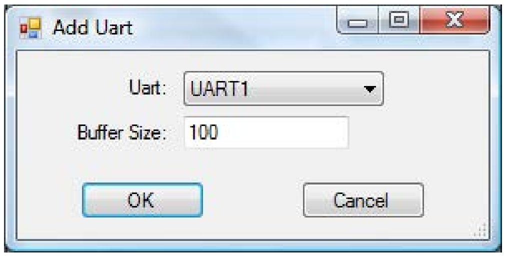


**Figure 11-5**

Click the **ADD** button. The Add Uart dialog will open. Select the UART to use from the drop-down Uart menu and enter the number of bytes to use for the  buffer size for the UART. See Figure 11-6.



Uart: UART1
Buffer Size: 100

**Figure 11-6**

Click **OK** to close and save the Add Uart Dialog. The UART is now listed in the Uarts Box of the Serial Print Properties window.  Click **OK** to close the Serial Print Properties Window.

The targets Devices window will close and the previous target properties window will now list the Serial Print as an installed device under the Device section.

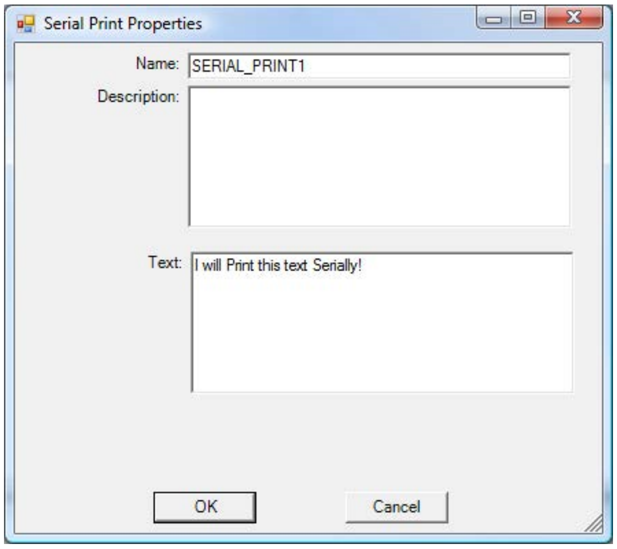
Click **OK** close the Target's properties and click **OK** again to close the Project Settings window.  Use the File Menu and Save the ladder diagram project.  The Serial Print can now be utilized from the ladder diagram project.

## Printing Data to a Serial Device using a Serial Port

With the Serial Print configured in the ladder diagram project, it is now possible to use the EZ LADDER Toolkit's function blocks to serially transmit text and set points. To serial print, the **SERIAL_PRINT** function block is used.

### Transmitting Text Serially

To transmit using the serial port, the **SERIAL_PRINT** function block is used. Using the SERIAL_PRINT function block is a two step process. When placing the function block, a new Serial Print Properties dialog box will open. See Figure 11-7. The *Text* field is where the message is typed that will be transmitted.


Serial Print Properties

Name: SERIAL_PRINT1

Description:

Text: I will Print this text Serially!

**Figure 11-7**

When all the information is entered, clicking **OK** will cause the function block to be placed in the ladder diagram project. Figure 11-8 is a sample of a complete SERIAL_PRINT circuit.

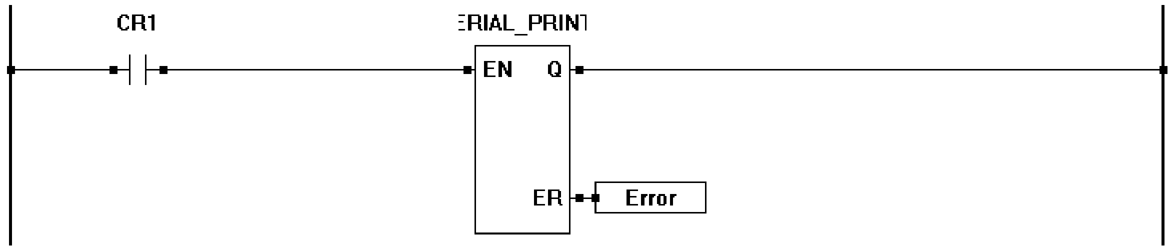
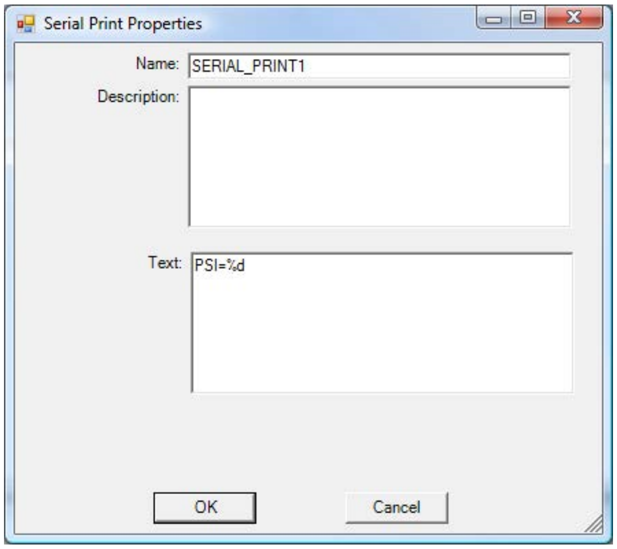


**Figure 11-8**

The SERIAL_PRINT function block supports special control characters. See the SERIAL_PRINT function block in **Appendix A - Function Reference**.

## Transmitting Variables Serially

In addition to transmitting static text, it is often desirable to be able to transmit variables to another device. To transmit a variable using the serial port, the same **SERIAL_PRINT** function block is still used. As in transmitting text, the text is entered into the *Text* field. In addition to the text, *control characters* may be inserted that represent variables and how to format the variable text. For a full listing of what control characters and formatting is supported, please see the SERIAL_PRINT function block in **Appendix A - Function Reference.** Figure 11-9 illustrates a sample text dialog with control characters.



Name: SERIAL_PRINT1
Description:
Text: PSI=%d

**Figure 11-9**

When a SERIAL_PRINT function is used to transmit variables, a new variable input is added to the function block automatically for each variable that will be transmitted.

The SERIAL_PRINT function block is rising edge sensitive. Therefore, it will only transmit one time as the ENable input goes high. If data is required to be transmitted repeatedly, it must be programmed into the ladder diagram project as part of the ENable control on the SERIAL_PRINT function block.

Every placement of a SERIAL_PRINT function block will use available RAM. For most ladder diagram projects, there is an more than enough RAM; however, ladder diagram projects with heavy memory usage functions could run short on RAM.

For more detail on all EZ LADDER Toolkit Function Blocks and objects, refer to **Appendix A - Function Reference**.

Additional Serial communications is available using Modbus and Structured Text. For Modbus Master / Slave, see **Chapter 13 - Modbus Networking**. For Structured Text, see **Chapter 26 - Structured Text**.

# CHAPTER 12

## Real Time Clock

This chapter provides basic information to understand how to install, configure and use the Real Time Clock in the EZ LADDER Toolkit.

## Chapter Contents

## Installing the Real Time Clock

P-Series targets (targets based on P-Series PLC on a Chip™) support the use of a Real Time Clock device. Most P-Series based targets support an on-chip Real Time Clock. Additional real time clock devices may be connected via an SPI port on the PLC on a Chip™.

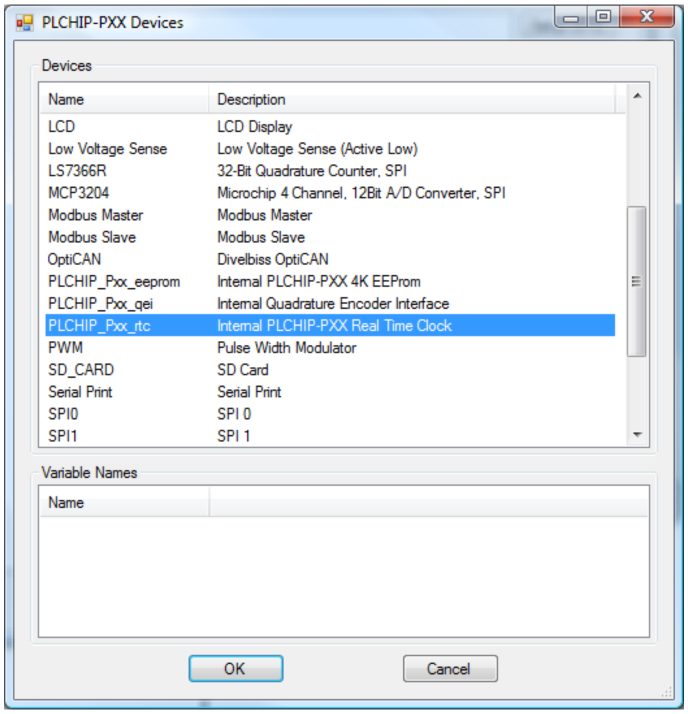
Real Time Clock support is target dependent. The PLC on a Chip™ target itself supports on-board and SPI real time clock devices. Refer to other target's user manuals or **Chapter 23 - Hardware Targets** for targets that support the real time clock. For proper operation the real time clock requires a battery and crystal. Standard product P-Series targets will include the battery and crystal as part of the product. When using the PLC on a Chip™ itself, the battery and crystal must be provided external to the chip itself.

Prior to using the Real Time Clock in EZ LADDER Toolkit, the Real Time Clock must be installed and configured in the target. The Real Time Clock is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the real time clock were installed, it would be listed in the *Devices pane* under the Internal section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find the PLCHIP_Pxx_rtc. Figure 12-1 shows the Target's Devices window.


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |

Variable Names

Name

**Figure 12-1**

Select the **PLCHIP_Pxx_rtc** and click **OK**. The real time clock will install and the Target's Devices window will close returning to the Target Properties Window. The real time clock will now be listed in the Devices pane. Refer to Figure 12-2.

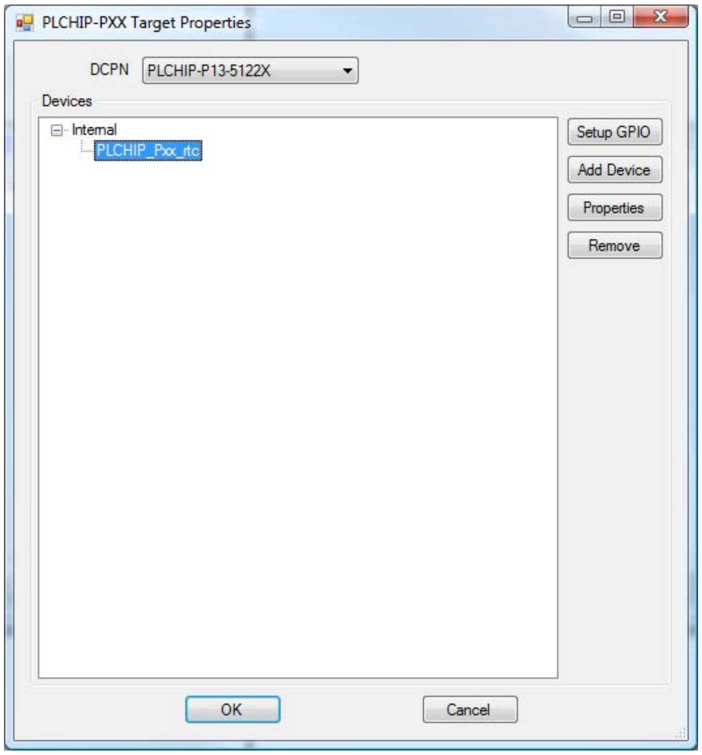


**Figure 12-2**

There are no additional configuration or setup required to use the real time clock. Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The real time clock is now ready to be used in the ladder diagram.

The real time clock must have it's time and date set before it can be used properly. The date/time can be set with different function blocks and in two formats: month, day, year, day of week, hour, minute, second or UTC time.

Certain features, such as MQTT require the real time clock to be set as UTC time. If these features are used in the ladder diagram project, then the real time clock must be set to UTC time for proper operation. Function blocks are provided to read the real time clock as normal date and time from the UTC set real time clock.

## Using the Real Time Clock

To use the real time clock in a ladder diagram project, functin blocks are used to set and read the date and time. Structured text functions are also available to use for accessing the real time clock. Depending on the features used in the ladder diagram project and your needs, the real time clock will either be set using normal time and date (month, day, year, day of week, hour, minute and second) or to UTC time.

## Real Time Clock with Normal Date/Time

To use the real time clock with normal date and time (also considered local time) features, four function blocks are provided. Two blocks (**GETDATE** and **GETTIME**) are used to read the current date and time from the real time clock. The remaining two blocks (**SETDATE** and **SETTIME**) are used to actually set the date and time on the real time clock. Figure 12-3 represents a ladder diagram program that will read the date and time.

When using features like SNTP or MQTT (VersaCloud M2M+IoT), the real time clock must be set using UTC Time (not Normal Date/Time). Setting to normal date/time when using these features will cause these features to not function properly.

When using MQTT or SNTP, other function blocks and structured text are provided to read and set the date time using normal type data (hour, minute, etc) while the actual time is still kept in UTC time. Refer to the **Real Time Clock Sync with SNTP** section of this chapter for SNTP information. Refer the **Real Time Clock as UTC Time** section of the chapter for using the real time clock with UTC time.

The real time clock function blocks use variables as inputs (for setting date and time) and outputs for reading date and time. These integer variables are global and may be used anywhere in the ladder diagram. Figure 12-4 represents a ladder diagram to set the date and time.

The real time clock can be set as normal time/date using the function blocks or from within EZ LADDER Toolkit when connected and in Monitor Mode with the target. See ***Setting the Real Time Clock using EZ LADDER Utilities*** later in this chapter.

For more details on using the real time clock function blocks or additional function blocks, see **Appendix A - Function Reference.**

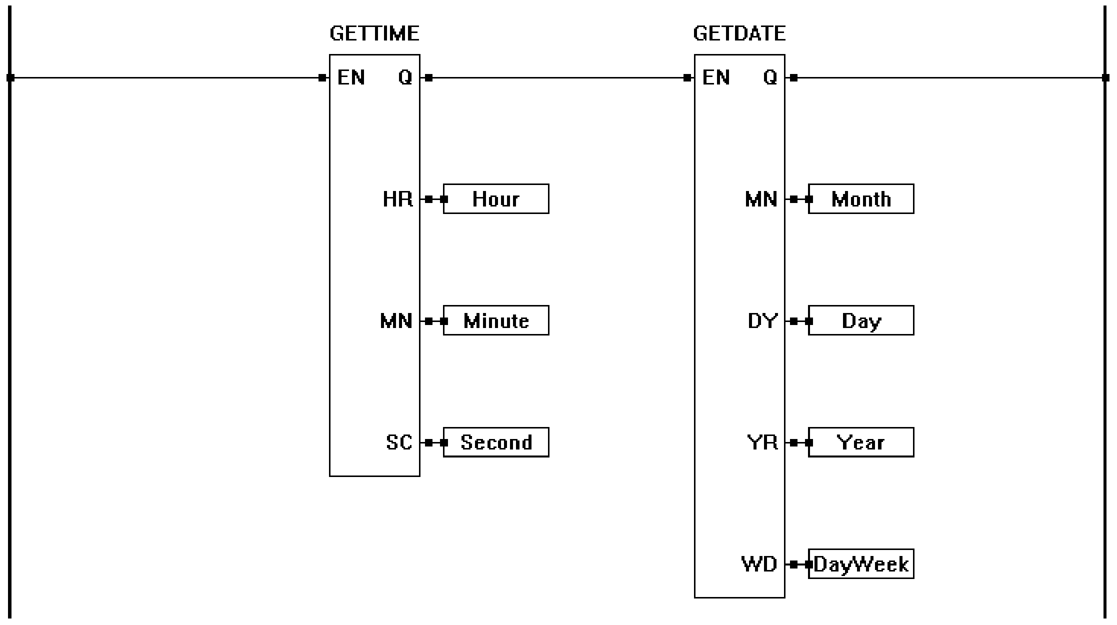


**Figure 12-3**

There are no structured text functions for setting or reading normal date/time of the real time clock.

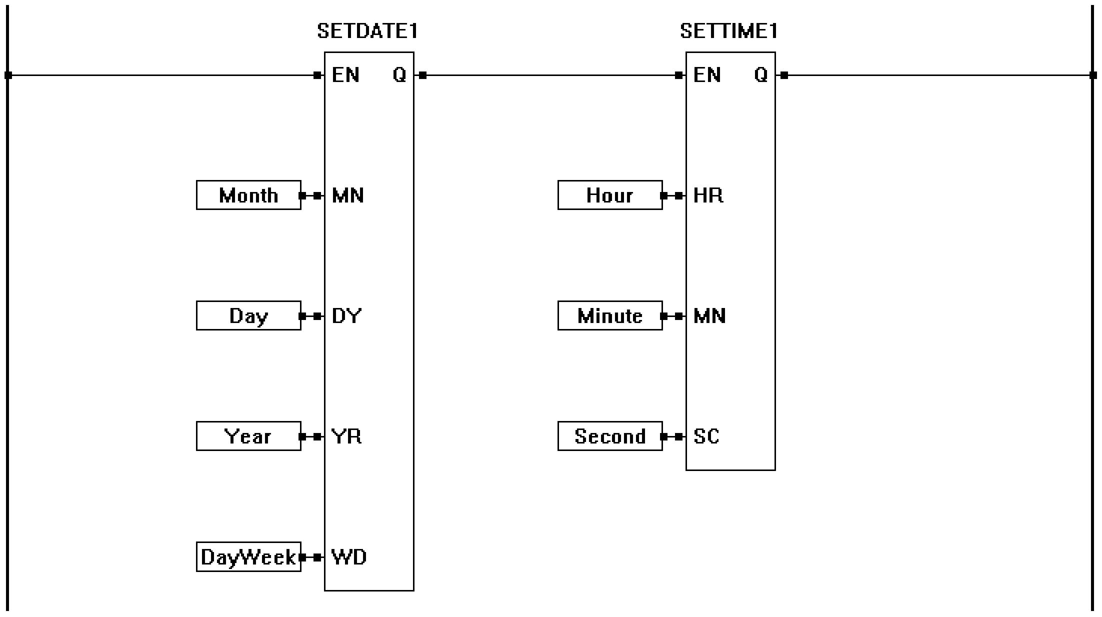


**Figure 12-4**

The real time clock can also be set from within EZ LADDER Toolkit when connected and in Monitor Mode with the target. See ***Setting the Real Time Clock using EZ LADDER Utilities*** later in this chapter.

The real time clock will return the time/date kept internally using the **GETDATE** and **GETTIME** function blocks. These blocks will return the values either in normal time or UTC (based on the time/date that is actually written and kept in the real time clock.)

The real time clock's time/date can be set using the **SETDATE** and **SETTIME** function blocks. These blocks will set the values in the real time clock that are provided as the setpoints to them (normal time or UTC).

## Real Time Clock with UTC Time

As mentioned earlier, some features (MQTT and SNTP) require the real time clock to be set using UTC time. You may choose to use UTC time for other reasons such as easier data logging and data-mining (keeping all time to UTC prevents dealing with time zones).

The real time clock can be set to UTC time using the function blocks, using the SNTP feature, using structured text or from within EZ LADDER Toolkit when connected and in Monitor Mode with the target. See ***Setting the Real Time Clock using EZ LADDER Utilities*** later in this chapter.

UTC time can be set using the SETDATE and SETTIME function blocks and read using the GETDATE and GETTIME function blocks, but three additional function blocks are provided that are used to set and read the date time. These function blocks allow you to set the local date and time, but the actual time is stored in UTC time. The function blocks read the UTC time from the real time clock and return the local date and time.

When using UTC time from the ladder diagram to set and read the date / time, it is recommended that these function blocks be used: **SETTZOFF**, **SETDTLOCAL** and **GETDTLOCAL**.

The **SETTZOFF** function block sets the time zone offset for the local time from UTC time. This value and function block must be ran before setting or reading the date/time using the **SETDTLOCAL** or **GETDTLOCAL** function blocks. This function must be ran each time the target (program) starts to set the time zone offset. The timezone offset is not stored as a value, but only used for conversion.

The SETTTZOFF function block and the EZ_RTC_SETTZOFF structured text function both set the time zone offset. If either is ran, then the time zone offset is set until a power cycle occurs (program restarts).

The **SETDTLOCAL** function block sets the date and time using local time entry (and the time zone offset using **SETTZOFF**) into the real time clock as UTC time.

The **GETDTLOCAL** function block reads the date and time from the real time clock (UTC time) and returns the local time based on the time zone offset that was set using **SETTZOFF**.

Figure 12-5 is an example using **SETTZOFF** and **SETDTLOCAL** to set the time zone offset and the real time clock (to UTC) using local date and time.

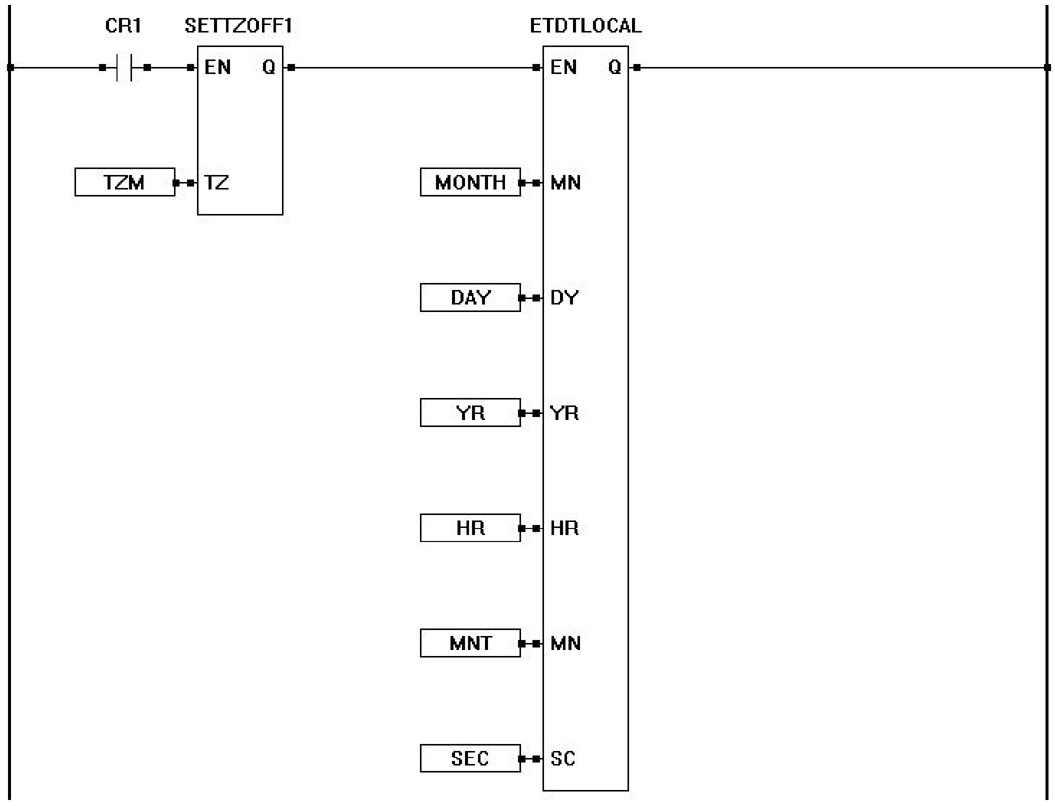


**Figure 12-5**

Figure 12-6 is an example using **GETDTLOCAL** to get the local date and time from the real time clock (stored as UTC).

The **SETTZOFF** must have been ran when the target (program)started with the current time zone off set in minutes for the **GETDTLOCAL** function block to return correct values. If the time zone offset (in minutes) is not correct or has not been set (**SETTZOFF**), the result will be incorrect local date and time returned.

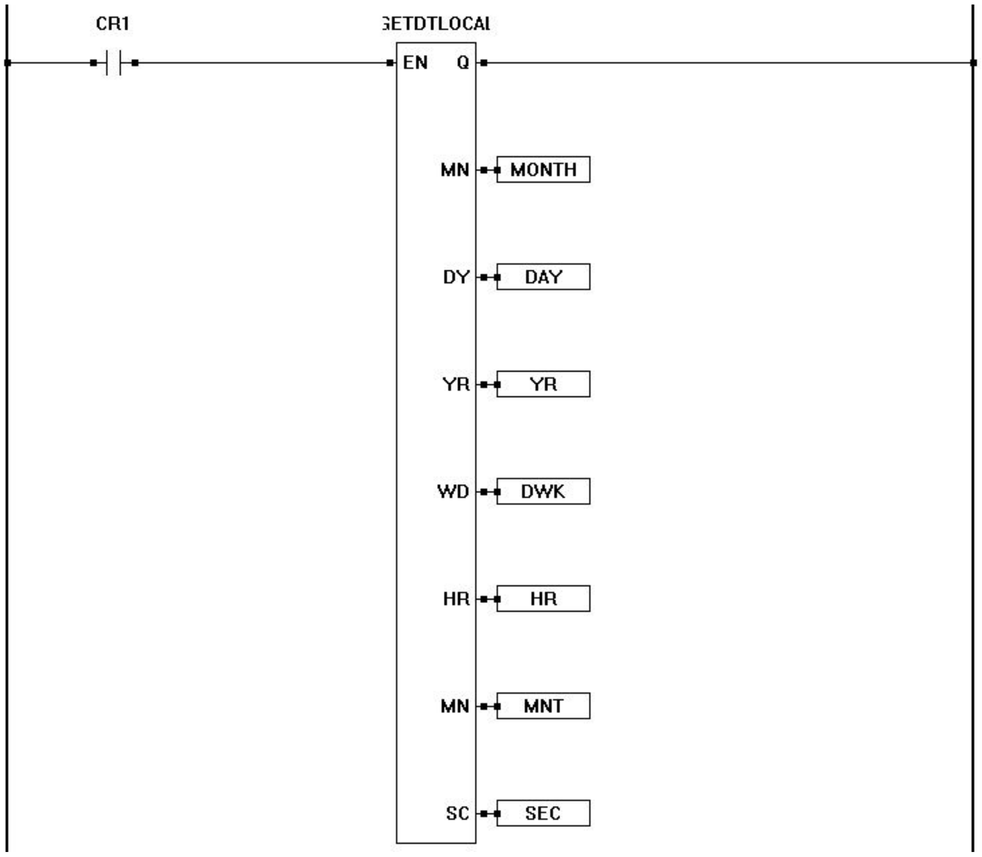


**Figure 12-6**

For more detials on the **SETTZOFF**, **SETDTLOCAL**, or **GETDTLOCAL** function blocks, refer to **Appendix A - Function Reference.**

The real time clock can also be accessed (set and read) using structured text. The structured text commands are similar to the function blocks, as they provide provisions for setting/reading the real time clock with local time and date (and time zone offset), while being stored as UTC time or setting/reading the real time clock using UTC time directly without using local date/time.

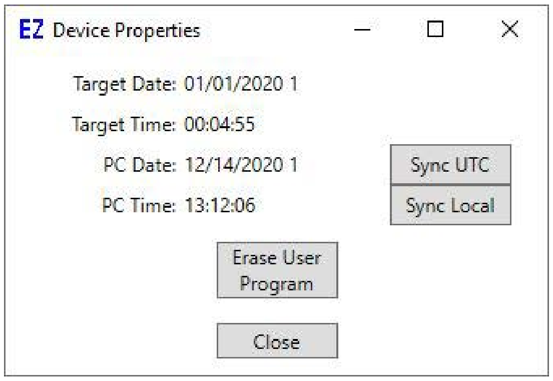
The structured text functions include: *EZ_RTC_SETTZOFF*, *EZ_RTC_GETDTUTC*, *EZ_RTC_SETDTUTC*, *EZ_SETDTLOCAL* and *EZ_GETDTLOCAL*. Other structured text function are also availabe for converting time types.

For more information on using these structured text functions, refer to **Appendix B - Target Specific ST Function Reference.** For more information on structured text, refer to **Chapter 26 - Structured Text**. Standard structured text function information can be found in **Appendix C - Standard ST Function Reference.**

## Setting the Real Time Clock using EZ LADDER Utilities

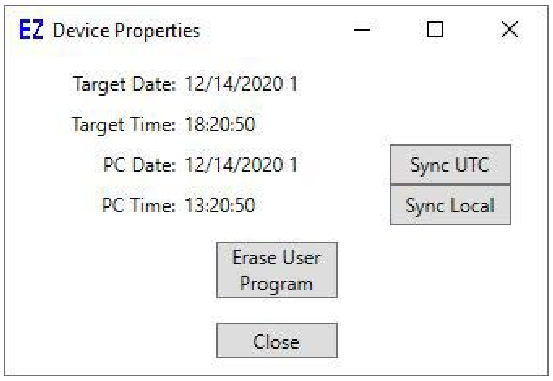
The real time clock's date and time can be set from EZ LADDER Toolkit when the hardware target is connected to EZ LADDER and running in the Monitor mode (connected to the target, with the same program in EZ LADDER Toolkit and the target and running in monitor mode).

When connected in this manner, click the **F11** button. This will open the *Device Properties* dialog. From this dialog, you can sync the target's real time clock to the PC's date and time in local format or UTC. Refer to Figure 12-7.


EZ Device Properties

Target Date: 01/01/2020 1
Target Time: 00:04:55
PC Date: 12/14/2020 1
PC Time: 13:12:06

**Figure 12-7**


EZ Device Properties

Target Date: 12/14/2020 1
Target Time: 18:20:50
PC Date: 12/14/2020 1
PC Time: 13:20:50

**Figure 12-8**

Figure 12-8 shows the real time clock set to UTC time (by pressing the Sync UTC button.

To sync the target's real time clock to the PC local time, click the **SYNC LOCAL** button. This will set the real time clock to local time and date. No UTC time is used in the real time clock.

To sync the target's real time clock to the PC local time, click the **SYNC UTC** button. This will set the real time clock to UTC time only.  Only the UTC time is stored. No time zone offset is used.

When you have finished setting the real time clock, click the **CLOSE** button to close the *Device Properties* dialog.

The real time clock when set using the EZ LADDER Toolkit utilities does not read, use or store the time zone offset for UTC time. The time syncing function just syncs the current date and time to the real time clock using either local time or UTC. You must still use the **SETTZOFF** function block to correctly use the real time clock and the **SETDTLOCAL** and **GETDTLOCAL** function blocks within the ladder diagram. Failure to do so will result in improper real time clock date/time.

## Real Time Clock Sync with SNTP

EZ LADDER Toolkit allows for the real time clock to be synced using SNTP (simple network time protocol). This syncs the real time clock to an external source using Ethernet / WiFi or Cellular. When this feature is used, the real time clock is synced to UTC time.

The external source for the SNTP (Ethernet/WiFi or Cellular) must be installed before the SNTP feature can be installed. The target must also support the Ethernet/WiFi or Cellular device. Refer to target's user manuals or **Chapter 23 - Hardware Targets** for targets that support these devices.

The Real time clock must be installed before the SNTP feature can be installed. The target must also support the Ethernet/WiFi or Cellular device. Refer to target's user manuals or **Chapter 23 - Hardware Targets** for targets that support the real time clock.

The SNTP sync feature is required to use other PLC on a Chip / EZ LADDER features like VersaCloud M2M-*IoT* solutions (MQTT, etc).

SNTP when implemented sets the real time clock to UTC time only (not local time). The UTC time is used during verification of communications to VersaCloud M2M+IoT (MQTT) solutions.

When using SNTP, the real time clock should not be set using any other methods (SETDATE, SET TIME functions or structured text). Setting the real time clock outside SNTP (when SNTP is used and required for MQTT) may result in loss of connection to the VersaCloud M2M+IoT solution.

As the SNTP sets the real time clock to UTC time, the GETTIME and GETDATE functions return values that are not directly usable without calculations. For reading the date and /or time to display local time (when SNTP is enabled and set to UTC time), use the GETLOCALTIME and GETLOCALDATE function blocks. You must know the number of hours to offset from UTC time for proper operation.

## Installing the SNTP in the Ladder Diagram Project

To be able to use the SNTP feature in an EZ LADDER Toolkit ladder diagram project, the SNTP must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the SNTP, it will be used as an example to install and configure the SNTP.

The SNTP is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the SNTP feature were installed, it would be listed in the *Devices pane* under the Network section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the SNTP. Figure 12-9 shows the Target's Devices window.

Click **OK**. The SNTP Properties dialog will open. See Figure 12-10.

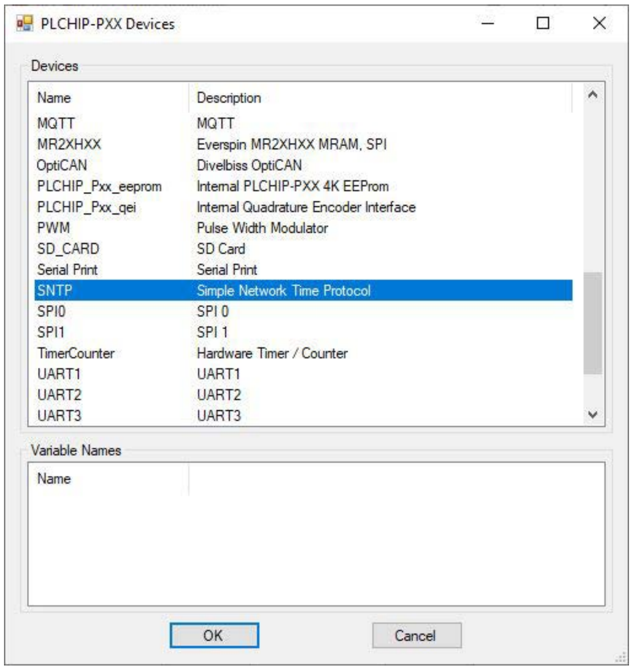
Using the *Available Interfaces* pane, select the interface to use as the source for the SNTP (external source) and click **ADD**. The selected source will now move to the *Selected Interfaces* pane.

Click the *Enabled* check box to enable the SNTP.

Using the *RTC* drop-down box, select the installed real time clock to sync.

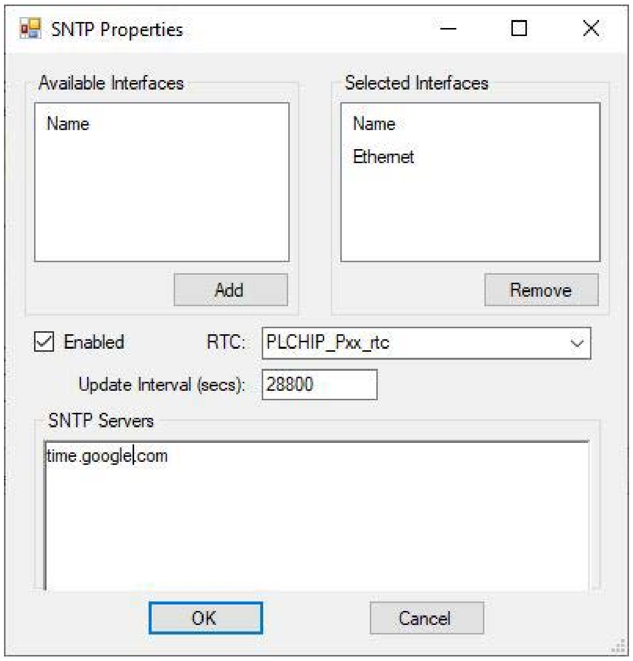
In the *Update Interval* (secs) box, use the default or enter a rate which the SNTP will sync the real time clock. This time interval may be limited based on the actual SNTP server that is used.

In the *SNTP Servers* pane, enter the SNTP servers to used to sync time. These can be local or internet sources, provided there is access (through the selected interfaces) to retrieve the date and time.



| Name | Description |
|---|---|
| MQTT | MQTT |
| MR2XHXX | Everspin MR2XHXX MRAM, SPI |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SNTP | Simple Network Time Protocol |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART1 | UART1 |
| UART2 | UART2 |
| UART3 | UART3 |

**Figure 12-9**



Selected Interfaces: Ethernet

☑ Enabled RTC: PLCHIP_Pxx_rtc

Update Interval (secs): 28800

SNTP Servers: time.google.com

**Figure 12-10**

When the SNTP is configured, click **OK** to close the SNTP Properties dialog. It is now listed in SPI section of the Devices window.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The SNTP feature is now installed.

The SNTP server will sync the real time clock to the provided SNTP time servers provided using the configured source. The successful syncing depends on the source and the actual SNTP servers being available.

## SNTP Structured Text Commands

There are two SNTP structured text commands to be used in structured text functions and function blocks: EZ_SNTP_START, EZ_SNTP_STOP.

Refer to **Appendix B - Target Specific ST Function Reference** for more detials on using the two functions.

# CHAPTER 13

## Modbus Networking

This chapter provides basic information to understand how to install, configure and use the Modbus Networking feature in the EZ LADDER Toolkit.

## Chapter Contents

## Modbus Overview

Modbus is a register based communication protocol connecting multiple devices to a single network. Devices on this network are divided into two types: Master and Slave. There is only one *master* device on a network. The master is in control and initiates communication to all the other devices. Each device that is not a master must be a *slave*. Multiple slaves may be located on a network. Slave devices *listen* for communication from the master and then respond as they are instructed. The master always controls the communication and can communicate to only one or all of the slaves. Slaves can not communicate with each other directly.

All modbus networking requires either a serial port (UART), Ethernet port or WiFi for the network traffic. The network device (UART, Ethernet or WiFi) must be installed prior to installing and using any Modbus devices in EZ LADDER Toolkit. For UART installation, see **Chapter 11 - UARTS and Serial Ports**. For Ethernet or WiFi, see **Chapter 19 - Ethernet/WiFi**

## Installing the Modbus Master

EZ LADDER Toolkit (P-Series based products) supports Modbus Master. With Master support, any of the P-Series targets may be used as a master on a modbus network.

All Modbus communication availability is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if Modbus (and what network hardware) is supported.

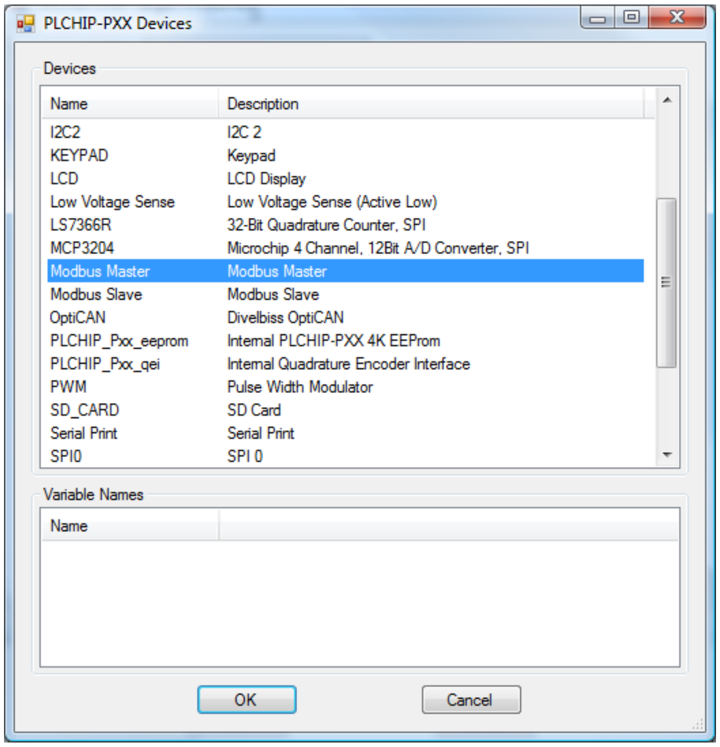
Prior to using the Modbus Master in EZ LADDER Toolkit, it must be installed and configured in the target. The Modbus Master is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the Modbus Master were installed, it would be listed in the *Devices pane* under the Network section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

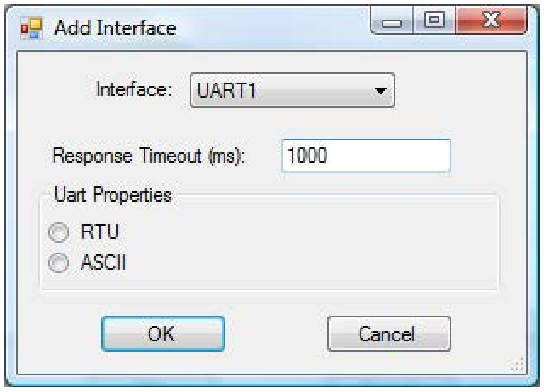
All the available devices and features for the target are shown in the Devices section. Scroll down and find the Modbus Master. Figure 13-1 shows the Target's Devices window.

Select the **Modbus Master** and click **OK**. The *Modbus Master Properties* dialog will open. This dialog is used to specify which hardware interface will be used for this Modbus Network (UARTx, Ethernet). Click the **ADD** button. The *Add Interface* dialog will open. Using the drop-down Interface menu, select the actual hardware interface to use (UART or Ethernet must already be installed prior to this step). Enter the **Response Timeout** in milliseconds and if a UART is used, select the **Uart Properties** as RTU or ASCII based on your needs. See Figure 13-2.



| Name | Description |
|---|---|
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |

**Figure 13-1**



Interface: UART1

Response Timeout (ms): 1000

**Figure 13-2**

Click **OK** to close the Add Interface dialog and click **OK** to close the Modbus Master Properties dialog and return to the Target Properties window. The Modbus Master is now listed in the Devices pane under the Network Section. See Figure 13-3.

The Modbus Master Properties dialog and Add Interface dialogs may also be used to adjust settings and change the actual hardware interface by selecting the Modbus Master in the Devices pane and clicking the Properties button.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The Modbus Master is now ready to be used in the ladder diagram.

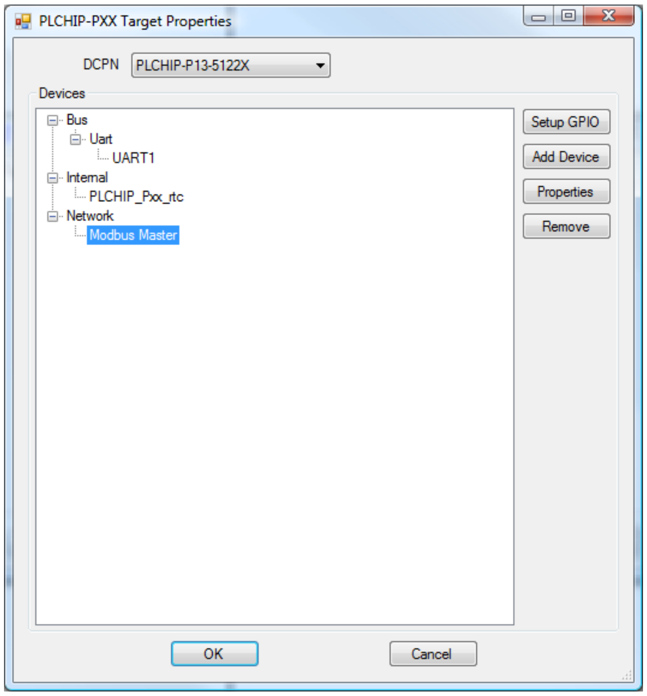


**Figure 13-3**

## Using the Modbus Master

The Modbus Master for the P-Series initiates communications to modbus slave(s) by using the **MODBUS_ MASTER** function block (additionally, the MODBUS_MASTER2 and MODBUS_MASTER3 function blocks are also available - see Appendix A). This section is provided as a base for how to use the Modbus Master on the network, not as a modbus tutorial. Previous knowledge of Modbus would ease the integration of a modbus network.

! The Modbus Master feature in EZ LADDER Toolkit supports multiple commands for network control. EZ LADDER Toolkit variables are used by the MODBUS_MASTER function block as the data storage for values using in the network communications. When values are read from slave(s), these values are stored in variables that are predefined in each instance of the MODBUS_MASTER function block. When values are to be written to slave(s), the values to be written are captured from variables that are predefined in each instance of the MODBUS_MASTER function block.

Only certain Modbus Master functions (commands) are supported in EZ LADDER Toolkit. See Figure 13-4.

| Functional Description | Function # | Functional Description | Function # |
|---|---|---|---|
| Read Coils | 1 | Write Single Register | 6 |
| Read Discrete Inputs | 2 | Write Multiple Coils | 15 |
| Read Holding Registers | 3 | Write Multiple Registers | 16 |
| Read Input Registers | 4 | Read / Write Multiple Registers | 23 |
| Write Single Coil | 5 | | |

**Figure 13-4**

When the MODBUS_MASTER is place in the ladder diagram workspace, a Modbus Master Properties dialog box is automatically displayed. See Figure 13-5.

Like most functions, a description can be added to the function block. Using the ***Interface*** drop-down selection menu, select the interface for the Modbus Master network (the UART or Ethernet) that was installed and configured previously. The ***Function Code*** drop-down selection box is used to select the type of function (command) for this instance of the MODBUS_MASTER function block to execute when active. These are listed in Figure 13-4. Select the Function Code and Interface as required. See Figure 13-5.

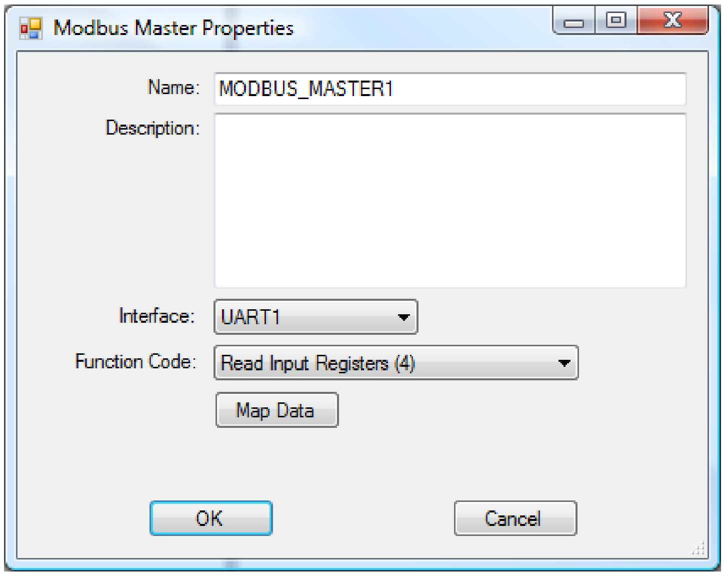

Name: MODBUS_MASTER1
Description:
Interface: UART1
Function Code: Read Input Registers (4)

**Figure 13-5**

With the Interface and Function Code set, it is now time to specify the variables used for this function block to capture values and send to the slave or read values from the slave and store to. To configure the variables on this function block instance, click the **MAP DATA** button. The *Modbus Master Map Data* window will open. See Figure 13-6.

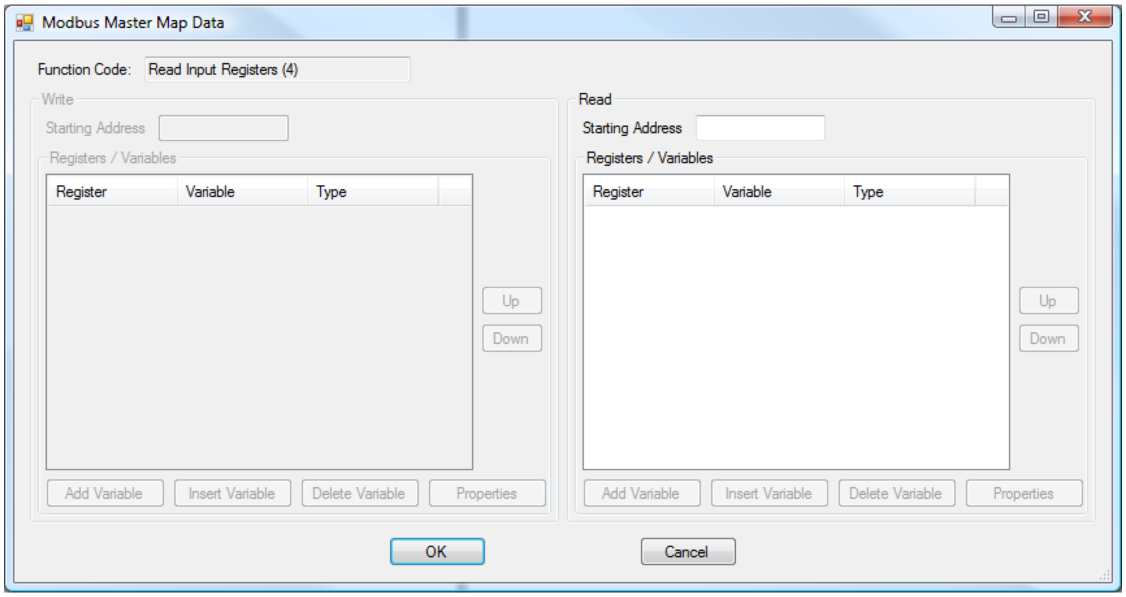

Function Code: Read Input Registers (4)

**Figure 13-6**

The *Modbus Master Map Data* window is divided into two sections: **Write** and **Read**. One or both of the window sections will be active depending upon what actual Function Code was selected in the *Modbus Master Properties* dialog.

The Write and Read sections are configured identically, with the main difference that the Write section is where variables are identified that will be used as the source point for writing to slave(s); while the Read section is where variables are identified that will be used to store data read from slave(s).

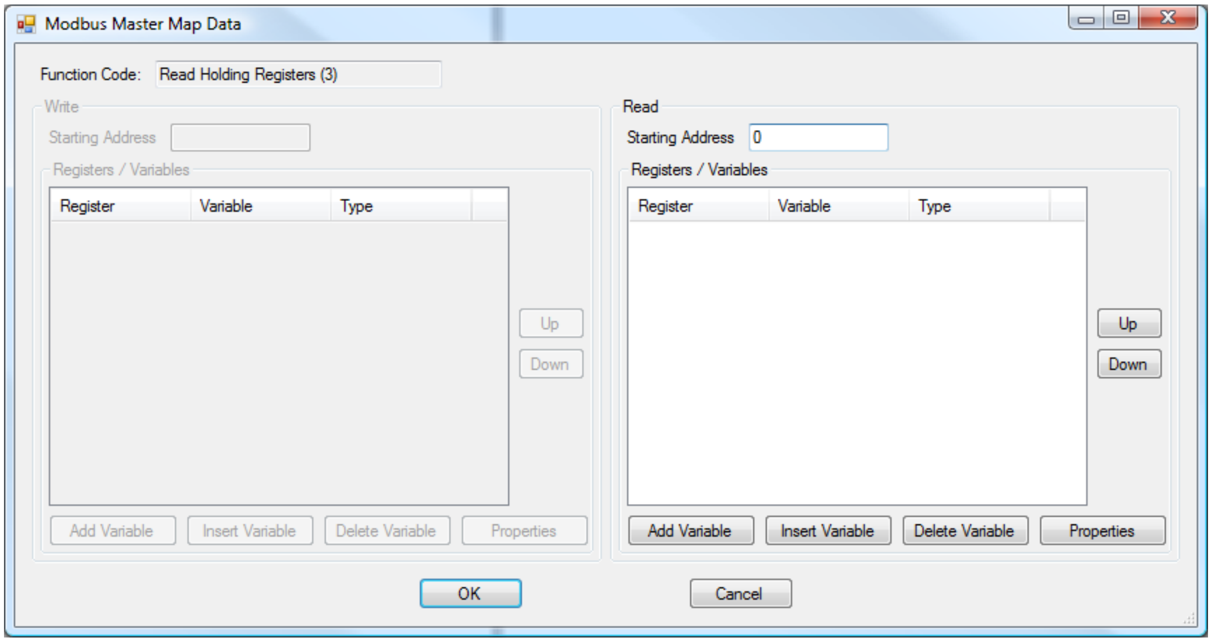
## Starting Address

The Starting Address box for both the Read and Write sections is the base register number where data will be read from / written to.

The register assignments for the Modbus Master is based on the Modbus specification and thus the starting address is 0 based. Register 0 will always be the first available register in any group of registers for a Function Code type (command). Each Function Code (command) type, per the Modbus specification support registers from 0 to 65535 and the register groups for each Function Code are independent. For example, Function Code 4 (Read Input Registers) will support from 0 to 65535 registers and Function Code 3 (Read Holding Registers) will support from 0 to 65535 registers; however, these two groups of registers are unique and independent from each other.

The actual number of registers supported on slave devices may vary by implementation of the Modbus specification on the devices. See the device’s documentation for actual supported sizes.

Enter the register address in the **Starting Address** box. Once a valid address is entered, several buttons and the Registers / Variables pane becomes active. See Figure 13-7.


Modbus Master Map Data

Function Code: Read Holding Registers (3)

Write

Starting Address

Read

Starting Address 0

**Figure 13-7**

## Adding / Specifying Variables

The buttons in the pane are used to Add, Insert, Delete, Edit and change the order of the variables for the MODBUS_SLAVE function block.

| | |
|---|---|
| **Add Variable**: | Used to add a new variable to the Registers / Variables list. Always adds the variable to the end of the currently listed variables. |
| **Insert Variable**: | When a variable is highlight in the list, this button will insert a variable above the current selection. |
| **Delete Variable**: | Deletes the highlighted variable in the list. |
| **Properties**: | Shows the highlighted variable's properties and allows it to be changed. |
| **Up**: | Moves the highlighted variable up in the list by one (1). |
| **Down**: | Moves the highlighted variable down in the list by one (1). |

Click the **ADD VARIABLE** button to add a new variable to the list. The *Add Variable* dialog will open. Click the **BROWSE** button. See Figure 13-8. The *Variables* window will open. Variables can be added or selected here just as was shown in **Chapter 5 - Creating Ladder Diagrams**.

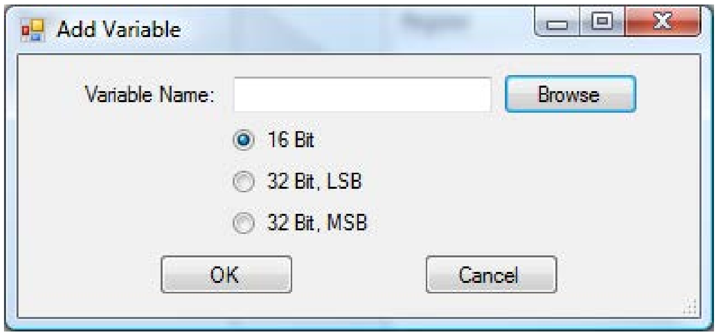


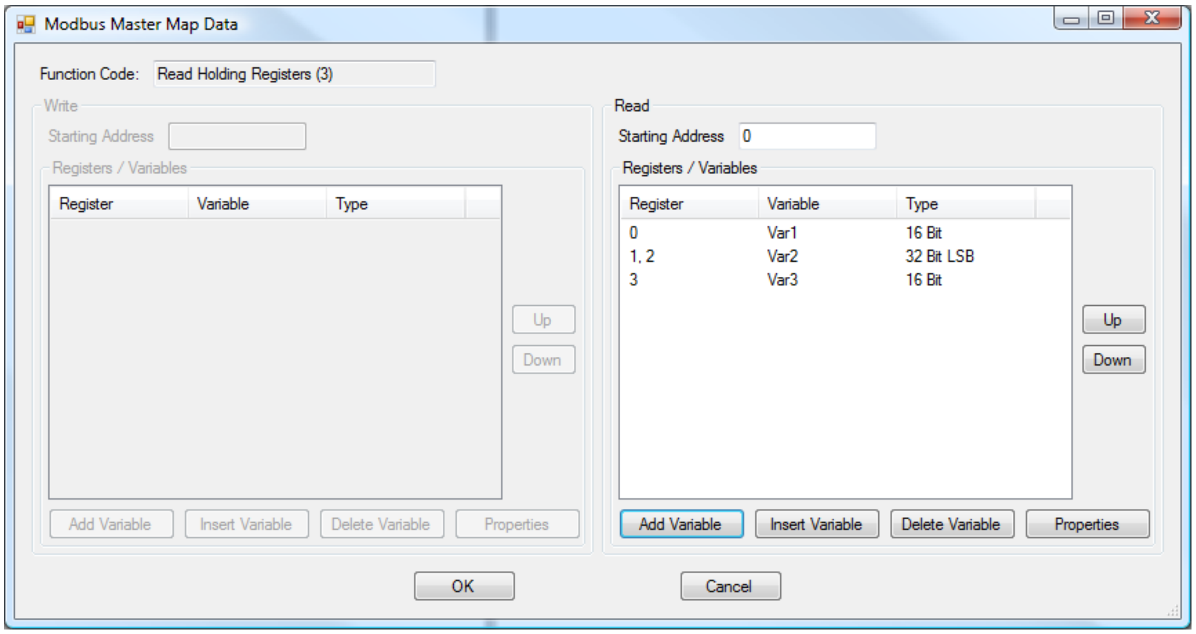
**Figure 13-8**

Add a new variable or select a variable from the list of variables in the *Variables* window and click **OK**. This new or selected variable will be transferred to the *Add Variable* Dialog and the *Variables* window will close. In the Add Variable dialog, select the type of variable to write or read: 16 bit, 32 Bit LSB (ordered Least Significant Bit) or 32 Bit MSB (ordered Most Significant Bit). 32 bit variables will use multiple registers while 16 bit variables use one.

The type of variable to read / write will be entirely application dependent and slave device dependent on supported types.

Once the variable has been added or selected and the type is set, click **OK**. The *Add Variable* dialog will close and the variable selected / added will now be listed in the *Registers / Variables* pane. Repeat the steps to add additional variables as needed for the application and this function block instance. As per indicated, the provided buttons allow the re-ordering of the variables.

Figure 13-9 illustrates 3 variables loaded, two 16 bit and one 32 bit. the types and actual registers that will be read / written to are displayed in the list with the variables.


Modbus Master Map Data

Function Code: Read Holding Registers (3)

Read

Starting Address: 0

Registers / Variables

| Register | Variable | Type |
|---|---|---|
| 0 | Var1 | 16 Bit |
| 1, 2 | Var2 | 32 Bit LSB |
| 3 | Var3 | 16 Bit |

**Figure 13-9**

Click **OK** to close the *Modbus Master Map Data* window and click **OK** to close the *Modbus Master Properties* dialog. One instance of the MODBUS_MASTER function block is now placed in the ladder program.

For each variable added, memory (RAM) is used. Large numbers of registers / variables will reduce the overall memory available for general ladder diagram object and program. Care should be taken to limit variables to only those needed.

As a MODBUS_MASTER function block supports only one Function Code (command), additional function block instances may need to be placed based on the application.

For Function 5 and Function 6, only one variable may be mapped.

## Understanding MODBUS_MASTER Function Block

The MODBUS_MASTER function block has two inputs (EN and ID) and two outputs (Q and ER). These inputs and outputs are all that are required to use the MODBUS_MASTER function block.

| | |
|---|---|
| **EN**: | Function block Enable Input. The EN is active on rising edge only. A rising edge on EN enables the function block to begin communications. |
| **ID**: | Slave ID Input. This number is the number of the slave ID that this function block communicates to. A valid address is 1 to 255. |
| **Q**: | Q Output. Only goes high for one scan when the communication initiated by the EN rising edge is completed or a time-out has occurred. |
| **ER**: | Error Output. Outputs an integer number for the status of any errors detected during the communications initiated by the rising edge on EN. Zero indicates NO errors. See the MODBUS_MASTER Function Block Errors section of this chapter for a list of error codes. |

When the EN rising edge is detected, the function block will attempt communication with the slave device. If an error occurs (including if the device is already busy or another MODBUS_MASTER function block is already communicating), the error will be present on the ER output. If an error occurs, the communication must be re-attempted in the ladder program as it is not buffered. While a function block is active, it's ER output will be set to 1. Figure 13-10 represents a ladder program using the MODBUS_MASTER function block.

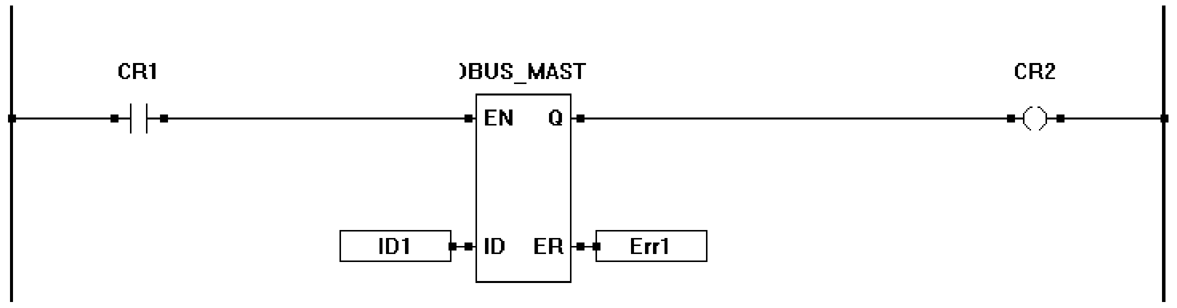


**Figure 13-10**

## MODBUS_MASTER Function Block Errors

The MODBUS_MASTER function block provides an error (ER) output to identify errors detected during Modbus Master communications to slave devices. Figure 13-11 lists the supported error ID codes.

| Error ID Code | Title | Description |
|---|---|---|
| 0 | No Error | No error was detected during communication. |
| 1 | Exception Code Illegal Function | The function code was not allowed by the slave. |
| 1 | Client Busy | A communications request is in process, busy. |
| 2 | Exception Code Illegal Data Address | The data address was not allowed by the slave. |
| 3 | Exception Code Illegal Data Value | A data value was not allowable for the slave. |
| 4 | Exception Code Slave Device Failure | An unrecoverable error occurred while the slave was attempting to perform the requested function. |
| 5 | Exception Code Acknowledge | The slave has accepted the request, but it will take a long duration time to complete. |
| 6 | Exception Code Slave Device Busy | The slave is processing a long duration command / function. |
| 8 | Exception Code Memory Parity Error | A Parity error was detected in memory during a attempt to read a record file. |
| 10 | Exception Code Gateway Path Unavailable | For Gateways only. Gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. |
| 11 | Exception Code Gateway Target No Response | For Gateways only. Indicates no response was received from the target device. |
| -1 | Client Receive Packet Error | An error was detected when receiving a packet from a slave |
| -2 | Client Timeout Waiting for Response | A time-out occurred while waiting for slave response. |
| -3 | Client Error Transmitting Packet | A error occurred transmitting a packet to a slave. |
| -4 | Client Invalid Request | The request was not valid. |
| -5 | Client Buffer Error | An error occurred access the communications buffer. |
| -6 | Client Invalid State | The client's current state is not valid. |
| -7 | Client Connect Failed | Unable to connect to the slave. |
| -8 | Client Checksum Error | A checksum error occurred. |
| -9 | Client Null Transport | A Null transport pointer was detected. |

**Figure 13-11**

## Modbus Slave (UART)

EZ LADDER Toolkit provides the ability to add Modbus slave functionality to a ladder diagram (making the device a Modbus Slave).

Modbus Slave support is based on actual hardware target specifications. PLC on a Chip$^{TM}$ Integrated Circuits and Modules support Modbus Slave as well as do some standard Divelbiss PLCs and Controllers. For PLCs and controllers, refer to the supported features. See **Chapter 23 - Hardware Targets**.

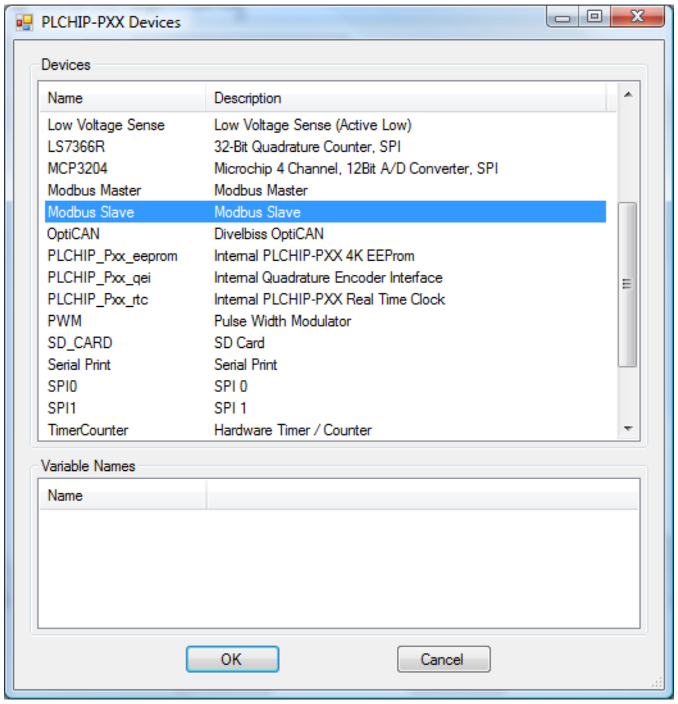
### Configuring for Modbus Slave

Prior to using the Modbus Slave in EZ LADDER Toolkit, it must be installed and configured in the target. The Modbus Slave is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the Modbus Slave were installed, it would be listed in the *Devices pane* under the Network section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find the Modbus Slave. Figure 13-12 shows the Target's Devices window.

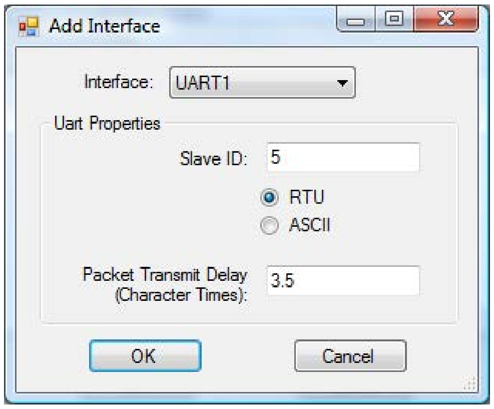


| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |

**Figure 13-12**

Select the **Modbus Slave** and click **OK**. The *Modbus Slave Properties* dialog will open. This dialog is used to specify which hardware interface will be used for this Modbus Network (UARTx, Ethernet). Click the **ADD** button. The *Add Interface* dialog will open. Using the drop-down Interface menu, select the actual hardware interface to use (UART or Ethernet must already be installed prior to this step).

For UART, enter the **Slave ID** of this unit (1-255), select the **Uart Properties** as RTU or ASCII based on your needs. See Figure 13-13. The packet Transmit Delay defaults to 3.5 and typically it does not need to be adjusted.


Add Interface

Interface: UART1

Uart Properties

Slave ID: 5

(●) RTU
( ) ASCII

Packet Transmit Delay (Character Times): 3.5

**Figure 13-13**

Each device on a Modbus network must have a unique ID number.  Duplicate ID numbers will result in a malfunctioning network and communication errors.

For UARTs, the slave ID is set in the Modbus Slave configuration in the Project Settings (Modbus Slave ID is set in the as shown above). For those needing to change the Slave ID on the fly, it can be done by using the MODBUS_SET_PROPERTY function block in the ladder diagram project.

Click **OK** to close the Add Interface dialog and click **OK** to close the Modbus Slave Properties dialog and return to the Target Properties window. The Modbus Slave is now listed in the Devices pane under the Network Section. See Figure 13-14.

The Modbus Slave Properties dialog and Add Interface dialogs may also be used to adjust settings and change the actual hardware interface by selecting the Modbus Slave in the Devices pane and clicking the Properties button.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window.  Be sure to save the ladder diagram project. The Modbus Slave is now ready to be used in the ladder diagram.

For all P-Series targets, the actual register numbers are always base zero per Function Code (Master Function Code) supported. When a modbus master queries a slave, the function code is identified and then the appropriate bank or block of registers are accessed. Register 0 will always be the first available register in any group of registers for a Function Code type (command). Each Function Code (command) type, per the Modbus specification support registers from 0 to 65535 and the register groups for each Function Code are independent. For example, Function Code 4 (Read Input Registers) will support from 0 to 65535 registers and Function Code 3 (Read Holding Registers) will support from 0 to 65535 registers; however, these two groups of registers are unique and independent from each other.

Typically the following register types are supported: Coils, Discrete Inputs, Inputs, and Holding Registers. Using these groups of registers, a master can read and write as needed for communication.

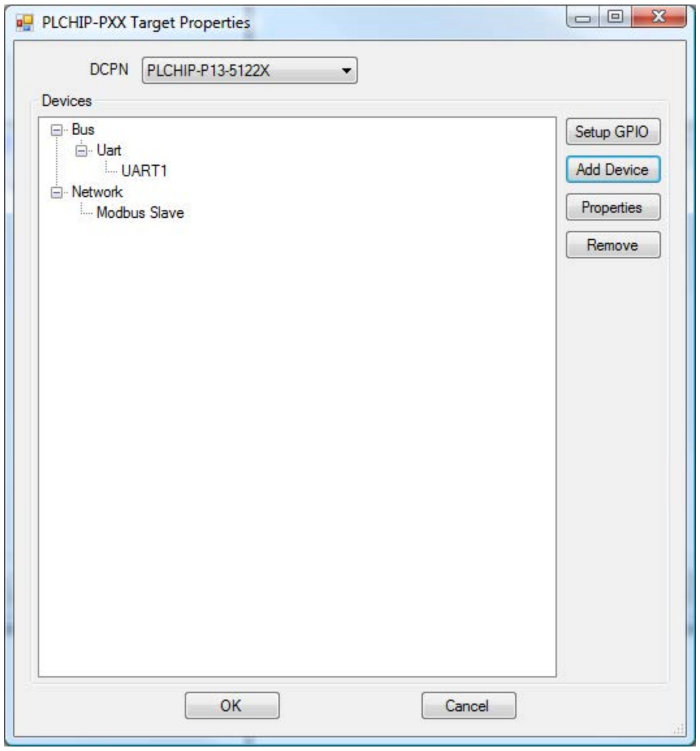


**Figure 13-14**

### Coil Registers

Coils registers are registers that are written to by the Master. Using these registers, the master can directly control coils located in the ladder diagram project (internal only).

### Discrete Input Registers

Discrete Input registers are registers that are read directly by the Master. Using these registers, the master can directly monitor the status of contacts located in the ladder diagram project (internal or real world).

### Input Registers

Input registers are registers that may be read by the Master, but can only be written to by the slave itself. Using these registers, the slave can set data that the master can view, but not modify.

### Holding Registers

Holding registers are registers that may be read and modified by both the Master and Slave. These are the most commonly used registers to pass data between the master and slave.

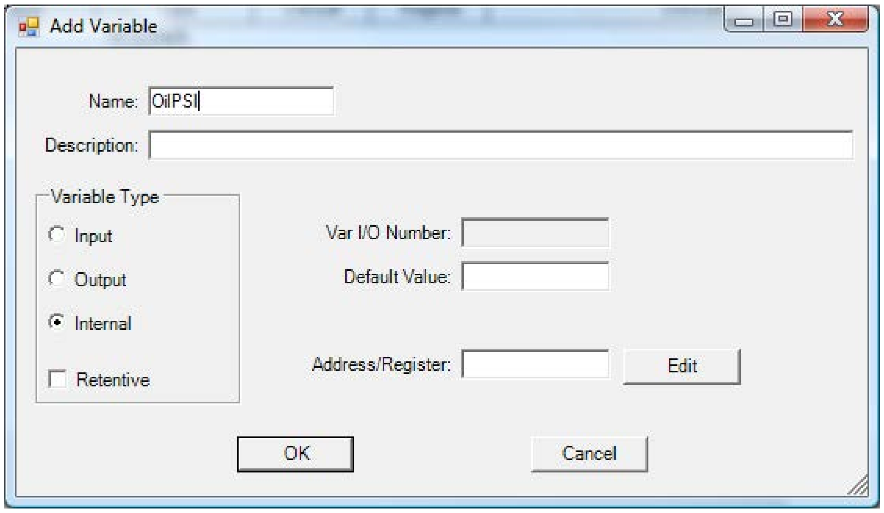
All Modbus registers are accessed as variables and must begin with *MB_* to identify a modbus slave register.

## Assigning and Setting Slave Registers

To use Modbus Slave registers, registers must be assigned to variables. For more information regarding variables, refer to **Chapter 5 - Creating Ladder Diagram Projects**. Modbus registers can be assigned by editing an existing variable when new variables are created.

Coils and Discrete Input registers are to be used with Boolean variable types while Holding and Input Registers are typically used with Integer variables.

To assign a Modbus register to a variable, using the Add Variable or Edit Variable dialog, click the **EDIT** button next to the Address / Register field. See Figure 13-15.


Add Variable

Name: OilPSI

Description:

Variable Type: ( ) Input ( ) Output [✓ Internal]

☐ Retentive

Var I/O Number:

Default Value:

Address/Register:

**Figure 13-15**

The Edit Address / Register dialog box will open. See Figure 13-16. Using the drop down menu, select ***MB_(Modbus)***. From the now visible Register Type drop down box, select the type of register to use (of the four supported types). In the Register Index box, type the address number of the modbus register (0-65535). This number depends on the type of register and it's range of allowed register numbers.

As the register type is selected and the number entered, the Address / Register displayed will be updated immediately. The MB_ and Function code is automatically entered. Only the actual register number needs to be added. This register number is always between 0 and 65535. This register number is automatically added to the MB_ X and type to create the register number in the correct range.

Select the type of variable to write or read: 16 bit, 32 Bit LSB (ordered Least Significant Bit) or 32 Bit MSB (ordered Most Significant Bit). 32 bit variables will use multiple registers while 16 bit variables use one.

Click **OK** to close the Edit Address / Register dialog box and return to the Add/Edit Variable dialog box. The register address is transferred to the text box automatically. Click **OK** to save the variable. The register is now assigned to a variable.

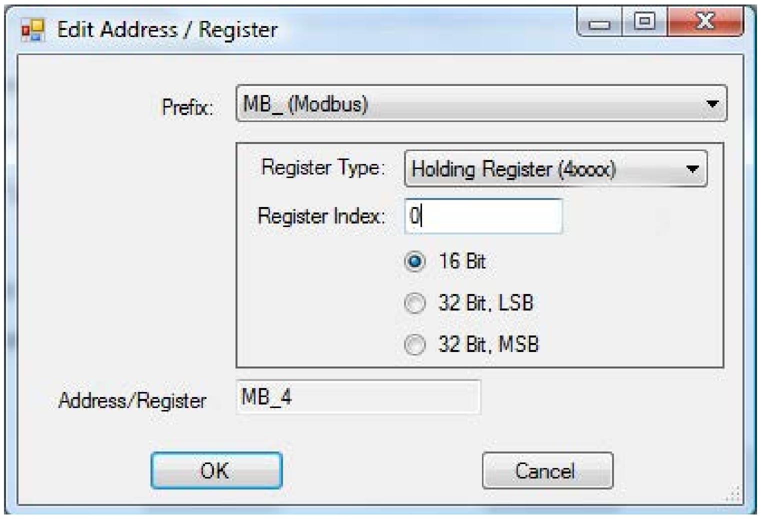


**Figure 13-16**

A Modbus address may be directly typed into the Address / Register box without using the **EDIT** button.

## Updating Network and Variable Values

When network registers are assigned to variables, any change to the variable locally in the ladder diagram project is available to the master to see without additional programming. If the master chooses to view the register, it will see the variables current value.

If the master chooses to modify a register (if the type is allowed to be modified), then any changes made by the master to the register will immediately change the value of the variable that is assigned to that register and will be used in the ladder diagram project locally.

## Modbus Slave Communication Errors

Modbus communications supports the use of error codes to identify and diagnose problems with the network and slaves. These errors are reported on master only. Typical error codes are:

| | |
|---|---|
| 2 - Illegal Data Address | This identifies that the master attempted to access an invalid register. |
| 3 - Illegal Data Value | This identifies the master attempted to access a register that is valid but not used in the ladder diagram project (on the slave unit).The largest register number is kept automatically by EZ LADDER Toolkit in the ladder diagram project. |

For more details regarding errors and error codes on a Modbus Network, refer to the network Master's documentation.

## Modbus Slave - Supported Master Functions

EZ LADDER Toolkit's Modbus Slave supports eight standard Modbus master functions (functions the master can use to access and update slave registers). While there is no way for the ladder diagram or EZ LADDER to use these functions, they are noted for reference.

Supported Functions include:

1 - Read Coil Status
2 - Read Discrete Input Status
3 - Read Holding Registers
4 - Read Input Registers
5 - Write to a Single Coil
6 - Write to a Single Register
15 - Write to Multiple Coils
16 - Write to Multiple Registers

## Modbus TCP over Ethernet / WiFi

In addition to supporting Modbus Master and Slave via UART(s) or Serial Ports, the P-Series based targets also support Modbus TCP over Ethernet / WiFi.

Modbus TCP over Ethernet/WiFi support is based on actual hardware target specifications. PLC on a Chip$^{TM}$ Integrated Circuits and Modules support Modbus Slave as well as do some standard Divelbiss PLCs and Controllers. For PLCs and controllers, refer to the supported features. See **Chapter 23 - Hardware Targets**.

### Configuring for Modbus TCP over Ethernet / WiFi

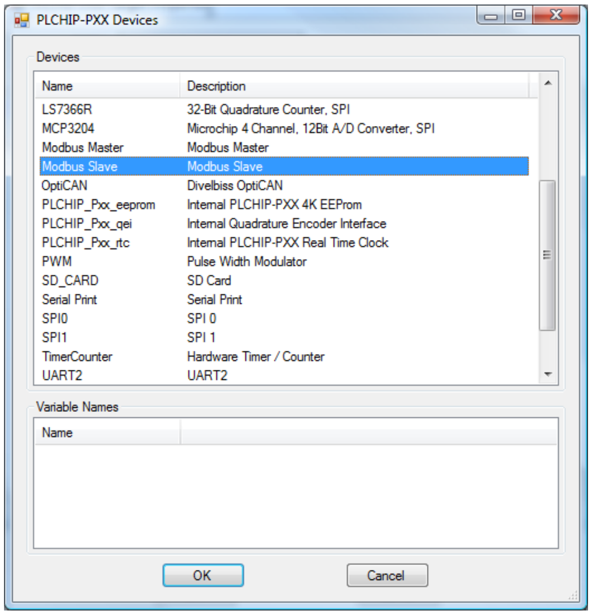
Before Modbus TCP over Ethernet or WiFi can be used, Ethernet / WiFi must be installed in the project settings prior to configuring Modbus. Refer to **Chapter 19 - Ethernet / WiFi**.

Prior to using the Modbus TCP over Ethernet in EZ LADDER Toolkit, Modbus must be installed and configured in the target. The Modbus TCP is configured using the Project Settings.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the Modbus were installed, it would be listed in the *Devices pane* under the Network section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

All the available devices and features for the target are shown in the Devices section. Scroll down and find the Modbus Slave or Modbus Master depending upon your exact needs. Figure 13-17 shows the Target's Devices window.


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |

Variable Names

Name

**Figure 13-17**

For this example select the **Modbus Slave** and click **OK**. The *Modbus Slave Properties* dialog will open. This dialog is used to specify which hardware interface will be used for this Modbus Network. Click the **ADD** button. The *Add Interface* dialog will open. Using the drop-down Interface menu, select the actual hardware interface: ***Ethernet***. The Ethernet must already be installed prior to this step.

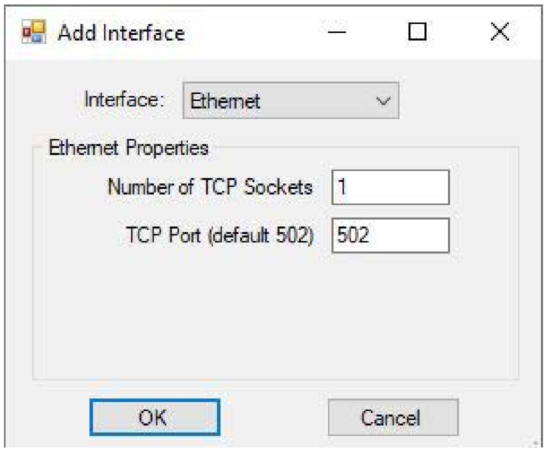
For Modbus Slave, enter the **Number of TCP Sockets** based on your needs. Selet the TCP Port (the default is 502). See Figure 13-18. For details on installing Ethernet, see **Chapter 19 - Ethernet / WiFi**.

For Modbus Slave, the TCP Pport can be changed in the ladder diagram at run-time by using the MODBUS_SET_PROPERTY function block.

For setting the Modbus TCP Port for Modbus Master, use the MODBUS_MASTER2 function block . See **Appendix A - Function Reference** for more detials on using the MODBUS_MASTER2 function block.

The WiFi interface is considered Ethernet in EZ LADDER Toolkit. When using WiFi for Modbus TCP, select Ethernet as the interface. When selecting Ethernet on WiFi enabled hardware, it will enable the Modbus TCP over WiFi.

If Modbus Master was selected, the Number of TCP Sockets is not available, but the Response Timeout will need to be configured.



Number of TCP Sockets: 1

TCP Port (default 502): 502

**Figure 13-18**

Click **OK** to close the Add Interface dialog and click **OK** to close the Modbus Slave Properties dialog and return to the Target Properties window. The Modbus Slave is now listed in the Devices pane under the Network Section.

The Modbus Slave Properties dialog and Add Interface dialogs may also be used to adjust settings and change the actual hardware interface by selecting the Modbus Slave in the Devices pane and clicking the Properties button.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The Modbus Slave TCP over Ethernet/WiFi is now ready to be used in the ladder diagram.

When configured as a Modbus Master, the target Slave ID (slave to communicate to) is set in the ladder diagram project using a variable and the **MODBUS_MASTER or MODBUS_MASTER2** function block. The variable connected to the ID pin of the MODBUS_MASTER function block will hold the IP address of the slave device. The slave ID is to be entered in an IP address format (eg: 192.168.1.55). Refer to Figures 13-19 and 13-20. Refer to Appendix A - Function Reference for more details on the **MODBUS_MASTER and MODBUS_MASTER2** function blocks.

## Setting Modbus TCP Master to Slave ID (Target Slave ID)

When configured as a Modbus Master, the target Slave ID (slave to communicate to) is set in the ladder diagram project using a variable and the **MODBUS_MASTER** or **MODBUS_MASTER** function blocks. The variable connected to the ID pin of the MODBUS_MASTER(2) function block will hold the IP address of the slave device. In the Default Value field, the slave ID is to be entered in an IP address format (eg: 192.168.1.55). Refer to Figures 13-19 and 13-20.

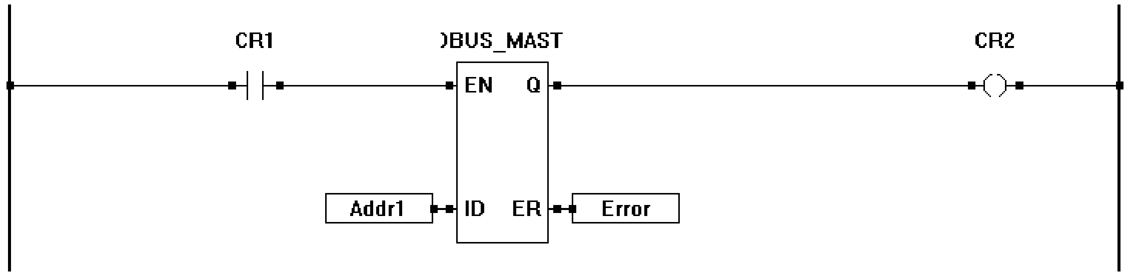
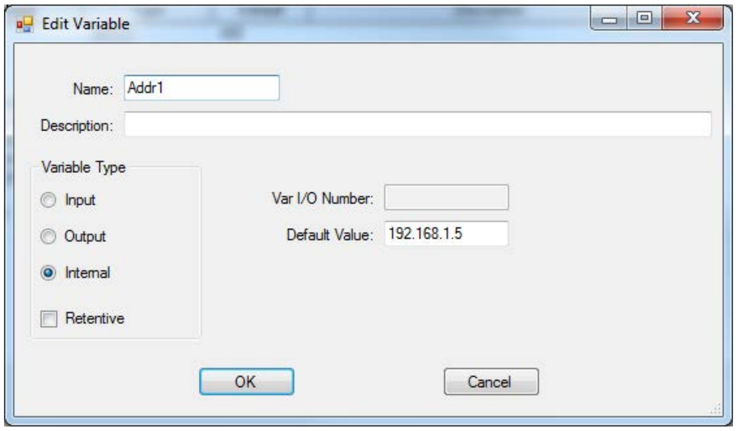


**Figure 13-19**



Name: Addr1
Description:
Variable Type
( ) Input
( ) Output
(●) Internal
[ ] Retentive
Var I/O Number:
Default Value: 192.168.1.5

**Figure 13-20**

## Setting Modbus TCP Slave ID

The Slave ID of the unit when configured for Modbus Slave is the actual IP address of the unit (the actual IP address that was configured in the unit's Bootloader. The address can be changed by editing the IP address by accessing the Bootloader. See **Chapter 4 - Configuring Targets** for details on accessing and setting the IP address within the Bootloader.

In addition to changing the Slave ID (IP address) in the Bootloader, it can be changed using Structured Text and the built-in Structured Text function *EZ_Eth_SetStaticIPV4Addr*. For information regarding structured text and using structured text functions, see **Chapter 26 - Structured Text** and **Appendix B - Structured Text Referece** of this manual.

## Modbus using Multiple Ports

The structure of the EZ LADDER Toolkit and P-Series hardware targets allow for modbus to be implemented using multiple ports that include serial ports and ethernet. As an example, two serial ports may be configured to be Modbus slaves with unique *slave ID* numbers or one serial port may be configured as a slave and the Ethernet / WiFiport may be configured to use Modbus TCP.

Regardless of the configuration, all the modbus registers are universal; meaning all the ports may use exactly the same register numbers (each port does not have individual register sets). This allows for multiple devices to access the same register and variable and therefore share the same data easily.

If the data in registers must be unique to each port, then the network registers must be mapped and divided based on register numbers on a per port basis. Using variables for each mapped network register section per port will allow this separate values to be accessed by the ladder variables (based on register number).

# CHAPTER 14

## CAN Networking

This chapter provides basic information to understand how to install, configure and use the OptiCAN Networking, J1939 Networking and NMEA 2000 Networking feature in the EZ LADDER Toolkit.

## Chapter Contents

## What is CAN?

CAN is short for Controller Area Network. CAN networks use a two-wire backbone to provide communication to each individual item on the network. There are specific requirements for the hardware, see the OptiCAN section of this chapter.

CAN networking refers to the hardware. There are multiple CAN protocols for communicating between devices on a CAN network. As with all networking and protocols, protocols have advantages and disadvantages. Some common protocols are: SAE J1939, CANopen, etc.

At this time the P-Series based targets only support the proprietary OptiCAN protocol, SAE J1939 and NMEA 2000.

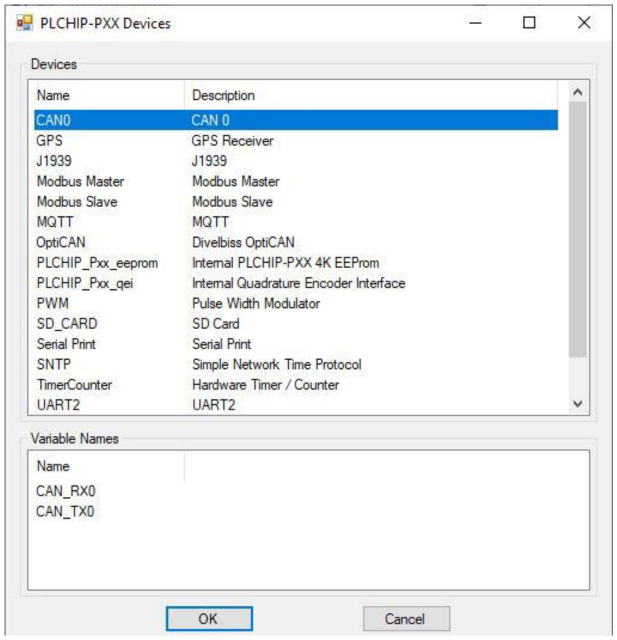
## Installing CAN Network Ports

Prior to installing and configuring any type of CAN protocol, the actual CAN port must installed in the P-Series target.

CAN Networking Port availability is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if CAN Network Ports are supported.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any CAN port were installed, it would be listed in the *Devices pane* under the Bus\CAN section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the CAN ports (CAN0, CAN1, etc). Figure 14-1 shows the Target's Devices window.



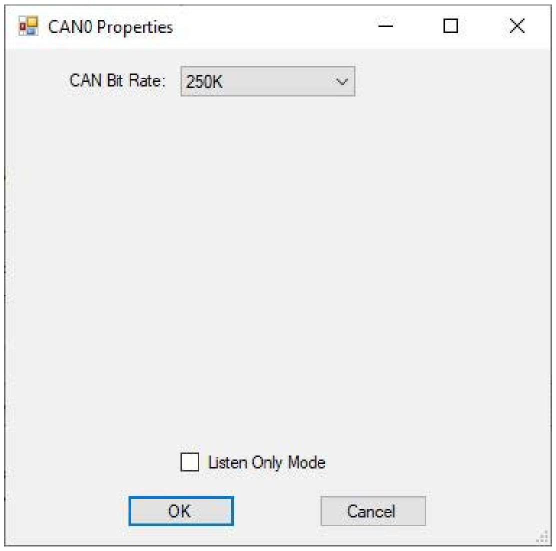
| Name | Description |
|---|---|
| CAN0 | CAN 0 |
| GPS | GPS Receiver |
| J1939 | J1939 |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SNTP | Simple Network Time Protocol |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |

| Name |
|---|
| CAN_RX0 |
| CAN_TX0 |

**Figure 14-1**

Select **CANx** and click **OK**. The *CANx Properties* dialog will open. This dialog set the CAN bit rate for communication for any **Native CAN** communications. Refer to figure 14-2. Native CAN communications is discussed later in this chapter.

Rate and other communication configuration items are controlled by the specific protocol (ie: OptiCAN, SAE J1939 and NMEA 2000) and are set in their respective target settings or hard set when the protocol is installed. OptiCAN bit rate is hard set at 250K. SAE J1939 and NMEA 2000 bit rate is set in the SAE J1939 and NMEA 2000 properties when added to the target.


CAN0 Properties

CAN Bit Rate: 250K

☐ Listen Only Mode

**Figure 14-2**

The **Listen Only Mode** checkbox applies to the **Native CAN** communications. When checked the Native Raw CAN communications can receive CAN data, but not transmit. Select the Listen Only Mode if you want to receive CAN data only (Native CAN).

When another protocol such as OptiCAN, SAE J1939 or NMEA 2000 is installed in the project, the **Listen Only Mode** checkbox and properties is disable and does not apply.

Click **OK**. The *CANx Properties* dialog will close and the previous target properties window will now list the CAN port as an installed device.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The CAN port is now installed and ready to be used in the ladder diagram.

## Divelbiss OptiCAN Network

OptiCAN is a Divelbiss proprietary CAN (Controller Area Network) that provides a communication link between Divelbiss OptiCAN enabled controllers and other OptiCAN enabled controllers and devices such as I/O modules or other controllers. The Divelbiss OptiCAN network supports up to 64 nodes (devices) and is register based. Each node supports up to 256 registers and communication can be triggered based on time or on an event.

Divelbiss OptiCAN can perform the following major functions:

1. Allow controllers to access external I/O Devices
2. Allow controllers to access other controllers
3. Allow the user to configure devices utilizing the CAN protocol

Only Divelbiss OptiCAN enabled devices will communicate on the network OptiCAN network. Connecting non-OptiCAN devices will result in network errors including loss of communication.

### Planning the OptiCAN Network

As with any network or communication scheme, the network should be planned taking into account the amount of communication, broadcast rate, communication triggers, register assignments and timing requirements. This plan is essential for a successful implementation of a network.

It is suggested that register needs should be identified and assigned for each device prior to the start of the programming. While any legal register may be used, it is recommended that register assignments start at the high end of available registers and work backward (i.e.: start with register 127 and then assign 126 and so on). As some devices utilize lower register numbers this will ensure that the controller register assignments will not interfere with the device register assignments.

All OptiCAN controllers have the ability to *broadcast* (send data) and *listen* (receive data). OptiCAN Controllers broadcast to all units (called *nodes*) on the network. This is called a *Global Broadcast*.

While all controllers may broadcast and listen, ideally one should be identified as a controlling agent for the network. This agent controller should be responsible for the network commands that start, stop and reset the OptiCAN network communications.

### Hardware Requirements & Recommendations

For optimal functionality, performance and noise immunity, all the hardware recommendations must be followed. A failure to follow recommended hardware requirements could result in decreased reliability of the OptiCAN Network.

Please adhere to the following requirements and recommendations:

1. The OptiCAN network cable should be of a *twisted pair with shield* variety and cannot exceed 40 meters in total length. Additional length or incorrect cable type may limit functionality or cause the network to fail.

Please adhere to the following specifications for cable requirements for all OptiCAN networks.

| Twisted Pair Shielded Cable Specifications | | | | | | |
|---|---|---|---|---|---|---|
| **Parameter** | **Symbol** | **Minimum** | **Nominal** | **Maximum** | **Unit** | **Comments** |
| Impedance | Z | 108 | 120 | 132 | Ω | |
| Specific Resistance | $r_b$ | 0 | 25 | 50 | mΩ/m | |
| Specific Capacitance | $C_b$ | 0 | 40 | 75 | pF/m | Between Conductors |
| | $C_s$ | 0 | 70 | 110 | pF/m | Conductor to Shield |

2. A 120Ω ohm resistor (load) is required at each end of the network.

   Please adhere to the following specifications for terminating resistor requirements for all OptiCAN networks.

| Terminating Resistor Specifications | | | | | | |
|---|---|---|---|---|---|---|
| **Parameter** | **Symbol** | **Minimum** | **Nominal** | **Maximum** | **Unit** | **Comments** |
| Resistance | $R_L$ | 110 | 120 | 130 | Ω | Min power dissipation 400mW(1) |
| Inductance | | | | 1 | µh | |
| (1) Assumes a short of 16V to $V_{CAN\_H}$ | | | | | | |

3. The cable shield should be grounded near the middle of the network (cable) run.

   The shield should only be connected to ground and one point on the network. Multiple ground points could cause a ground loop, decrease noise immunity and adversely affect network performance.

4. If wiring as a network bus with stub connections, the maximum stub length from bus to node is 1 Meter.

   See Figure 14-3 for a sample connection diagram.

## OptiCAN Specifications

| | |
|---|---|
| **Bandwidth**: | 250 KBits / Sec |
| **Maximum Cable Length:** | Up to 270 Meters [1] |
| **Maximum Number of Nodes:** | Up to 254 Nodes [1] |
| **Registers per Node:** | 256 Registers |
| [1] Dependent on cable selection and bus loading. See Application Note for CAN Transceiver (NXP AN00020 or your CAN transceiver) | |

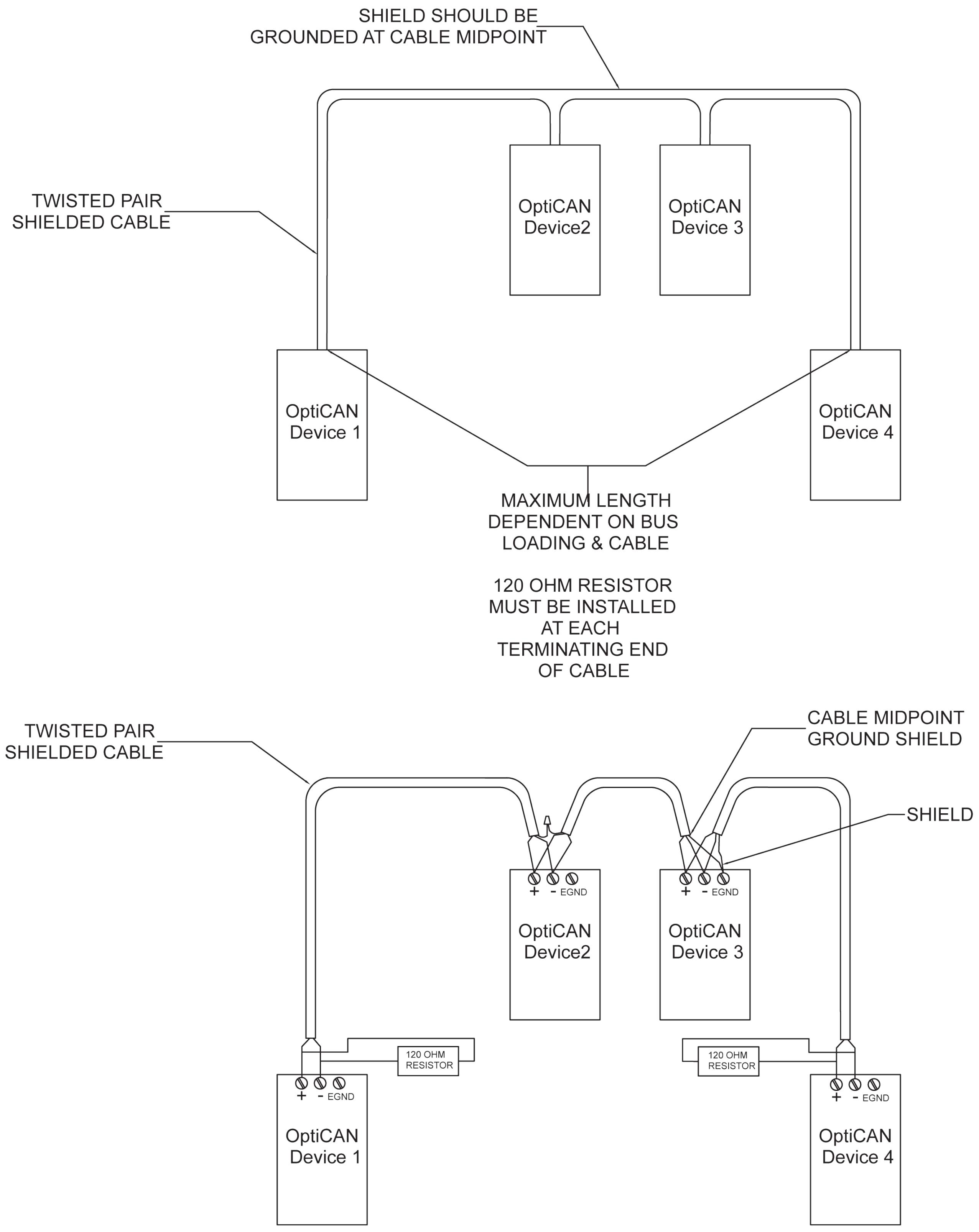

SHIELD SHOULD BE
GROUNDED AT CABLE MIDPOINT
TWISTED PAIR
SHIELDED CABLE
OptiCAN
Device2
OptiCAN
Device 3
OptiCAN
Device 1
OptiCAN
Device 4
MAXIMUM LENGTH
DEPENDENT ON BUS
LOADING & CABLE
120 OHM RESISTOR
MUST BE INSTALLED
AT EACH
TERMINATING END
OF CABLE
TWISTED PAIR
SHIELDED CABLE
CABLE MIDPOINT
GROUND SHIELD
SHIELD
+ - EGND
OptiCAN
Device2
+ - EGND
OptiCAN
Device 3
120 OHM
RESISTOR
120 OHM
RESISTOR
+ - EGND
OptiCAN
Device 1
+ - EGND
OptiCAN
Device 4

**Figure 14-3**

## Using Controllers on the OptiCAN Network

A typical application involves a controller running it's own ladder diagram project, monitoring inputs and controlling outputs based upon the project that is running. When connected to an OptiCAN network the controller will operate the same, but now using OptiCAN, it can communicate to other devices including other controllers with OptiCAN and OptiCAN I/O Modules.

The following describes how a controller can operate when used on a active OptiCAN network.

1. Local Control: Monitor Inputs and Control Outputs
2. Globally *broadcast* data to all OptiCAN Nodes on the OptiCAN Network
3. *Listen* for Broadcasts from a specific Node on the OptiCAN Network

All EZ LADDER Toolkit programmed OptiCAN network controllers are configured using the EZ LADDER Toolkit and maintain their network settings, parameters and register settings in the actual ladder diagram project. Each controller on the OptiCAN Network may have different settings and all will be required to have a different Node ID (address).

## OptiCAN Controller Heartbeat

Each OptiCAN controller has the ability to broadcast a signal called a *heartbeat*. This signal is broadcast at a regular interval and is used to ensure that all devices on the network are communicating properly. Each node automatically listens for this heartbeat and adjusts OptiCAN registers based on the network condition. These conditions may be monitored using function blocks.

One node on the OptiCAN network **MUST** broadcast the heartbeat message for the network to function properly. Although it is possible to have multiple controllers on one network sending heartbeats, it is recommended only one controller broadcast a heartbeat per network.

In the event the heartbeat is lost, then the local ladder diagram project should ignore data from the network as the loss of heartbeat signifies that communication with part or all of the network has been lost. How a controller responds to a network loss is entirely dependent on the ladder diagram project.

## Installing a Controller on the OptiCAN Network

Before a controller may communicate on the OptiCAN network, it must be installed in the EZ LADDER Toolkit. Once the OptiCAN settings are configured, they are stored in the actual ladder diagram project. Please see the following steps required to install and configure the OptiCAN network feature on a controller. Actual menus steps to reach the OptiCAN configuration may vary based on the actual controller used, but the configuration itself is always the same. Divelbiss standard controllers based on PLC on a Chip (Enhanced Baby Bear, PCS-XXX, etc) are configured based on the part number. For details on specific targets, please see **Chapter 23 - Hardware Targets**

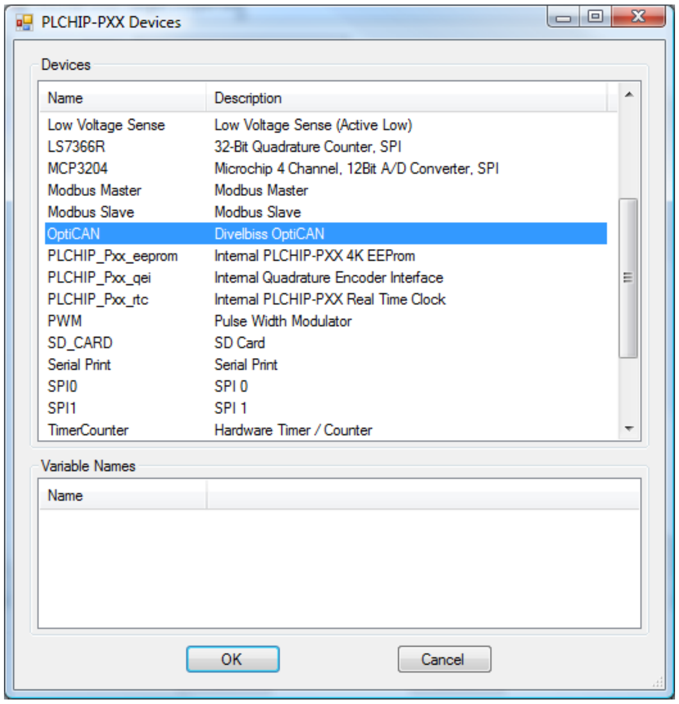
OptiCAN Networking availability is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if OptiCAN is supported.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If OptiCAN were installed, it would be listed in the *Devices pane* under the Device section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

A CAN port must be installed prior to installing OptiCAN. See **Installing CAN Network Ports** earlier in this chapter.

All the available devices and features for the target are shown in the Devices section. Scroll down and find the OptiCAN. Figure 14-4 shows the Target's Devices window.



| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |

**Figure 14-4**

Click **OK**. The OptiCAN Properties dialog will open allowing configuration of the OptiCAN communications settings. Refer to Figure 14-5.

Set the paramters as needed:

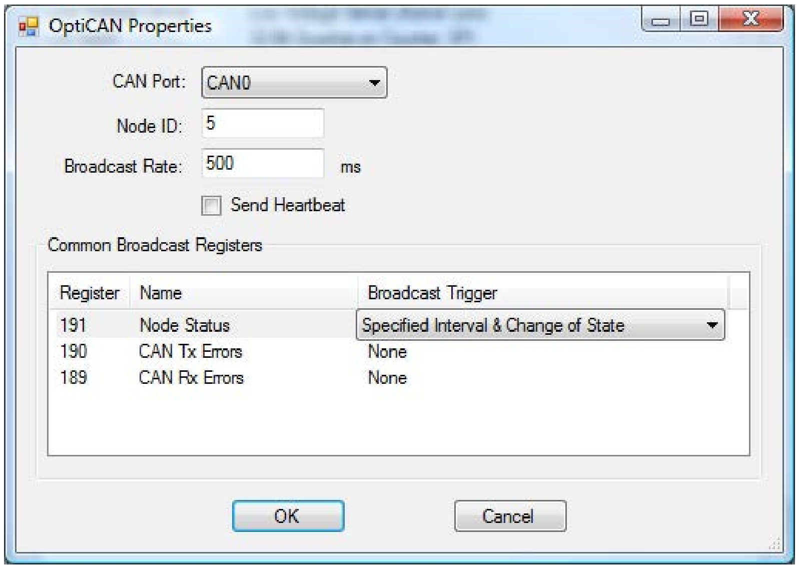
1. CAN Port Using the drop down menu, select the physical CAN port that will be connected to the OptiCAN network. All installed and not used CAN ports are displayed.

2. Node ID — The Node ID serves as the controller's address on the network. It may be numbered up to the maximum number of nodes allowed.

All Node IDs on an OptiCAN Network must be unique. Duplicate Node IDs will result in communication errors or communication loss.

The node ID may also be set from the ladder diagram project using a variable. This variable must be configured as node 255. The variable (integer value) then becomes the OptiCAN node ID. It is important to keep this ID number in the proper range.

3. Broadcast Rate — The rate that the controller will broadcast registers is entered here in milliseconds. This timing requirement should be identified during network planning.



CAN Port: CAN0
Node ID: 5
Broadcast Rate: 500 ms
☐ Send Heartbeat

Common Broadcast Registers

| Register | Name | Broadcast Trigger |
|---|---|---|
| 191 | Node Status | Specified Interval & Change of State |
| 190 | CAN Tx Errors | None |
| 189 | CAN Rx Errors | None |

**Figure 14-5**

In addition to the parameters listed above that are required, the following additional configuration points are optional based on the requirements of the controller, program and network configuration.

1. Send Heartbeat — Check this box configure this controller to send a network heartbeat signal.

One node on the OptiCAN network **MUST** broadcast the heartbeat message for the network to function properly. Although it is possible to have multiple controllers on one network sending heartbeats, it is recommended only one controller broadcast a heartbeat per network.

2. 191 Node Status — This setting identifies when to broadcast the status of this controller (node). The broadcast trigger may be selected by clicking in the table. The selections are Specified Interval, Change of State and Specified Interval and Change of State.

| | |
|---|---|
| 3. CAN Tx Errors | This setting identifies when to broadcast the CAN Transmit errors identified by this controller (node).The broadcast trigger may be selected by clicking in the table. The selections are Specified Interval, Change of State and Specified Interval and Change of State. |
| 4. CAN Rx Errors | This setting identifies when to broadcast the CAN Receive errors identified by this controller (node).The broadcast trigger may be selected by clicking in the table. The selections are Specified Interval, Change of State and Specified Interval and Change of State. |

When the OptiCAN properties have been configured, click **OK** to close the OptiCAN Properties dialog and return to the the Target Properties window. OptiCAN is now listed in the Devices pane under Device.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. OptiCAN is now installed and ready to be used in the ladder diagram.

## Controller OptiCAN Network Register Assignments

The OptiCAN network operates based on preset and user defined registers. The following are general register assignments and information common for all OptiCAN enabled controllers. For non-controller devices, please consult the product's data sheet for detailed register assignments and preset functions.

**General Register Assignments:** These are the overall general register assignments common to all OptiCAN enabled devices.

| Register Number | Assigned Function / Use |
|---|---|
| 0-127 | User Defined Controller Registers and I/O Defined Registers |
| 128-191 | Common Broadcast Registers |
| 192-255 | Common Configuration and Command Registers |

User Defined registers for controllers are available for the user to define the use of during the ladder diagram project development. Device Defined registers for I/O and other devices have preset definitions of register use and cannot be changed.

**Common Configuration / Command Register Assignments:** These registers are pre-assigned and cannot be altered. These register's contents may only be modified by a controller and can only change the I/O setting.

| Register Number | Name | Description | Read / Write |
|---|---|---|---|
| 255 | Node ID | The node's ID Number | Read / Write |
| 254 | Serial Number | The node's Serial Number | Read |
| 253 | Broadcast Interval | Interval for Broadcasting (ms) | Read / Write |
| 252 | Broadcast Trigger 0 | Broadcast Trigger for Registers 0-15 | Read / Write |
| 251 | Broadcast Trigger 1 | Broadcast Trigger for Registers 16-31 | Read / Write |
| 250 | Broadcast Trigger 2 | Broadcast Trigger for Registers 32-47 | Read / Write |
| 249 | Broadcast Trigger 3 | Broadcast Trigger for Registers 48-63 | Read / Write |
| 248 | Broadcast Trigger 4 | Broadcast Trigger for Registers 64-79 | Read / Write |
| 247 | Broadcast Trigger 5 | Broadcast Trigger for Registers 80-95 | Read / Write |
| 246 | Broadcast Trigger 6 | Broadcast Trigger for Registers 96-111 | Read / Write |
| 245 | Broadcast Trigger 7 | Broadcast Trigger for Registers 112-127 | Read / Write |
| 244 | Broadcast Trigger 8 | Broadcast Trigger for Registers 128-143 | Read / Write |
| 243 | Broadcast Trigger 9 | Broadcast Trigger for Registers 144-159 | Read / Write |
| 242 | Broadcast Trigger 10 | Broadcast Trigger for Registers 160-175 | Read / Write |
| 241 | Broadcast Trigger 11 | Broadcast Trigger for Registers 176-191 | Read / Write |

**Common Broadcast Register Assignments:** These registers are pre-assigned and cannot be altered

| Register Number | Name | Description | Read / Write |
|---|---|---|---|
| 191 | Node Status | This Node's Status | Read |
| 190 | CAN Transmit Errors | CAN Transmit Error Counter | Read |
| 189 | CAN Receive Errors | CAN Receive Error Counter | Read |

The Node Status register (191) is represented by a 32 bit number. The lower 16 bits represents the current **status code** while the upper 16 bits represents the **error code**.

There are three status codes supported on the OptiCAN network. The status codes are:
1 = Reset, 2 = Active, and 4 = Error.

If the error code returned is 0, then typically, the network was not started.

Error codes are divided into two groups. Device specific errors are numbered 0-32767 while common error codes are numbered 32768-65535.

Common Error Codes are as follows:

65535 = CAN Controller Receive Error
65534 = CAN Controller Receive Warning
65533 = CAN Controller Transmit Error
65532 = CAN Controller Transmit Warning
65531 = CAN Controller Bus Off State
65530 = CAN Controller Data Overrun
65519 = OptiCAN Heartbeat Timeout
65518 = CAN Controller Error

## Broadcasting to Other Controllers  and Devices

To broadcast from one controller to other controllers and devices, the following steps should be completed before proceeding.

1. All OptiCAN Devices and Controllers on the network must be identified with unique Node ID numbers and configured properly.
2. Register assignments and uses should be defined as these register number will be needed to broadcast and listen.
3. The heartbeat node should be identified.

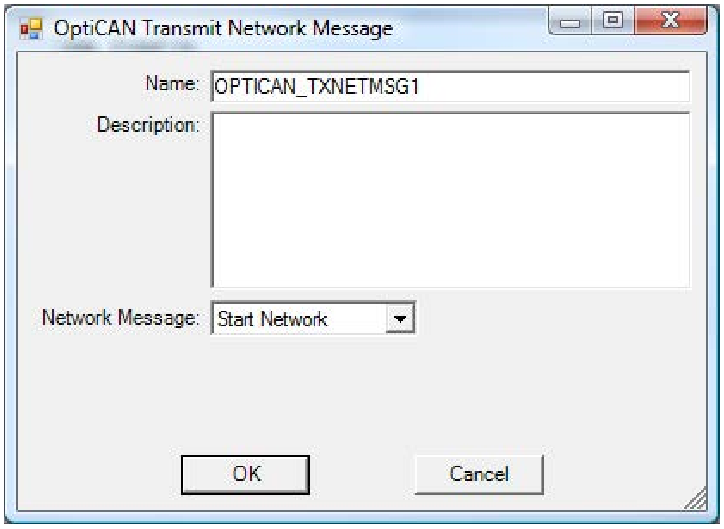
To broadcast to nodes, several steps must take place in addition to the configurations listed above. For the OptiCAN network to function correctly, several steps must be taken.  Before any node can broadcast or listen, the OptiCAN Network must be *started*.

### To Start the OptiCAN Network

The OPTICAN_TXNETMSG function block is used to send global network commands to all OptiCAN nodes on a network.  Using this function block, a controller may send a ***Network Start***, ***Network Stop*** or a **Network Reset** command.

On power up, the OptiCAN network does NOT start by default and will not begin communication until one controller has sent the ***Network Start*** command.

To send the start command, the OPTICAN_TXNETMSG function block is used.  Using the OPTICAN_ TXNETMSG function block is a two step process.  When placing the function block, the OptiCAN Transmit Network Message dialog box will open.  See Figure 14-6.  Using the drop down menu, select the Network Message *Start Network*.


OptiCAN Transmit Network Message

Name: OPTICAN_TXNETMSG1

Description:

Network Message: Start Network

**Figure 14-6**

Click **OK** to place the function block in the ladder diagram project. Figure 14-7 is a sample of a complete OPTICAN_TXNETMSG circuit. Note the use of contacts to control when the Start Network is sent.

The Network Start should be sent based on two conditions: The network needs to start as in a start-up or if communication errors are detected. If a single node loses power, it will appear as communication loss. When node regains power, it will not communicate on the network unless another Network Start is sent (since nodes do not start on power up). If at any time a communication is lost to a node, re-send the Network Start.

Restarting upon an error is entirely application dependent. Some applications would benefit from an automatic restart while other applications may find it beneficial to stop all functions when an error is detected. Safety should be paramount when deciding when to automatically restart.

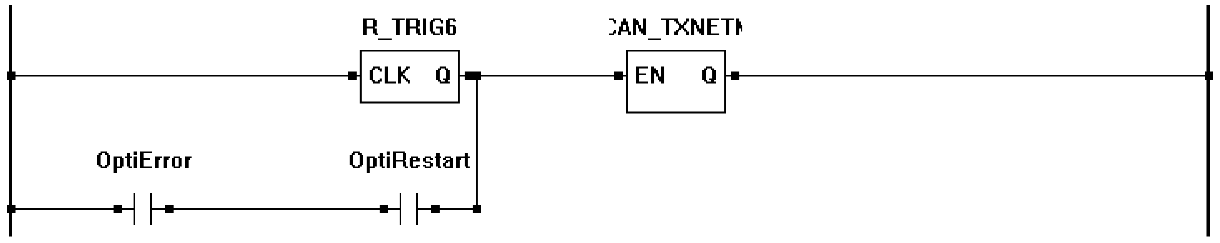


**Figure 14-7**

All nodes on the network should begin communication upon receipt of the Start Network command. With the network started and communicating, it is now possible to broadcast to nodes and listen for node broadcasts.

## Global Broadcasting to all Nodes

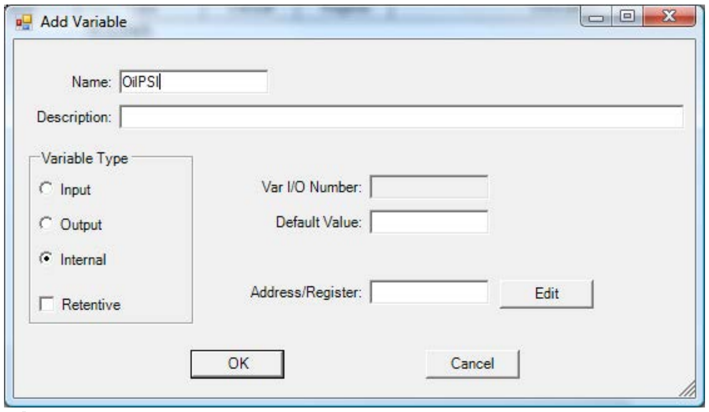
To broadcast or listen, a basic understanding of registers is required. Typically, controller registers 0-127 are available to be user-defined and used while 128-255 are pre-defined and cannot be altered. The user-defined registers are commonly used to communicate between controllers.

It is recommended that before programming is started that all nodes are identified, assigned a node ID and documented. For each device, their register requirements should be identified, registers assigned and registers documented. This will verify the all requirements are met and help to promote proper functionality and design.

To send a global broadcast, a variable must be identified and assigned to use an OptiCAN register. To assign an OptiCAN register to a variable, using the Add Variable or Edit Variable dialog, click the **EDIT** button next to the Address / Register field. See Figure 14-8.

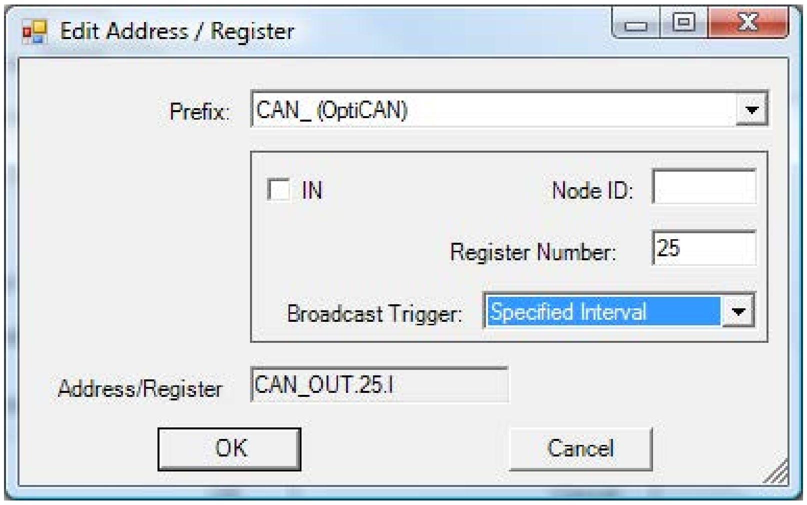
The Edit Address / Register dialog box will open. See Figure 14-9. Using the drop down menu, select ***CAN_(OptiCAN)***. Fill in the Register Number only to transmit to the same register on all nodes. The register number must be a user-defined register (0-127). When broadcasting, leave the Node ID empty or blank.

Leaving the Node ID blank causes this variable to be sent to the same register number on all nodes on the network (Global Broadcast). This method will allow for any node to have the ability to *listen* for this register broadcast.



Name: OilPSI
Description:
Internal [✓]
Retentive
Var I/O Number:
Default Value:
Address/Register:

Figure 14-8



Prefix: CAN_ (OptiCAN)
IN
Node ID:
Register Number: 25
Broadcast Trigger: Specified Interval
Address/Register CAN_OUT.25.I

Figure 14-9

Using the Broadcast drop down menu, select a broadcast trigger. The choices are: None, Specified Interval, Change of State and Specified Interval and Change of State. This register will be broadcast when this condition is met. Click **OK** to close the dialog and click **OK** to close the Add / Edit Variable dialog.

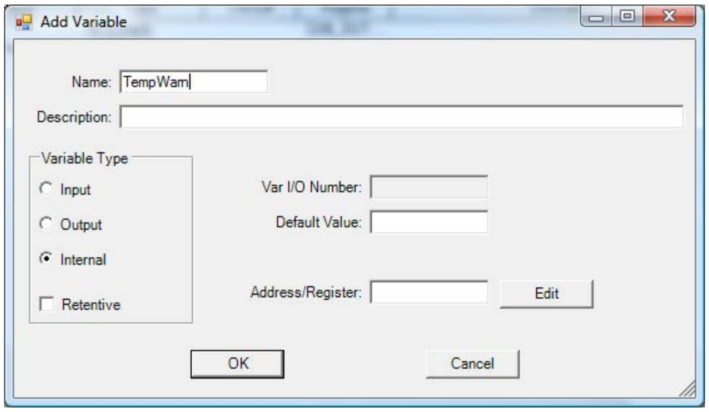
As the register type is selected and the number entered, the Address / Register displayed will be updated immediately. The CAN_ and the register number is automatically entered.

In this example, when this ladder diagram project is running, value of the variable OilPSI will be transmitted globally to all nodes at register 25. The interval will the same as Broadcast Rate that was configured in the Project Settings.

## Listening for Broadcasts

While broadcasting can be global or to a specific node ID, all listening for broadcasts are specific. For a controller to listen for a broadcast, the specific node ID and register are required.

To listen for a broadcast, a variable must be identified and assigned to use an OptiCAN register. To assign an OptiCAN register to a variable, using the Add Variable or Edit Variable dialog, click the **EDIT** button next to the Address / Register field. See Figure 14-10.



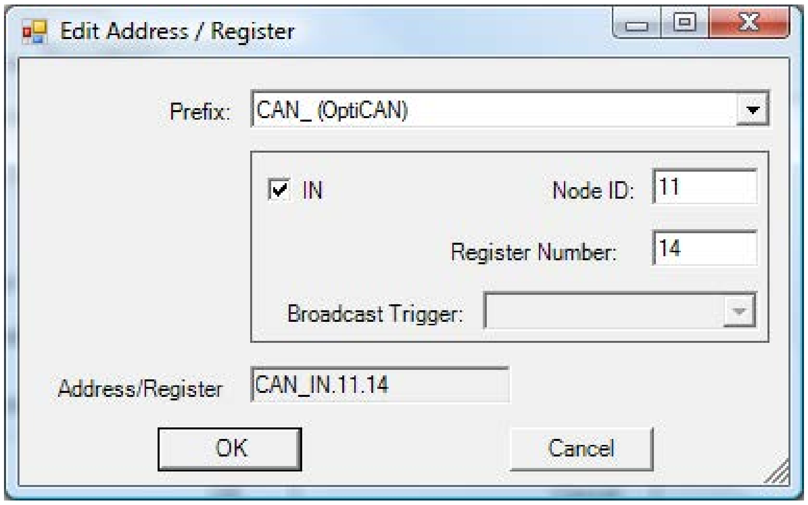
Name: TempWarn
Description:
Address/Register:

**Figure 14-10**

The Edit Address / Register dialog box will open. See Figure 14-11. Using the drop down menu, select ***CAN_(OptiCAN)***. Fill in the Node ID with the node ID number of the device you wish to listen for.

Fill in the Register Number that you are listening for on the specific node. The register number must be a user-defined register (0-127).

Click the IN check box. This identifies that this variable will be listening, not broadcasting. The Broadcast Trigger drop down is no longer available. Click **OK** to close the dialog and click **OK** to close the Add / Edit Variable dialog.



Prefix: CAN_ (OptiCAN)
☑ IN
Node ID: 11
Register Number: 14
Broadcast Trigger:
Address/Register CAN_IN.11.14

**Figure 14-11**

Leaving the node ID blank while is allowable, but is not a valid address and no data will be received.

When this ladder diagram project is running, if a broadcast from node ID 11, register 4 is seen on the network, it will be heard and the register (variable) on this controller will be updated.

## Determining Node Status

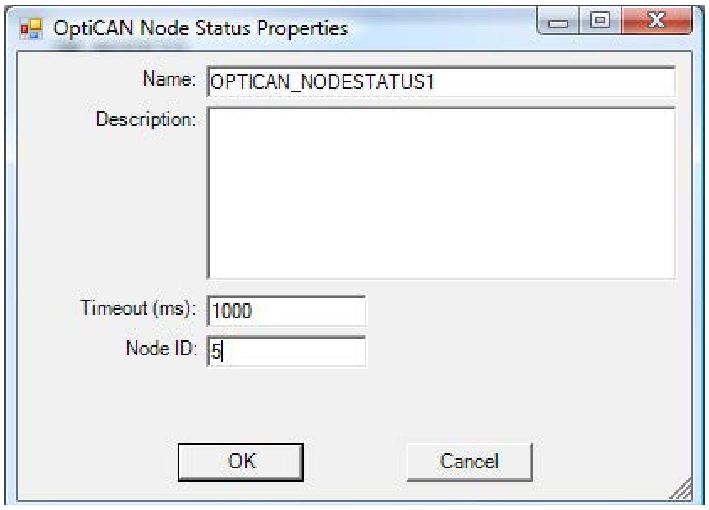
As discussed earlier in this chapter, that a Start Network command must be transmitted to start the network communicating and that it should happen on start up. In addition, it was recommended to monitor nodes status and possible resent the Start Network command in the event of a communications loss to a node.

To determine the status of a specific node, the OPTICAN_NODESTATUS function block is used. Using the OPTICAN_NODESTATUS function block is a two step process. When placing the function block, the OptiCAN Node Status Properties dialog box will open. See Figure 14-12.

In the Node ID field, enter the node ID that is to be monitored for communication loss and in the Timeout (ms) field, enter the amount of time that communication is lost before the node status is changed (in milliseconds).

When enabled, the OPTICAN_NODESTATUS block's Q output will be true if messages are received from the actual node. If the Q output is false, then the node status is not valid as no messages are received from it. When this occurs, the VAL output will be set to zero.

The OPTICAN_NODESTATUS function block is node specific, meaning that for each node that must be monitored for a network restart, a separate function block is required. In addition, nodes that are considered critical in overall operation may require multiple uses of the OPTICAN_NODESTATUS function block to identify errors and handle them correctly. Restarting or stopping based on communication errors is application dependent. Keep in mind, there will always be some communication errors on any network. The amount and type of what is allowable is application dependent.


OptiCAN Node Status Properties

Name: OPTICAN_NODESTATUS1

Description:

Timeout (ms): 1000

Node ID: 5

**Figure 14-12**

Click **OK** to place the function block in the ladder diagram project. Figure 14-13 is a sample of a complete OPTICAN_NODESTATUS circuit. The Error variable shown will be equal to the status of the node that was programmed into the function block. The error codes are listed earlier in this chapter.

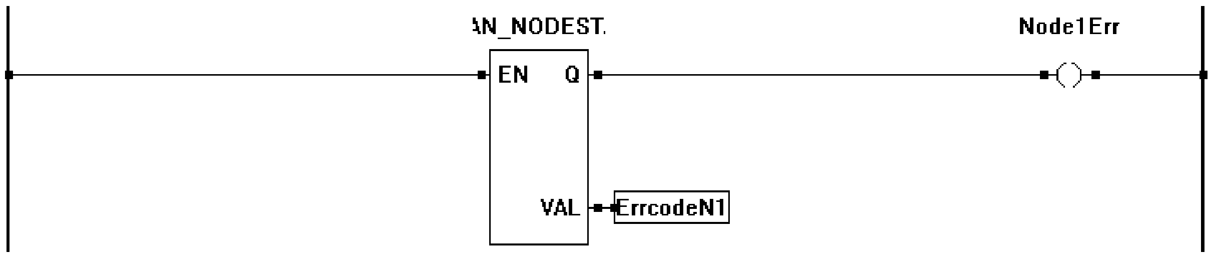


Figure 14-13

## Using the OptiCAN Configuration Tool

Up to this point, we have configured controllers on the OptiCAN network. In addition to controllers, other devices support OptiCAN including I/O modules. As these devices are not programmed with an EZ LADDER Toolkit ladder diagram project, other means must be used to identify and configure them. There are two tools that may be used to configure non-controller OptiCAN devices.

The first option is to purchase the OptiCAN Configuration Tool Professional. The Professional version is sold separately and requires additional hardware (included in the purchase). The Professional version does not have a limitation on the number of nodes that may configured. In addition, it also has more advanced controls, diagnostics and reporting features. The OptiCAN Configuration Tool Professional has it's own User's Manual.

If the OptiCAN network will host more than 10 non-controller nodes, then you must purchase and use the OptiCAN Configuration Tool Professional to configure the non-controller nodes.

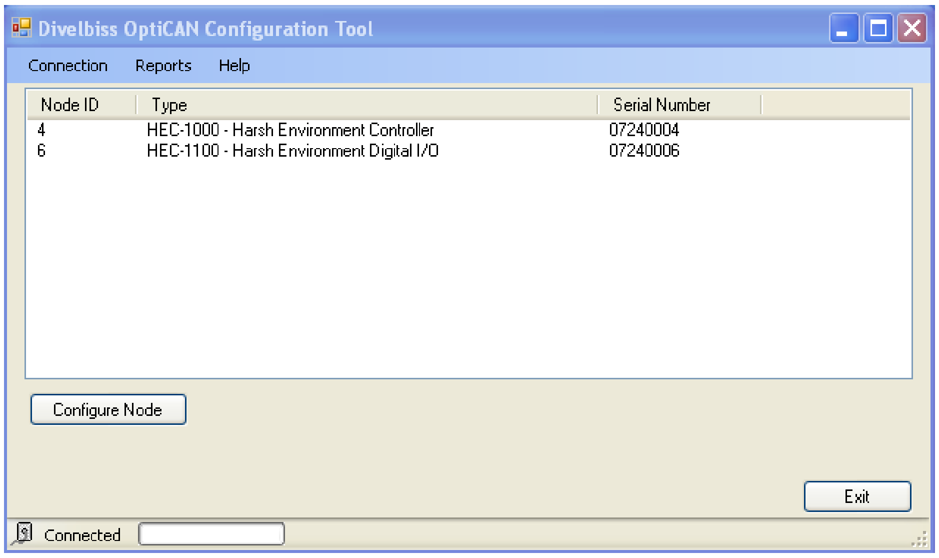
The OptiCAN Configuration Tool Basic is part of the EZ LADDER Toolkit and is capable of configuring up to 10 total non-controller nodes on an OptiCAN network. As this is a feature of the EZ LADDER Toolkit and does not require additional hardware or software, it will covered in detail.

The can detect up to 10 nodes, returning the Node ID, Device Type / Name and it's Serial Number.

To use the OptiCAN Configuration Tool, a ladder diagram project with OptiCAN enabled must be loaded, compiled and running on a target. Using EZ LADDER Toolkit, change to the Monitor Mode. In the Monitor Mode, using the ***Project Menu***, select *OptiCAN*. The Divelbiss OptiCAN Configuration Tool will open in a new window. See Figure 14-14.

EZ LADDER Toolkit must have a project loaded and be in Monitor mode (with OptiCAN enabled) to open the OptiCAN Configuration Tool. It is not necessary to connect to the target controller. If connected to the controller, the OptiCAN Configuration Tool will disconnect EZ LADDER Toolkit from the controller when it opens.

Whenever the OptiCAN Configuration tool connects, it automatically sends the Stop Network command. The network will have to be restarted for proper operation.



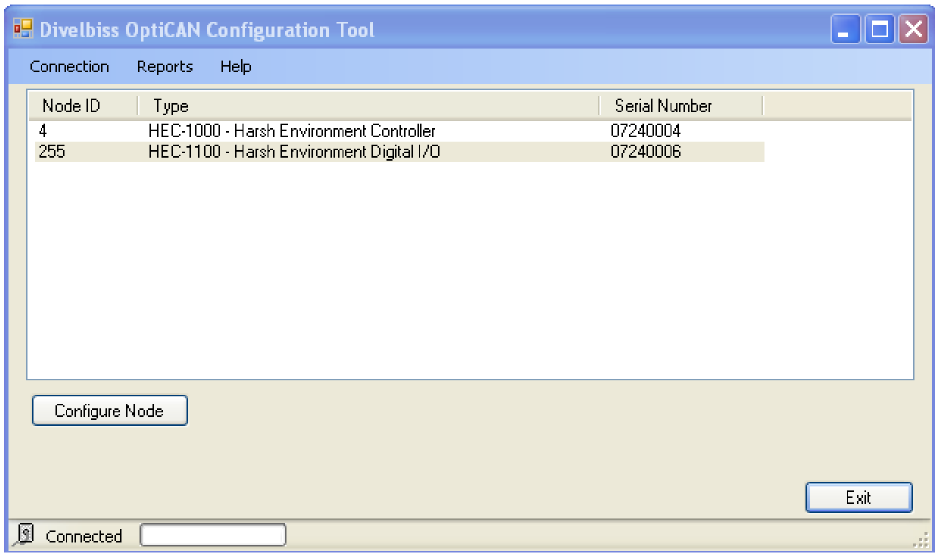
| Node ID | Type | Serial Number |
|---|---|---|
| 4 | HEC-1000 - Harsh Environment Controller | 07240004 |
| 6 | HEC-1100 - Harsh Environment Digital I/O | 07240006 |

**Figure 14-14**

As shown in Figure 14-14, there are two devices on the OptiCAN network. The tool shows the Node ID, Type and Serial Number for each of the devices. These two devices are already configured as they have Node ID's assigned.

When configuring a non-controller device for the first time, the device will display with a Node ID of 255. The 255 designation is reserved for devices that have not been configured. See Figure 14-15. For multiple new devices, they will all be assigned the same 255 Node ID. The controller can differentiate between devices that have not been configured using their serial number. The serial number is programmed at the factory and is not user changeable.

Only non-controller device Node IDs may be configured using this tool. Controller Node IDs are only changeable in the actual ladder diagram project loaded on the controller.



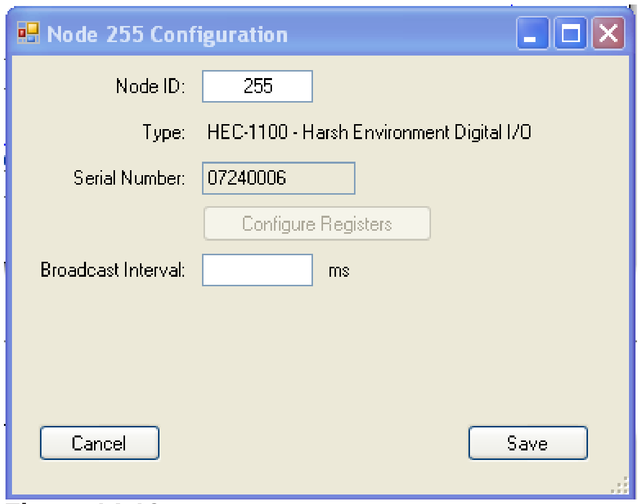
| Node ID | Type | Serial Number |
|---|---|---|
| 4 | HEC-1000 - Harsh Environment Controller | 07240004 |
| 255 | HEC-1100 - Harsh Environment Digital I/O | 07240006 |

**Figure 14-15**

## To Configure a Node

To configure a node, select the node (highlight) in the list and click the **CONFIGURE NODE** button. The Node Configuration dialog box will open. See Figure 14-16. The following can be viewed from the Node Configuration dialog. Some may be configured while others may not.

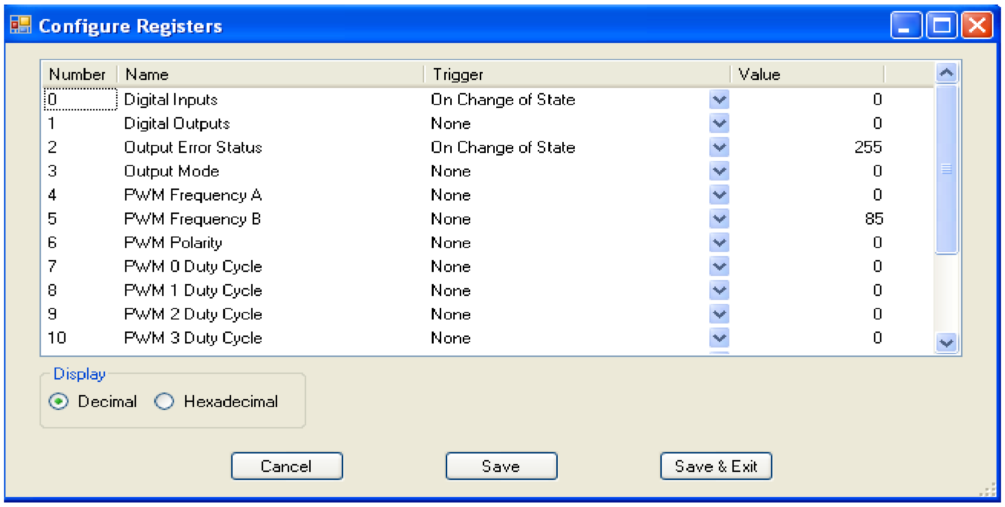
| | |
|---|---|
| **Node ID**: | This is where the node ID number is set. |
| **Type**: | This is the description of the device (cannot be edited). |
| **Serial Number**: | This is the serial number of the device (programmed at factory and cannot be edited). |
| **Broadcast Interval**: | This is the interval (rate) at which the registers will be broadcast on the net work. |



Node ID: 255
Type: HEC-1100 - Harsh Environment Digital I/O
Serial Number: 07240006
Broadcast Interval: ms

Figure 14-16

The **CONFIGURE REGISTERS** button is used to configure each register of the device including it's trigger and value. **CONFIGURE REGISTERS** button to open the Configure Registers dialog box. See Figure 14-17.

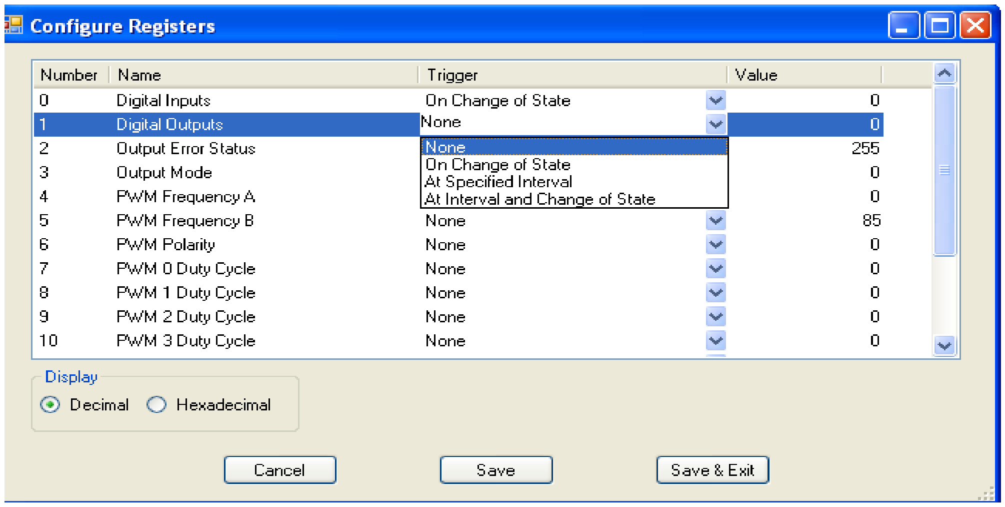


| Number | Name | Trigger | Value |
|---|---|---|---|
| 0 | Digital Inputs | On Change of State | 0 |
| 1 | Digital Outputs | None | 0 |
| 2 | Output Error Status | On Change of State | 255 |
| 3 | Output Mode | None | 0 |
| 4 | PWM Frequency A | None | 0 |
| 5 | PWM Frequency B | None | 85 |
| 6 | PWM Polarity | None | 0 |
| 7 | PWM 0 Duty Cycle | None | 0 |
| 8 | PWM 1 Duty Cycle | None | 0 |
| 9 | PWM 2 Duty Cycle | None | 0 |
| 10 | PWM 3 Duty Cycle | None | 0 |

Display: [✓ Decimal] Hexadecimal

Figure 14-17

To change the Trigger for any register, select the register and click the down arrow in the trigger column for that register. This will open a small list of trigger options. Each register maintains its own individual trigger setting. See Figure 14-18.



| Number | Name | Trigger | Value |
|---|---|---|---|
| 0 | Digital Inputs | On Change of State | 0 |
| 1 | Digital Outputs | None | 0 |
| 2 | Output Error Status | None | 255 |
| 3 | Output Mode | On Change of State | 0 |
| 4 | PWM Frequency A | At Specified Interval | 0 |
| 5 | PWM Frequency B | None | 85 |
| 6 | PWM Polarity | None | 0 |
| 7 | PWM 0 Duty Cycle | None | 0 |
| 8 | PWM 1 Duty Cycle | None | 0 |
| 9 | PWM 2 Duty Cycle | None | 0 |
| 10 | PWM 3 Duty Cycle | None | 0 |

Trigger drop-down list: None, On Change of State, At Specified Interval, At Interval and Change of State

Display: [✓ Decimal] Hexadecimal

**Figure 14-18**

When each of the registers of the node have been configured, click the **SAVE & EXIT** button to save changes and close the Configure Registers dialog box and return to the Node Configuration dialog.

In addition to changing the trigger, from this dialog, the Value for each register can be changed (providing the register is writable). The numbers can be represented in decimal or in hex. Figure 14.16 is set to display in decimal. As an example, register number 1 (Digital Outputs) will directly control the outputs on the node. By changing the Value, the outputs can be set to be on or off.

The values that may be entered are decimals that represent the binary bits that correspond to each individual output.

| Decimal Number Value | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| Corresponding Real World Output | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | All Off |

Examples: If Decimal Number Value = 128, then Real World Output 8 is ON.
If Decimal Number Value = 8, then Real World Output 4 is ON.
If Decimal Number Value = 40, then Real World Outputs 4 and 6 are both ON.

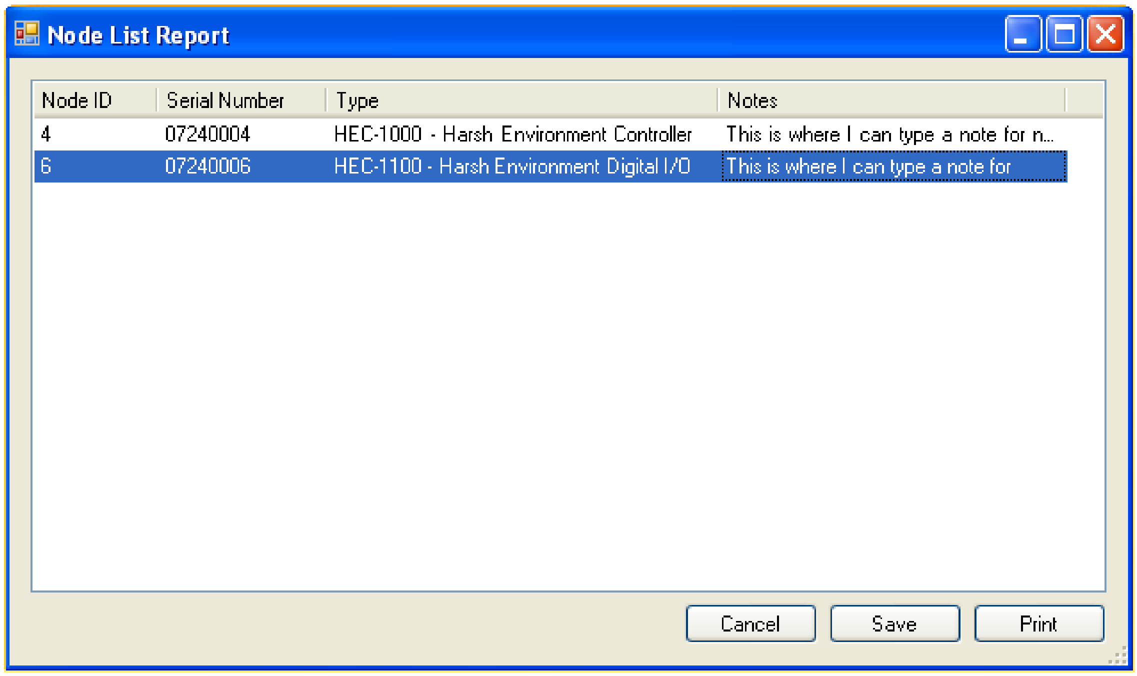
Each non-controller node has unique register assignments. Refer to the actual product manual for details regarding register assignments.

## OptiCAN Node List Notes

Notes can be added to the list of nodes to help with documentation and service later.  This is accessed from the OptiCAN Configuration Tool. To access this feature, in the Divelbiss OptiCAN Configuration Tool window, use the ***Reports Menu*** and select *Node List*.

The Node List Report window will open. Place the cursor under the Note Heading next to the node of choice. Simply type in the notes for that node.  See Figure 14-19.

The node list and notes may be saved and printed for future reference.



| Node ID | Serial Number | Type | Notes |
|---|---|---|---|
| 4 | 07240004 | HEC-1000 - Harsh Environment Controller | This is where I can type a note for n... |
| 6 | 07240006 | HEC-1100 - Harsh Environment Digital I/O | This is where I can type a note for |

**Figure 14-19**

## J1939 Networking / NMEA 2000

J1939 is a standard maintained by the Society of Automotive Engineers (SAE) that defines how information is transferred across a network to allow devices such as engine ECUs to communicate information such as engine speed, engine RPM, etc. to other ECUs or devices. J1939 is essentially a software specification / protocol that operates on a CAN network. NMEA 2000 is based on J1939 with additional special message requirements. As NMEA 2000 is based on J1939, many of the menus and configuration items are generally the same and interchangeable except for noted differences.

J1939 uses CAN 2.0B as its structure. It is typically used on commercial vehicles such as tractor trailers and construction equipment.

This manual section is provides the basics for installing and configuring J1939 support in EZ LADDER Toolkit, using the function blocks and additional features available in EZ LADDER for J1939. It is not intended as a complete J1939 reference document. As such, prior knowledge of J1939 is recommended.

EZ LADDER Toolkit support for J1939 allow for receive and transmitting data on the J1939 bus including items such as PGN request, BAM (global) and CM (specific destination). EZ LADDER also supports J1939 bus address claim.

### J1939 PGN Overview

J1939 communication is based on the Parameter Group Number, otherwise known as the PGN. PGNs are defined based on a compilation of information. For example, PGN 61444 is identified as *Electronic Engine Controller 1*. Items transmitted with this PGN designation would be engine speed, driver's demand torque, etc. PGNs are typically transmitted by ECUs (and other devices) at a set rate. By reading appropriate PGNs, controllers and targets can use the data as setpoints and values in the ladder diagram program. Additionally, EZ LADDER Toolkit programs can transmit data using J1939 (P-Series targets only).

Most PGNs are eight bytes in length, but can be a different length. If data isn't available (typically if the PGN is not supported), the data will return a 0xFF. If a 0xFE is returned, this signifies an error.

### J1939 SPN Overview

When PGNs are received, they typically include multiple data points (transmitted as part of the PGN). For example, PGN 61444 would be *Electronic Engine controller 1*. When PGN 61444 is received, it contains multiple data points and these points are stored in the data based on byte number. In EZ LADDER Toolkit, these parameters are accessed by their Suspect Parameter Numbers (SPNs).

### Installing J1939 in EZ LADDER Toolkit

The information contained in this manual regarding J1939 is specifically for the P-Series PLC on a Chip based targets only. For M-Series based targets, refer to the **M-Series EZ LADDER Toolkit Manual**.

When installing the J1939 support in the project, many configuration settings and parameters (including optional items) must be completed. It is recommended that you have a complete picture of how you want to implement J1939 and use it.

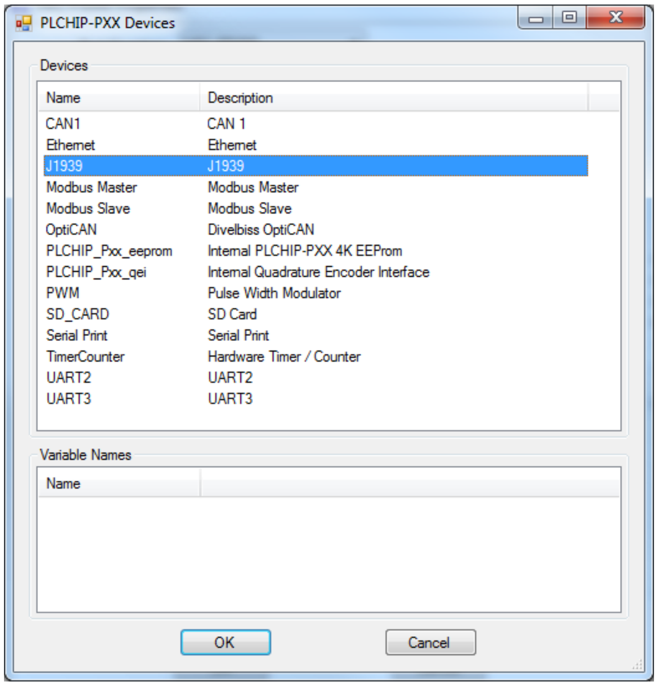
J1939 is installed on a per project (program) basis. Once the J1939 settings are configured, they are stored in the actual ladder diagram project. Actual menus steps to reach the J1939 configuration may vary based on the actual controller used, but the configuration itself is always the same. Divelbiss standard controllers based on P-Series PLC on a Chip (HEC-P5000, VB-2000, etc) are configured based on the part number. For details on specific targets, please see **Chapter 23 - Hardware Targets**.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If J1939 were installed, it would be listed in the *Devices pane* under the Device section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open.

A CAN port must be installed prior to installing SAE J1939/NMEA 2000. See **Installing CAN Network Ports** earlier in this chapter.

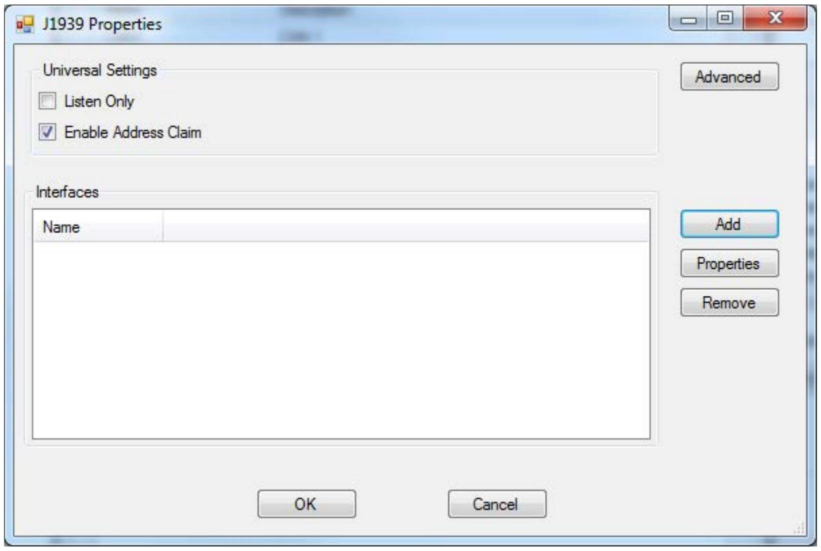
All the available devices and features for the target are shown in the Devices section. Scroll down and find the J1939. Figure 14-20 shows the Target's Devices window.



| Name | Description |
|---|---|
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| J1939 | J1939 |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |
| UART3 | UART3 |

**Figure 14-20**

Select the **J1939** and click **OK**. The *J1939 Properties* dialog will open. This dialog is used to specify which CAN port is to be used as well as the J1939 Universal Settings. The configuration here will decide how J1939 is implemented. Refer to Figure 14-21.



Listen Only [ ]
Enable Address Claim [✓]

**Figure 14-21**

**Listen Only checkbox**
When this box is checked, J1939 is configured to receive J1939. The controller with this EZ LADDER program will receive data, but cannot transmit on the J1939 bus.

**Enable Address Claim checkbox**
When this box is checked, J1939 is configured to receive and transmit J1939 broadcast data. The controller with this EZ LADDER program will attempt to claim an address on the J1939 bus. With the address claimed, it not only receives data, but can transmit on the J1939 bus.

You should select between Listen Only mode and Enable Address Claim mode. No check boxes are required to receive J1939, but per the J1939 specification, to transmit the unit must claim an address on the network.

If you are only needing to receive J1939 broadcast data (no transmit), it is recommended you select **Listen Only**.

The **ADVANCED** button accesses advanced configuration features such as receive and transmit buffer sizes, PGN and BAM connections. Changing these parameters would be required if larger numbers of simultaneous PGNs or lager buffers were required. This information will be discussed in detail later.

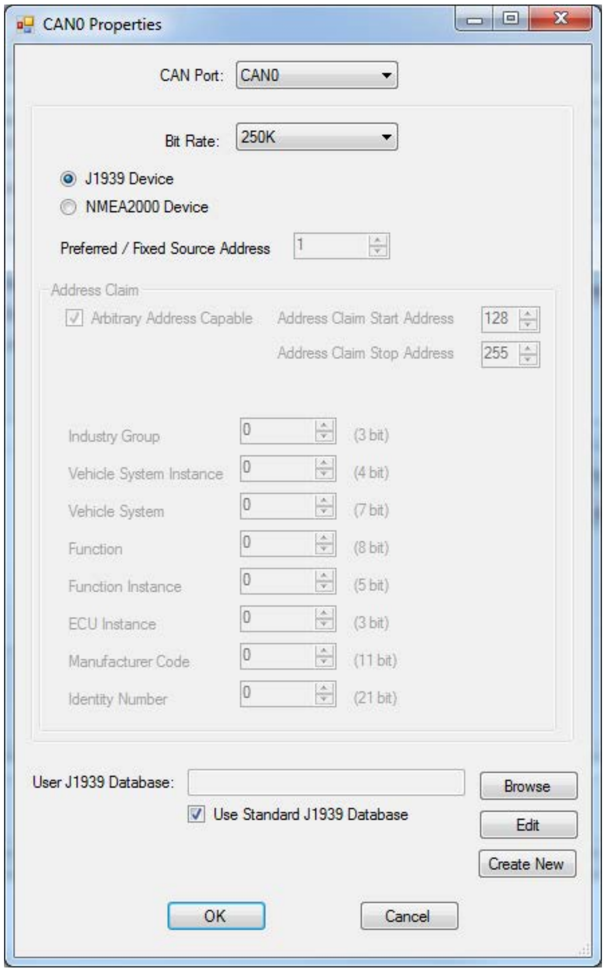
Once the Universal Setting (Listen Only or Enable Address Claim) has been configured, click the **ADD** button. The J1939 Properties dialog will open. Refer to Figure 14-22.

Using the provided *CAN Port* drop down box, select the CAN port to use for the J1939 communication. This should be the CAN port connected to the J1939 network.

With the CAN port selected, other items then become available to configure. The available configuration items is dependent upon the mode selected previously (listen or claim address). Figure 14-22 illustrates the dialogs configured each way.

In listen mode, the amount of configuration items is reduced significantly. Configure the items as required for the specific application requirements for interfacing to J1939.

**Listen Only Mode**


CAN0 Properties

CAN Port: CAN0

Bit Rate: 250K

● J1939 Device

○ NMEA2000 Device

Preferred / Fixed Source Address: 1

Address Claim

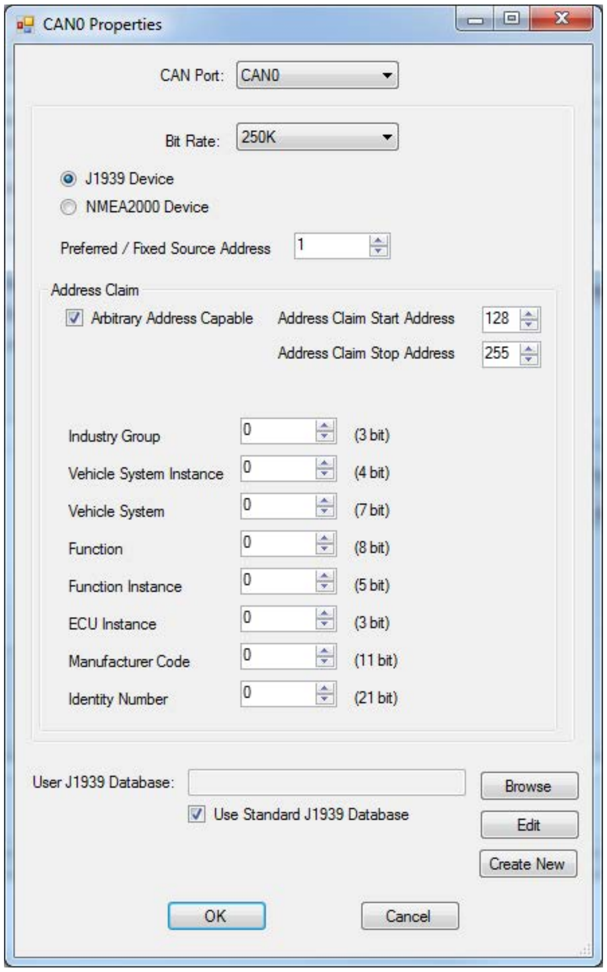
✓ Arbitrary Address Capable

| | | |
|---|---|---|
| Address Claim Start Address | 128 | |
| Address Claim Stop Address | 255 | |
| Industry Group | 0 | (3 bit) |
| Vehicle System Instance | 0 | (4 bit) |
| Vehicle System | 0 | (7 bit) |
| Function | 0 | (8 bit) |
| Function Instance | 0 | (5 bit) |
| ECU Instance | 0 | (3 bit) |
| Manufacturer Code | 0 | (11 bit) |
| Identity Number | 0 | (21 bit) |

User J1939 Database:

✓ Use Standard J1939 Database

**Claim Address Mode**


CAN0 Properties

CAN Port: CAN0

Bit Rate: 250K

● J1939 Device

○ NMEA2000 Device

Preferred / Fixed Source Address: 1

Address Claim

✓ Arbitrary Address Capable

| | | |
|---|---|---|
| Address Claim Start Address | 128 | |
| Address Claim Stop Address | 255 | |
| Industry Group | 0 | (3 bit) |
| Vehicle System Instance | 0 | (4 bit) |
| Vehicle System | 0 | (7 bit) |
| Function | 0 | (8 bit) |
| Function Instance | 0 | (5 bit) |
| ECU Instance | 0 | (3 bit) |
| Manufacturer Code | 0 | (11 bit) |
| Identity Number | 0 | (21 bit) |

User J1939 Database:

✓ Use Standard J1939 Database

**Figure 14-22**

## Configuration Items

The items that are to be configured must be configured specifically for your application requirements for proper operation.

| | |
|---|---|
| **Bit Rate:** | Drop down box. Select 250K or 500K. Standard bus uses 250K. |
| **J1939 Device/NMEA 2000 Device:** | Select between J1939 and NMEA 2000 network. Choose the appropriate network. |
| **Preferred/Fixed Source Address:** | Number from 1 to 100. This is the first address that the controller will try and use on the J1939 network. This box works with the Arbitrary Address Capable box and addresses. If the Arbitrary Address Capable box is checked, then the Preferred/ Fixed Address is tried first. If the address is not available on the network, then the controller will arbitrarily try additional addresses based on the limits set. If the Arbitrary Address Capable box is not checked, then only this one address will be attemped, even if it is not available. This box is not available in listen mode. |

| | |
|---|---|
| **Arbitrary Address Capable:** | If this box is checked, then the Preferred/Fixed Address is tried first. If the address is not available on the network, then the controller will arbitrarily try additional addresses based on the Address Claim Start/Stop addresses set. This box is not available in listen mode. |
| **Address Claim Start Address:** | If arbitrary addressing is required, this is the first arbitrary address that will be tried. |
| **Address Claim Stop Address:** | If arbitrary addressing is required, this is the last arbitrary address that will be tried. |
| **Industry Group, Vehicle System Instance, Function, Function Function Instance, ECU Instance, Manufacturer Code, Identity Number** | These items are specific items that may be used on the J1939 bus for communication and arbitration. These items are optional and would be provided by the actual J1939 bus or equipment manufacturer. Note: if NMEA 2000 is selected instead of J1939 above, then some of these names will change as required by NMEA 2000. |
| **User J1939 Database:** | This location box and check box for Use Standard J1939 Database identifies the J1939 database of PGNs and SPNs to be used on this project. The Use Standard J1939 Database checkbox forces the use of the EZ LADDER Toolkit standard PGN and SPN database. |

EZ LADDER Toolkit supports the ability to add new PGNs and SPNs that are not currently in the standard database as needed. This topic will be covered in a later section.

With all the items configured, click the **OK** button to close dialog and save the settings. Click **OK** to close and save the J1939 properties dialog.

Up to two J1939 networks may be accessed (one for each CAN port, up to 2 maximum) by repeating the same steps for the additional J1939 network with a different CAN port.

Click the **OK** button to close the target's properties. Click **OK** to close and save the Project Settings. The J1939 is now installed and configured and is ready to be used in the ladder diagram program.

## Standard J1939 Database

Included as part of EZ LADDER Toolkit is the Divelbiss Standard J1939 database. This database has been preloaded with common J1939 PGNs (and SPNs). During the configuration previously shown, the database that is used for the storing J1939 PGNs is identified. The Standard J1939 database cannot be modified (added to or deleted from). If additional PGNs are required that are not included in the Standard J1939 database, then the User J1939 database must be identified and used during the configuration. To use the Standard J1939 database, ensure the check box *Use Standard J1939 Database* is checked. If a user J1939 database is to be used, then uncheck the box.

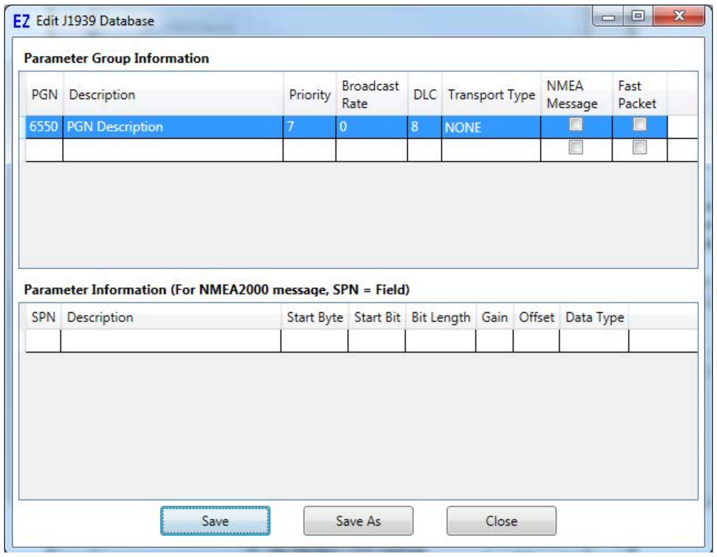
## User J1939 Database

If the Standard J1939 database does not include all the PGNs that are required, the option is provided to use a *User J1939 database*. This custom database may be configured and used to store the PGN list for your projects.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If J1939 were installed, it would be listed in the *Devices pane* under the Device section. Select the J1939 (under Network) and click the **PROPERTIES** button. The J1939 Properties dialog will open. Select the CAN port for the J1939 and click the **PROPERTIES** button. The CAN port Properties window will open.

A built-in editor is included in EZ LADDER Toolkit for creating and editing user J1939 databases. Refer to Figure 14-23. To create a new user J1939 database, click the **CREATE NEW** button. The Edit J1939 database window will open. Refer to Figure 14-23.



Parameter Group Information

| PGN | Description | Priority | Broadcast Rate | DLC | Transport Type | NMEA Message | Fast Packet |
|---|---|---|---|---|---|---|---|
| 6550 | PGN Description | 7 | 0 | 8 | NONE | ☐ | ☐ |
| | | | | | | ☐ | ☐ |

Parameter Information (For NMEA2000 message, SPN = Field)

| SPN | Description | Start Byte | Start Bit | Bit Length | Gain | Offset | Data Type |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

Figure 14-23

Click into the fields provided and add the PGN (PGN number), PGN Description, Priority Broadcast Rate DLS, and Transport Type for each PGN. If this for NMEA 2000, click the NMEA Message checkbox and if it is a Fast Packet for NMEA 2000, click the Fast Packet checkbox.

For each PGN added in the top pane, SPNs may be added in the bottom pane. With the PGN selected in the top PGN pane (Parameter Group Information), add SPNs in the SPN pane at the bottom (Parameter Information). The data for the PGN/SPN is provided by the manufacturer of the device or the J1939/NMEA 2000 specification. Enter the data for SPN, Description, Start Byte, Start Bit, Bit Length, Gain, Offset and Data Type. For NMEA 2000, the SPN is the Field.

Use the **SAVE** and **SAVEAS** buttons to set the filename of the user J1939 database and save (update) as new PGNs are added. The **CLOSE** button is used to close the editor.

Since the J1939 database cannot be edited, if you require PGNs from the Standard J1939 database and additional not in the database, you will need to add **ALL** the PGNs that you require to the User J1939 database.

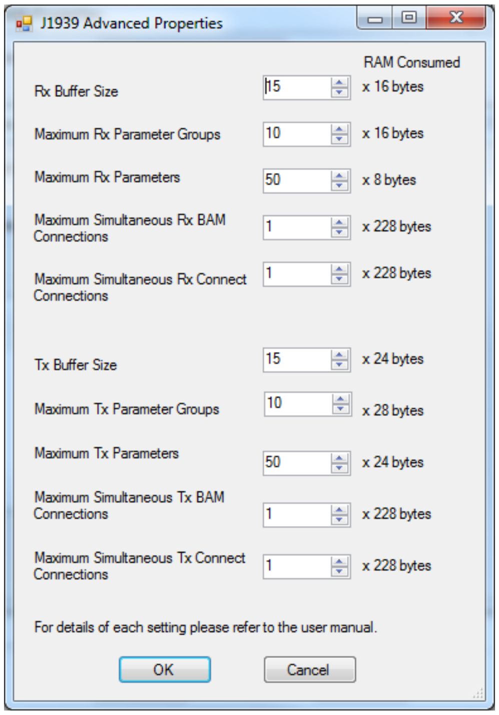
If you need some additional PGNs not included in the standard database, you may also copy the standard database to a new file, rename it, then edit it adding additional PGNs needed for your project.

Refer to Figure 14-22 To select your User J1939 database, uncheck the Use Standard J1939 Database and click the **BROWSE** button. Browse to the location your J1939 database is located and click **OPEN**. The location of your User J1939 database should now be locate in the box. Ensure all the J1939 settings are correct and click **OK** with the User J1939 Database selected to close the CAN port Properties window. Exit out of all the remaining project settings windows. Be sure to save your ladder diagram project.

## Advanced J1939 Configuration

EZ LADDER Toolkit provides advanced configuration options for J1939. These configuration items may be adjusted if the default configuration is not sufficient for the application.

With the J1939 Properties window open (See Figure 14-20), click the **ADVANCED** button. The J1939 Advanced Properties window will open. The currently shown settings are the default transmit and receive configuration items (assuming that no advanced changes were made previously). See Figure 14-24.


J1939 Advanced Properties

| | | RAM Consumed |
|---|---|---|
| Rx Buffer Size | 15 | x 16 bytes |
| Maximum Rx Parameter Groups | 10 | x 16 bytes |
| Maximum Rx Parameters | 50 | x 8 bytes |
| Maximum Simultaneous Rx BAM Connections | 1 | x 228 bytes |
| Maximum Simultaneous Rx Connect Connections | 1 | x 228 bytes |
| Tx Buffer Size | 15 | x 24 bytes |
| Maximum Tx Parameter Groups | 10 | x 28 bytes |
| Maximum Tx Parameters | 50 | x 24 bytes |
| Maximum Simultaneous Tx BAM Connections | 1 | x 228 bytes |
| Maximum Simultaneous Tx Connect Connections | 1 | x 228 bytes |

For details of each setting please refer to the user manual.

**Figure 14-24**

From this window, items such as transmit and receive buffer size, number of PGNs, etc. may be adjusted.

| | |
|---|---|
| **Rx Buffer Size:** | Size of Rx buffer to hold data before it is processed. (Should not typically need adjustment). |

| | |
|---|---|
| **Maximum Rx Parameter Groups:** | Total number of Rx PGNs that can be setup to receive simultaneously. |
| **Maximum Rx Parameters:** | Total number of Rx Parameters that can be setup to receive simultaneously. |
| **Maximum Simultaneous Rx BAM Connections:** | Maximum number of Rx BAM messages that can be received simultaneously. |
| **Maximum Simultaneous Rx Connect Connections:** | Maximum number of Rx Multipacket Connection sessions that can be open at one time. |
| **Tx Buffer Size:** | Size of Tx buffer holding data to be sent immediately. (Should not typically need adjustment). |
| **Maximum Tx Parameter Groups:** | Maximum number of PGNs that can be in the cyclic send queue at a time. |
| **Maximum Tx Parameters:** | Maximum number of Parameters that can be in the cyclic send queue at a time. |
| **Maximum Simultaneous Tx BAM Connections:** | Maximum number of Tx BAM messages that can be setup at one time. |
| **Maximum Simultaneous Tx Connect Connections:** | Maximum number of Tx Multipacket Connection sessions that can be open at one time. |

These advanced parameters should only be adjusted if necessary for the application. Adjusting these numbers affects the amount of RAM used and available for the entire ladder diagram project during Compilation. Caution must be taken to not use excessive RAM for J1939 unnecessarily as it is possible to run out of RAM memory.

## Receiving Data with J1939

With the J1939 network configured in the Project Settings of EZ LADDER Toolkit, data may be received in the ladder diagram. To receive data using J1939 (NMEA 2000), the J1939_RX_PGN Function block is used. This function block is automatically added to the functions drop down list when J1939/NMEA 2000 is installed in the project settings.

To receive data, use the Functions drop down box on the toolbar and select **J1939_RX_PGN**. Click the **INSERT FUNCTION** button and then click in the ladder diagram workspace where you want to insert the J1939_RX_PGN function block.

The J1939 RX Properties dialog box will open. This box is used to configure this specific instance of the J1939_RX_PGN function used in the program. Refer to Figure 14-25.

The J1939 RX Properties must be configured for each insert of a J1939_RX_PGN function block as each insert is a unique instance of the function block.

Using the drop down menu, select the CAN port for the J1939/NMEA 2000 network. Along the left side of the dialog, below the CAN port drop down, is a pane with a list of PGNs (PGNs in the configured database). By selecting a PGN, the SPN pane below populates with the supported SPNs for the highlighted PGN. As different PGNs are selected (highlighted) in the PGN pane, the PGN settings to the right change as do the SPN pane.

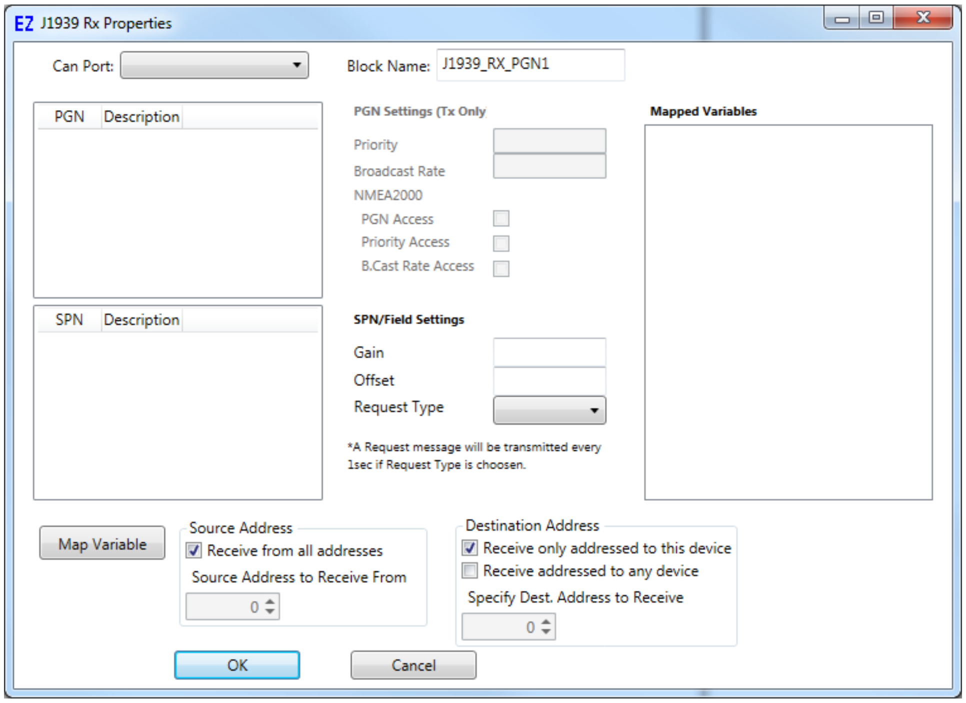


**Figure 14-25**

Select the PGN to receive data from. The SPN pane will update. Select the desired SPN in the SPN pane. The SPN/Field setting will update with information based on the J1939/User database.

The Gain, Offset and Request Type are stored in the database but may be overridden for this instance (of the J1939_RX_PGN) function block.

The Source Address area determines what J1939 address are may be received from for the selected PGN. If the ***Receive from all addresses*** is checked, then any device on the network that broadcasts this PGN will be received from. If only specific a specific J1939 network address should be received from, uncheck the ***Receive from all addresses*** checkbox and use the ***Source Address to Receive From*** box to set the J1939 network address.

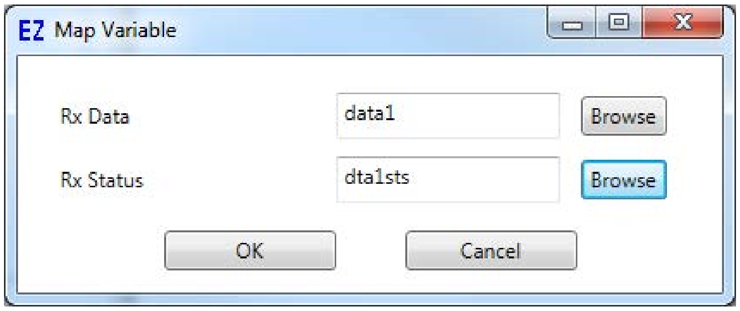
The source of the J1939 broadcasts received may be configured as Receive from all addresses or may be limited to a specific address based on the Source Address area settings.

In addition to controlling which Source Addresses are received, the receive function block may be configured based on a Destination address in the Destination Address section. Using these settings, the function block will receive all global broadcasts of a PGN (Receive addressed to any device), broadcasts to this specific controller or device (Receive only addressed to this device) or it may listen in on a broadcast to a specific address broadcast (Specify Dest. Address to Receive). These configuration items are set by check boxes and an Address box in the Destination Address area.

The function block instance may be configured to receive all global broadcasts to this PGN, only broadcasts to this specific address or to any specific address based on the settings in the Destination Address area.

The next step is to identify where the SPN data that is received will be stored. All J1939 data is stored in variables. To set the variable(s) used, click the **MAP VARIABLE** button. The Map Variable dialog will open.

Using the **BROWSE** buttons, select the variable to store the received data to (Rx Data) and optionally, select the variable to store the received data status (ie valid, not seen, etc). Click **OK** when the variables have been set. Refer to Figure 14-26.



Rx Data: data1

Rx Status: dta1sts

**Figure 14-26**

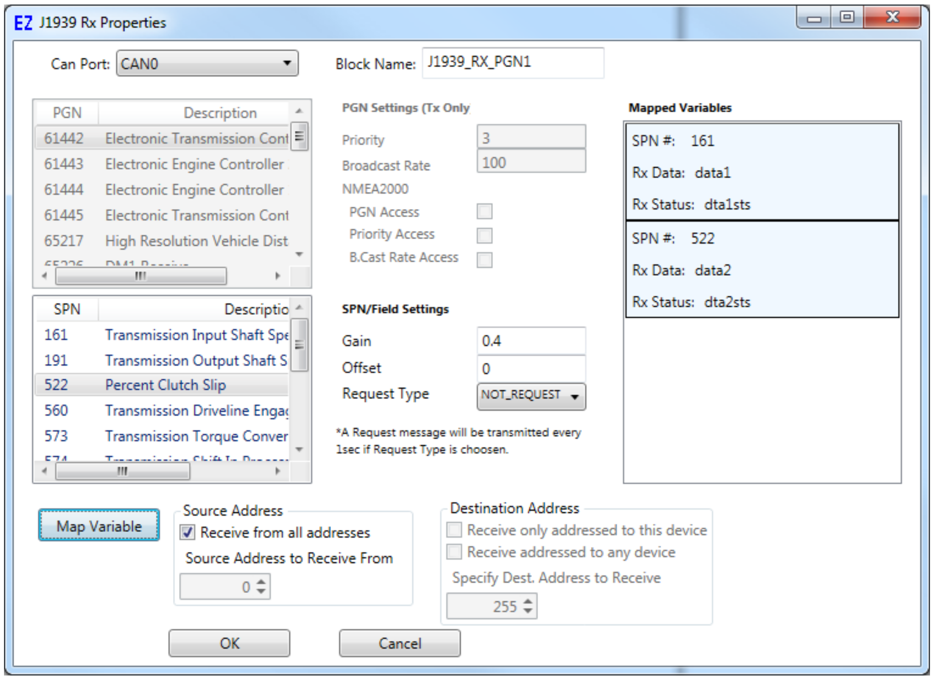
Repeat these steps for any SPN of the selected PGN you want to receive data from in this instance of the J1939_RX_PGN function block. Figure 14-27 shows two SPNs selected and mapped to variables for PGN 61442.

Multiple SPNs may be mapped to a variables in one J1939_RX_PGN function block instance providing that all the SPNs are part of the same PGN (selected PGN).

Click **OK** when all the SPNs required have been mapped to variables. The function block is now placed in the ladder diagram.

The EN (enable) should be tied to the left power rail or using a contact to control when the specific J1939_ RX_PGN function block is enabled and receives data. The Q output goes true when the data is received for one ladder diagram scan and then goes false on the next ladder diagram scan.

The ER is the error output. It must be connected to an integer variable. This variable will store the status of the last receive (0= Valid Data or No Error, -1 = PGN Not Seen, -2 = PGN Failed Update or No data received in last 5 seconds).



Can Port: CAN0

Block Name: J1939_RX_PGN1

| PGN | Description |
|---|---|
| 61442 | Electronic Transmission Cont |
| 61443 | Electronic Engine Controller |
| 61444 | Electronic Engine Controller |
| 61445 | Electronic Transmission Cont |
| 65217 | High Resolution Vehicle Dist |

| SPN | Descriptio |
|---|---|
| 161 | Transmission Input Shaft Spe |
| 191 | Transmission Output Shaft S |
| 522 | Percent Clutch Slip |
| 560 | Transmission Driveline Engag |
| 573 | Transmission Torque Conver |

PGN Settings (Tx Only

Priority: 3

Broadcast Rate: 100

NMEA2000

PGN Access ☐

Priority Access ☐

B.Cast Rate Access ☐

SPN/Field Settings

Gain: 0.4

Offset: 0

Request Type: NOT_REQUEST

*A Request message will be transmitted every 1sec if Request Type is choosen.

Mapped Variables

SPN #: 161
Rx Data: data1
Rx Status: dta1sts

SPN #: 522
Rx Data: data2
Rx Status: dta2sts

Source Address

☑ Receive from all addresses

Source Address to Receive From: 0

Destination Address

☐ Receive only addressed to this device

☐ Receive addressed to any device

Specify Dest. Address to Receive: 255

**Figure 14-27**

The SA is the Source Address output. It must be connected to an integer variable. This variable will store the Source Address of the last receive (which address this PGN was received from).

The DA is the Destination Address output. It must be connected to an integer variable. This variable will store the Destination Address of the last receive if the PGN had a specific destination in the broadcast packet. If the broadcast was a global broadcast, then the DA does not apply.

Figure 14-28 illustrates a complete J1939_RX_PGN function block inserted in a ladder diagram program.

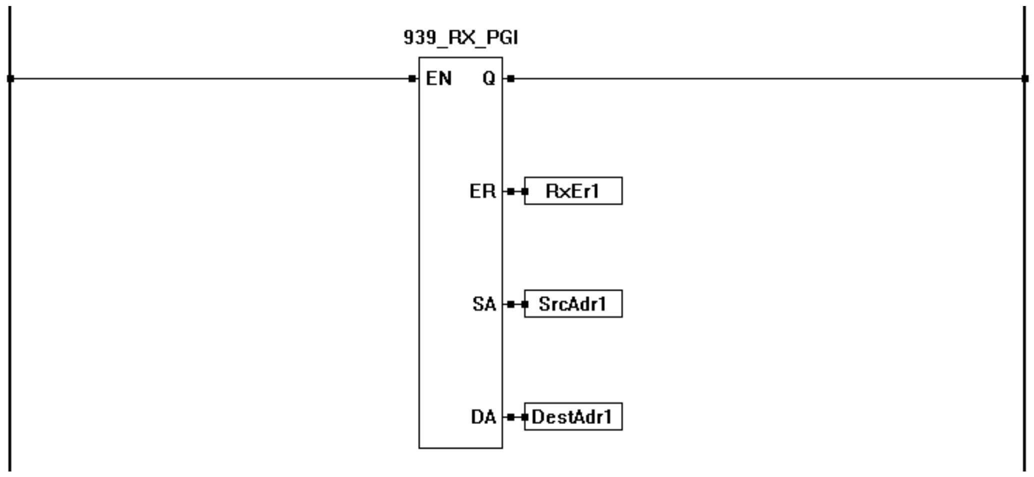


**Figure 14-28**

### SPN Status Mapped Variables

The mapped status (not data) variables for the SPNs of the function block have their own error and status reporting. The SPN error status reporting is only valid if the SPN is set as STANDARD or STATE in the J1939 Database). The SPN status is 0 = Valid, -1 = Not Seen, -2 = Data Error, -3 = Not Supported -4 = Data Reserved.

## Transmitting Data with J1939

With the J1939 network configured in the Project Settings of EZ LADDER Toolkit, data may be transmitted using the ladder diagram. To transmit data using J1939 (NMEA 2000), the J1939_TX_PGN Function block is used. This function block is automatically added to the functions drop down list when J1939/NMEA 2000 is installed in the project settings.

To transmit data, use the Functions drop down box on the toolbar and select **J1939_TX_PGN**. Click the **INSERT FUNCTION** button and then click in the ladder diagram workspace where you want to insert the J1939_TX_PGN function block.

The J1939 TX Properties dialog box will open. This box is used to configure this specific instance of the J1939_TX_PGN function used in the program. Refer to Figure 14-29.

The J1939 RX Properties must be configured for each insert of a J1939_RX_PGN function block as each insert is a unique instance of the function block.

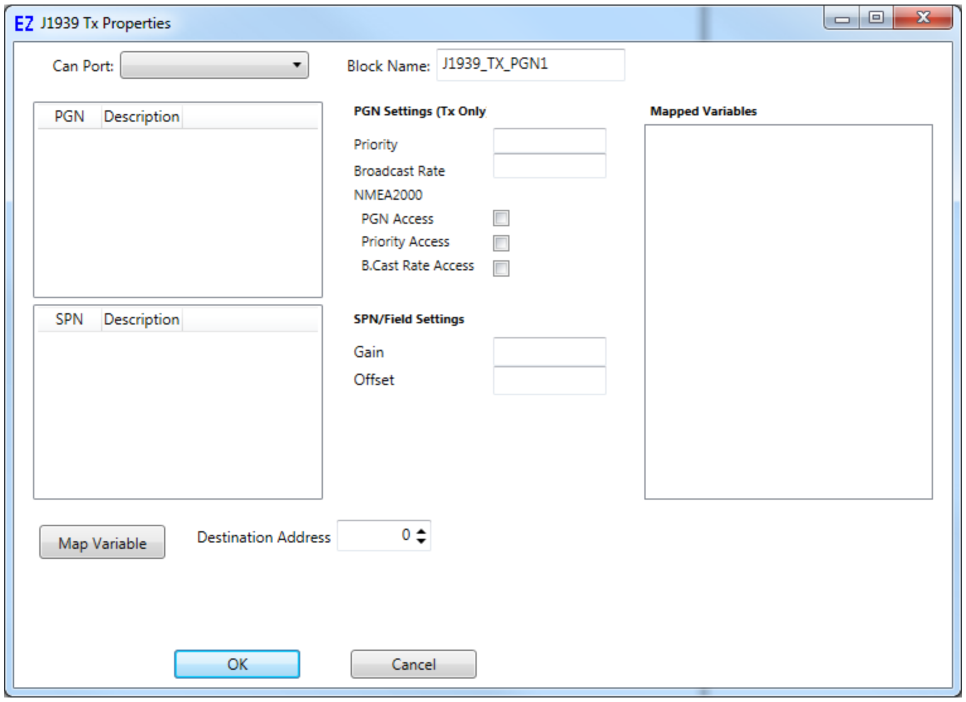


**Figure 14-29**

Using the drop down menu, select the CAN port for the J1939/NMEA 2000 network. Along the left side of the dialog, below the CAN port drop down, is a pane with a list of PGNs (PGNs in the configured database). By selecting a PGN, the SPN pane below populates with the supported SPNs for the highlighted PGN. As different PGNs are selected (highlighted) in the PGN pane, the PGN settings to the right change as do the SPN pane.

Select the PGN to transmit data as. The SPN pane will update. Select the desired SPN in the SPN pane. The SPN/Field setting will update with information based on the J1939/User database.

The Destination Address area determines the type of J1939 broadcast. If set to 255, then the broadcast is global and will not have a specific destination address. If the Destination Address is set to a specific number, then the broadcast is sent with a that specific destination address.

Destination address may not be settable to a specific address. Specific address or global is also dependent upon the PGN/SPN selected and the database settings.

The PGN Settings are based on the settings from the J1939/User database. These values may be adjusted for this instance of the function block. The adjustable values include the Priority and Broadcast Rate. Additionally, checkboxes are provided for NMEA 2000 optional settings (PGN Access, Priority Access and B.Cast Rate Access). Refer to the NMEA 2000 specification for details on these configuration items.

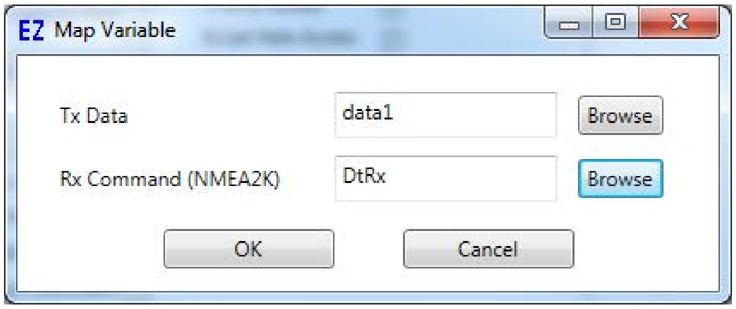
The Priority and Broadcast Rate for the PGN are stored in the database but may be overridden for this instance (of the J1939_TX_PGN) function block.

The SPN Settings are based on the settings from the J1939/User database. These values may be adjusted for this instance of the function block. The adjustable values include the Gain and Offset.

The Gain and Offset for the SPN are stored in the database but may be overridden for this instance (of the J1939_TX_PGN) function block.

The next step is to identify where the SPN data that will be broadcast will be gathered from. All J1939 broadcast data must be mapped from variables. To set the variable(s) used, click the **MAP VARIABLE** button. The Map Variable dialog will open. Using the **BROWSE** buttons, select the variable to gather the transmit data from (Tx Data). For NMEA 2000 Commanded Data, select the variable for the Rx Command (NMEA2K). Click **OK** when the variables have been set. Refer to Figure 14-30. Refer to the NMEA 2000 specification regarding Commanded Data.

Double-clicking the SPN will open the Map Variable dialog.



| | |
|---|---|
| Tx Data | data1 |
| Rx Command (NMEA2K) | DtRx |

**Figure 14-30**

Repeat these steps for any SPN of the selected PGN you want to transmit data as in this instance of the J1939_TX_PGN function block. Figure 14-31 shows one SPN selected and mapped to variables for PGN 61442.

Multiple SPNs may be mapped to a variables in one J1939_TX_PGN function block instance providing that all the SPNs are part of the same PGN (selected PGN).

Click **OK** when all the SPNs required have been mapped to variables. The function block is now placed in the ladder diagram.

The EN (enable) should be tied to the left power rail or using a contact to control when the specific J1939_ TX_PGN function block is enabled and transmits data. The Q output is true when the function block is enabled.

When the Enable input is true, the function block is active and the PGN/SPNs will broadcast based on the rates set in the database or the overridden values when the function block was placed. If the Enable is false, the function block is disabled and the PGN/SPNs will not transmit (broadcast).

Figure Figure 14-32 illustrates a complete J1939_TX_PGN function block inserted in a ladder diagram program.

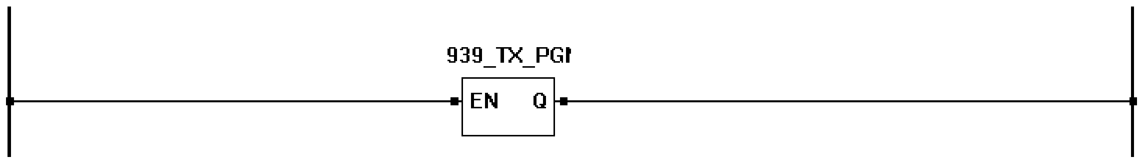
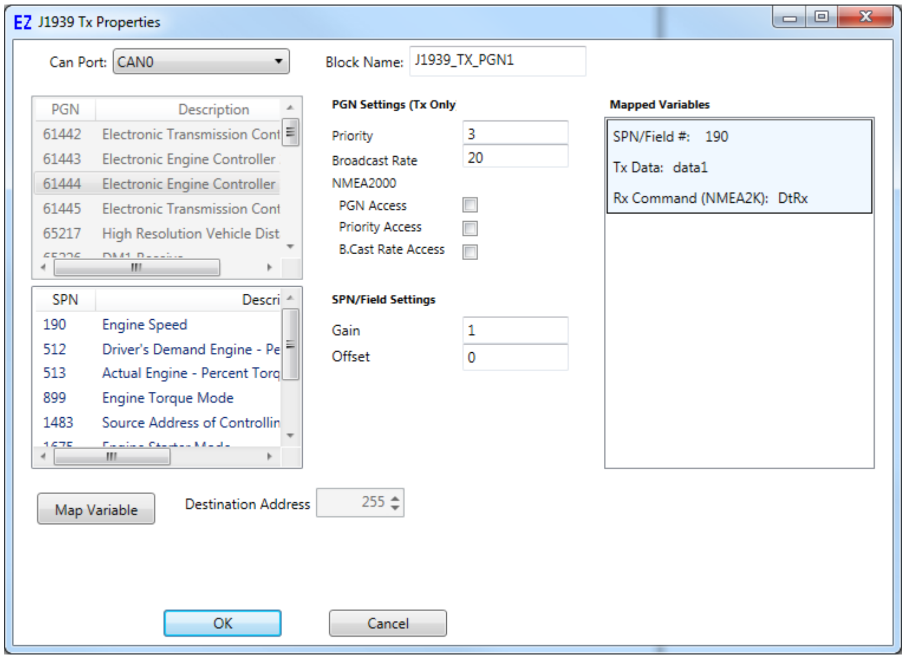


**Figure 14-32**


J1939 Tx Properties
Can Port: CAN0
Block Name: J1939_TX_PGN1
PGN
Description
61442 Electronic Transmission Cont
61443 Electronic Engine Controller
61444 Electronic Engine Controller
61445 Electronic Transmission Cont
65217 High Resolution Vehicle Dist
SPN
Descri
190 Engine Speed
512 Driver's Demand Engine - Pe
513 Actual Engine - Percent Torq
899 Engine Torque Mode
1483 Source Address of Controllin
PGN Settings (Tx Only
Priority 3
Broadcast Rate 20
NMEA2000
PGN Access
Priority Access
B.Cast Rate Access
SPN/Field Settings
Gain 1
Offset 0
Mapped Variables
SPN/Field #: 190
Tx Data: data1
Rx Command (NMEA2K): DtRx
Map Variable
Destination Address 255
OK
Cancel
**Figure 14-31**

## PGN Request

J1939/NMEA 2000 PGN request is supported in EZ LADDER Toolkit. If a specific PGN is required and it is not broadcast on a schedule or if the PGN data is required more often than the device is broadcasting it (if the broadcast rate of the PGN is 10 seconds, but you need the data every 4 seconds), the PGN Request may be used. The PGN Request is exactly as it's name implies, it re quests the device to broadcast the PGN.

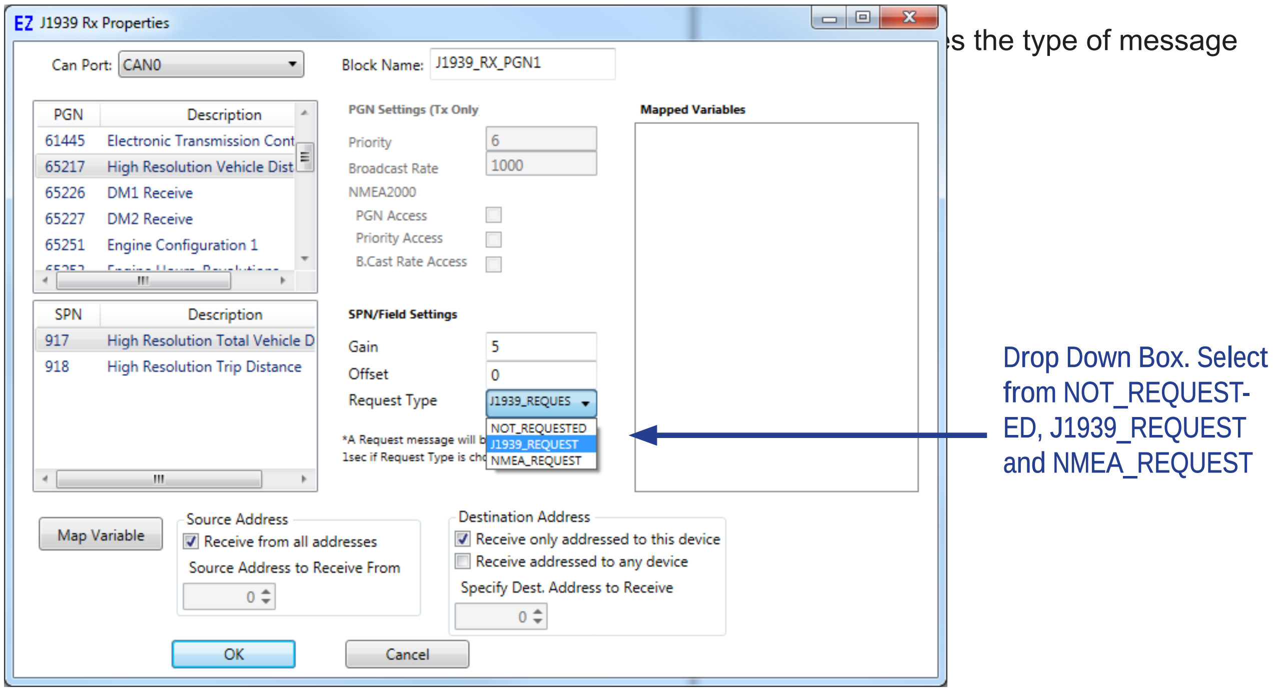
This feature is accessed in the J1939_RX_PGN function block. With a PGN and SPN selected, a drop-down box is available in the SPN/Field Settings area of the Properties window. Refer to Figure 14-33. Mapped variables to receive the data also required.

When the Request Type is set to J1939_REQUEST, the controller will request the device broadcast the PGN. The request is sent every 1 second. The NMEA_Request is the NMEA 2000 version of the PGN request.

## BAM

In the event that data needs to be transmitted that is larger than the standard 16 bytes for the normal J1939 PGN/SPN communication, J1939 also provides the BAM message. EZ LADDER Toolkit support the BAM message. Generally, the BAM message will take larger amounts of data that need transmitted an break it into smaller sizes. The J1939 BAM message will transmit the information regarding the message size and number of packets that are required and then transmit the packets of data. The data is transmitted with a 50 millisecond transmit time then waits 50 milliseconds to send the next packet. This repeats until all the data of the BAM message has been transmitted. The BAM message is a global transmit.

s the type of message


Drop Down Box. Select from NOT_REQUEST-ED, J1939_REQUEST and NMEA_REQUEST

J1939 Rx Properties

Can Port: CAN0

Block Name: J1939_RX_PGN1

| PGN | Description |
|---|---|
| 61445 | Electronic Transmission Cont |
| 65217 | High Resolution Vehicle Dist |
| 65226 | DM1 Receive |
| 65227 | DM2 Receive |
| 65251 | Engine Configuration 1 |

| SPN | Description |
|---|---|
| 917 | High Resolution Total Vehicle D |
| 918 | High Resolution Trip Distance |

PGN Settings (Tx Only

Priority: 6

Broadcast Rate: 1000

NMEA2000

PGN Access ☐

Priority Access ☐

B.Cast Rate Access ☐

SPN/Field Settings

Gain: 5

Offset: 0

Request Type: J1939_REQUES

NOT_REQUESTED

J1939_REQUEST

NMEA_REQUEST

*A Request message will b

1sec if Request Type is ch

Mapped Variables

Map Variable

Source Address

☑ Receive from all addresses

Source Address to Receive From: 0

Destination Address

☑ Receive only addressed to this device

☐ Receive addressed to any device

Specify Dest. Address to Receive: 0

OK Cancel

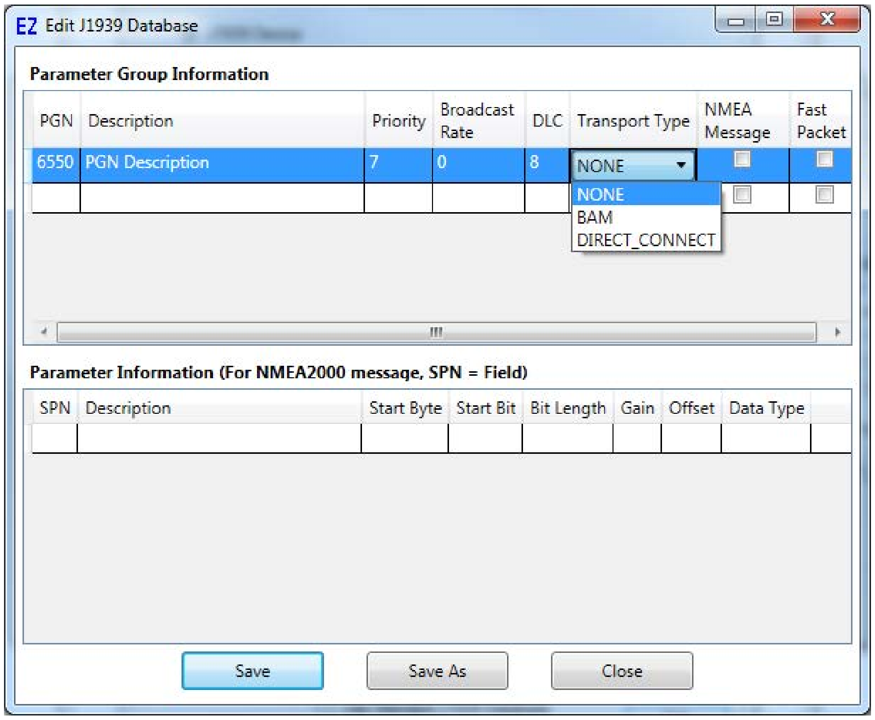
**Figure 14-33**

(None, BAM or DIRECT_CONNECT). To set a PGN as a BAM message, it must be configured in the J1939/User database. Figure 14-34 illustrates the configuration options drop-down boxes.

The controller will handle all the data manipulation required to transmit BAM messages if configured to use BAM in the database.

## DIRECT CONNECT

J1939 Direct Connect is supported in EZ LADDER Toolkit and can be selected as the Transport Type for a PGN in the J1939/User database. No additional configuration is required. Refer to the J1939 specification for more information regarding Direct Connect.


Edit J1939 Database
Parameter Group Information
Parameter Information (For NMEA2000 message, SPN = Field)
NONE
BAM
DIRECT_CONNECT
Save
Save As
Close

| PGN | Description | Priority | Broadcast Rate | DLC | Transport Type | NMEA Message | Fast Packet |
|---|---|---|---|---|---|---|---|
| 6550 | PGN Description | 7 | 0 | 8 | NONE | ☐ | ☐ |
|  |  |  |  |  |  | ☐ | ☐ |

| SPN | Description | Start Byte | Start Bit | Bit Length | Gain | Offset | Data Type |
|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |

**Figure 14-34**

## Native CAN Communications

P-Series EZ LADDER Toolkit and targets support Native CAN communications that allow transmitting and receiving CAN data frames via the CAN port.  The Native CAN communications is only available using Structured Text commands. Using these commands and custom structured text functions and function blocks, you can control the CAN bus, receive data, send data, minitor the status and set the bit rate.

Native CAN communications is only available using Structured Text functions.

The use of Native CAN communications provides a versatile method of interacting with CAN net works using 11-bit or 29-bit communications. Since the communications is at the raw frame level. a detailed understanding of CAN networking is required for programming, including building and under standing CAN frames.

The following EZ LADDER structured text built-in functions are used for Native CAN communications.

| | |
|---|---|
| **EZ_CAN_Reset** | **EZ_CAN_Status** |
| **EZ_CAN_Rx** | **EZ_CAN_Tx** |
| **EZ_CAN_SetBitRate** | |

As the Native CAN communications are all at the frame level, the programming requires the structured text to manage the communications from monitoring the CAN network status, resetting it if necessary, receiving the CAN frames and transmitting CAN frames. The CAN data is built as arrays, so the data arrays will need to be built using structured text. Each function has its own return status bit with error detection. It is up to the programming to implement the entire communications structure necessary from configuration, data in , data out and error detection using the available structured text functions.

For more details on using Structured Text and Structured Text target specific functions, refer to **Chapter 26 - Structured Text** and **Appendix B - Target Specific ST Function Reference**.

# CHAPTER 15

## SPI Devices and Support

This chapter provides basic information to understand how to install, configure and use the SPI Devices and SPI features in the EZ LADDER Toolkit.

## Chapter Contents

## SPI Bus Devices

EZ LADDER Toolkit provides built-in support for the use of several SPI devices. These supported devices are easily integrated with PLC on a Chip™ and used via EZ LADDER Toolkit variables, function blocks and Structured Text. Generally, these devices are installed and configured using the Project Settings and are not supported on all targets. These devices may be factory implemented on other hardware targets.

## Installing an SPI Bus

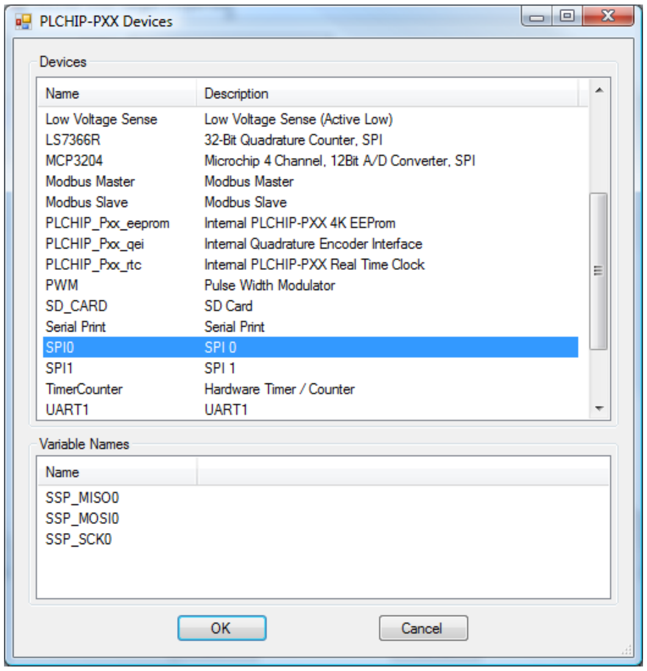
Before any of the above listed / supported SPI devices may be installed in EZ LADDER Toolkit, an SPI bus must be installed.

SPI Port availability is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI Ports are supported. SPI devices must be connected to the P-Series PLC on a Chip™ per design guidelines for proper operation.

Installing SPI or copying ladder program components with SPI into a ladder diagram for a target that does not support SPI or has factory set SPI settings will cause the hardware target to malfunction.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any SPI port were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the SPI ports (SPI0, SPI1, etc). Figure 15-1 shows the Target's Devices window.



| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART1 | UART1 |

| Name |
|---|
| SSP_MISO0 |
| SSP_MOSI0 |
| SSP_SCK0 |

**Figure 15-1**

Select **SPIx** and click **OK**. The *SPI Properties* dialog will open. See Figure 15-2. All SPI Devices require Chip Select pins, but they are typically selected / identified when the actual SPI device is installed. This dialog allows for reserving Chip Select pins specifically when using structured text only.

Each SPI Device requires the use of a Chip Select pin (GPIO) from the P-Series PLC on a Chip™. As chip select pins are reserved, they should not / cannot be used in a program as digital I/O. Designs must identify all chip select pins while mapping the digital I/O. Each device on an SPI bus required its own individual chip select pin.

While each device requires a single chip select pin. Multiple devices may be connected to an SPI port provided that SPI device is compatible and supported.

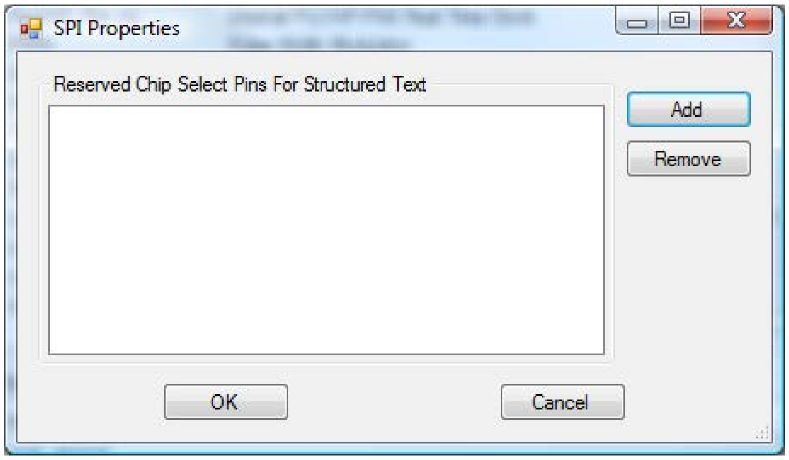
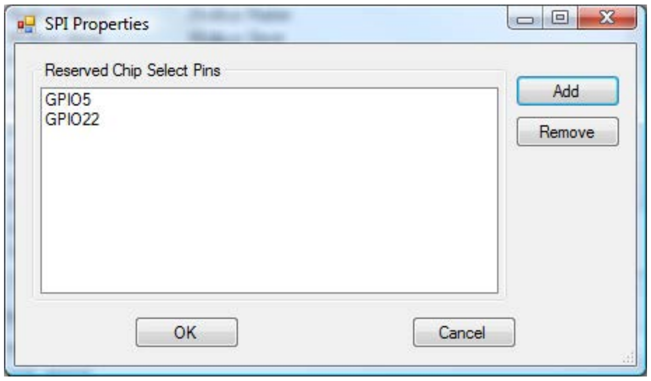


**Figure 15-2**

If using structured text, Click the **ADD** button. The SPI CS dialog will open. Using the provided drop-down CS Output menu, select the GPIO to reserve as a chip select pin for the SPI bus for structured text only. Repeat this step until all the required chip selects have been installed and are listed in the Reserved Chip Select Pins pane of the SPI Properties dialog. See Figure 15-3. If not using structured text, click **OK** to close and exit the SPI Properties dialog and return to the Target Properties window.



GPIO5
GPIO22

**Figure 15-3**

If using structured text, when all the chip selects have been reserved, click **OK**. The Target's Devices window will close and the previous target properties window will now list the SPI port as an installed device.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The SPI port is now installed and ready to be used and additional SPI devices may be installed and used in the ladder diagram.

## Installing Supported SPI Bus Devices

The following SPI devices are supported by EZ LADDER Toolkit.

### LS7366R 32 Bit Quadrature Counter

The LS7366R is a 32 bit Quadrature Counter integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port.

The LS7366R is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the LS7366R in the ladder diagram and minor references to hardware when needed.

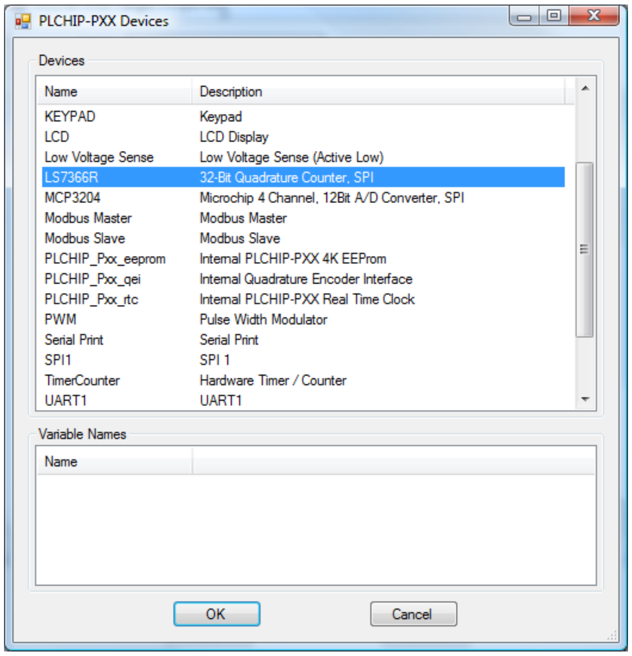
LS7366R / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

#### Installing the LS7366R in the Ladder Diagram Project

To be able to use the LS7366R in an EZ LADDER Toolkit ladder diagram project, the LS7366R must first be installed and configured. As the PLC on a Chip$^{TM}$ is the most commonly used target for the LS7366R, it will be used as an example to install and configure the LS7366R.

The LS7366R is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If a LS7366R device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the LS7366R. Figure 15-4 shows the Target's Devices window.



| Name | Description |
|---|---|
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| Serial Print | Serial Print |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART1 | UART1 |

**Figure 15-4**

Click **OK**. The LS7366R Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this LS7366R device. See Figure 15-5.

Multiple LS7366R devices may be added to an SPI port provided that each device has a unique chip select output specified.

The SPI port must be installed previously or no SPI ports will show available in drop down configuration menus.

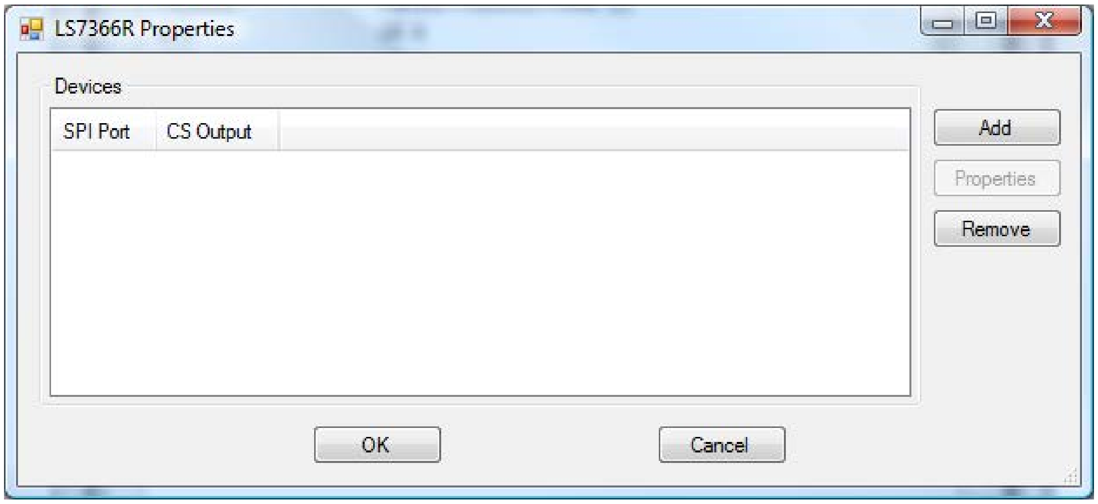


**Figure 15-5**

Click the **ADD** button. An additional LS7366R Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). With these two devices selected, additional channel information will be available to configure. See Figure 15-6. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

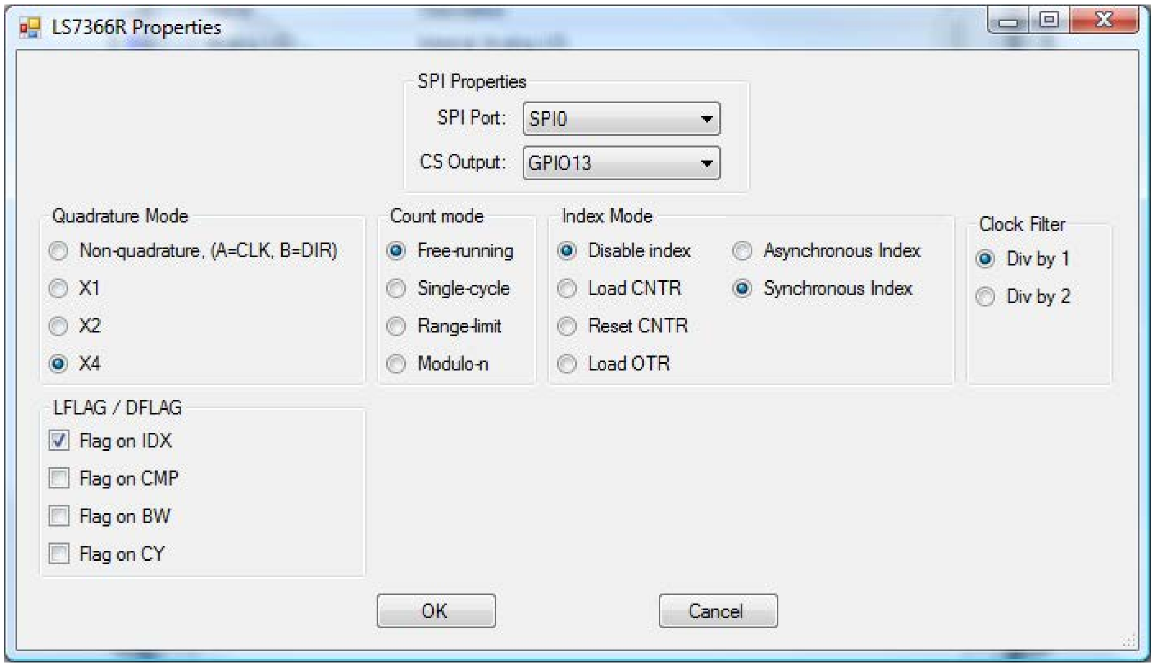
The LS7366R may be configured to run in several modes. Each mode has specific operation parameters and features that may be utilized in the ladder diagram project. These modes and parameters are configured in this dialog box.

Refer to the LS7366R integrated circuit data sheet for details to understand options, features and configurations. Failure to review the data sheet may result in a loss of understanding of how to configure and use this device.

As a difference between the LS7366R counter and other SPI devices, the LS7366R does not use variables, but instead relies on a function block to provide access to counter functionality in the ladder diagram project.

When the LS7366R is configured, click **OK** to close the LS7366R Properties dialog # 2. It is now listed in the LS7366R Properties dialog #1. Click **OK** to close the LS7366 Properites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The SPI port is now installed and ready to be used and additional SPI devices may be installed and used in the ladder diagram.


LS7366R Properties

SPI Properties
SPI Port: SPI0
CS Output: GPIO13

Quadrature Mode
- ○ Non-quadrature, (A=CLK, B=DIR)
- ○ X1
- ○ X2
- ● X4

Count mode
- ● Free-running
- ○ Single-cycle
- ○ Range-limit
- ○ Modulo-n

Index Mode
- ● Disable index
- ○ Load CNTR
- ○ Reset CNTR
- ○ Load OTR
- ○ Asynchronous Index
- ● Synchronous Index

Clock Filter
- ● Div by 1
- ○ Div by 2

LFLAG / DFLAG
- ☑ Flag on IDX
- ☐ Flag on CMP
- ☐ Flag on BW
- ☐ Flag on CY

**Figure 15-6**

## LS7366R Configuration Parameters

All modes are controlled by the hardware settings listed. Functionality is achieved using a function block in the EZ LADDER Toolkit.

**Quadrature Mode:**

***Non Quadrature (A=CLK, B=DIR)***
A pulse on the A input will increment the counter or decrement the counter based on the B input.

***X1***
The A and B inputs are used in X1 mode for use with biphase encoders. The count value changes once for each biphase cycle.

***X2***
The A and B inputs are used in X2 mode for use with biphase encoders. The count value changes 2 times X1 mode given the same input signal.

***X4***
The A and B inputs are used in X4 mode for use with biphase encoders. The count value changes with each input transition, 4 times faster than X1 given the same input signal.

**Count Mode:**

***Free Running***
When counting pulses, the counter will continue to count in either direction and wrap if the count goes larger (or smaller) than the standard integer (32 bit).

***Single Cycle***

When counting pulses, the counter will stop counting in either direction if the count goes larger (or smaller) than the standard integer (32 bit). A Reset or Load is required to restart counting.

***Range Limit***
Counting range is limited between zero (0) and the PD (DTR) input on the CNTR_LS7366R function block. The LD function block input must be used to load the DTR (one scan cycle).

***Modulo N***
The counter value will be equal to the value of the input signal divided by the value loaded in DTR, plus 1 (DTR+1). If DTR is 1, then the count will equal the input signal divided by 2 or will count at 1/2 the rate of the input signal.

**Index Mode:**

***Disable Index***
The device’s Index input will have no affect on operation.

***Load CNTR***
Configures the device’s Index input to act as a *load counter.* This will load the value of the function block input PD (DTR) as the actual count.

***Reset CNTR***
Configures the device’s Index input to act as a *reset counter.* This will reset the actual counter to zero.

***Load OTR***
Configures the devices Index input to transfer the actual count into the OTR register. The OTR register is a temporary register for storing the count.

***Asynchronous Index***
Asynchronous index mode. Valid in all modes.

***Synchronous Index***
Synchronous index mode. Only valid in quadrature mode.

**Clock Filter:**

***Divide by 1***
The clock input to the device is divided by 1 to create a filter frequency.  This filter frequency must be at least 4 times larger than the frequency on the device A input.

***Divide by 2***
The clock input to the device is divided by 2 to create a filter frequency.  This filter frequency must be at least 4 times larger than the frequency on the device A input.

**LFLAG / DFLAG:**

***Flag on IDX***
This check box enable the index flag bit that is output on the status register (ST of the CNTR_LS7366R function block).

***Flag on CMP***
This check box enable the compare flag bit that is output on the status register (set if DTR = actual count).

***Flag on BW***
This check box enable the borrow flag bit that is output on the status register (set if counter wraps negative (borrow)).

***Flag on CY***
This check box enable the borrow flag bit that is output on the status register (set if counter wraps positive (carry)).

## Using the LS7366R in the Ladder Diagram Project

To gain the functionality of the SPI LS7366R counter integrated circuit, you must use the CNTR_LS7366R function block.  This function block has multiple inputs and outputs.  These inputs and outputs can function in different modes based on the configuration of the actual LS7366R in the ladder diagram projects.

It is important to reference the LS7366R data sheet for operation modes and to understand registers. A thorough understanding of the LS7366R is required to properly configure and use the device correctly.

For details on the use of the CNTR_LS7366 function block, refer to **Appendix A - Function Reference**.

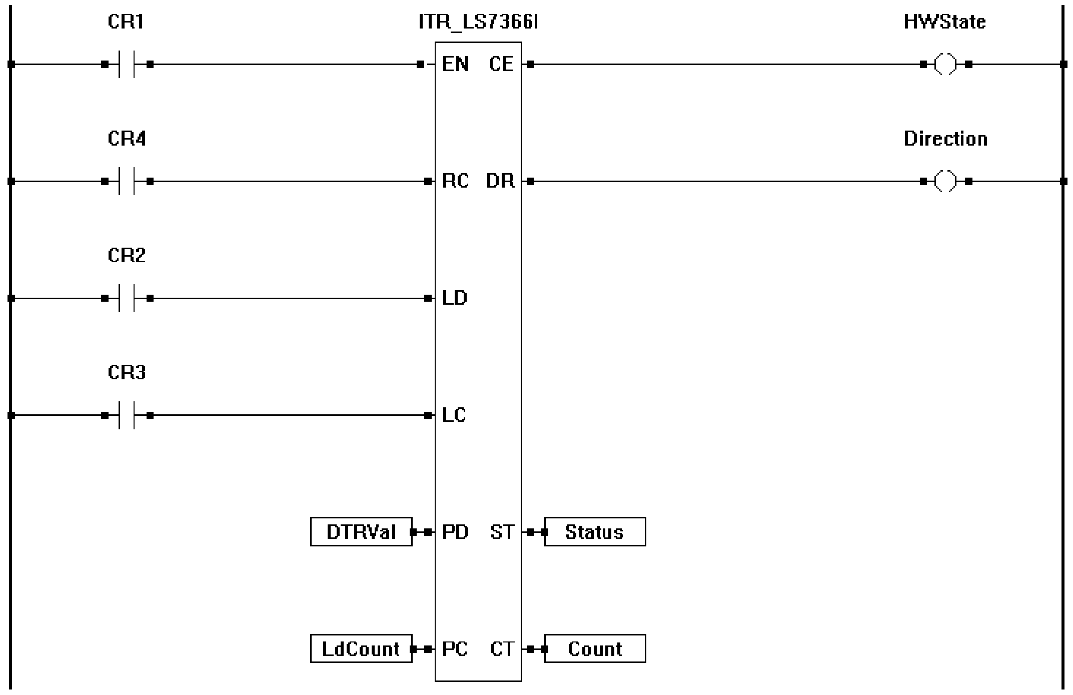


Figure 15-7

## MCP3204 4 Channel 12 Bit Analog to Digital Converter

The MCP3204 is a 12 bit, 4 channel analog to digital converter integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port.

The MCP3204 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the MCP3204 in the ladder diagram and minor references to hardware when needed.

MCP3204 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

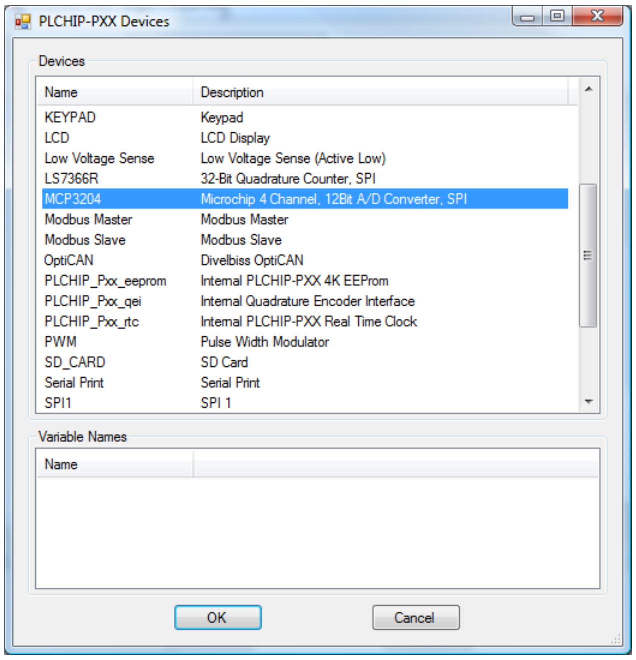
### Installing the MCP3204 in the Ladder Diagram Project

To be able to use the MCP3204 in an EZ LADDER Toolkit ladder diagram project, the MCP3204 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MCP3204, it will be used as an example to install and configure the MCP3204.

The MCP3204 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an MCP3204 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MCP3204. Figure 15-8 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.



| Name | Description |
|---|---|
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI1 | SPI 1 |

**Figure 15-8**

Click **OK**. The MCP3204 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this MCP3204 device. See Figure 15-9.

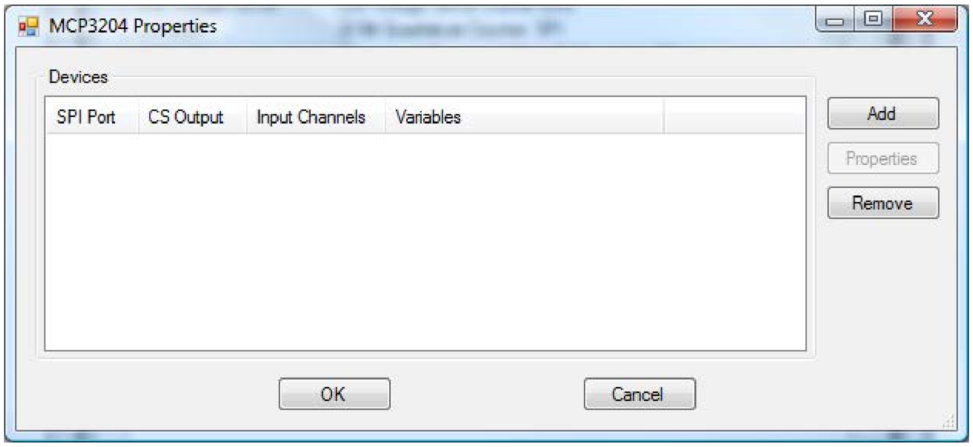


**Figure 15-9**

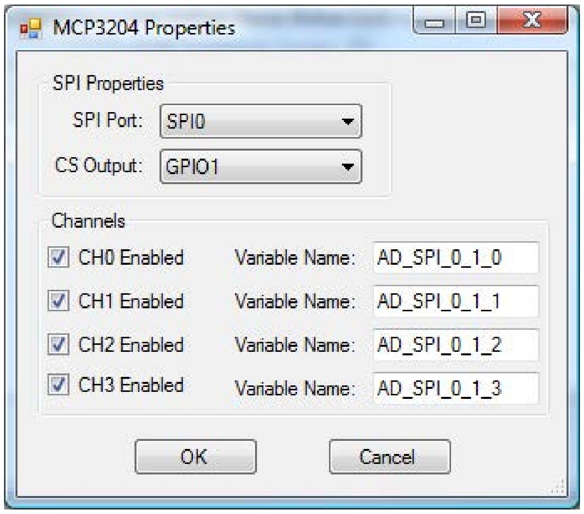
Multiple MCP3204 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional MCP3204 Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-10. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, the Channel enable check boxes are now functional. Place a checkmark in the channels that are to be used by clicking in the check box next to CH0 to CH3. As a channel is enabled, an automatic Variable name is shown.

The variable names are automatically created when a channel is enabled. The name can be changed by typing the desired variable name in the Variable Name box for each channel.


MCP3204 Properties

SPI Properties

SPI Port: SPI0

CS Output: GPIO1

Channels

| | |
|---|---|
| [✓] CH0 Enabled | Variable Name: AD_SPI_0_1_0 |
| [✓] CH1 Enabled | Variable Name: AD_SPI_0_1_1 |
| [✓] CH2 Enabled | Variable Name: AD_SPI_0_1_2 |
| [✓] CH3 Enabled | Variable Name: AD_SPI_0_1_3 |

**Figure 15-10**

Refer to the MCP3204 integrated circuit data sheet for details on the MCP3204 and its limitations.

The MCP3204 analog input (analog to digital converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MCP3204 Properties # 2 dialog.

When the MCP3204 is configured, click **OK** to close the MCP3204 Properties dialog # 2. It is now listed in the MCP3204 Properties dialog #1. Click **OK** to close the MCP3204 Propererites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The MCP3204 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the MCP3204 as a device in EZ LADDER Toolkit and the target. The MCP3204 requires additional circuitry to amplify and or scale real world analog signals that are separate from the MCP3204 and EZ LADDER Toolkit.

### Using the MCP3204 in the Ladder Diagram Project

The MCP3204 analog input (analog to digital converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MCP3204 Properties # 2 dialog.

During the installation process of the MCP3204, the variables should automatically be created as Integers. As the MCP3204 is a 12 bit analog to digital converter, the actual integer value for each channel will range from 0 to 4095 with 0 being 0 or the low end and 4095 being the maximum or high end. These variables may be used as inputs to function blocks in the ladder diagram program.

Figure 15-11 is an example ladder diagram using a variable from the MCP3204.

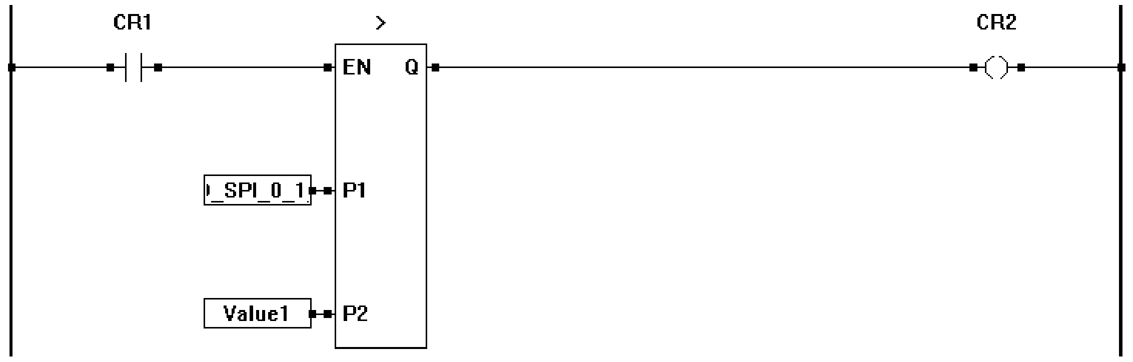


**Figure 15-11**

## MCP3208 8 Channel 12 Bit Analog to Digital Converter

The MCP3208 is a 12 bit, 8 channel analog to digital converter integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port.

The MCP3208 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the MCP3208 in the ladder diagram and minor references to hardware when needed.

MCP3208 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

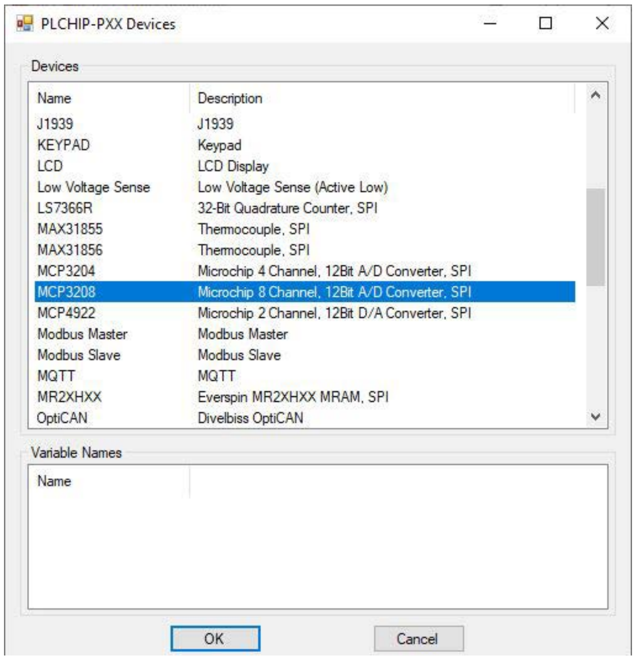
### Installing the MCP3208 in the Ladder Diagram Project

To be able to use the MCP3208 in an EZ LADDER Toolkit ladder diagram project, the MCP3208 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MCP3208, it will be used as an example to install and configure the MCP3208.

The MCP3208 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an MCP3208 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MCP3208. Figure 15-12 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| J1939 | J1939 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |
| MAX31856 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| MCP3208 | Microchip 8 Channel, 12Bit A/D Converter, SPI |
| MCP4922 | Microchip 2 Channel, 12Bit D/A Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| MR2XHXX | Everspin MR2XHXX MRAM, SPI |
| OptiCAN | Divelbiss OptiCAN |

Figure 15-12

Click **OK**. The MCP3208 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this MCP3208 device. See Figure 15-13.

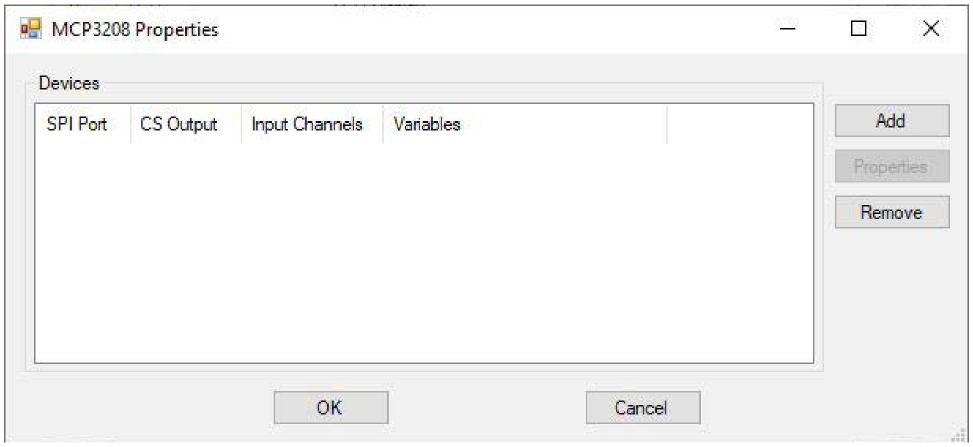


Figure 15-13

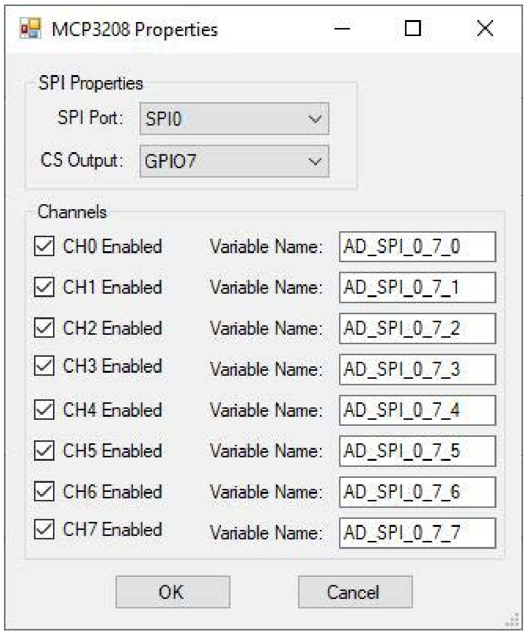
Multiple MCP3208 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional MCP3208 Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-14. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, the Channel enable check boxes are now functional. Place a checkmark in the channels that are to be used by clicking in the check box next to CH0 to CH7. As a channel is enabled, an automatic Variable name is shown.

The variable names are automatically created when a channel is enabled. The name can be changed by typing the desired variable name in the Variable Name box for each channel.


MCP3208 Properties

SPI Properties

SPI Port: SPI0

CS Output: GPIO7

Channels

| | | |
|---|---|---|
| ☑ CH0 Enabled | Variable Name: | AD_SPI_0_7_0 |
| ☑ CH1 Enabled | Variable Name: | AD_SPI_0_7_1 |
| ☑ CH2 Enabled | Variable Name: | AD_SPI_0_7_2 |
| ☑ CH3 Enabled | Variable Name: | AD_SPI_0_7_3 |
| ☑ CH4 Enabled | Variable Name: | AD_SPI_0_7_4 |
| ☑ CH5 Enabled | Variable Name: | AD_SPI_0_7_5 |
| ☑ CH6 Enabled | Variable Name: | AD_SPI_0_7_6 |
| ☑ CH7 Enabled | Variable Name: | AD_SPI_0_7_7 |

**Figure 15-14**

Refer to the MCP3208 integrated circuit data sheet for details on the MCP3208 and its limitations.

The MCP3208 analog input (analog to digital converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MCP3208 Properties # 2 dialog.

When the MCP3208 is configured, click **OK** to close the MCP3208 Properties dialog # 2.  It is now listed in the MCP3208 Properties dialog #1. Click **OK** to close the MCP3208 Properites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window.  Be sure to save the ladder diagram project. The MCP3208 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the MCP3208 as a device in EZ LADDER Toolkit and the target. The MCP3208 requires additional circuitry to amplify and or scale real world analog signals that are separate from the MCP3208 and EZ LADDER Toolkit.

### Using the MCP3208 in the Ladder Diagram Project

The MCP3208 analog input (analog to digital converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MCP3208 Properties # 2 dialog.

During the installation process of the MCP3208, the variables should automatically be created as Integers. As the MCP3208 is a 12 bit analog to digital converter, the actual integer value for each channel will range from 0 to 4095 with 0 being 0 or the low end and 4095 being the maximum or high end. These variables may be used as inputs to function blocks in the ladder diagram program.

Figure 15-15 is an example ladder diagram using a variable from the MCP3204.

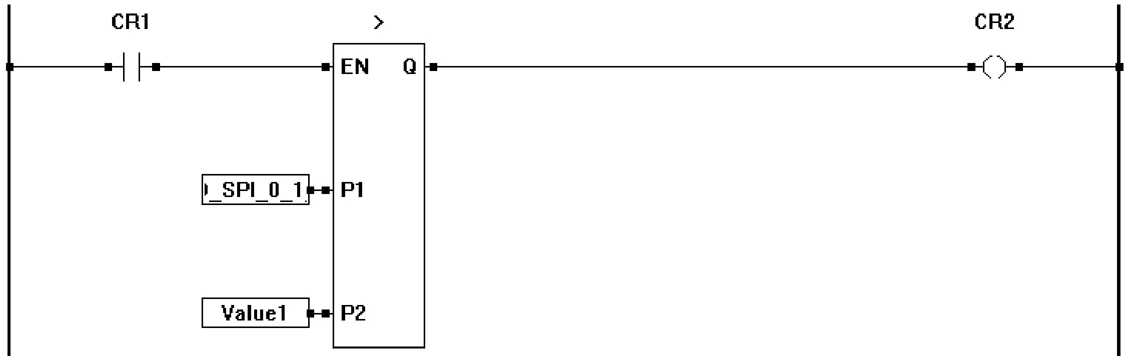


**Figure 15-15**

## ADS8341 4 Channel 16 Bit Analog to Digital Converter

The ADS8341 is a 16 bit, 4 channel analog to digital converter integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port.

The ADS8341 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the ADS8341 in the ladder diagram and minor references to hardware when needed.

ADS8341 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

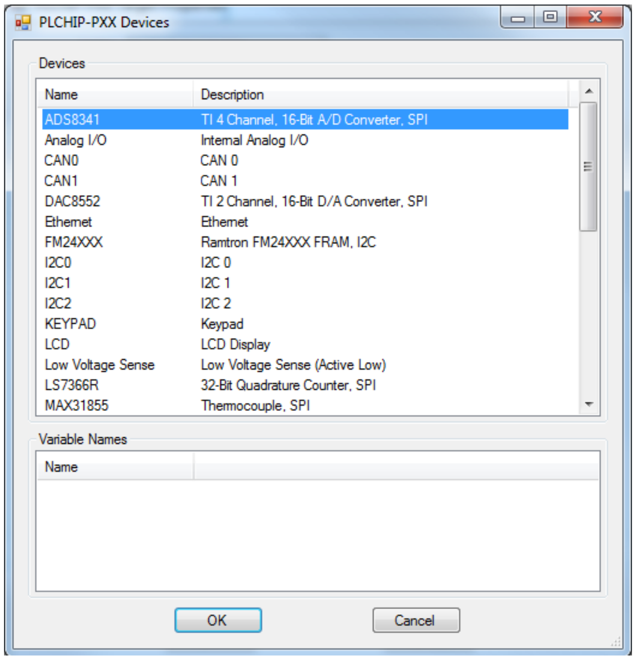
### Installing the ADS8341 in the Ladder Diagram Project

To be able to use the ADS8341 in an EZ LADDER Toolkit ladder diagram project, the ADS8341 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the ADS8341, it will be used as an example to install and configure the ADS8341.

The ADS8341 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an ADS8341 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the ADS8341. Figure 15-16 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.



| Name | Description |
|---|---|
| ADS8341 | TI 4 Channel, 16-Bit A/D Converter, SPI |
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| DAC8552 | TI 2 Channel, 16-Bit D/A Converter, SPI |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |

**Figure 15-16**

Click **OK**. The ADS8341 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this ADS8341 device. See Figure 15-17.

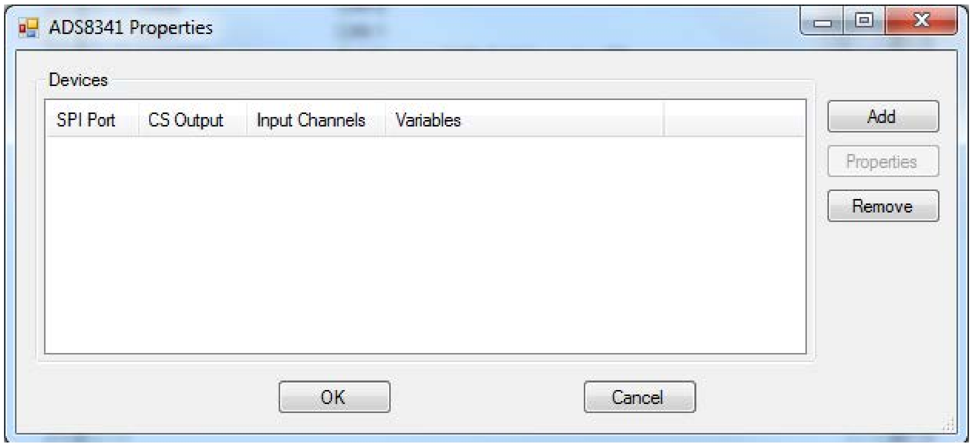


**Figure 15-17**

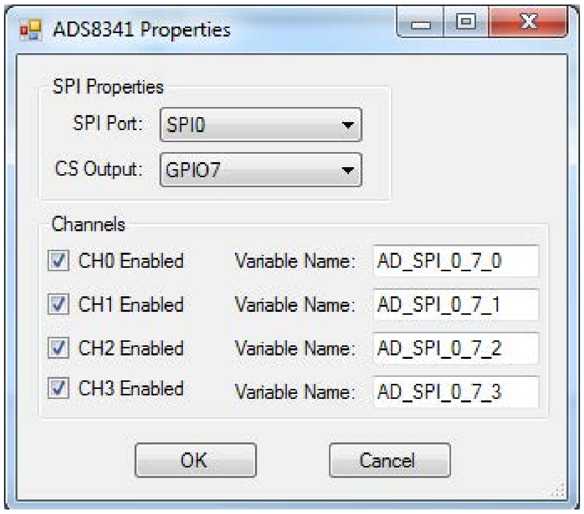
Multiple ADS8341 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional ADS8341 Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-18. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, the Channel enable check boxes are now functional. Place a checkmark in the channels that are to be used by clicking in the check box next to CH0 to CH3. As a channel is enabled, an automatic Variable name is shown.

The variable names are automatically created when a channel is enabled. The name can be changed by typing the desired variable name in the Variable Name box for each channel.



SPI Port: SPI0
CS Output: GPIO7

Channels

| | | |
|---|---|---|
| ✓ CH0 Enabled | Variable Name: | AD_SPI_0_7_0 |
| ✓ CH1 Enabled | Variable Name: | AD_SPI_0_7_1 |
| ✓ CH2 Enabled | Variable Name: | AD_SPI_0_7_2 |
| ✓ CH3 Enabled | Variable Name: | AD_SPI_0_7_3 |

**Figure 15-18**

Refer to the ADS8341 integrated circuit data sheet for details on the ADS8341 and its limitations.

The ADS8341 analog input (analog to digital converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the ADS8341 Properties # 2 dialog.

When the ADS8341 is configured, click **OK** to close the ADS8341 Properties dialog # 2. It is now listed in the ADS8341 Properties dialog #1. Click **OK** to close the ADS8341 Properperites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The ADS8341 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the ADS8341 as a device in EZ LADDER Toolkit and the target. The ADS8341 requires additional circuitry to amplify and or scale real world analog signals that are separate from the ADS8341 and EZ LADDER Toolkit.

### Using the ADS8341 in the Ladder Diagram Project

The ADS8341 analog input (analog to digital converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the ADS8341 Properties # 2 dialog.

During the installation process of the ADS8341, the variables should automatically be created as Integers. As the ADS8341 is a 16 bit analog to digital converter, the actual integer value for each channel will range from 0 to 32767 with 0 being 0 or the low end and 32767 being the maximum or high end. These variables may be used as inputs to function blocks in the ladder diagram program.

Figure 15-19 is an example ladder diagram using a variable from the ADS8341.

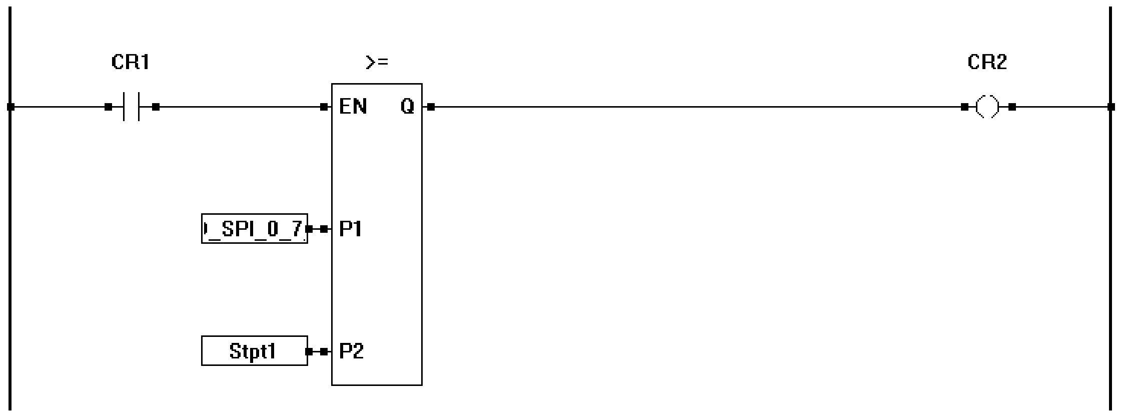


**Figure 15-19**

## DAC8552 2 Channel 16 Bit Digital to Analog Converter

The DAC8552 is a 16 bit, 2 channel digital to analog converter integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port.

The DAC8552 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the DAC8552 in the ladder diagram and minor references to hardware when needed.

DAC8552 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

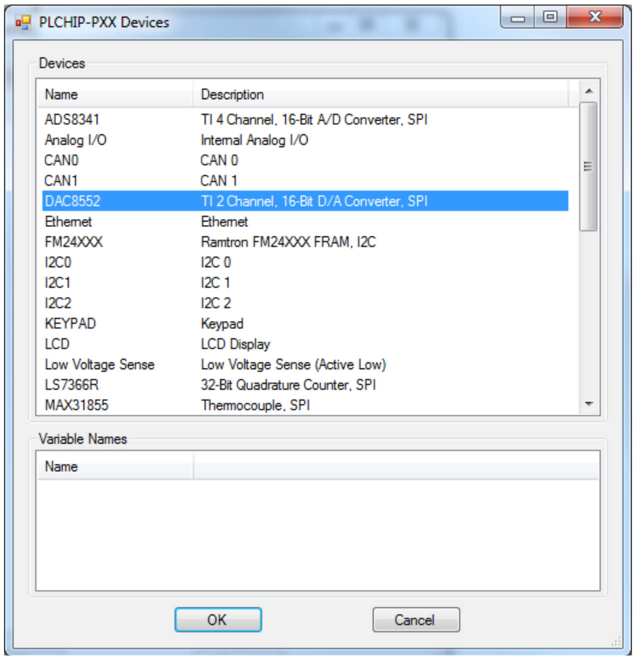
### Installing the DAC8552 in the Ladder Diagram Project

To be able to use the DAC8552 in an EZ LADDER Toolkit ladder diagram project, the DAC8552 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the DAC8552, it will be used as an example to install and configure the DAC8552.

The DAC8552 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an DAC8552 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the DAC8552. Figure 15-20 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.


PLCHIP-PXX Devices

Devices

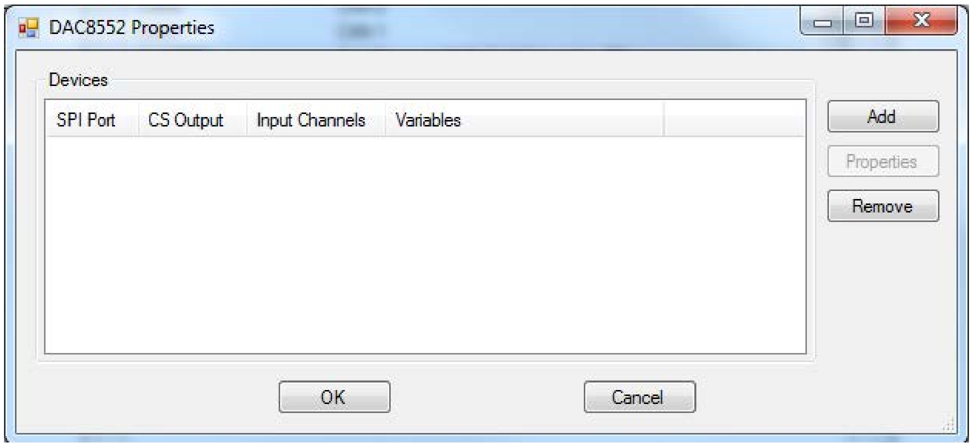
| Name | Description |
|---|---|
| ADS8341 | TI 4 Channel, 16-Bit A/D Converter, SPI |
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| DAC8552 | TI 2 Channel, 16-Bit D/A Converter, SPI |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |

Variable Names

Name

**Figure 15-20**

Click **OK**. The DAC8552 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this DAC8552 device. See Figure 15-21.


DAC8552 Properties

Devices

| SPI Port | CS Output | Input Channels | Variables |
|---|---|---|---|

**Figure 15-21**

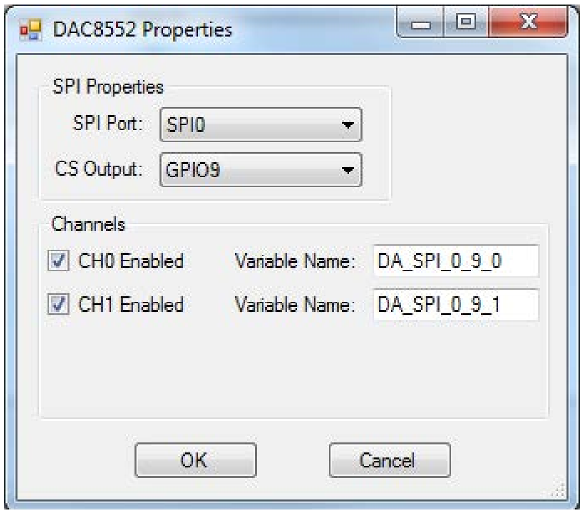
Multiple DAC8552 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional DAC8552 Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-22. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, the Channel enable check boxes are now functional. Place a checkmark in the channels that are to be used by clicking in the check box next to CH0 to CH1. As a channel is enabled, an automatic Variable name is shown.

The variable names are automatically created when a channel is enabled. The name can be changed by typing the desired variable name in the Variable Name box for each channel.


DAC8552 Properties

SPI Properties

SPI Port: SPI0

CS Output: GPIO9

Channels

[✓] CH0 Enabled Variable Name: DA_SPI_0_9_0

[✓] CH1 Enabled Variable Name: DA_SPI_0_9_1

**Figure 15-22**

Refer to the DAC8552 integrated circuit data sheet for details on the DAC8552 and its limitations.

The DAC8552 analog output (digital to analog converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the DAC8552 Properties # 2 dialog.

When the DAC8552 is configured, click **OK** to close the DAC8552 Properties dialog # 2. It is now listed in the DAC8552 Properties dialog #1. Click **OK** to close the DAC8552 Properites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The DAC8552 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the DAC8552 as a device in EZ LADDER Toolkit and the target. The DAC8552 requires additional circuitry to amplify and or scale real world analog signals that are separate from the DAC8552 and EZ LADDER Toolkit.

### Using the DAC8552 in the Ladder Diagram Project

The DAC8552 analog output (digital to analog converter channel values) are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the DAC8552 Properties # 2 dialog.

During the installation process of the DAC8552, the variables should automatically be created as Integers. As the DAC8552 is a 16 bit digital to analog converter, the actual integer value for each channel will range from 0 to 65535 with 0 being 0 or the low end and 65535 being the maximum or high end. These variables may be used as outputs from function blocks in the ladder diagram program.

Figure 15-23 is an example ladder diagram using a variable from the DAC8552.

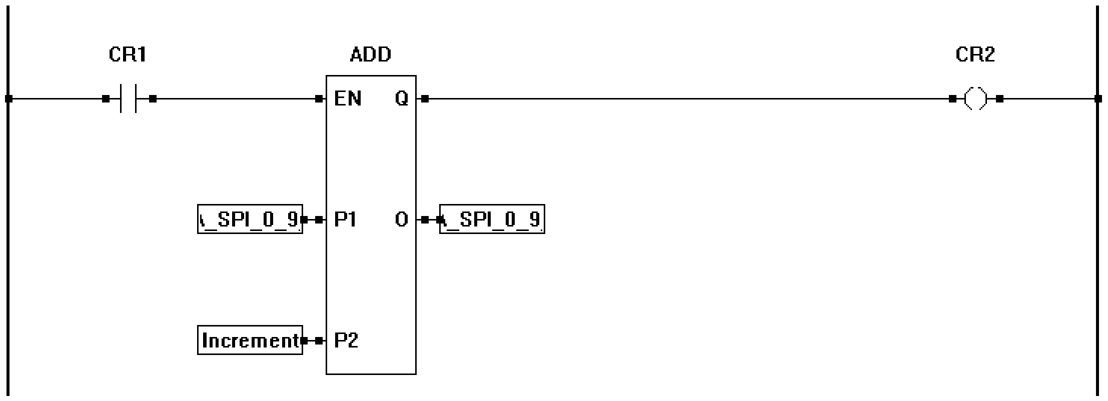


**Figure 15-23**

## MAX31855 Single Channel Thermocouple Input

The MAX31855 is a single channel cold junction compensated thermocouple input integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port. The suffix to the MAX31855 determines the thermocouple type and range of operation. Refer to the integrated circuit's datasheet.

The MAX31855 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the MAX31855 in the ladder diagram and minor references to hardware when needed.

MAX31855 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

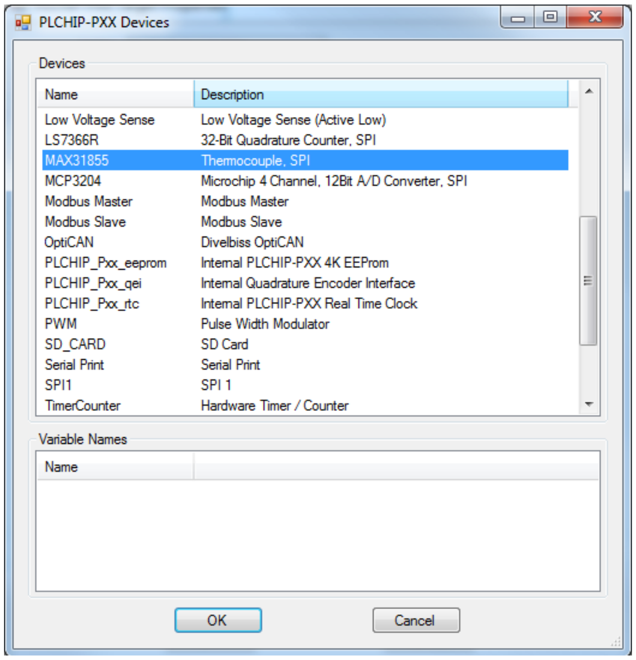
### Installing the MAX31855 in the Ladder Diagram Project

To be able to use the MAX31855 in an EZ LADDER Toolkit ladder diagram project, the MAX31855 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MAX31855, it will be used as an example to install and configure the MAX31855.

The MAX31855 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an MAX31855 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MAX31855. Figure 15-24 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.


PLCHIP-PXX Devices

Devices

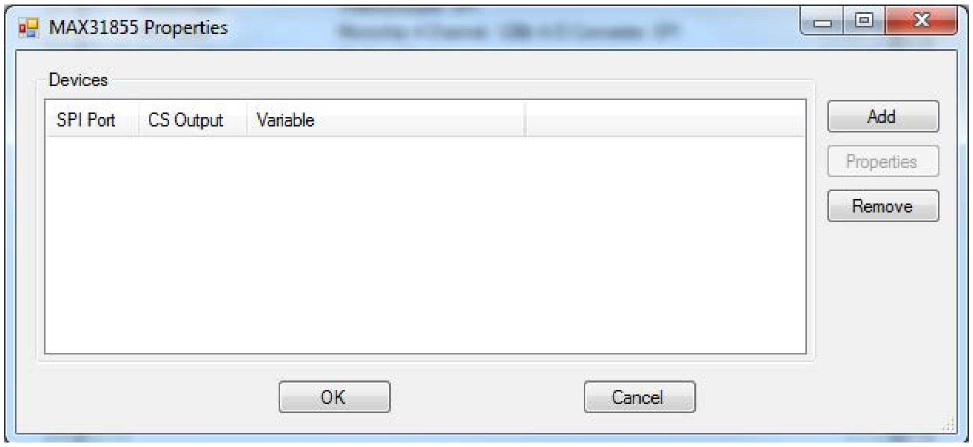
| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |

Variable Names

Name

**Figure 15-24**

Click **OK**. The MAX31855 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this MAX31855 device. See Figure 15-25.


MAX31855 Properties

Devices

| SPI Port | CS Output | Variable |
|---|---|---|

**Figure 15-25**

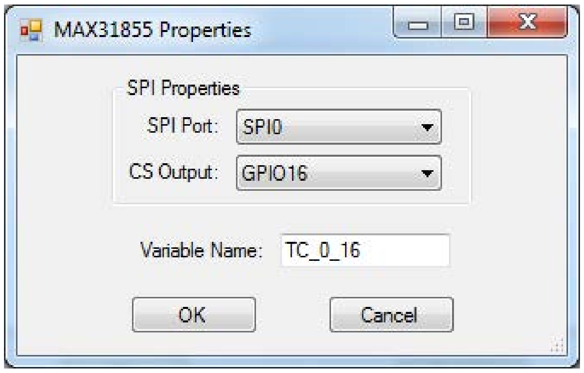
Multiple MAX31855 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional MAX31855 Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device’s chip select (CS). See Figure 15-26. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, an automatic Variable name is shown.

The variable name is automatically created when SPI port and CS are selected. The name can be changed by typing the desired variable name in the Variable Name box.


MAX31855 Properties

SPI Properties

SPI Port: SPI0

CS Output: GPIO16

Variable Name: TC_0_16

**Figure 15-26**

Refer to the MAX31855 integrated circuit data sheet for details on the MAX31855 and its limitations.

The MAX31855 thermocouple input values are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MAX31855 Properties # 2 dialog.

When the MAX31855 is configured, click **OK** to close the MAX31855 Properties dialog # 2. It is now listed in the MAX31855 Properties dialog #1. Click **OK** to close the MAX31855 Properperites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The MAX31855 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the MAX31855 as a device in EZ LADDER Toolkit and the target. The MAX31855 requires additional circuitry to control and filter real world thermocouple signals that are separate from the MAX31855 and EZ LADDER Toolkit.

## Using the MAX31855 in the Ladder Diagram Project

The MAX31855 thermocouple input values are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MAX31855 Properties # 2 dialog.

During the installation process of the MAX31855, the variable should have been automatically be created as a real. As the MAX31855 is a thermocouple input, the actual value will be equal to the current sensed temperature in degrees celsius (°C). The type of thermocouple and range is dependent upon the actual part number of the MAX31855 used (model determines the thermocouple type and thermocouple temperature range).This variable may be used as inputs to function blocks in the ladder diagram program.

Figure 15-27 is an example ladder diagram using a variable from the MAX31855.

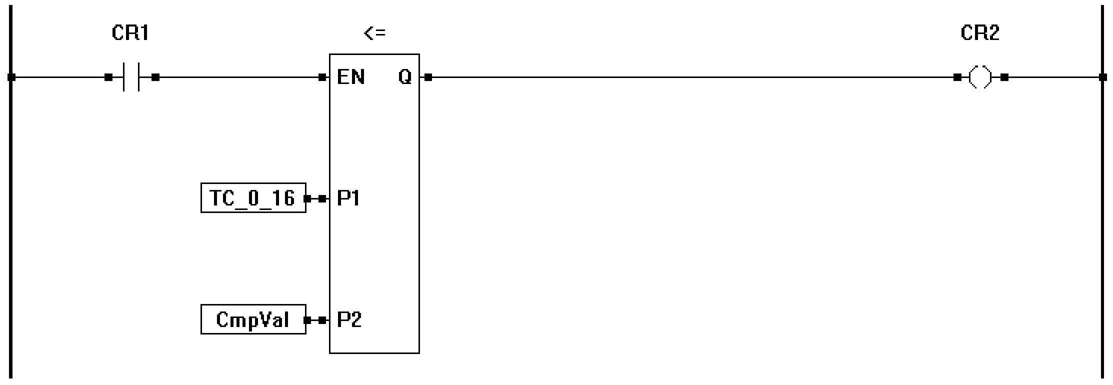


**Figure 15-27**

## MAX31855 Error Codes

The MAX31855 may return an error code instead of a temperature based on errors detected. The following error codes are supported:

| | |
|---|---|
| -9999 | Thermocouple is reading open |
| -9998 | Thermocouple is reading shorted to ground |
| -9997 | Thermocouple is reading shorted to +V |
| -9996 | Undefined error, could be invalid or unexpected data |

## MAX31856 Single Channel Thermocouple Input

The MAX31856 is a single channel cold junction compensated thermocouple input integrated circuit with an SPI interface.  EZ LADDER Toolkit has built-in software support for using this device on an SPI port. The suffix to the MAX31856 supports multiple thermocouple types (B, E, J, K, N, R, S, T), selected and configured in EZ LADDER Toolkit. Refer to the MAX31856 circuit's datasheet for more details on thermocouple types and interface circuitry.

The MAX31856 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the MAX31856 in the ladder diagram and minor references to hardware when needed.

MAX31856 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

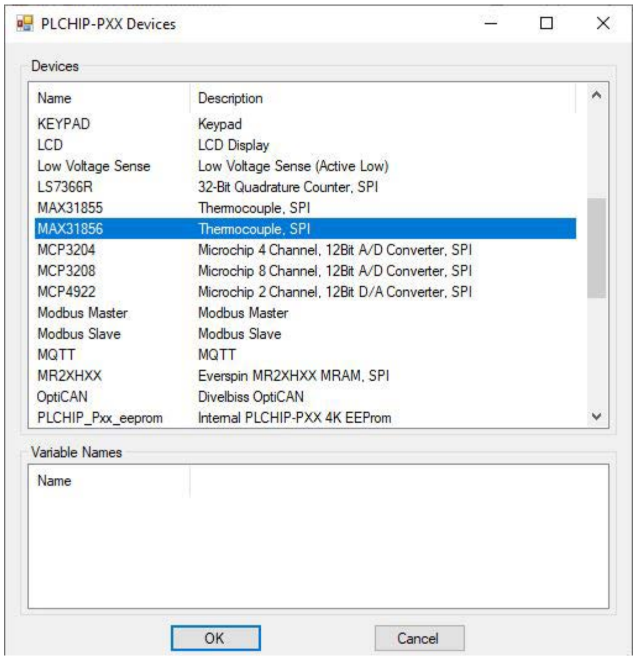
## Installing the MAX31856 in the Ladder Diagram Project

To be able to use the MAX31856 in an EZ LADDER Toolkit ladder diagram project, the MAX31856 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MAX31856, it will be used as an example to install and configure the MAX31856.

The MAX31855 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an MAX31856 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MAX31856. Figure 15-28 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |
| MAX31856 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| MCP3208 | Microchip 8 Channel, 12Bit A/D Converter, SPI |
| MCP4922 | Microchip 2 Channel, 12Bit D/A Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| MR2XHXX | Everspin MR2XHXX MRAM, SPI |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |

Variable Names

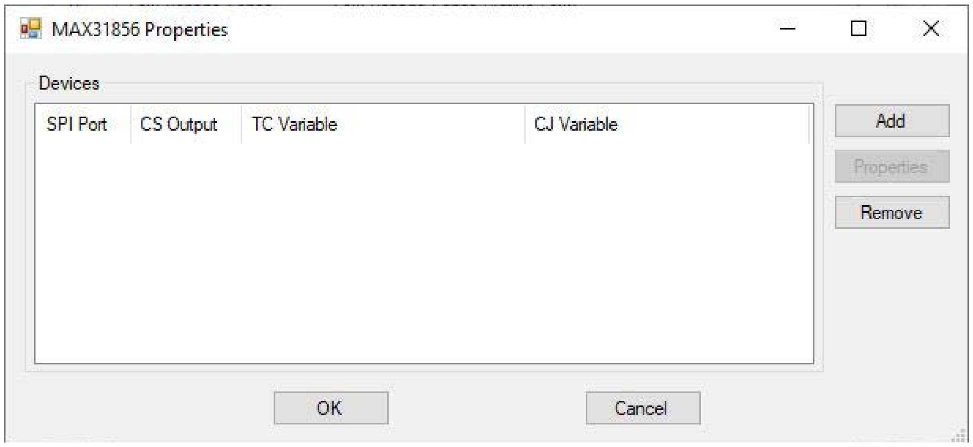
Name

**Figure 15-28**

Click **OK**. The MAX31856 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this MAX31856 device. See Figure 15-29.

Multiple MAX31856 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional MAX31856 Properties window will open.



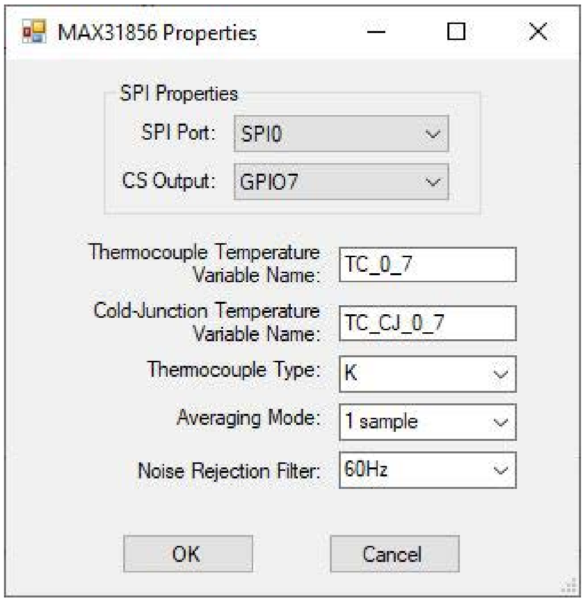
| SPI Port | CS Output | TC Variable | CJ Variable |
|---|---|---|---|

Figure 15-29

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-30. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, an automatic Thernocouple Temperature Variable name and Cold-Junction Temperature Variable name is shown.

The variable names are automatically created when SPI port and CS are selected. The names can be changed by typing the desired variable name(s) in the Variable Name(s) boxes.



SPI Port: SPI0
CS Output: GPIO7
Thermocouple Temperature Variable Name: TC_0_7
Cold-Junction Temperature Variable Name: TC_CJ_0_7
Thermocouple Type: K
Averaging Mode: 1 sample
Noise Rejection Filter: 60Hz

Figure 15-30

Using the ***Thermocouple Type*** drop-down menu, select the type of thermocouple that will be connected to the device from the allowed choices.

Using the ***Averaging Mode*** drop-down menu, select the number of averaging samples to use for the temperature readings from the allowed choices.

Using the ***Noise Rejection Filter*** drop-down menu, select the type noise rejection filter to use (50 Hz or 60 Hz).

Refer to the MAX31856 integrated circuit data sheet for details on the MAX31856 and its limitations.

The MAX31856 thermocouple input values are always stored and can be accessed in the ladder diagram program by the variable (names) configured in the MAX31856 Properties # 2 dialog.

When the MAX31856 is configured, click **OK** to close the MAX31856 Properties dialog # 2. It is now listed in the MAX31856 Properties dialog #1. Click **OK** to close the MAX31856 Properites dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The MAX31856 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the MAX31855 as a device in EZ LADDER Toolkit and the target. The MAX31855 requires additional circuitry to control and filter real world thermocouple signals that are separate from the MAX31855 and EZ LADDER Toolkit.

## Using the MAX31856 in the Ladder Diagram Project

The MAX31856 thermocouple input values are always stored and can be accessed in the ladder diagram program by the variables (names) configured in the MAX31856 Properties # 2 dialog (Thermocouple Temperature and Cold-Junction Temperature).

During the installation process of the MAX31856, the variable(s) should have been automatically be created as a reals. As the MAX31856 is a thermocouple input, the actual value will be equal to the current sensed temperature in degrees celsius (°C) and the cold-junction temperature in degrees (°C). The type of thermocouple and range is dependent upon how the device was configured in the Project Settings. These variables may be used as inputs to function blocks in the ladder diagram program.

Figure 15-31 is an example ladder diagram using a variable from the MAX31856.

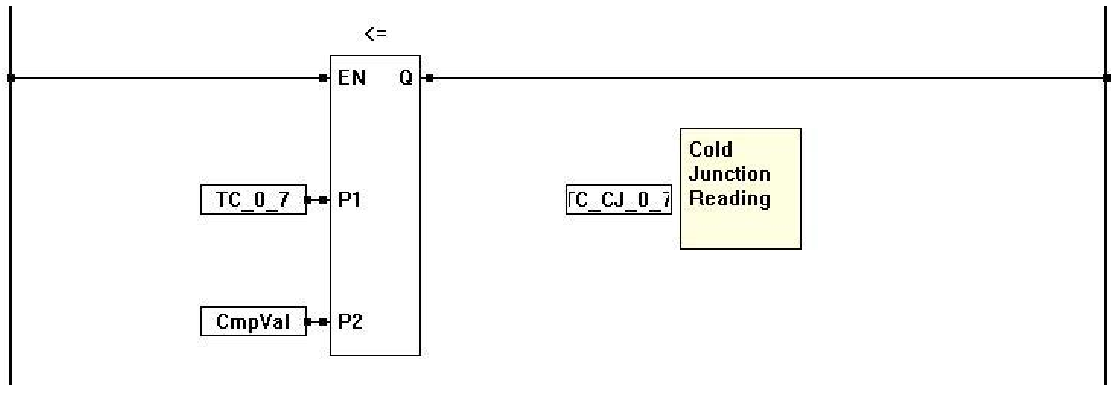


**Figure 15-31**

### MAX31856 Error Codes

The MAX31856 may return an error code instead of a temperature based on errors detected. The following error codes are supported:

| | |
|---|---|
| -9999 | Thermocouple is reading open |
| -9998 | Thermocouple is reading shorted to ground |
| -9997 | Thermocouple is reading shorted to +V |
| -9996 | Undefined error, could be invalid or unexpected data |

## MCP4922 2 Channel 12 Bit Digital to Analog (D/A) Converter

The MCP4922 is a two channel digital to analog converter integrated circuit with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port. Refer to the MCP4922 circuit's datasheet for more details on interface circuitry.

The MCP4922 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the MCP4922 in the ladder diagram and minor references to hardware when needed.

MCP4922 / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

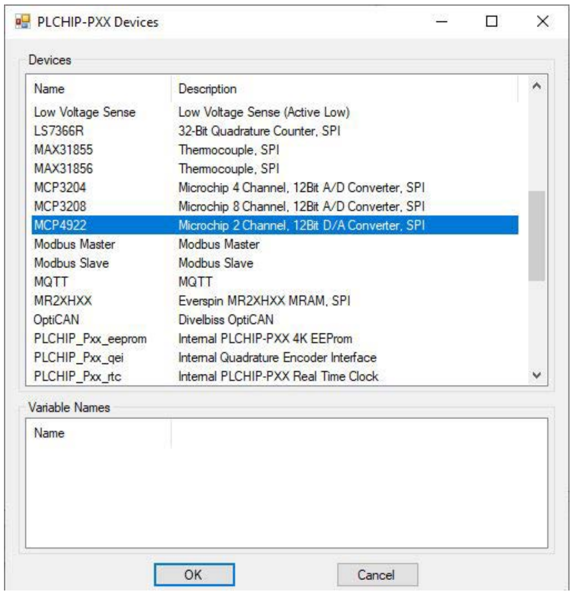
### Installing the MCP4922 in the Ladder Diagram Project

To be able to use the MCP4922 in an EZ LADDER Toolkit ladder diagram project, the MCP4922 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MCP4922, it will be used as an example to install and configure the MCP4922.

The MCP4922 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If an MCP4922 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MCP4922. Figure 15-32 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.



PLCHIP-PXX Devices

Devices

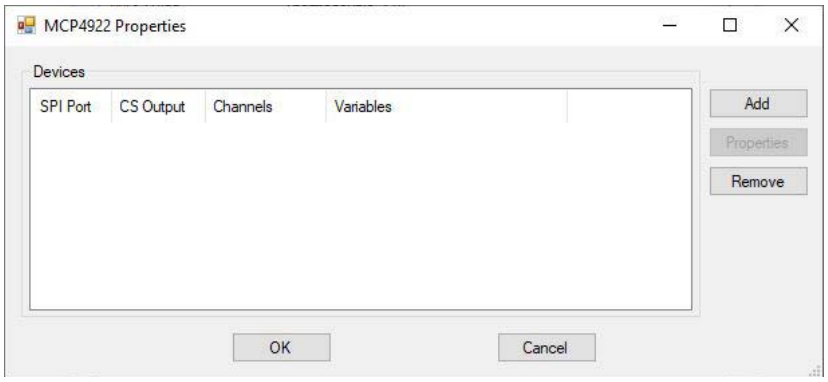
| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |
| MAX31856 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| MCP3208 | Microchip 8 Channel, 12Bit A/D Converter, SPI |
| MCP4922 | Microchip 2 Channel, 12Bit D/A Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| MR2XHXX | Everspin MR2XHXX MRAM, SPI |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |

Variable Names

Name

**Figure 15-32**

Click **OK**. The MCP4922 Properties dialog will open. This dialog selects the SPI Port and Chip Select to be used for this MCP4922 device. See Figure 15-33



MCP4922 Properties

Devices

| SPI Port | CS Output | Channels | Variables |
|---|---|---|---|

**Figure 15-33**

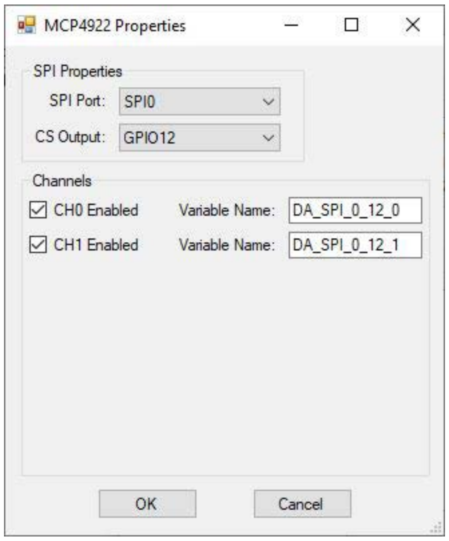
Multiple MCP4922 devices may be added to an SPI port provided that each device has a unique chip select output specified.

Click the **ADD** button. An additional MCP4922 Properties window will open.

Select the SPI port from the drop down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-34. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

With the SPI Port and CS Output selected, the Channel enable check boxes are now functional. Place a checkmark in the channels that are to be used by clicking in the check box next to CH0 to CH1. As a channel is enabled, an automatic Variable name is shown.

The variable names are automatically created when a channel is enabled. The name can be changed by typing the desired variable name in the Variable Name box for each channel.



SPI Properties
SPI Port: SPI0
CS Output: GPIO12

Channels
☑ CH0 Enabled Variable Name: DA_SPI_0_12_0
☑ CH1 Enabled Variable Name: DA_SPI_0_12_1

**Figure 15-34**

Refer to the MCP4922 integrated circuit data sheet for details on the MCP4922 and its limitations.

The MCP4922 analog output (digital to analog converter channel values) are set and can be accessed in the ladder diagram program by the variable (names) configured in the MCP4922 Properties # 2 dialog.

When the MCP4922 is configured, click **OK** to close the MCP4922 Properties dialog # 2. It is now listed in the MCP4922 Properties dialog #1. Click **OK** to close the MCP4922 Properiters dialog #1.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The MCP4922 device is now installed and ready to be used in the ladder diagram.

This installation is to configure the MCP4922 as a device in EZ LADDER Toolkit and the target. The MCP4922 requires additional circuitry to amplify and or scale to real world analog signals that are separate from the MCP4922 and EZ LADDER Toolkit.

## Using the MCP4922 in the Ladder Diagram Project

The MCP4922 analog output (digital to analog converter channel values) are always controlled and can be accessed in the ladder diagram program by the variable (names) configured in the MCP4922 Properties # 2 dialog.

During the installation process of the MCP4922, the variables should automatically be created as Integers. As the MCP4922 is a 12 bit digital to analog converter, the actual integer value for each channel will range from 0 to 4095 with 0 being 0 or the low end and 4095 being the maximum or high end. These variables may be used as outputs from function blocks in the ladder diagram program.

Figure 15-35 is an example ladder diagram using a variable from the MCP4922.

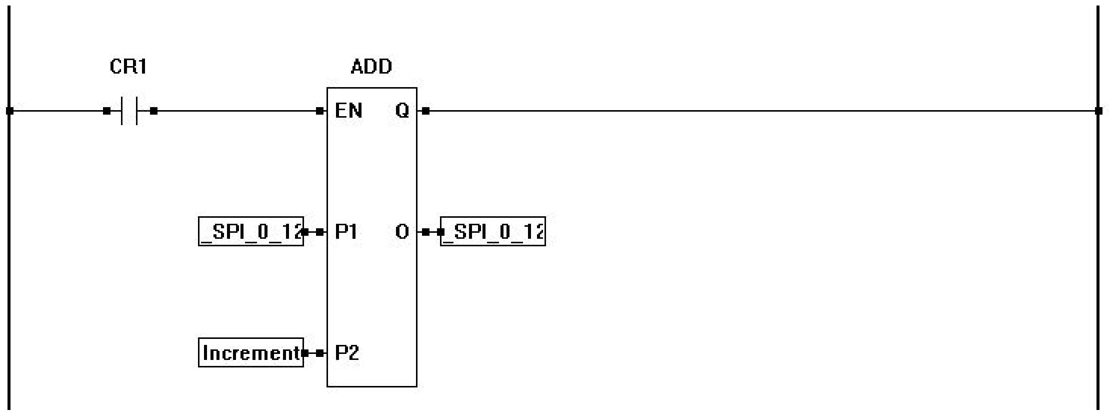


Figure 15-35

## Everspin MRxHxx Magnetoresistive RAM

The Everspin MRxHxx is magnetoresistive RAM integrated circuit (device) with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port. The MRxHxx device is used in the ladder diagram as retentive memory for automatically storing retentive variables on power loss or as memory storage (using the EEPROM_Read and EEPROM_Write) functions.

The MRxHxx device is avaiable in multiple memory sizes. The currently supported sizes and part numbers are:

| | |
|---|---|
| MR25H128A | 16K Bytes |
| MR25H256 | 32K Bytes |
| MR25H10 | 128K Bytes |
| MR25H40 | 512K Bytes |

(Refer to the Everspin MRxHxx circuit's datasheet for more details on interface circuitry and other requirements.

The Everspin MRxHxx is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the Everspin MRxHxx in the ladder diagram and minor references to hardware when needed.

Everspin MRxHxx / SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

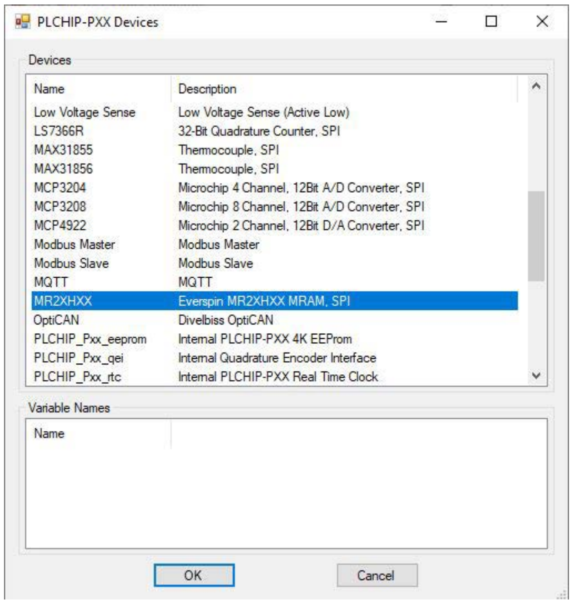
### Installing the MRxHxx in the Ladder Diagram Project

To be able to use the MRxHxx in an EZ LADDER Toolkit ladder diagram project, the MRxHxx must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MRxHxx, it will be used as an example to install and configure the MRxHxx.

The MRxHxx is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the MRxHxx device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MRxHxx. Figure 15-36 shows the Target's Devices window.

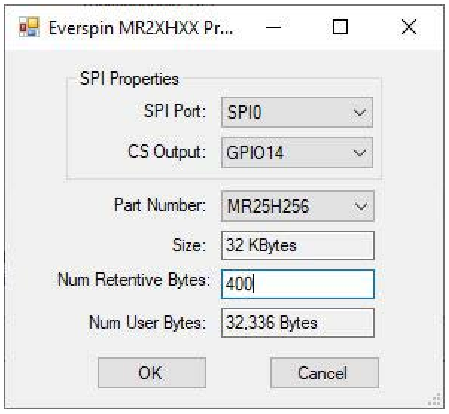
The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus.



| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |
| MAX31856 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| MCP3208 | Microchip 8 Channel, 12Bit A/D Converter, SPI |
| MCP4922 | Microchip 2 Channel, 12Bit D/A Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| MR2XHXX | Everspin MR2XHXX MRAM, SPI |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |

**Figure 15-36**

Click **OK**. The Everspin MRxHxx Properties dialog will open. This dialog selects the SPI Port, Chip Select, Part number and memory allocation to be used for this Everspin MRxHxx device. See Figure 15-37



SPI Port: SPI0
CS Output: GPIO14
Part Number: MR25H256
Size: 32 KBytes
Num Retentive Bytes: 400
Num User Bytes: 32,336 Bytes

**Figure 15-37**

Select the SPI port from the drop-down menu and select the general purpose output pin (GPO) that will serve as this device's chip select (CS). See Figure 15-37. Any chip select pins reserved for structured text will not be visible in the drop-down menu.

Select the Everspin MR2xHxx part number using the ***Part Number*** drop-down menu. Once selected, the ***Size*** will update automatically and a default number of retentive bytes will be added. Adjust the number of Retentive bytes to the desired amount using the ***Num Retentive Bytes*** box. The ***Num User Bytes*** will adjust automatically from the Num Retentive bytes entered.

All memory on the device not configured as retentive will be configured as Num User Bytes.

All Retentive memory is used to store retentive variables automatically on power loss (provided enough power is provided to complete the write cycle after a power loss is detected (hardware). All Num User bytes are accessed in the ladder using the EEPROM_READ and EEPROM_WRITE function blocks.

When the Everspin MR2xHxx is configured, click **OK** to close the Everspin MR2xHxx Properties dialog. It is now listed in SPI section of the Devices window.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The Everspin MR2xHxx device is now installed and ready to be used in the ladder diagram.

Only one Everspin MR2xHxx can be installed in a ladder diagram project. EZ LADDER does not support multiple Everspin MR2xHxx devices simultaneously.

### Using the Everspin MR2xHxx in the Ladder Diagram Project

The Everspin MR2xHxx memory is accessed as retentive memory or as general memory storage using the EEPROM_READ and EEPROM_WRITE commands (same as EEPROM).

For retentive memory and using the EEPROM_READ and EEPROM_WRITE, see **Chapter 7 - Retentive Variables & EEPROM Memory** and **Chapter 5 - Creating Ladder Diagrams Projects**.

## MAX22007 Digital to Analog Converter (DAC) IC

The MAX22007 is a 4 channel, digital to analog output (DAC) integrated circuit (device) with an SPI interface. EZ LADDER Toolkit has built-in software support for using this device on an SPI port. The MAX22007 can be configured to be either 0-10VDC (0 to 12.5VDC) or 0-20mADC (0-25mADC) via software (MAX22007 Device Project Settings). The MAX22007 is a TQFN-CU 56 pin device.

At the time the MAX22007 was added to EZ LADDER, it had two part numbers:

MAX22007ETN+
MAX22007ETN+T

Refer to the MAX22007 circuit's datasheet for more details on interface circuitry and other requirements.

The MAX22007 is a hardware device and requires additional circuitry and knowledge to interface it an EZ LADDER supported target. This chapter discusses the basics of using the MAX22007 in the ladder diagram and minor references to hardware when needed.

The MAX22007 SPI support is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if SPI devices are supported.

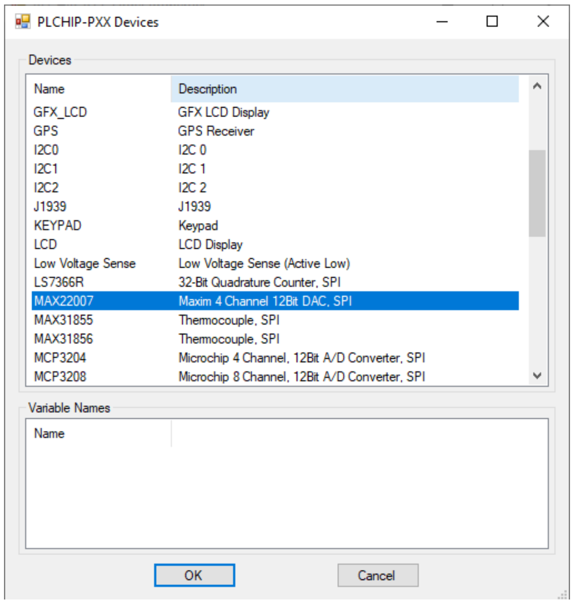
## Installing the MAX22007 in the Ladder Diagram Project

To be able to use the MAX22007 in an EZ LADDER Toolkit ladder diagram project, the MAX22007 must first be installed and configured. As the PLC on a Chip™ is the most commonly used target for the MAX22007, it will be used as an example to install and configure the MAX22007.

The MAX22007 is configured using the Project Settings. Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If the MAX22007 device were installed, it would be listed in the *Devices pane* under the Bus\SPI section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the MAX22007. Figure 15-38 shows the Target's Devices window.

The SPI port must be installed individually or no SPI ports will show available in later drop down configuration menus. For full functionality, the MAX22007 datasheet implementation of hardware must be followed.



| Name | Description |
|---|---|
| GFX_LCD | GFX LCD Display |
| GPS | GPS Receiver |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| J1939 | J1939 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX22007 | Maxim 4 Channel 12Bit DAC, SPI |
| MAX31855 | Thermocouple, SPI |
| MAX31856 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| MCP3208 | Microchip 8 Channel, 12Bit A/D Converter, SPI |

**Figure 15-38**

Click **OK**. The MAX22007 Properties dialog will open. See Figure 15-39.

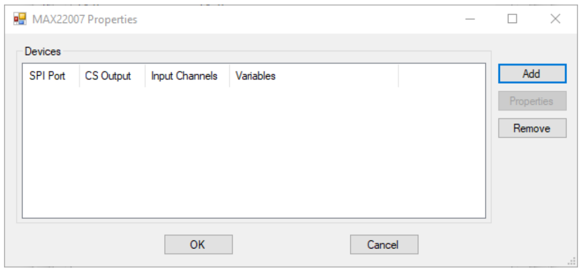
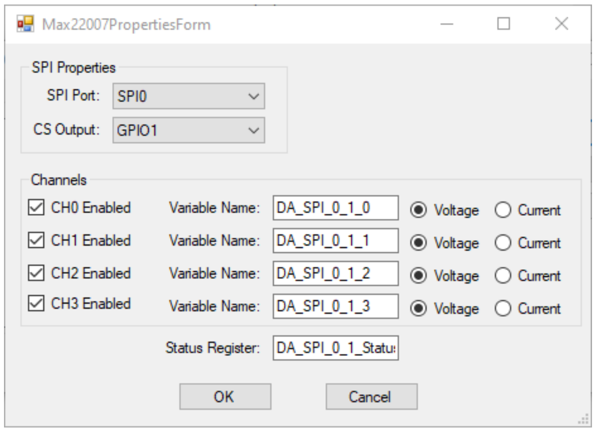


**Figure 15-39**

Click **ADD**. The MAX22007PropertiesForm dialog will open. This box is used to select the SPI port, the CS (chip select), enable the channels and select their operating mode (current or voltage).

The variable names are automatically created when the channel is enabled, but can be changed in the field boxes.  See Figure 14-40.


Max22007PropertiesForm

SPI Properties

SPI Port: SPI0

CS Output: GPIO1

Channels

| | | | |
|---|---|---|---|
| ☑ CH0 Enabled | Variable Name: DA_SPI_0_1_0 | ● Voltage | ○ Current |
| ☑ CH1 Enabled | Variable Name: DA_SPI_0_1_1 | ● Voltage | ○ Current |
| ☑ CH2 Enabled | Variable Name: DA_SPI_0_1_2 | ● Voltage | ○ Current |
| ☑ CH3 Enabled | Variable Name: DA_SPI_0_1_3 | ● Voltage | ○ Current |

Status Register: DA_SPI_0_1_Statu

**Figure 15-40**

Enable the Channels by using the check boxes.
Select either Voltage or Current by using the Radio butons.
The variable names can be changed if desired.

Once the settings are complete, click **OK**. On the MAX22007 Properties dialog, click **OK** (provided you aren't adding additional MAX22007's). Click **OK** to close the PLCHIP-PXX Target Properties window and Project Settings.

The DAC (Analog outputs) are now accessible in the ladder diagram using the variable names above. Based on current or voltage, each channel will operate over it's range based on the variable value of 0-4095.

The DAC varlables will accept numbers greater than 4095, so the ladder diagram should provide sufficient range and limits. Larger numbers will cause the analog output to 'wrap' and begin from 0.

The Status variable configured in the ladder diagram provides a status of the DAC output channels on the MAX22007. Refer to the MAX22007 datasheet.

# CHAPTER 16

## I$^2$C Devices

This chapter provides basic information to understand how to install, configure and use the I2C Devices in the EZ LADDER Toolkit.

## Chapter Contents

## I$^2$C Overview

EZ LADDER Toolkit supports the use of I$^2$C Devices. These devices expand the capability of the P-Series PLC on a Chip$^{TM}$. Not all P-Series PLC on a Chip targets will support these devices. Refer to **Chapter 23 - Hardware Targets** for complete target information including supported devices and commands. Natively supported I2C devices are available to install from the Project Settings Menu. Other devices can be used with a P-Series PLC on a Chip I$^2$C bus using structured text commands, see **I$^2$C Custom Device Communications** later in this chapter.

For supported I2C devices, they must be connected to the P-Series PLC on a Chip$^{TM}$ correctly, using standard pull-up resistors (typically 1.5K Ohms). In addition, the device must be installed in EZ LADDER Toolkit.

The following I$^2$C devices are currently supported in EZ LADDER Toolkit and P-Series targets:

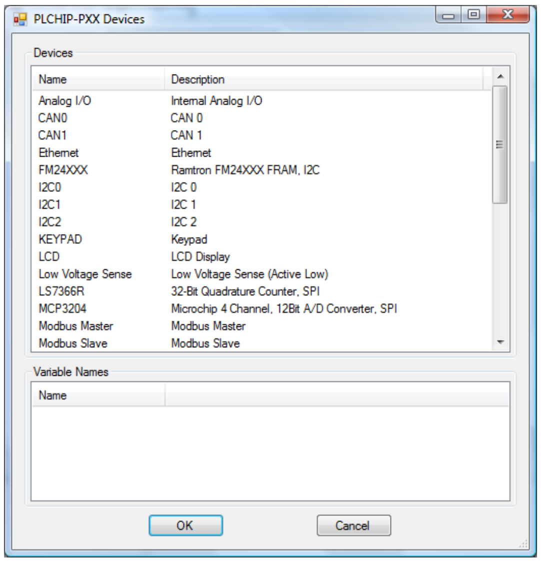
| | |
|---|---|
| **FM24XXX** | Ramtron 24xxx Series FRAM |

## Installing an I$^2$C Bus in EZ LADDER

I$^2$C bus availability is based on actual hardware targets. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if I$^2$C is supported. I$^2$C devices must be connected to the P-Series PLC on a Chip$^{TM}$ per design guidelines for proper operation using pull-up resistors.

Using the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any I$^2$C bus is installed, it will be listed in the *Devices pane* under the Bus\I2C section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the I$^2$C buses (I2C0, I2C1, etc). Figure 16-1 shows the Target's Devices window.



| Name | Description |
|---|---|
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |

| Name |
|---|

**Figure 16-1**

Select **I2Cx** and click **OK**. The I2Cx bus will be installed. The Target's ***Devices*** window will close and the I2C bus is now listed in the *Target Properties* window in the Devices pane.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The I$^2$C bus is now installed and ready to be used and additional I$^2$C devices may be installed and used in the ladder diagram.

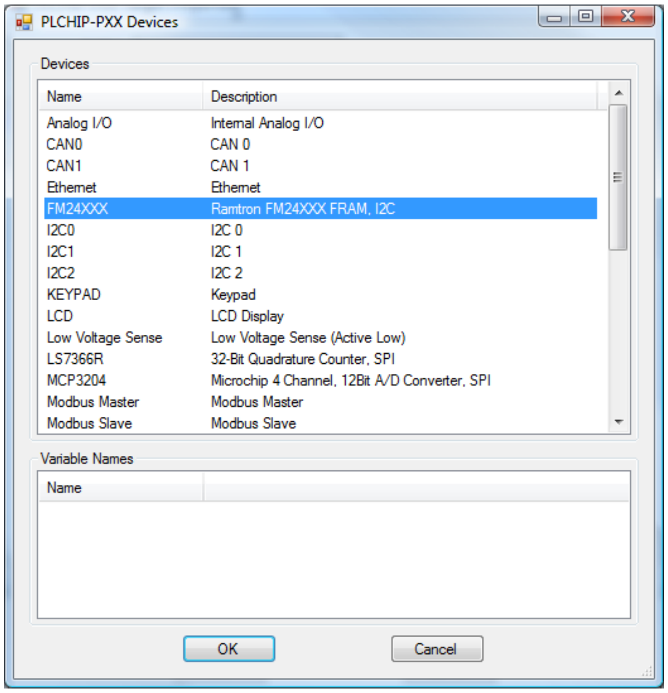
## FM24XXX RAMTRON FRAM Storage Devices

F24XXX FRAM devices are memory storage devices based on FRAM technology. These FRAM devices are the memory used for retentive memory and/or used as non-volatile (EEPROM) storage on P-Series targets. As multiple sizes are available, the xxx in the part number represents the model / size.

To install an FM24XXX FRAM device, use the **Project Menu**, choose **Settings**. The Project Settings window will open as previously covered in **Chapter 4 - Configuring Targets**.

An I$^2$C bus must be installed for the FM24XXX to install and function. If an I2C bus is not installed, see **Installing an I$^2$C bus in EZ LADDER** earlier in this chapter.

Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the FM24XXX. Figure 16-2 shows the Target's Devices window.


PLCHIP-PXX Devices

Devices

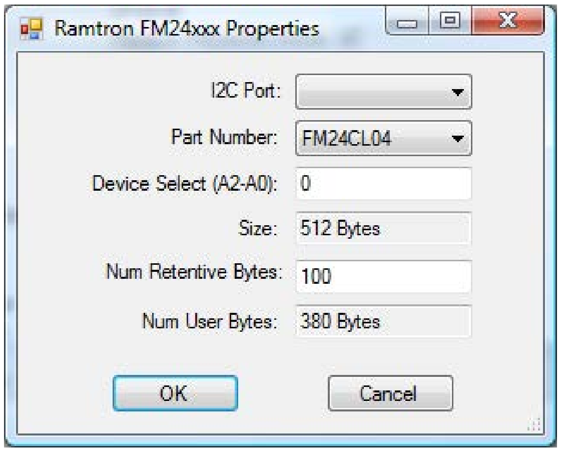
| Name | Description |
|---|---|
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |

Variable Names

Name

**Figure 16-2**

Select **FM24XXX** and click **OK**. The Ramtron FM24XXX Properties Window will open. See Figure 16-3.


Ramtron FM24xxx Properties

I2C Port:

Part Number: FM24CL04

Device Select (A2-A0): 0

Size: 512 Bytes

Num Retentive Bytes: 100

Num User Bytes: 380 Bytes

**Figure 16-3**

Using the I2C Port Drop down select box, select the I2C port that will interface to the Ramtron device. This bus should have been installed prior to this step.

Select the Ramtron Device part number from the Part Number Drop down select box.

The Device Select (0) should not be edited unless multiple devices are installed on the same bus. Consult the factory for configuring and using multiple I2C devices on one I2C port for P-Series Targets.

The Num Retentive Bytes may be adjusted to reflect the amount of retentive memory required. See **Chapter 7 - Retentive Variables** and **Chapter 4 - Configuring Targets** for more details.

Click **OK** to install the FM24XXX device and close the Target's ***Devices*** window. The FM24XXX is now installed and should appear in the Target Properties window in the Devices pane under Bus/I2C/I2Cx/.

Click **OK** to close the Target Properties window and click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The FM24XXX device is now installed and ready to be used.

## I$^2$C Custom Device Communications

EZ LADDER Toolkit supports the use of I$^2$C Devices that are not natively supported in the hardware target. These devices expand the capability of the P-Series PLC on a Chip™. Refer to **Chapter 23 - Hardware Targets** for complete target information including natively supported devices.

For supported I2C devices, they must be connected to the P-Series PLC on a Chip™ correctly, using standard pull-up resistors (typically 1.5K Ohms). In addition, the device is only accessible using structured text functions.

Two structured text functions (**EZ_I2CReadData** and **EZ_I2CWriteData**) are provided to give receive and transmit data from and to I2C devices that are not natively supported in the hardware target.

Using structured text functions for I2C Custom Device Communications requires knowledge of I2C bus architecture and an understanding of implementing I2C communications to devices with different configuration parameters. Each I2C device had individual requirements for communications.

This section and the structured text function details in Appendix B are provided as a resource of what is available in P-Series PLC on a Chip based targets and EZ LADDER Toolkit. It is not intended as a manual or guide for programming and writing code for I2C communications.

**Appendix B - Target Specific ST Function Reference** provides details and syntax on using the structured text I2C functions.

# CHAPTER 17

## Analog I/O

This chapter provides basic information on installing and using analog inputs and outputs. This chapter covers standard product and P-13 Series PLC on a Chip™ targets.

## Chapter Contents

## Analog Inputs

As analog inputs are a common requirement in today's control world, EZ LADDER Toolkit provides an easy to use interface to read analog inputs and then using the built-in function blocks, act on the analog input values.

Analog inputs provide a digital representation of an analog input signal. Analog inputs values are ranged as integers based on the resolution of the analog input. The on-board analog inputs of the P-Series PLC on a Chip™ are 12 bit resolution and the integer values that represent the signal ranges from 0 to 4095.

As the ladder diagram scans, it reads the analog signal level and digitizes it and converts it into an integer that represents it. For example, if the analog input signal can range from 0-5VDC, then the integer representation at 0V would be approximately 0 and at 5V would be approximately 4095. This integer number can then be scaled in the ladder diagram into engineering units.

The integer representation of the analog input is typically zero at the lower end (0V, 0mA, etc.) and 4095 at the high end of the scale (5VDC, 20mA). The highest allowed for the analog input resolution is hardware dependent (10 bit = 1023, 12 bit = 4095, 15 bit - 32767). SPI analog input devices may support different resolutions than the on-board 12 bit.

There are two ways to achieve analog input functionality in the EZ LADDER Toolkit hardware target.

One way is to add supported SPI bus analog input devices (integrated circuits). This requires additional hardware circuitry and interfacing. **See Chapter 15 - SPI Devices and Support** for a list of the supported devices, how to install and use them.

The seconds is using the on-board analog inputs (if supported).

All analog input support whether on-board or SPI device based are subject to the actual hardware target. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if analog inputs are supported.

### Analog Input Installation / Configuration

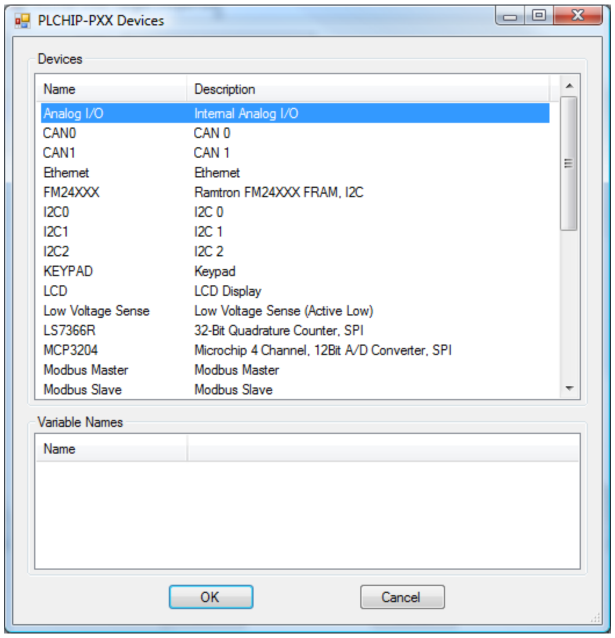
Some hardware targets require analog inputs to be installed and prior to being available in the EZ LADDER Toolkit ladder diagram project while others automatically configure the analog inputs in the EZ LADDER Toolkit when the target is selected.

Generally, off-the-shelf controllers that have analog inputs will automatically install in the EZ LADDER Toolkit and their variables are created automatically for the analog inputs.

P-Series PLC on a Chip™ targets (and others) typically require the analog inputs be installed in the Project Settings before they can be used in an ladder diagram project.

## Installing Analog Inputs for PLC on a Chip™ Targets

Select the target (PLCHIP-PXX) and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any Analog Inputs are installed, they will be listed in the *Devices pane* under the Internal/Analog I/O section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the Analog I/O. Figure 17-1 shows the Target's Devices window.



| Name | Description |
|---|---|
| Analog I/O | Internal Analog I/O |
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |

**Figure 17-1**

Select **Analog I/O** and click **OK**. The *Internal Analog I/O Properties* Window will open. See Figure 17-2. Click the **ADD INPUTS** button to add analog inputs. A new *Add Analog Inputs* dialog will open. Select the analog input channels that are required (AI0 through AI7) (holding the CTRL key will allow multiple selections) and click **OK**. The *Add Analog Inputs* dialog closes automatically and the analog inputs should now appear under the Input section of the *Internal Analog I/O Properties* window.

Click **OK** to close the Internal Analog I/O Properites window. The Analog I/O should now be listed in the Devices pane, under Internal. Click **OK** to close the Target Properties window.

Click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The analog inputs are now installed and ready to be used.

EZ LADDER Toolkit automatically creates variables that represent the analog inputs. They are labeled AI0 through AI7 for analog input 1 through analog input 8 respectively.

The P-Series PLC on a Chip Target supports one (1) on-board analog output and it also uses pin 12 (AI3). Therefore functionality on pin 12 may be configured as and analog input (AI3) or an analog output (AO0) but not both. Care should be taken when mapping analog I/O as to fit this into the requirements.

Removing Analog Inputs is accomplished in the same manner as adding inputs except the analog input(s) to remove is selected and the **REMOVE INPUTS** button is used in the Internal Analog I/O Properties window.

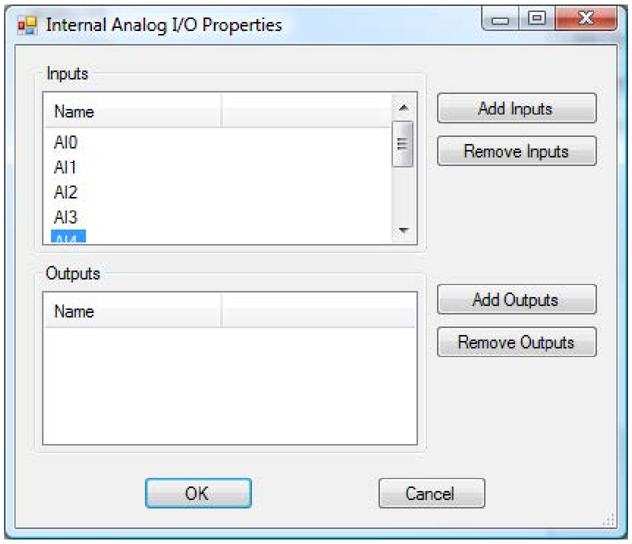


**Figure 17-2**

## Using Analog Inputs in the Ladder Diagram Project

With the hardware target selected (and analog inputs installed if required), it is now simple to use these analog input readings in the ladder diagram project.

For each analog input, an integer variable exists that will be equal to the digital representation of the real world analog input signal. Typically, this number ranges from 0-4095 with zero representing the low end (0VDC, 0mADC, etc) and 4095 representing the upper end of the range (5VDC, 20mADC, etc.).

As these variables represent the analog inputs, they can be tied directly to function blocks that have integer inputs and if necessary these variables may be converted to REAL variables using the REAL function block. Figure 17-3 shows a ladder diagram using the analog input variable AI0 as an input to a function block.

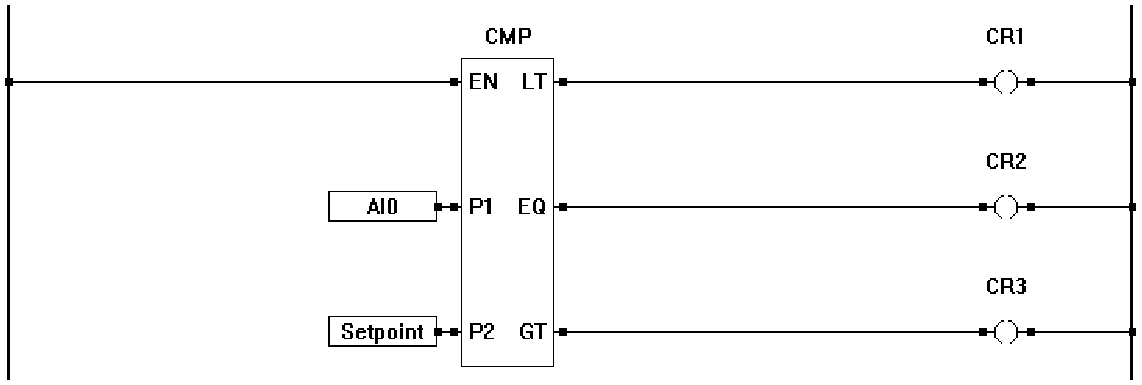


**Figure 17-3**

## Averaging Analog Input Readings

As analog signals are susceptible to many environmental factors such as noise, etc, when connected to analog inputs, the analog input variables values will change frequently. Typically, analog inputs will toggle normally +/- one bit of resolution. To minimize the effect of this bit toggle and environmental conditions, it is recommended to average each analog input.

It is recommended to use the MAVG function block (Moving Average). When placing this block, you must enter the number of samples to be averaged.

The larger the number of samples, the more RAM is used and the slower the reaction time of the block output to input changes. Size the number of samples to give the best suited reaction time and to use the least amount of RAM needed accomplish to meet the operation specifications.

Figure 17-4 illustrates an analog input being averaged by the MAVG function block.

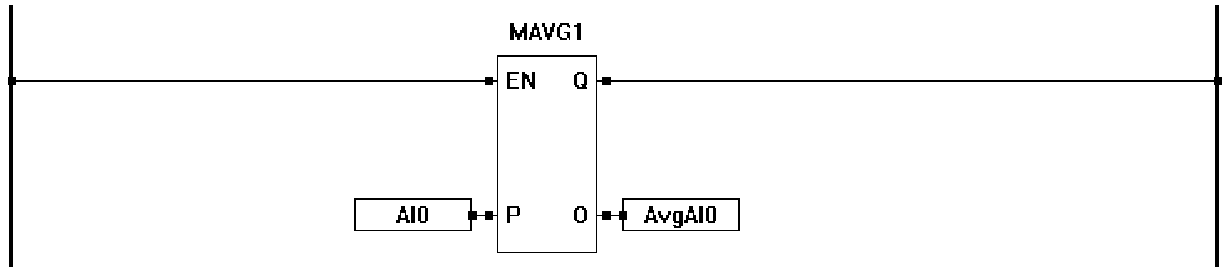


**Figure 17-4**

## Scaling Analog Input Readings

It is often desirable to scale analog input reading to match the range of some control parameter such as pressure, etc. An analog input reading can be converted to another scale by using some math and conversion function blocks.

For scaling to operate properly, the analog input sensor must be sized correctly or the scaled analog input will not truly represent the range of operation.

### Simple Scaling

If the analog input and sensor are sized accordingly (analog input 0-5VDC and the sensor = 0-100 PSI), then scaling is a simple matter. It is recommended that averaging be used prior to converting to any scale. Figure 17-5 illustrates a simple scaling circuit taking the analog input, averaging it and then converting it as above 0-100 PSI to represent 0-5VDC on the analog input.

It uses this formula:

***Scaled Reading = ((Analog Input Reading / Max Resolution) X Max Scale)***

in this case:

***Scaled Reading = ((AI0 / 4095.0) X 100)***

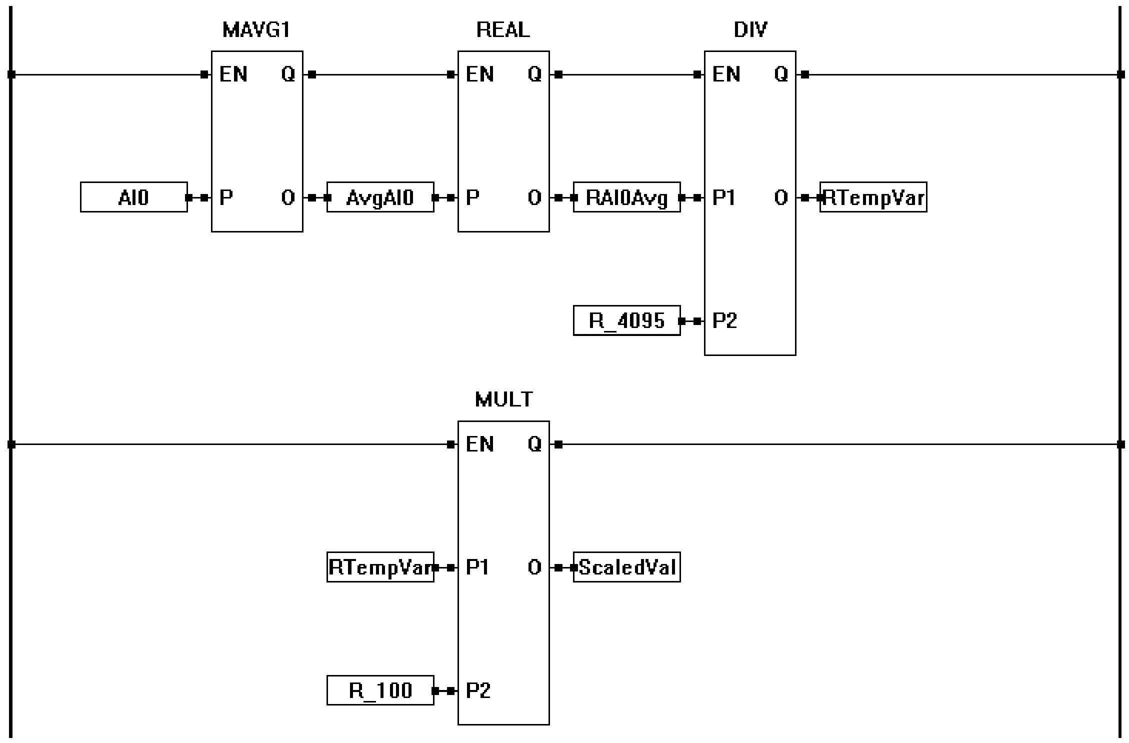


**Figure 17-5**

## Advanced Scaling

If the analog input and sensor are designed with a range that does not start at zero as in the previous example, The analog input reading is still scalable, but requires a more complex formula. See Figure 17-6. It will take an analog input and scale it to 50 to 250 PSI.

It uses this formula:

***Scaled Reading = (((Analog Input Reading / Max Resolution) X ( Range Max - Range Min)) + Range Min)***

in this case:

***Scaled Reading = (((AI0 / 4095.0) X (250 - 50)) + 50)***

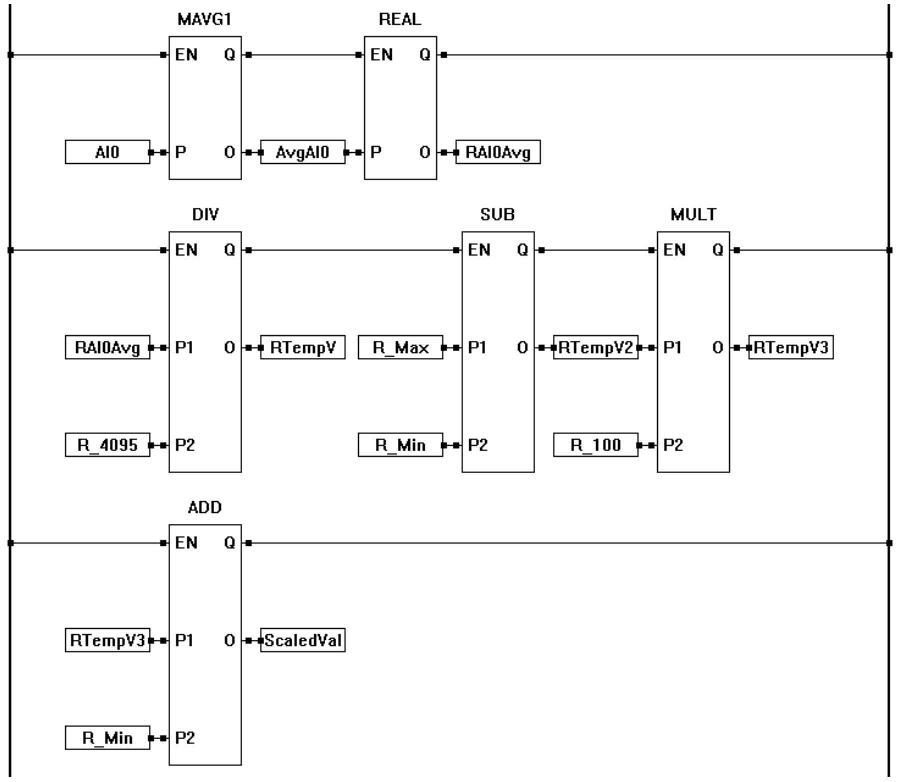


**Figure 17-6**

## Analog Outputs

As analog outputs are a common requirement in today's control world, EZ LADDER Toolkit provides an easy to use interface to control analog outputs.

Analog outputs convert a digital value into an analog signal. Analog outputs values are ranged as integers based on the resolution of the analog input. The on-board analog output of the P-Series PLC on a Chip$^{TM}$ is 10 bit resolution and the integer values that represent the signal ranges from 0 to 1023.

There are two ways to achieve analog output functionality in the EZ LADDER Toolkit hardware target.

One way is to add supported SPI bus analog output devices (integrated circuits). This requires additional hardware circuitry and interfacing. **See Chapter 15 - SPI Devices and Support** for a list of the supported devices, how to install and use them.

The seconds is using the on-board analog output(s) (if supported).

All analog output support whether on-board or SPI device based are subject to the actual hardware target. Refer to the target's User Manual or **Chapter 23 - Hardware Targets** to determine if analog outputs are supported.

The P-Series PLC on a Chip Target supports one (1) on-board analog output and it also uses pin 12 (AI3). Therefore functionality on pin 12 may be configured as and analog input (AI3) or an analog output (AO0) but not both. Care should be taken when mapping analog I/O as to fit this into the requirements.

## Installing Analog Inputs for PLC on a Chip™ Targets

Select the target (PLCHIP-PXX) and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any Analog Outputs are installed, they will be listed in the *Devices pane* under the Internal/Analog I/O section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the Analog I/O. Figure 17-7 shows the Target's Devices window.

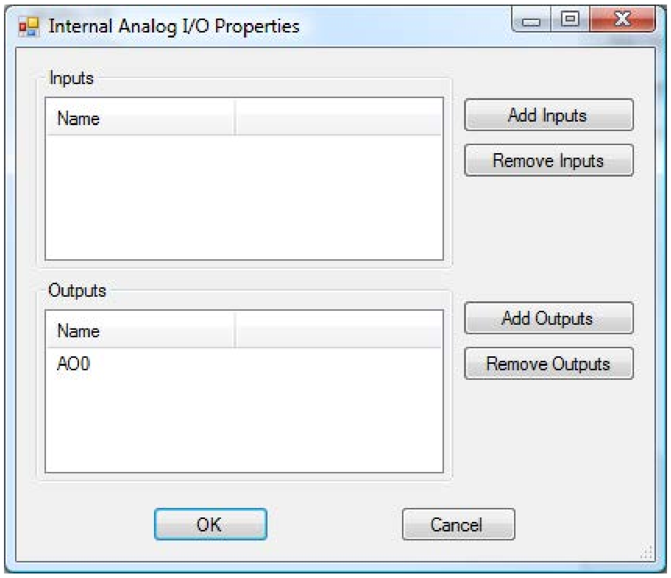


**Figure 17-7**

Select **Analog I/O** and click **OK**. The *Internal Analog I/O Properties* Window will open. See Figure 16-2. Click the **ADD OUTPUTS** button to add analog outputs. A new *Add Analog Outputs* dialog will open. Select the analog output channel(s) that are required (AOx) (holding the CTRL key will allow multiple selections if necessary) and click **OK**. The *Add Analog Outputs* dialog closes automatically and the analog outputs should now appear under the Outputs section of the *Internal Analog I/O Properties* window.

Click **OK** to close the Internal Analog I/O Properites window. The Analog I/O should now be listed in the Devices pane, under Internal. Click **OK** to close the Target Properties window.

Click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The analog output(s) are now installed and ready to be used.

EZ LADDER Toolkit automatically creates variables that represent the analog outputs. They are labeled AOx for analog outputs.

## Using Analog Outputs in the Ladder Diagram Project

With the hardware target selected (and analog outputs installed if required), it is now simple to use these analog outputs in the ladder diagram project.

For each analog output, an integer variable exists that will be the digital representation of the real world analog output signal. Typically, this number ranges from 0-1023 with zero representing the low end (0VDC, 0mADC, etc) and 1023 representing the upper end of the range (5VDC, 20mADC, etc.). Actual values and ranges are dependent upon the resolution of the analog output; in this case, the on-board analog output is 10 bit or 0-1023.

As these variables represent the analog outputs, they can be tied directly to function blocks that have integer outputs. As these function blocks *change* the variable values, the actual analog output's voltage or current will change accordingly.

Figure 17-8 shows a ladder diagram using the analog output variable AO0 as an output from a function block.

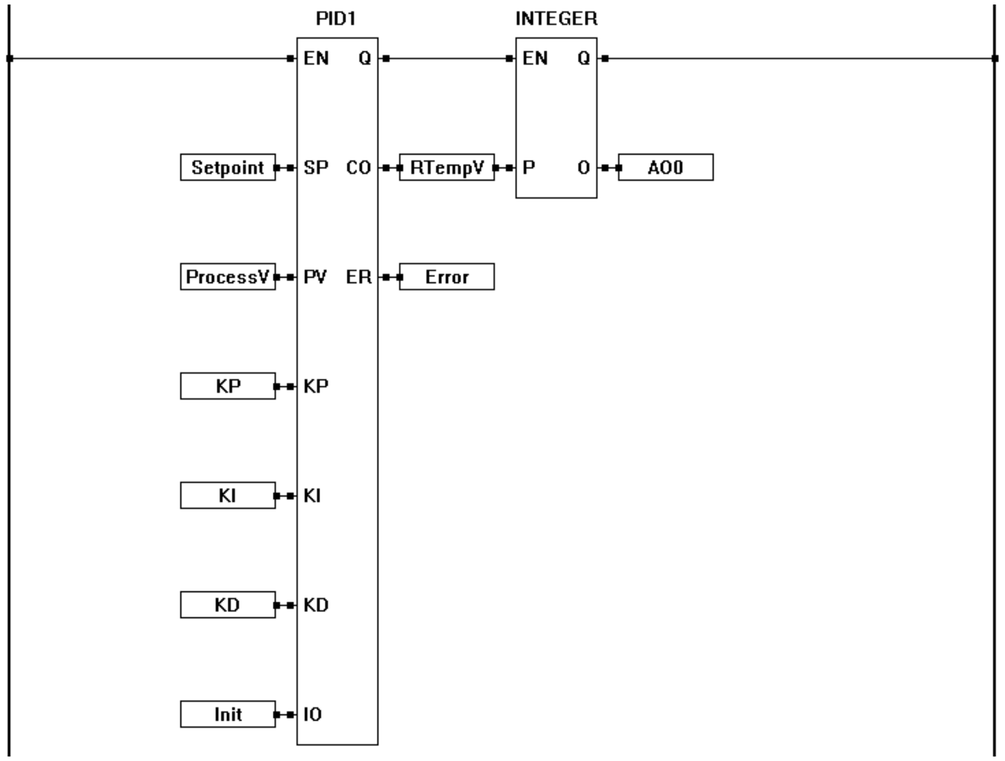


**Figure 17-8**

# CHAPTER 18

## Counters & Timers

This chapter provides basic information on installing and using on-board EZ LADDER based P-Series PLC on a Chip$^{TM}$ Timers and Counters.

## Chapter Contents

## Counter - Timer Capture Inputs

The P-Series PLC on a Chip$^{TM}$ based products provide up to three hardware capture inputs that may be configured and utilized as either timers or counter inputs. These inputs are based on actual hardware inputs and internal frequencies, therefore, are more accurate than standard software timers and counters.

Additional EZ LADDER software Counter (CTU, CTD) and Timer (TON, TOF) functions exist. See **Appendix A - Function Reference**.

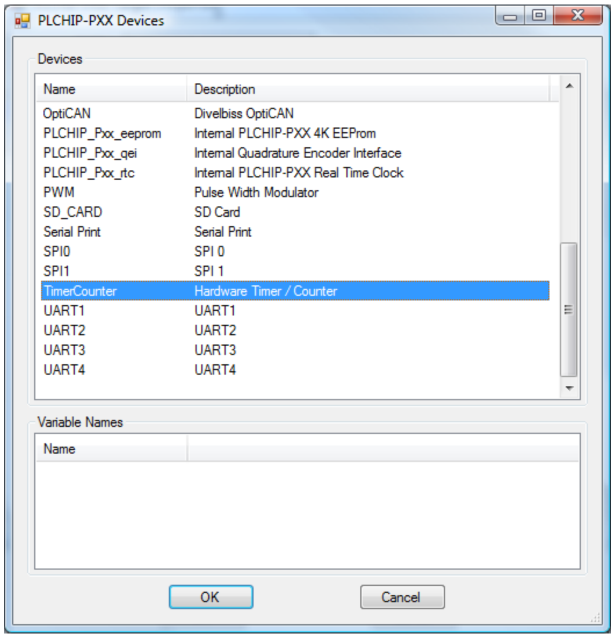
As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip$^{TM}$ target programming, the counter and timer capture inputs are covered. To determine if the actual hardware target (model) supports these inputs, refer to **Chapter 23 - Hardware Targets**.

All the counter - timer capture inputs are based off the PLC on a Chip$^{TM}$ CAPx.x pins. Depending upon the hardware target, only some or even none of the inputs may be available.

## Installing Counter - Capture Inputs

Prior to using the TIMERCOUNTER function block, the actual capture input(s) must be installed in EZ LADDER Toolkit and configured for the operational mode desired. The examples shown are for the P-Series PLC on a Chip$^{TM}$. Other targets will operate and configure similarly.

Select the target (PLCHIP-PXX) and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any of the Timer/Counter capture inputs are installed, they will be listed in the *Devices pane* under the Internal/TimerCounter section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the TimerCounter. Figure 18-1 shows the Target's Devices window.



| Name | Description |
|---|---|
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| TimerCounter | Hardware Timer / Counter |
| UART1 | UART1 |
| UART2 | UART2 |
| UART3 | UART3 |
| UART4 | UART4 |

**Figure 18-1**

Select **TimerCounter** and click **OK**. The *Timer/Counter Properties* Window will open. See Figure 18-2.

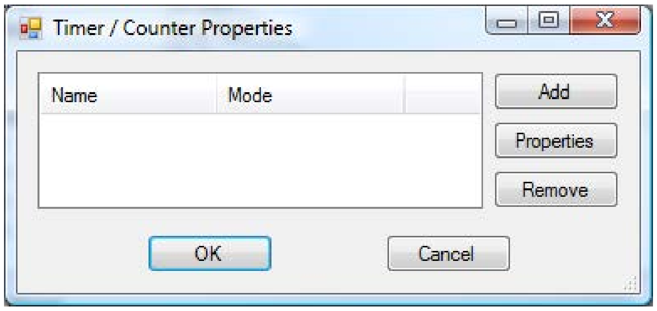


**Figure 18-2**

Click the **ADD** button to add capture inputs. A new *Select Timer / Counter Channel* dialog will open. Select the capture input channels that are required (TmrCntr0 - TmrCntr2) (holding the CTRL key will allow multiple selections) and click **OK**. See Figure 18-3.

The TmrCntrX (X=Channel) Properties for dialog will open. Using the drop down Mode menu, select the mode of operation for the timer-counter capture input. The choices are Free-running Timer, Timer or Counter. When selecting the mode as a free-running timer, no additional configuration is required. When configuring as either a timer or a counter, additional configuration is required by selecting the Timer Mode (or Counter Mode) and the Pin (capture pin). For the capture pin, refer to the target's hardware manual. See Figure 18-4.

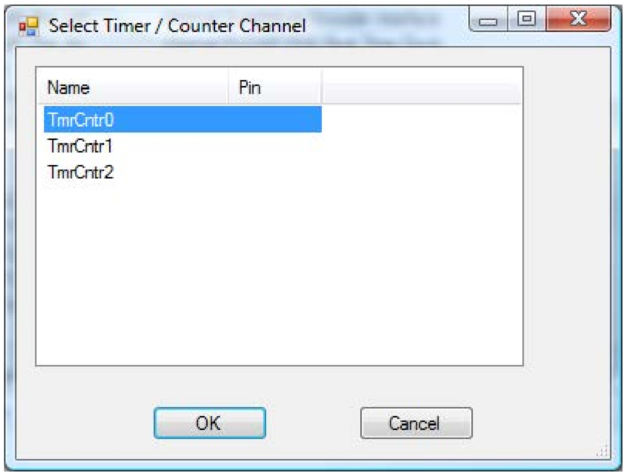
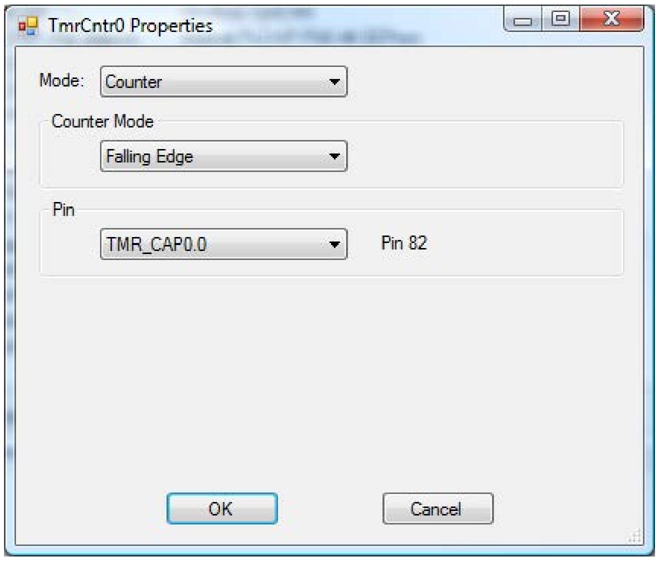


**Figure 18-3**


TmrCntr0 Properties

Mode: Counter

Counter Mode: Falling Edge

Pin: TMR_CAP0.0 Pin 82

**Figure 18-4**

Click **OK** to close the TmrCntrX Properites window. Click **OK** to close the Timer / Counter Properites window. The TimerCounter should now be listed in the Devices pane, under Internal. Click **OK** to close the Target Properties window.

Click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The timer / counter capture input(s) are now installed and ready to be used.

## Capture Inputs Configured as Timers

Capture inputs may be configured as either standard timers or free-running timers and must be configured using the Project...Settings Menu as shown earlier this chapter.

### Free-Running Timer

When configured as a free-running timer, the timer channel (capture input) may not be used for any other function and is based on a 1MHz reference clock. The TIMERCOUNTER function block is used to read the current value from the timer (capture input). See **Appendix A - Function Reference** for details on specific function blocks. When configured and the TIMERCOUNTER function block is enabled, the timer channel (capture input) will time based on 1MHz or one count for each microsecond. The timer may be reset by utilizing the reset (R) input.

When a timer capture channel reaches its maximum value, it will reset to zero and begin again. Care should be taken to not allow this to occur in the ladder diagram.

### Standard Timer (Timer)

When configured as a standard timer, the timer channel (capture input) may not be used for any other function and is based on a 24MHz clock. When using the capture input(s) as standard timers, they may be used and configured to measure either the input frequency or the period between input pulses.

The TIMERCOUNTER function block is used to read the current value from the timer (capture input). See **Appendix A - Function Reference** for details on specific function blocks. When configured and the TIMERCOUNTER function block is enabled, the timer channel (capture input) will provide either the actual input frequency on the capture input or the period between pulses on the capture input.

The timer may be reset by utilizing the reset (R) input.

## Capture Inputs Configured as Counters

Capture inputs may be configured as counters and must be configured using the Project...Settings Menu as shown earlier this chapter.

Capture inputs configured as counters may be configured to count on rising edge, falling edge or both edges. The Counter Mode (edge) is configured in the Project....Menu Settings. As timers, a input capture pin must be selected. Refer to the actual hardware target's datasheet or manual for identifying the proper capture pin.

When a counter capture channel reaches its maximum value, it will reset to zero and begin again. Care should be taken to not allow this to occur in the ladder diagram.

## Quadrature Counter Inputs

In addition to timer-counter capture inputs, P-Series targets may support quadrature counter inputs. These inputs are ideal for connecting encoders for motion control.

As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip$^{TM}$ target programming, the Quadrature Counter inputs are covered. To determine if the actual hardware target (model) supports these inputs, refer to **Chapter 23 - Hardware Targets**.

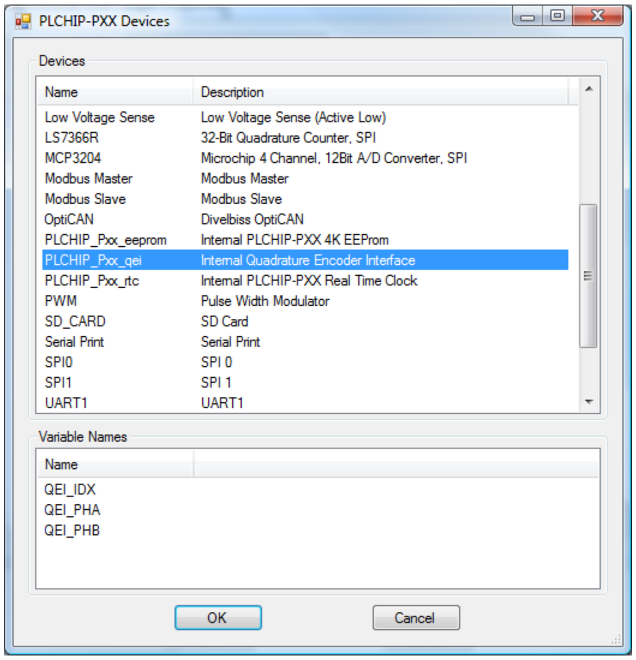
The quadrature counter consists of three inputs QEI_PHA (Phase A), QEI_PHB (Phase B) and QEI_IDX (Index). Pulses on these inputs cause internal hardware counters to count up, down or reset.

The internal hardware counters consist of the Position counter which holds the actual count of the input device connected to the A and B inputs and the Index Counter which hold the number of times the counter has passed the maximum allowed position (wrapped back to zero).

## Installing Quadrature Counter Inputs

Prior to using the quadrature counter function block(s), the actual quadrature input(s) must be installed in EZ LADDER Toolkit and configured for the operational mode desired. The examples shown are for the P-Series PLC on a Chip$^{TM}$. Other targets will operate and configure similarly.

Select the target (PLCHIP-PXX) and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If any of the Quadrature Counter inputs are installed, they will be listed in the *Devices pane* under the Internal/PLCHIP_Pxx_qei section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find the PLCHIP_Pxx_qei. Figure 18-5 shows the Target's Devices window.



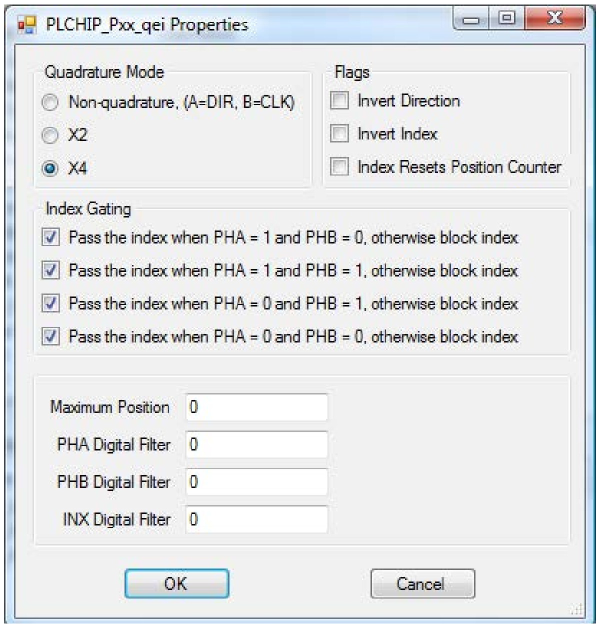
| Name | Description |
|---|---|
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PLCHIP_Pxx_rtc | Internal PLCHIP-PXX Real Time Clock |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SPI0 | SPI 0 |
| SPI1 | SPI 1 |
| UART1 | UART1 |

Variable Names

| Name |
|---|
| QEI_IDX |
| QEI_PHA |
| QEI_PHB |

**Figure 18-5**

Select **PLCHIP_Pxx_qei** and click **OK**. The *PLCHIP_Pxx_qei Properties* Window will open. See Figure 18-6.


PLCHIP_Pxx_qei Properties

Quadrature Mode
- ○ Non-quadrature, (A=DIR, B=CLK)
- ○ X2
- ● X4

Flags
- ☐ Invert Direction
- ☐ Invert Index
- ☐ Index Resets Position Counter

Index Gating
- ☑ Pass the index when PHA = 1 and PHB = 0, otherwise block index
- ☑ Pass the index when PHA = 1 and PHB = 1, otherwise block index
- ☑ Pass the index when PHA = 0 and PHB = 1, otherwise block index
- ☑ Pass the index when PHA = 0 and PHB = 0, otherwise block index

| | |
|---|---|
| Maximum Position | 0 |
| PHA Digital Filter | 0 |
| PHB Digital Filter | 0 |
| INX Digital Filter | 0 |

**Figure 18-6**

This PLCHIP_Pxx_qei properties window is used to configure specific parameters for the quadrature counter inputs.

## Quadrature Mode

The quadrature mode sets how the quadrature counter inputs will operate.

| | |
|---|---|
| **Non-quadrature** | Counter increments for each pulse on B (QEI_PHB) input. A (QEI_PHA) input sets the direction of count. |
| **X2** | X2 quadrature mode. Only two edges are counted per quadrature pulses on inputs A and B. |
| **X4** | X4 quadrature mode. All edges are counted per quadrature pulses on inputs A and B. |

## Flags

Optional flags may be configured for added versatility. These flags alter the way inputs are handled.

| | |
|---|---|
| **Invert Direction** | Inverts the count direction. |
| **Invert Index** | Inverts the active state of the index input (QEI_IDX). |
| **Index Resets Counter Position** | A pulse on the Index input (QEI_IDX) will cause the Position Counter to reset. |

### Index Gating

Index Gating flags may be configured for added versatility. These flags alter the way the index pulses are handled base on conditions of the A and B inputs (Phase A, QEI_PHA and Phase B, QEI_PHB).

Care must be taken when altering the Index Gating flags as changes may cause undesired results.

### Additional Settings

| | |
|---|---|
| **Maximum Position** | This is the maximum position for the encoder (counter) position. In the forward direction, when the position counter exceeds this value, the index counter is incremented and the position counter is set to zero. In a reverse direction, when the position counter his zero, the index counters is decremented and the Position counter is set to the this value (maximum position). |
| **PHA Digital Filter** | Sampling counter for digital input filter for Phase A Input. If set to zero, the filter is disabled. When not zero, the value is the number of sample clocks that the input signal must remain in a new state (high/low) for the new state to be seen] as a valid state. The sample clock is 120MHz (8.33333ns). |
| **PHB Digital Filter** | Sampling counter for digital input filter for Phase B Input. If set to zero, the filter is disabled. When not zero, the value is the number of sample clocks that the input signal must remain in a new state (high/low) for the new state to be seen] as a valid state. The sample clock is 120MHz (8.33333ns). |
| **INX Digital Filter** | Sampling counter for digital input filter for Index Input. If set to zero, the filter is disabled. When not zero, the value is the number of sample clocks that the input signal must remain in a new state (high/low) for the new state to be seen] as a valid state. The sample clock is 120MHz (8.33333ns). |

Click **OK** to close the PLCHIP_Pxx_qei Properites window. The PLCHIP_Pxx_qei should now be listed in the Devices pane, under Internal. Click **OK** to close the Target Properties window.

Click **OK** to close the Project Settings window.  Be sure to save the ladder diagram project. The quadrature counter inputs are now installed and ready to be used.

## Using the Quadrature Counter Inputs

With the quadrature counter inputs installed, they may be used in a ladder diagram. Three function blocks are provided that access the quadrature inputs: CNTR_PXX_QEI, CNTR_PXX_QEI_CMP and CNTR_PXX_QEI_VEL.

Using combinations of these function blocks provide maximum versatility in the ladder diagram to handle a multitude of applications.

The CNTR_PXX_QEI function block is used to read the current count (position counter), direction of travel, status and index count with optional reset control for the position counter and index counter.

The CNTR_PXX_QEI_CMP function block is used to set internal compare registers and then monitor the compare status of these registers. The position counter and index counter each have three (3) internal hardware compare registers that this function uses to compare values input to the function block as compare values to the actual counts of the position counter and index counter.

The CNTR_PXX_QEI_VEL function block is used to calculate a velocity based on the quadrature counter inputs.

See **Appendix A - Function Reference** for details on specific function blocks.

# CHAPTER 19

## Ethernet / Wi-Fi

This chapter provides basic information on installing and using Ethernet features.

## Chapter Contents

## Ethernet Overview

P-Series PLC on a Chip™ targets may support Ethernet connectivity. This Ethernet port may be used in four ways: as the programming port using EZ LADDER Toolkit, as a communication port (using Modbus TCP), a webserver and as a VersaCloud M2M+IoT communications port to the VersaCloud M2M+IoT portals.

As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip™ target programming, Ethernet connectivity is covered. To determine if the actual hardware target (model) supports ethernet, refer to **Chapter 23 - Hardware Targets**.

## Ethernet Physical Port (Layer) Support

P-Series PLC on a Chip™ targets natively support 3 different physical (PHY) integerated circuits. These circuits supports 1, 3 and 5 Ethernet ports.

**2 Port provides one port to the PLC on a Chip and 1 External Port**

| | | | |
|---|---|---|---|
| Uses: | Texas Instrument | DP83848IVV/NOPB | 10/100 |

**3 Port provides one port to the PLC on a Chip and 2 External Ports**

| | | | |
|---|---|---|---|
| Uses: | Microchip | KSZ8873RLLI | 10/100 |

**5 Port provides one port to the PLC on a Chip and 4 External Ports**

| | | | |
|---|---|---|---|
| Uses: | MIcrochip | KSZ8775CLXIC | 10/100 |

When installing the Ethernet option in the Target Settings, EZ LADDER and the P-Series PLC on a Chip automatically identify which physical integrated circuit is connected. No additional configuration is needed, provided the physical integrated circuit has been configured, properly applied and connected per the design guidelines.

## Installing Ethernet Support

By default, Ethernet ports are not enabled or configured on P-Series hardware targets. The Ethernet port must be installed and configured using the Project....Bootloader Menu.

The first step is to open an existing program or create a new program (a simple one-rung program). This program is required to switch EZ LADDER Toolkit from the EDIT mode to the RUN mode; which is required for using the Bootloader Menu option. Make sure the serial port is configured correctly in the Project...Settings menu. Refer to **Chapter 4 - Configuring Targets** and **Chapter 5 - Creating Ladder Diagram Projects**.

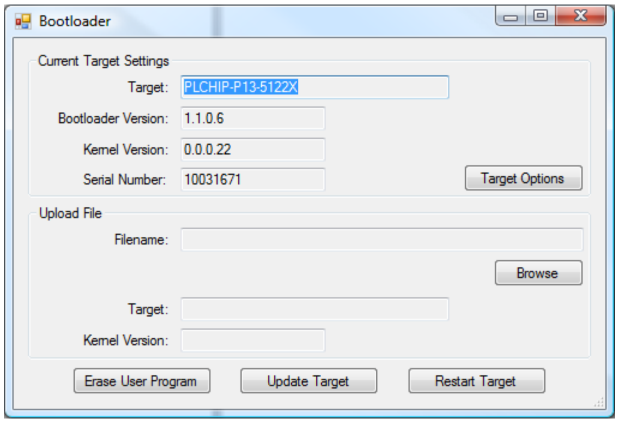
The Bootloader Menu option is only available when EZ LADDER Toolkit is in the RUN mode and connected to the actual target.

Open or create the ladder diagram project and configure the serial port using the Project...Settings menu. Click the **MONITOR** (MON) button located on the toolbar. EZ LADDER will switch from the EDIT mode to the RUN mode. The toolbars will change appropriately.

In the RUN mode, select the ***Project*** menu and the select ***Bootloader***. EZ LADDER will now attempt to connect to the hardware target and initialize it's bootloader. During this time, you may see a small status window . If the Bootloader does not respond or an error is encountered, check the serial port settings and cables.

The bootloader is a factory installed utility program on all PLC on a Chip$^{TM}$ hardware targets. This utility is used for configuring options on the target and to update or install target software kernels.

When the Bootloader utility responds, the Bootloader screen will automatically appear. See Figure 19-1.

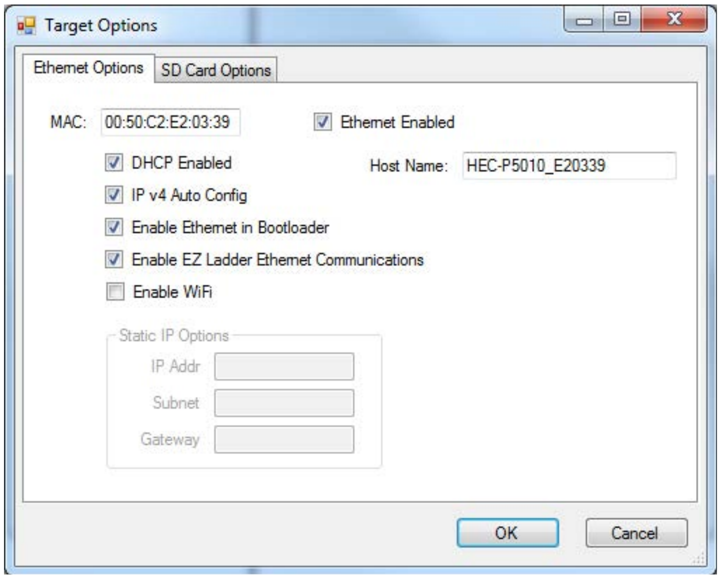

Bootloader

Current Target Settings
Target: PLCHIP-P13-5122X
Bootloader Version: 1.1.0.6
Kernel Version: 0.0.0.22
Serial Number: 10031671

Upload File
Filename:
Target:
Kernel Version:

**Figure 19-1**

Click the **TARGET OPTIONS** button. A new Target Options window will open. This window had two tabs: Ethernet Options and SD Card Options. See Figure 19-2.


Target Options

Ethernet Options | SD Card Options

MAC: 00:50:C2:E2:03:39
[✓] Ethernet Enabled
[✓] DHCP Enabled
Host Name: HEC-P5010_E20339
[✓] IP v4 Auto Config
[✓] Enable Ethernet in Bootloader
[✓] Enable EZ Ladder Ethernet Communications
[ ] Enable WiFi

Static IP Options
IP Addr
Subnet
Gateway

**Figure 19-2**

If not selected, select the **Ethernet Options** Tab. Click the Ethernet Enabled check-box to enable Ethernet.

The Ethernet configuration defaults to DHCP Enabled and IP v4 Auto Config enabled (the target gets it's IP address from the network's DHCP server. Static IP Addressing may also be configured.

To be able to connect to the Bootloader via Ethernet in the future (this screen without using serial ports), check the **Enable Ethernet in Bootloader** check box.

To be able to use the Ethernet port as the EZ LADDER programming port, check the **Enable EZ Ladder Ethernet Communications** check box.

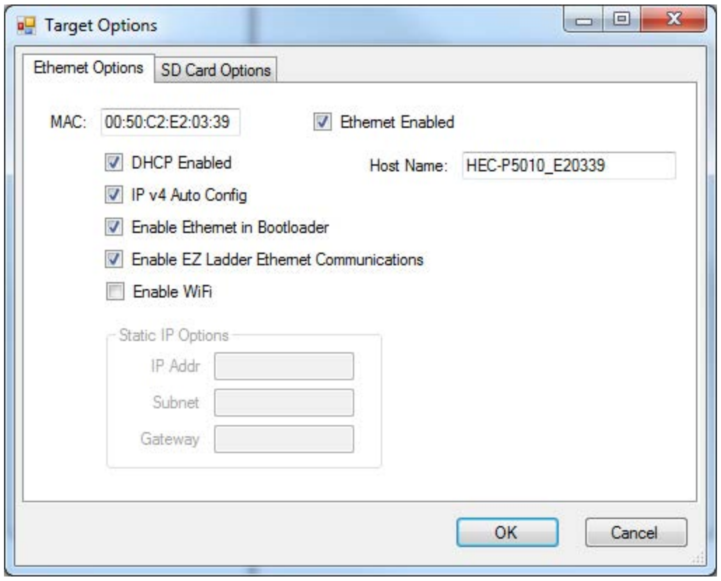
The **Enable WiFi** check box is used for models with Wi-Fi capability. Refer to the Wi-Fi section of this chapter for more information. This box should only be used for Wi-Fi models only.

P-Series PLC on a Chip based targets can support an actual Ethernet Port or Wi-Fi Connectivity individually. Both are not supported simultaneously. Internally, the PLC on a Chip itself considers the Wi-Fi connectivity as an Ethernet Port.

## DHCP Configuration

When configuring for receiving the IP address from the network, a Host name is recommended to identify the hardware target on the Ethernet network. See Figure 19-3. Enter a Host name in the *Host Name* box.

DHCP IP addressing is assigned by the network's DHCP server and therefore can be any address and is controlled (can change at will) by the DHCP server.



MAC: 00:50:C2:E2:03:39

[✓] Ethernet Enabled

[✓] DHCP Enabled

Host Name: HEC-P5010_E20339

[✓] IP v4 Auto Config

[✓] Enable Ethernet in Bootloader

[✓] Enable EZ Ladder Ethernet Communications

[ ] Enable WiFi

**Figure 19-3**

Click **OK** to accept the DHCP settings and close the window. Click **RESTART TARGET** to exit the Bootloader and force the software kernel to restart, accepting the changes. The Ethernet port is now configured to operate as the hardware target's programming port with the network's DHCP.

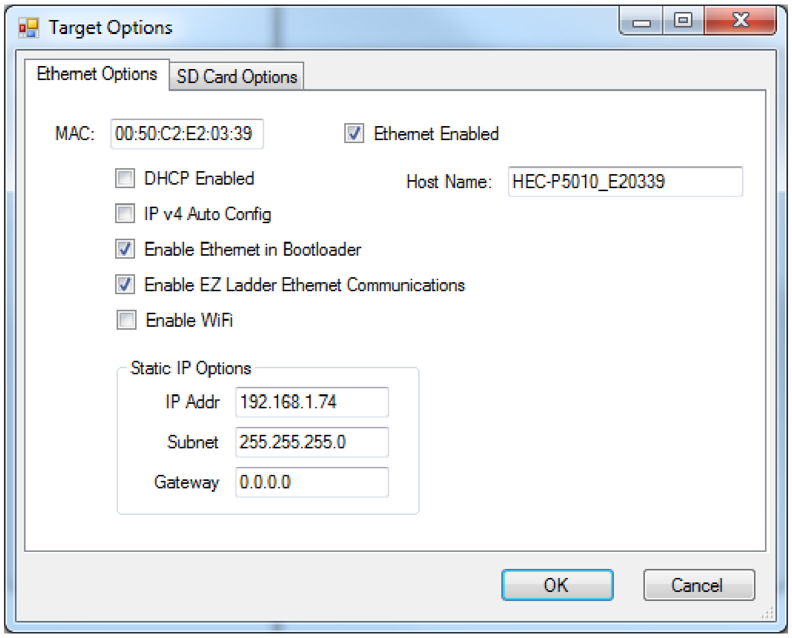
The MAC: field is factory set and should not be changed.

When configured for DHCP and no DHCP server is found (time-out), the port will revert to a default Auto-IP address

## Static IP Configuration

To configure Ethernet for a static IP mode, de-select the DHCP Enabled and IP v4 Auto Config check-boxes. Additional configuration items are required. See Figure 19-4.

When using the static IP mode, a Host Name is recommended to identify the hardware target on the Ethernet network.



MAC: 00:50:C2:E2:03:39

[✓] Ethernet Enabled

[ ] DHCP Enabled

Host Name: HEC-P5010_E20339

[ ] IP v4 Auto Config

[✓] Enable Ethernet in Bootloader

[✓] Enable EZ Ladder Ethernet Communications

[ ] Enable WiFi

Static IP Options

IP Addr 192.168.1.74

Subnet 255.255.255.0

Gateway 0.0.0.0

**Figure 19-4**

The IP Address (IP Addr), Subnet (Subnet) and Gateway (Gateway) information must be entered into the appropriate boxes. Enter the information.

Generally, the Subnet (mask) is 255.255.255.0; however, this setting as with the IP Address and Gateway is network specific. This information may be obtained from you network administrator.

Click **OK** to accept the static IP settings and close the window. Click **RESTART TARGET** to exit the Bootloader and force the software kernel to restart, accepting the changes. The Ethernet port is now configured to operate as the hardware target's programming port with a static IP address.

The MAC: field is factory set and should not be changed.

## Wi-Fi Overview

P-Series PLC on a Chip$^{TM}$ targets may support Wi-Fi connectivity. This Wi-Fi connectivity may be used in three ways: as the programming port using EZ LADDER Toolkit , as a communication port (using Modbus TCP), a webserver and as a VersaCloud M2M+IoT communications port to VersaCloud M2M+IoT portals.

As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip$^{TM}$ target programming, Wi-Fi connectivity is covered. To determine if the actual hardware target (model) supports Wi-Fi, refer to **Chapter 23 - Hardware Targets**.

Wi-Fi may be configured to operate in Client or Host Mode. Client mode is the default mode where the Wi-Fi connects to a Wi-Fi network or router. The Host Mode configures the Wi-Fi to accept connections from other Wi-Fi enabled devices like phones or tablets. This is typically used with the Webserver feature.

## Installing Wi-Fi Support

By default, Wi-Fi connectivity is not enabled or configured on P-Series hardware targets. For all P-Series based PLC on a Chip targets, the PLC on a Chip itself treats the Wi-Fi connectivity as an Ethernet port. To use Wi-Fi, the Ethernet port and Wi-Fi options must be installed and configured using the Project....Bootloader Menu.

The first step is to open an existing program or create a new program (a simple one-rung program). This program is required to switch EZ LADDER Toolkit from the EDIT mode to the RUN mode; which is required for using the Bootloader Menu option. Make sure the serial port is configured correctly in the Project...Settings menu. Refer to **Chapter 4 - Configuring Targets** and **Chapter 5 - Creating Ladder Diagram Projects**.

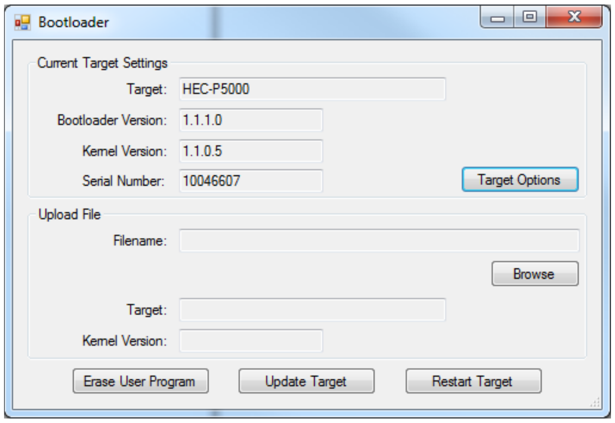
The Bootloader Menu option is only available when EZ LADDER Toolkit is in the RUN mode and connected to the actual target.

Open or create the ladder diagram project and configure the serial port using the Project...Settings menu. Click the **MONITOR** (MON) button located on the toolbar. EZ LADDER will switch from the EDIT mode to the RUN mode. The toolbars will change appropriately.

In the RUN mode, select the ***Project*** menu and the select ***Bootloader***. EZ LADDER will now attempt to connect to the hardware target and initialize it's bootloader. During this time, you may see a small status window . If the Bootloader does not respond or an error is encountered, check the serial port settings and cables.

The bootloader is a factory installed utility program on all PLC on a Chip™ hardware targets. This utility is used for configuring options on the target and to update or install target software kernels.

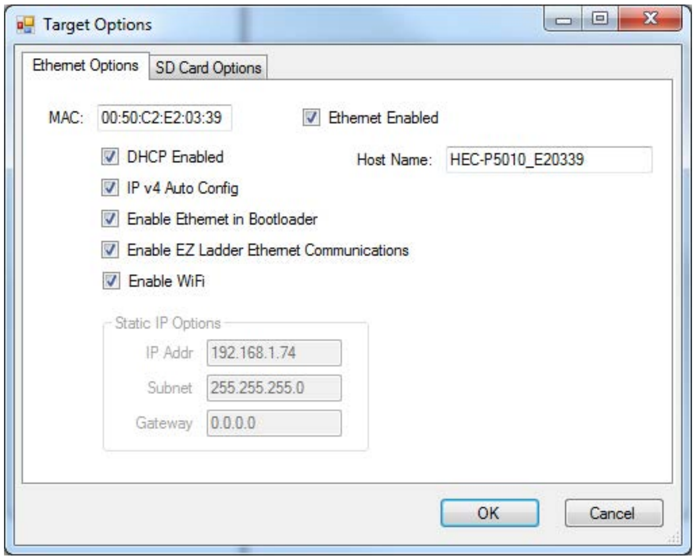
When the Bootloader utility responds, the Bootloader screen will automatically appear. See Figure 19-5.



Target: HEC-P5000
Bootloader Version: 1.1.1.0
Kernel Version: 1.1.0.5
Serial Number: 10046607
Filename:
Target:
Kernel Version:

**Figure 19-5**

Click the **TARGET OPTIONS** button. A new Target Options window will open. This window had two tabs: Ethernet Options and SD Card Options. See Figure 19-6.



MAC: 00:50:C2:E2:03:39
[✓] Ethernet Enabled
[✓] DHCP Enabled
Host Name: HEC-P5010_E20339
[✓] IP v4 Auto Config
[✓] Enable Ethernet in Bootloader
[✓] Enable EZ Ladder Ethernet Communications
[✓] Enable WiFi

Static IP Options
IP Addr 192.168.1.74
Subnet 255.255.255.0
Gateway 0.0.0.0

**Figure 19-6**

If not selected, select the **Ethernet Options** Tab. Click the Ethernet Enabled check-box to enable Ethernet.

The Ethernet configuration defaults to DHCP Enabled and IP v4 Auto Config enabled (the target gets it's IP address from the network's DHCP server. Static IP Addressing may also be configured.

To be able to connect to the Bootloader via Ethernet (Wi-Fi) in the future (this screen without using serial ports), check the **Enable Ethernet in Bootloader** check box.

To be able to use the Ethernet port (Wi-Fi) as the EZ LADDER programming port, check the **Enable EZ  Ladder Ethernet Communications** check box.

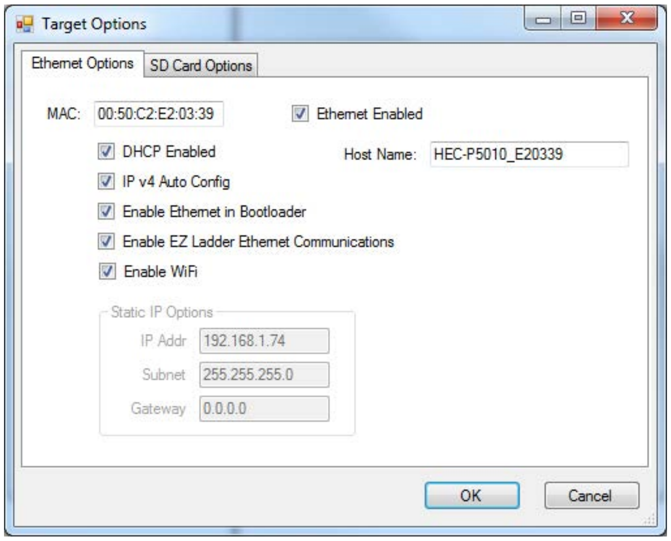
Check the Enable WiFi checkbox. This enables the Wi-Fi and configures it to be treated as an Ethernet port from the PLC on a Chip and EZ LADDER Toolkit.

P-Series PLC on a Chip based targets can support an actual Ethernet Port or Wi-Fi Connectivity individually. Both are not supported simultaneously. Generally, the PLC on a Chip itself treats the Wi-Fi connectivity as a Ethernet Port.

## DHCP Configuration

When configuring for receiving the IP address from the network, a Host name is recommended to identify the hardware target on the Wi-Fi network. See Figure 19-7. Enter a Host name in the *Host Name* box.

DHCP IP addressing is assigned by the network's DHCP server and therefore can be any address and is controlled (can change at will) by the DHCP server.



MAC: 00:50:C2:E2:03:39
☑ Ethernet Enabled
☑ DHCP Enabled
Host Name: HEC-P5010_E20339
☑ IP v4 Auto Config
☑ Enable Ethernet in Bootloader
☑ Enable EZ Ladder Ethernet Communications
☑ Enable WiFi

Static IP Options
IP Addr 192.168.1.74
Subnet 255.255.255.0
Gateway 0.0.0.0

**Figure 19-7**

Click **OK** to accept the DHCP settings and close the window. Click **RESTART TARGET** to exit the Bootloader and force the software kernel to restart, accepting the changes. The Ethernet port and Wi-Fi connectivity is now configured to operate as the hardware target's programming port with the network's DHCP.

The MAC: field is factory set and should not be changed.

When configured for DHCP and no DHCP server is found (time-out), the port will revert to a default Auto-IP address if the IP v4 Auto Config checkbox is enabled. This setting can be altered at run-time using structured text.

## Static IP Configuration

To configure the Wi-Fi for a static IP mode, de-select the DHCP Enabled and IP v4 Auto Config check-boxes. Additional configuration items are required. See Figure 19-8.

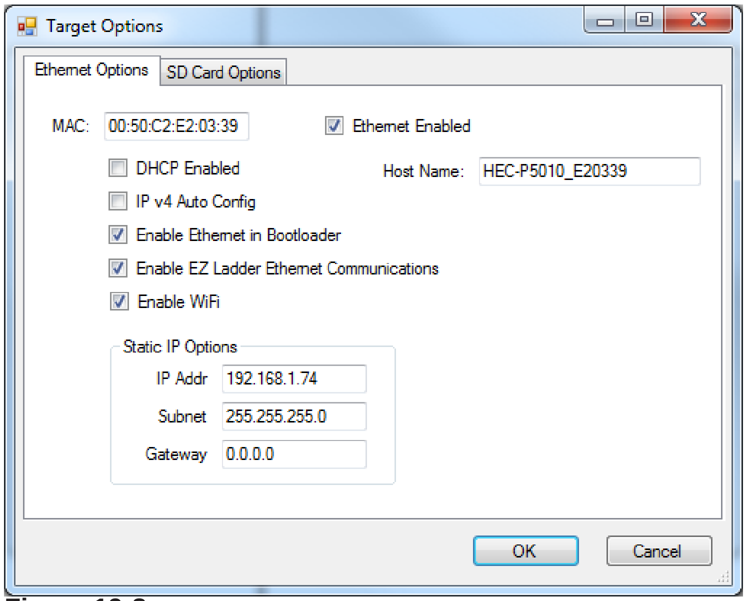
When using the static IP mode, a Host Name is recommended to identify the hardware target on the Ethernet network.

The IP Address (IP Addr), Subnet (Subnet) and Gateway (Gateway) information must be entered into the appropriate boxes. Enter the information.

Generally, the Subnet (mask) is 255.255.255.0; however, this setting as with the IP Address and Gateway is network specific. This information may be obtained from you network administrator.

Click **OK** to accept the static IP settings and close the window. Click **RESTART TARGET** to exit the Bootloader and force the software kernel to restart, accepting the changes. The Ethernet port / Wi-Fi connectivity is now configured to operate as the hardware target's programming port with a static IP address.

The MAC: field is factory set and should not be changed.



MAC: 00:50:C2:E2:03:39
[✓] Ethernet Enabled
Host Name: HEC-P5010_E20339
[ ] DHCP Enabled
[ ] IP v4 Auto Config
[✓] Enable Ethernet in Bootloader
[✓] Enable EZ Ladder Ethernet Communications
[✓] Enable WiFi

Static IP Options

| | |
|---|---|
| IP Addr | 192.168.1.74 |
| Subnet | 255.255.255.0 |
| Gateway | 0.0.0.0 |

**Figure 19-8**

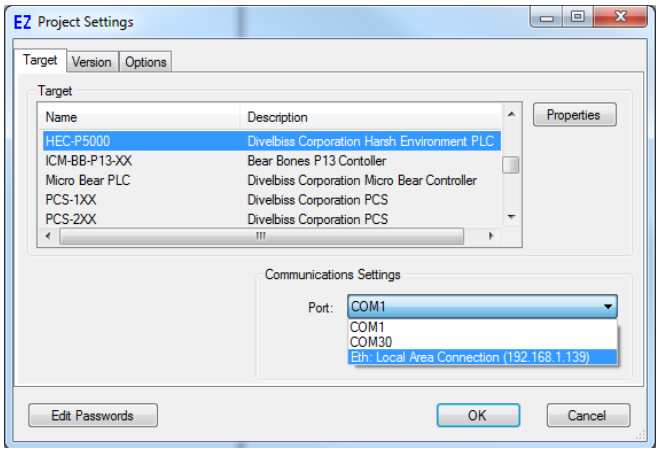
## Ethernet / Wi-Fi as the Programming Port

By completing the configuration of the Bootloader Ethernet / Wi-Fi settings (DHCP or Static), the Ethernet port or Wi-Fi connectivity (model dependent) is configured to be used as the hardware target's Programming Port for configuration and downloading ladder diagram programs.

For Modbus functionality, additional configuration items are required.

Before a target's Wi-Fi connectivity may be used as the programming port, the Ethernet and Wi-Fi must be enabled in the Bootloader and the Wi-Fi must be configured to connect to a network with all the setup and credentials required to do so. See / follow the steps in **Connecting to a Wi-Fi Network** later in this chapter.

To use the Ethernet port as the programming port, in the ladder diagram's Project Settings, select Eth:XXXX from the Communications Settings drop-down box to configure the Ethernet as the port to use. See Figure 19-9. The Eth:XXXX is the selected is for the Ethernet / Wi-Fi IP address of the unit to connect to and program. This IP address was set in the Bootloader or is DHCP.


EZ Project Settings

Target | Version | Options

Target

| Name | Description |
|---|---|
| HEC-P5000 | Divelbiss Corporation Harsh Environment PLC |
| ICM-BB-P13-XX | Bear Bones P13 Contoller |
| Micro Bear PLC | Divelbiss Corporation Micro Bear Controller |
| PCS-1XX | Divelbiss Corporation PCS |
| PCS-2XX | Divelbiss Corporation PCS |

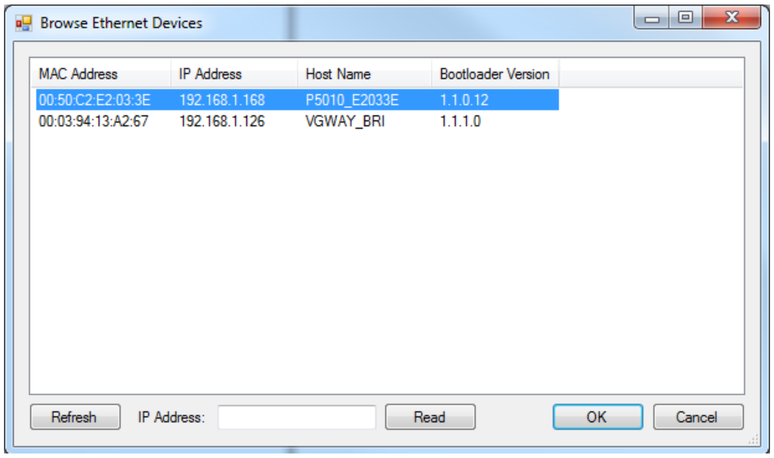
Communications Settings

Port: COM1

COM1
COM30
Eth: Local Area Connection (192.168.1.139)

**Figure 19-9**

When connecting to the target (in RUN mode), a dialog box (Browse Ethernet Devices) will appear listing all the hardware targets found on the Ethernet / Wi-Fi network. Select the target to use and click **OK**. See Figure 19-10.


Browse Ethernet Devices

| MAC Address | IP Address | Host Name | Bootloader Version |
|---|---|---|---|
| 00:50:C2:E2:03:3E | 192.168.1.168 | P5010_E2033E | 1.1.0.12 |
| 00:03:94:13:A2:67 | 192.168.1.126 | VGWAY_BRI | 1.1.1.0 |

IP Address:

**Figure 19-10**

## Ethernet / Wi-Fi Connectivity as Modbus Port

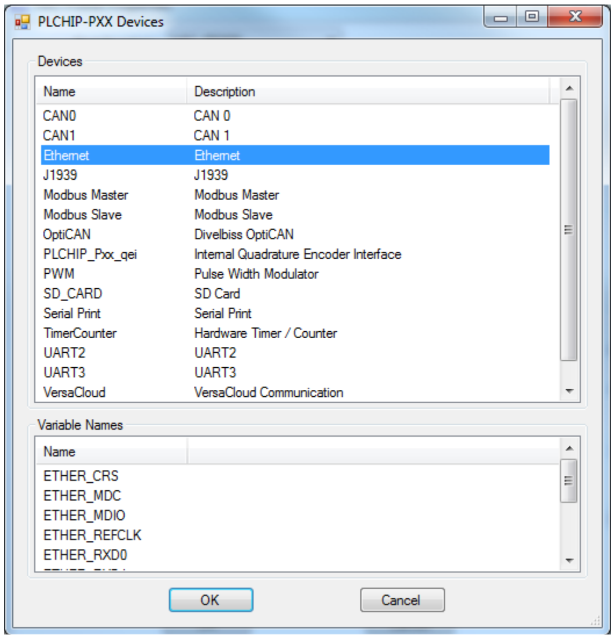
The Ethernet port / Wi-Fi connectivity may also be used as a Modbus communications port (Modbus Master or Slave - Modbus TCP). When using the Ethernet port as a Modbus communications port, additional configuration is required.

! As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip$^{TM}$ target programming, Ethernet / Wi-Fi connectivity is covered. To determine if the actual hardware target (model) supports Ethernet or Wi-Fi connectivity, refer to **Chapter 23 - Hardware Targets**

Prior to using the Ethernet port or Wi-Fi connectivity for Modbus communications, the Ethernet port must be installed in the EZ LADDER project (this is different than previously installed using the Bootloader). This is required for Wi-Fi as the PLC on a Chip treats Wi-Fi as an Ethernet port. It is installed using the Projects.... Settings Menu.

Before a target's Wi-Fi connectivity may be used as Modbus TCP port, the Ethernet and Wi-Fi must be enabled in the Bootloader and the Wi-Fi must be configured to connect to a network with all the setup and credentials required to do so. See / follow the steps in **Connecting to a Wi-Fi Network** later in this chapter.

Select the target (PLCHIP-PXX) and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. If Ethernet Port is installed, they will be listed in the *Devices pane* under the Internal/Ethernet section. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and find Ethernet. Figure 19-11 shows the Target's Devices window.



| Name | Description |
|---|---|
| CAN0 | CAN 0 |
| CAN1 | CAN 1 |
| Ethernet | Ethernet |
| J1939 | J1939 |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |
| UART3 | UART3 |
| VersaCloud | VersaCloud Communication |

| Name |
|---|
| ETHER_CRS |
| ETHER_MDC |
| ETHER_MDIO |
| ETHER_REFCLK |
| ETHER_RXD0 |

**Figure 19-11**

Select **Ethernet** and click **OK.** The *Target Properties* window. Ethernet should now be listed in the Devices pane, under Internal. Click **OK** to close the Target Properties window.

Click **OK** to close the Project Settings window. Be sure to save the ladder diagram project. The Ethernet port / Wi-Fi connectivity is now installed and ready to be used for Modbus communication.

The Ethernet / Wi-Fi is now installed, but will require additional configuration prior to using Modbus. Refer to **Chapter 13 - Modbus Networking** for details on Modbus communication, Modbus function blocks and selecting/configuring Ethernet / Wi-Fi for use with Modbus.

## Connecting to a Wi-Fi Network (Client Mode)

With Wi-Fi configured in the Bootloader (for targets that support Wi-Fi connectivity), additional steps must be taken to connect the target to a Wi-Fi network (the most common configuration). These steps include selecting the Wi-Fi (wireless) network, entering an SSID and an access password.

Before Wi-Fi connectivity may be used as the programming port for the first time, the Wi-Fi network must be selected and configured using EZ LADDER Toolkit with serial port connection to the target. This configuration includes selecting the network, entering the SSID and the Password.

### To configure for a Wi-Fi network

1. Open a program or create a simple program with the target info.
2. Change EZ LADDER to the Monitor mode by clicking the MON button.
3. Make sure the target is connected to the computer and click the button to connect EZ LADDER Toolkit to the hardware target.
4. From the menu at the top, select Project then select WiFi Setup. Refer to Figure 19-12.

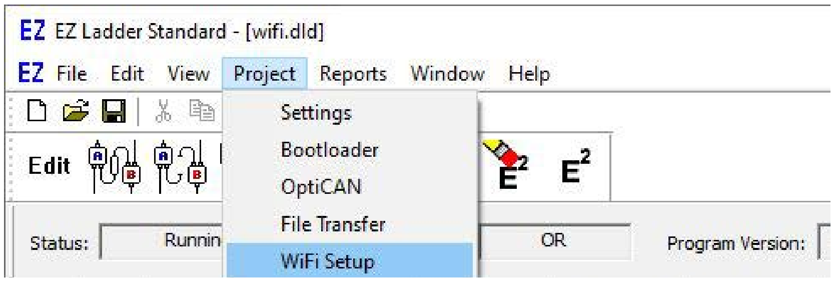
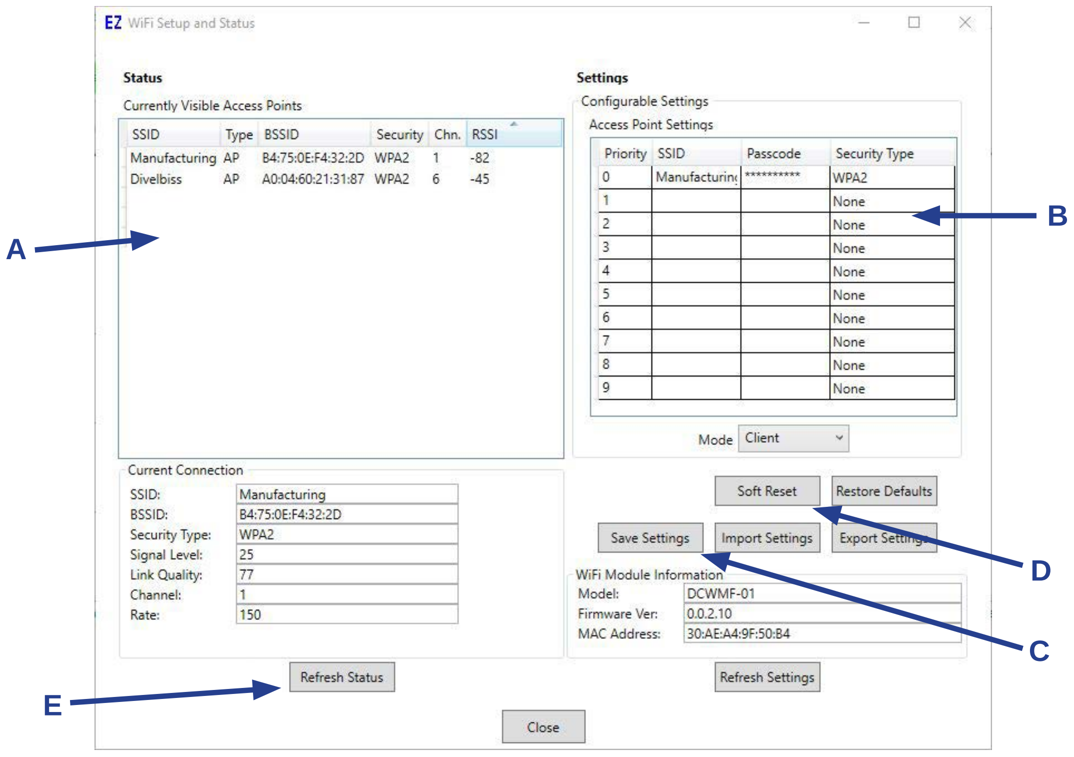


**Figure 19-12**

5. The ***WiFi Setup and Status*** window will open. An intermediate temporary dialog may be seen while the setup is accessed. Refer to Figure 19-13.
6. The ***Mode*** must be set to Client. For host mode, see *Configuring and Using Wi-Fi (Host Mode)* later in this chapter.
7. Referring to Figure 19-13, the *Currently Visible Access Points* (item A) pane shows all the networks currently in-range for the Wi-Fi to detect. **The network must be in-range to be configured**.
8. In the Access Points Settings, enter the **SSID** and **Passcode** in their respective places (item B). It will be necessary to double-click to enter the values. Refer to Figure 19-13.
9. Select the **Security Type** for the network (item B).
10. With the information entered, click the **SAVE SETTINGS** button (Item C) to save the current settings for the Wi-Fi network.



**Status**

Currently Visible Access Points

| SSID | Type | BSSID | Security | Chn. | RSSI |
|---|---|---|---|---|---|
| Manufacturing | AP | B4:75:0E:F4:32:2D | WPA2 | 1 | -82 |
| Divelbiss | AP | A0:04:60:21:31:87 | WPA2 | 6 | -45 |

Current Connection

| | |
|---|---|
| SSID: | Manufacturing |
| BSSID: | B4:75:0E:F4:32:2D |
| Security Type: | WPA2 |
| Signal Level: | 25 |
| Link Quality: | 77 |
| Channel: | 1 |
| Rate: | 150 |

**Settings**

Configurable Settings

Access Point Settings

| Priority | SSID | Passcode | Security Type |
|---|---|---|---|
| 0 | Manufacturin | ********** | WPA2 |
| 1 | | | None |
| 2 | | | None |
| 3 | | | None |
| 4 | | | None |
| 5 | | | None |
| 6 | | | None |
| 7 | | | None |
| 8 | | | None |
| 9 | | | None |

Mode: Client

WiFi Module Information

| | |
|---|---|
| Model: | DCWMF-01 |
| Firmware Ver: | 0.0.2.10 |
| MAC Address: | 30:AE:A4:9F:50:B4 |

**Figure 19-13**

Multiple Wi-Fi networks may be saved by adding them to list shown in Figure 19-13. Each setting is stored in the hardware target and is maintained during a power loss. The priority of Wi-Fi network to connect to is based on the priority number in the list.

Up to 10 SSID / Passwords may be saved on the on-board Wi-Fi module. The module searches through the list for an in-range SSIDs (APs) and attempts to connect with it. When removing (deleting) SSIDs, the list should be edited as all remaining SSIDs are listed beginning with the top and leaving no empty spaces in the list. When operating, as the module searches the list in order, if an empty location is detected, the module will stop searching for an SSID match. There should be no empty locations except at the end of the list (if less than 10 entries).

11. Click the **SOFT RESET** button (Item D). This forces the Wi-Fi connectivity to reset. After the reset, the target should connect to the Wi-Fi network.

12. Click the **REFRESH STATUS** button (Item E). The information under the Current Connection should update and show the network currently connected to.

13. Click **CLOSE** to close the WiFi Setup and Status window.

The Wi-Fi connectivity is now configured and connected to Wi-Fi network and can be used as the programming port, for Modbus TCP or webserver. VersaCloudM2M+IoT communications requires additional configuration in the project settings.

Wi-Fi connectivity depends upon the target being in range, with sufficient signal strength and being configured properly for communications over the Wi-Fi network.

## Configuring and Using Wi-Fi (Host Mode)

With Wi-Fi configured in the Bootloader (for targets that support Wi-Fi connectivity), the Wi-Fi may be configured in Host mode. This mode allows configures the on-board Wi-Fi module to accept connections from other smart devices (phones, tablets, computers). The on-board Wi-Fi module assigns the IP address to the connected device(s). This is useful for stand-alone applications using a webserver with local connections for data retrieval.

Wi-Fi can only operate in either Client or Host mode. It cannot operate in both modes.

### To configure for Host Mode

1. Open a program or create a simple program with the target info.

2. Change EZ LADDER to the Monitor mode by clicking the MON button.

3. Make sure the target is connected to the computer and click the button to connect EZ LADDER Toolkit to the hardware target.

4. From the menu at the top, select Project then select WiFi Setup. Refer to Figure 19-14.

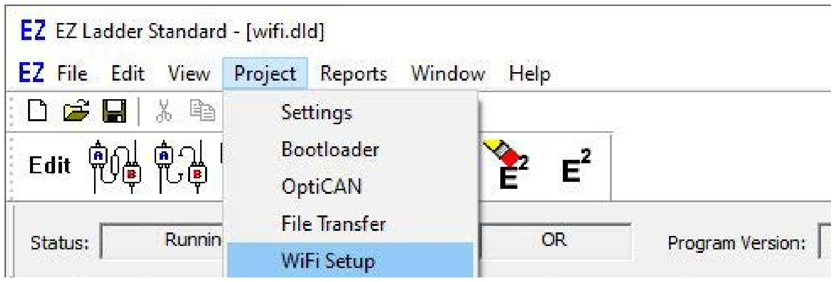
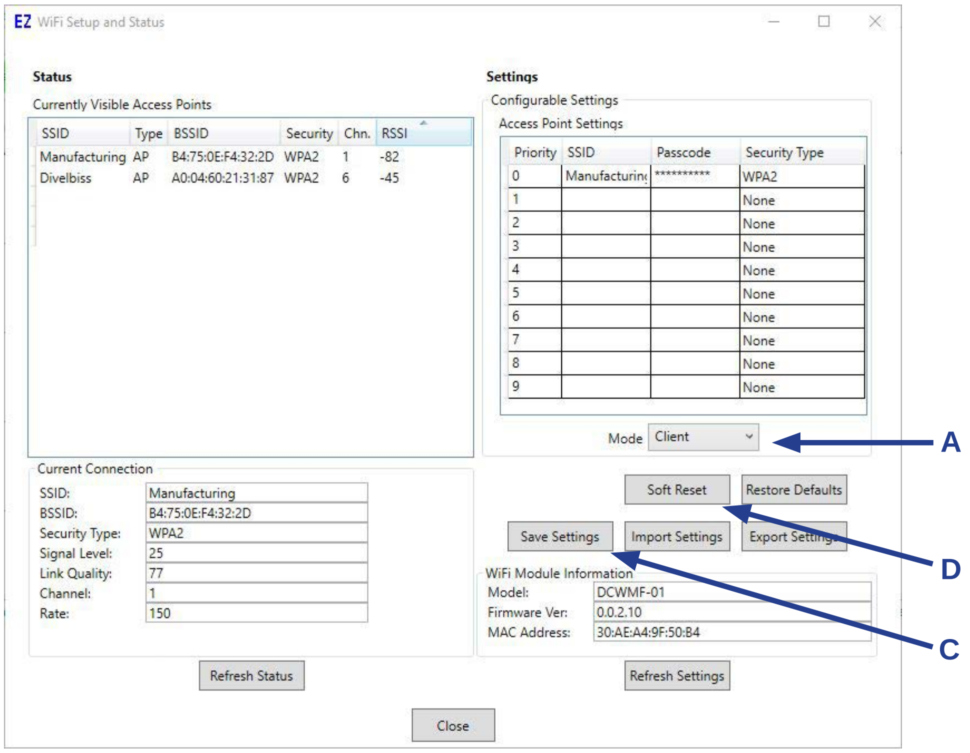


**Figure 19-14**

5. The ***WiFi Setup and Status*** window will open. An intermediate temporary dialog may be seen while the setup is accessed. Refer to Figure 19-15.

6. Set the ***Mode*** to Host (Item A). For client mode, see *Connecting to a Wi-Fi Network (Client Mode)* earlier in this chapter.

Ignore the *Currently Visible Access Points* pane, the *CurrentConnection* area (the settings shown do not apply and will disappear after configuation) and the other information.



**Status**

Currently Visible Access Points

| SSID | Type | BSSID | Security | Chn. | RSSI |
|---|---|---|---|---|---|
| Manufacturing | AP | B4:75:0E:F4:32:2D | WPA2 | 1 | -82 |
| Divelbiss | AP | A0:04:60:21:31:87 | WPA2 | 6 | -45 |

Current Connection

| | |
|---|---|
| SSID: | Manufacturing |
| BSSID: | B4:75:0E:F4:32:2D |
| Security Type: | WPA2 |
| Signal Level: | 25 |
| Link Quality: | 77 |
| Channel: | 1 |
| Rate: | 150 |

**Settings**

Configurable Settings

Access Point Settings

| Priority | SSID | Passcode | Security Type |
|---|---|---|---|
| 0 | Manufacturing | ********** | WPA2 |
| 1 | | | None |
| 2 | | | None |
| 3 | | | None |
| 4 | | | None |
| 5 | | | None |
| 6 | | | None |
| 7 | | | None |
| 8 | | | None |
| 9 | | | None |

Mode: Client

WiFi Module Information

| | |
|---|---|
| Model: | DCWMF-01 |
| Firmware Ver: | 0.0.2.10 |
| MAC Address: | 30:AE:A4:9F:50:B4 |

**Figure 19-15**

7. Click the **SAVE SETTINGS** button (Item C) to save the current settings for the Wi-Fi module.

8. Click the **SOFT RESET** button (Item D). This forces the Wi-Fi module to reset. After the reset, the screen should update changed for Host mode with grayed out sections.

7. Refer to Figure 19-16 for the rest of the configuration.

8. In the Access Points Settings - use **Priortity 0** (item B) slot and enter a name (or SSID) in the **SSID** field to use for the access point and enter a passcode in the **Passcode** field that all devices will use to connect. It will be necessary to double-click to enter the values.

9. Select the **Security Type** desired for the access point (item B).

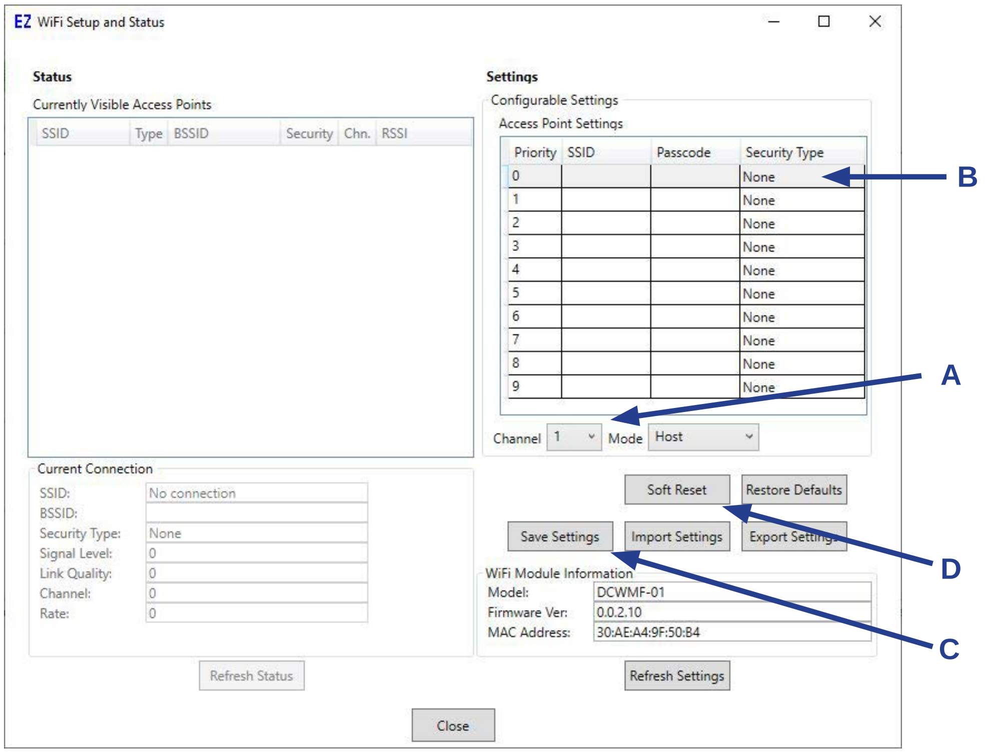
10. Select a **Channel** number (item A) (if needed) to prevent any interference with other Wi-Fi access points nearby.

11. With the information entered, click the **SAVE SETTINGS** button (Item C) to save the current settings for the acess point.

12. Click the **SOFT RESET** button (Item D). This forces the Wi-Fi module to reset.

13. Click **CLOSE** to close the WiFi Setup and Status window.

The Wi-Fi connectivity (Host Mode) is now configured and should be visible as an access point to Wi-Fi enabled smart devices.



| Priority | SSID | Passcode | Security Type |
|---|---|---|---|
| 0 | | | None |
| 1 | | | None |
| 2 | | | None |
| 3 | | | None |
| 4 | | | None |
| 5 | | | None |
| 6 | | | None |
| 7 | | | None |
| 8 | | | None |
| 9 | | | None |

Channel: 1 Mode: Host

SSID: No connection
BSSID:
Security Type: None
Signal Level: 0
Link Quality: 0
Channel: 0
Rate: 0

Model: DCWMF-01
Firmware Ver: 0.0.2.10
MAC Address: 30:AE:A4:9F:50:B4

**Figure 19-16**

This Wi-Fi (Host Mode) connectionis for connecting smart devices directly to a P-Series PLC on a Chip based controller to use direct communications features such as webserver. It does not provide any other connection or internet access and cannot be used as any internet hot-spot.

The Wi-Fi mode and other configurations may also be programmed using Structured Text, hard-codeded into the application program.  This chapter lists the Ethernet and Wi-Fi Structured Text commands. For more details on them and Strutured Text, refer to **Appendix B - Structured Text Function Reference** and **Chapter 26 - Structured Text**.

## Structured Text Wi-Fi / Ethernet Control

Both Wi-Fi and Ethernet may be configured based on needs and often are controlled automatically for communications from the ladder diagram program (such as Modbus, Programming Port, webserver or Versa-Cloud M2M+IoT).

Several target specific structured text functions exist that allow for direct interaction to the Ethernet or Wi-Fi hardware and connection. The functions are described in detail in **Appendix B - Target Specific ST Function Reference**. Structured text information is found in **Chapter 26 - Structured Text**.

Some of the functions are:

| | |
|---|---|
| **EZ_Eth_DHCPRelease** | **EZ_WiFi_Get_Connection_Status_1** |
| **EZ_Eth_DHCPRenew** | **EZ_WiFi_Get_Connection_Status_2** |
| **EZ_Eth_GetHostname** | **EZ_WiFi_Get_Mode** |
| **EZ_Eth_GetIPV4Addr** | **EZ_WiFi_Get_Passcode** |
| **EZ_Eth_GetLinkActive** | **EZ_WiFi_Get_Security** |
| **EZ_Eth_SetEnableAutoIpV4** | **EZ_WiFi_Get_SSID** |
| **EZ_Eth_SetEnableDHCPIpV4** | **EZ_WiFi_Set_Channel** |
| **EZ_Eth_SetHostname** | **EZ_WiFi_Set_Passcode** |
| **EZ_Eth_SetStaticIPV4Addr** | **EZ_WiFi_Set_Security** |
| **EZ_WiFi_Get_Access_Points** | **EZ_WiFi_Set_SSID** |
| **EZ_WiFi_GetChannel** | **EZ_WiFi_Soft_Reset** |

Using structured text to access the information and control the Ethernet and Wi-Fi requires programming experience with structured text and an understanding of Ethernet and Wi-Fi.

Care must be taken implementing control and interfacing to Wi-Fi and Ethernet using structured text. If not handled properly, Ethernet or Wi-Fi communications may become unstable or non-functioning entirely.

All structured text Wi-Fi commands are passed through to the on-board Wi-Fi module. The Wi-Fi module then processes the command and forms a response (that is returned to the calling function). All the Wi-Fi structured text commands will need to be polled until they have returned with a complete flag as some could take several seconds (up to 20 seconds) to complete.

After a complete flag is received by the calling function, the response must be checked for errors. Refer to **Appendix B - Target Specific ST Function Reference** for function response and error codes.

The Wi-Fi module stores up to 10 SSID / password combinations and when the Wi-Fi module boots, it searches beginning with the top of the list and continues to the end of the list searching for a matching (AP) SSID. If an empty ‘slot’ is detected in the list, it will stop searching (all entries after an empty are ignored). For example, if slot #1 were empty, the module would never search slots 2-9. Care must be taken when deleting / adding SSIDs to the list to not have empty slots.  See **Connecting to a Wi-Fi Network** section earlier in this chapter for adding / deleting SSIDs.

# CHAPTER 20

## SD Card Support

This chapter provides basic information to understand install and use SD Card features.

## Chapter Contents

# SD Card Support

EZ LADDER Toolkit provides features for installing and using SD Card features for P-Series PLC on a Chip$^{TM}$ targets. The SD Card may be used to install and update EZ LADDER target kernels and/or ladder diagram programs, as part of the webserver as well as logging data using structured text.

As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip$^{TM}$ target programming, SD Card Support is covered. To determine if the actual hardware target (model) supports the use of SD Cards, refer to **Chapter 23 - Hardware Targets**.

## Installing SD Card Support

By default, SD Card features may not enabled or configured on P-Series hardware targets. The SD Card Support port must be installed and configured using the Project....Bootloader Menu.

The first step is to open an existing program or create a new program (a simple one-rung program). This program is required to switch EZ LADDER Toolkit from the EDIT mode to the RUN mode; which is required for using the Bootloader Menu option. Make sure the serial port is configured correctly in the Project...Settings menu. Refer to **Chapter 4 - Configuring Targets** and **Chapter 5 - Creating Ladder Diagram Projects**.

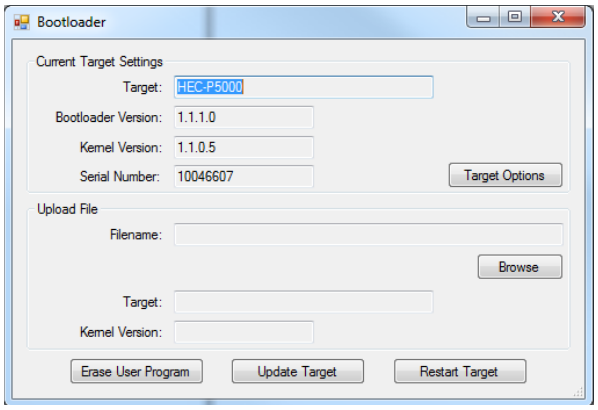
The Bootloader Menu option is only available when EZ LADDER Toolkit is in the RUN mode and when actually connected to the hardware target.

Open or create the ladder diagram project and configure the serial port using the Project...Settings menu. Click the **MONITOR** (MON) button located on the toolbar. EZ LADDER will switch from the EDIT mode to the RUN mode. The toolbars will change appropriately.

In the RUN mode, select the ***Project*** menu and the select ***Bootloader***. EZ LADDER will now attempt to connect to the hardware target and initialize it's bootloader. During this time, you may see a small status window. If the Bootloader does not respond or an error is encountered, check the serial port settings and cables.

The bootloader is a factory installed utility program on all PLC on a Chip$^{TM}$ hardware targets. This utilility is used for configuring options on the target and to update or install target software kernels.

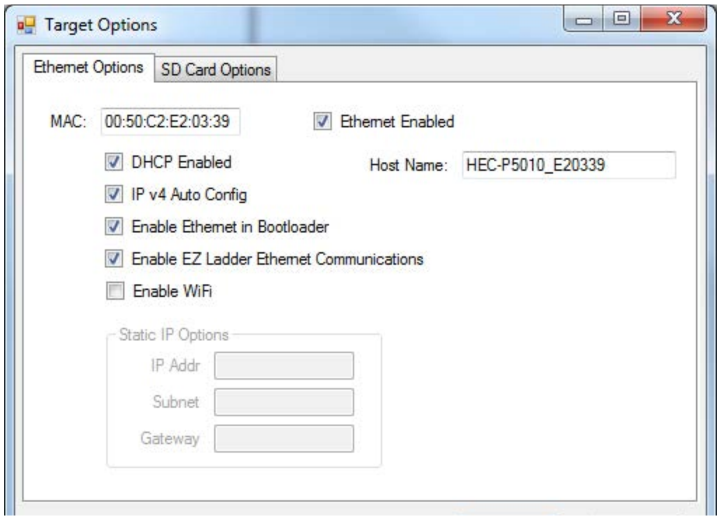
When the Bootloader utility responds, the Bootloader screen will automatically appear. See Figure 20-1.



Target: HEC-P5000
Bootloader Version: 1.1.1.0
Kernel Version: 1.1.0.5
Serial Number: 10046607

**Figure 20-1**

Click the **TARGET OPTIONS** button. A new Target Options window will open. This window has two tabs: Ethernet Options and SD Card Options. See Figure 20-2.



MAC: 00:50:C2:E2:03:39
☑ Ethernet Enabled
☑ DHCP Enabled
Host Name: HEC-P5010_E20339
☑ IP v4 Auto Config
☑ Enable Ethernet in Bootloader
☑ Enable EZ Ladder Ethernet Communications
☐ Enable WiFi

Static IP Options
IP Addr
Subnet
Gateway

**Figure 20-2**

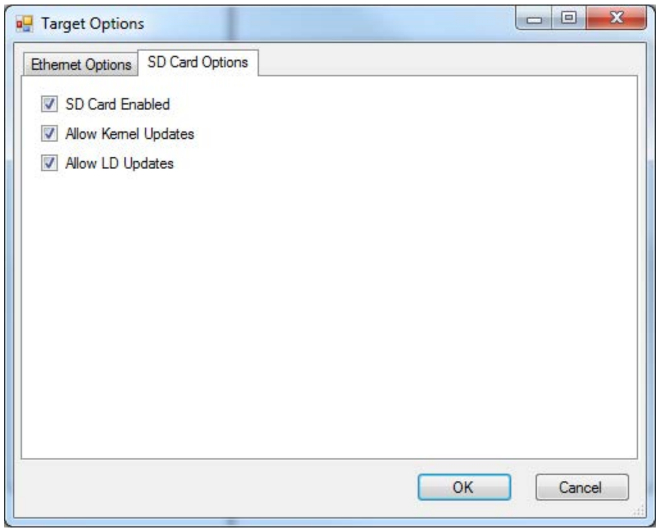
Select the **SD Card Options** Tab. Refer to Figure 20-3.

To enable the SD Card, click (check) the **SD Card Enabled** check-box. This enables the SD Card Features in the target.

If you wish to allow kernel updates from an installed SD Card, click (check) the **Allow Kernel Updates** check-box. How kernels are installed or updated is covered later in this chapter.

If you wish to allow Ladder Diagram updates from an installed SD Card, click (check) the **Allow LD Updates** check-box. How ladder diagrams are installed or updated is covered later in this chapter.

Click **OK** to accept the SD Card settings and close the window. Click **RESTART TARGET** to exit the Bootloader and force the software kernel to restart, accepting the changes. The SD Card is now configured to operate as configured.



☑ SD Card Enabled
☑ Allow Kernel Updates
☑ Allow LD Updates

**Figure 20-3**

## Using the SD Card

The SD Card Support must be installed using the Bootloader before any SD Card may be used.

As this manual is a generic EZ LADDER Toolkit manual for P-Series PLC on a Chip™ target programming, SD Card Support is covered. To determine if the actual hardware target (model) supports the use of SD Cards, refer to **Chapter 23 - Hardware Targets**.

For targets that support the SD Card (Secure Digital Card), it is typically plugged-in (or inserted) into a socket located on the hardware target.

The type of SD Card and socket is target dependent. Refer to the hardware target's User Manual for details on the type of SD Card supported and details for accessing and installing a supported SD Card.

### Updating the Kernel and Ladder Diagram

An installed SD Card may be used to install or update a kernel or ladder diagram (or both). The SD Card must have a directory named: ***update***. The files to be installed or updated are located in the ***update*** directory of the SD Card.

If the kernel is to be updated or installed, the actual target kernel file (.dat) must be copied or loaded into the ***update*** directory on the SD Card. This file can be found in the EZ LADDER kernel directory or other kernel files may used from other sources such as downloading from the website, etc. This file loading on the SD Card is done outside of EZ LADDER and the hardware target or using the Project.... ***File Transfer*** tool option in EZ LADDER.

The kernel will only update from the SD Card when the kernel name (.dat) matches the kernel name installed on the hardware target and the version of the kernel on the SD CARD is newer. The kernel will install on a new target (without kernel installed already) if there is only one kernel file (only 1 xxx.dat) in the update directory.

If the ladder diagram is to be updated or installed, the actual compiled ladder diagram file (.hex) must be copied or loaded into the ***update*** directory on the SD Card. This file is found where the actual ladder diagram (.dld) file is located. This file loading on the SD Card is done outside of EZ LADDER and the hardware target or using the Project.... ***File Transfer*** tool option in EZ LADDER.

The compiled ladder diagram program (.hex) will update from the SD Card when the controller starts (powered on) and the program on the SD card is not the exact same program (version and build number) that is already installed (provided the compiled .hex file matches the product type and is compatible with the kernel version installed).

Additionally, if there is no ladder diagram installed on the target (blank) and there is only one .hex (compiled ladder diagram) file in the SD card's update directory and the installed target's kernel matches what is expected in the compile ladder diagram file (.hex) and the kernel is new enough to support the ladder diagram file (.hex), then the .hex file (ladder diagram) will in installed on the target.

The controller will automatically load the .hex file provided the above conditions are met regardless if the currently installed program is the same. Care must be taken when downloading programs and having programs on the SD card. The SD card will over-write the downloaded program when power is cycled or the target is restarted.

Kernel and ladder diagram updates are performed only on power-up of the hardware target (after the SD Card has been installed).

To install an update, install the pre-loaded SD Card into the target's SD Card socket and cycle power. If the above conditions are met, the file(s) will be updated on the target. The SD Card may be removed after the update.

At times, it may be required to cycle power a total of two times. This seen usually when updating the **kernel and ladder diagram** using the same SD card **and if the kernel update required remapping** of the PLC on a Chip's internal memory (this is occasionally remapped to accommodate new features). By watching the Status / Watchdog LED, you can determine if the ladder program is running. If after an update, the LED is flashing slowly indicating the ladder diagram is not running, cycle the power again. If the problem still persists, check that the restrictions of the filename, etc. are not preventing the ladder diagram from loading.

## SD Card File Operations / Data Logging

P-Series EZ LADDER Toolkit supports file write, read and seek operations that may be used to store information to/from the SD card including logging data during normal operation beginning with P-Series EZ LADDER Toolkit V1.2.2.0.

All SD Card file operations including data logging to the SD card is accessed via structured text using target specific functions.

Application Note(s) are available to download from www.divelbiss.com with descriptions and actual ladder diagrams with examples of logging data to an SD card.

For more details on using Structured Text and Structured Text target specific functions, refer to **Chapter 26 - Structured Text** and **Appendix B - Target Specific ST Function Reference**. All File System (SD Card) structured text functions begin with EZ_FS_.

As the SD card may be accessed using file operations, it may be used to store and read additional parameters, store historical data (data-logging) and many more.

SD card data can be retrieved and sent to the cloud using **VersaCloud M2M Portal** Solutions. Contact Divelbiss for more information regarding sending SD card data to a VersaCloud M2M Portal.

### File System Operations Description

The file system (SD Card file operations) controlled by structured text functions provides the basic functionality for creating, accessing, reading and writing data to the SD Card. Each individual function must be controlled and combined with additional functions (EZ_FS functions and other functions) to take full advantage of the file system (SD Card storage).

To use the file system (SD Card), each operation necessary must be performed in the correct order with appropriate error detection. The following are guidelines for using the file system. For details on each function, refer to **Chapter 26 - Structured Text** and **Appendix B - Target Specific ST Function Reference.**

For most steps, error detection and handling must be observed. The basic steps to using the file system are:

1. Open (or create) the file on the SD Card. This uses the ST function **EZ_FS_Open**. This function has optional control flags for opening, creating and appending. Multiple files may be opened.

2. Locate the point in the file to read data from or write data to (this also known as the file location pointer). To read or write data, the actual location in the file must be identified and a pointer positioned for subsequent use of read / write functions using the **EZ_FS_Seek** function. Depending upon the EZ_FS_Open flag implementation, the pointer may be located at the end of the file (using the append flag)

3. Read and Write functions **EZ_FS_Read**, **EZ_FS_ReadStr**, **EZ_FS_Write** and **EZ_FS_WriteStr** functions are used to read and write data to / from the SD Card.

4. The file should be closed after the reads and writes are completed using the **EZ_FS_Close** function. It is recommended to close the file after each read or write as any file left open with cached data is susceptible to losing data in the event of a power loss.

Other actions may include truncating a file using **EZ_FS_Truncate** function and flushing the cached data using the **EZ_FS_Flush** function and getting the file size using **EZ_FS_Size**.

### File Identification

The file system using a unique file identification for each file. This identification is automatically assigned by the **EZ_FS_Open** function when it is called (unique for each file open). This identification number (also called the *FileHandle* must be kept (as a variable) in structured text and used to identify the file in use for all additional functions (read, write, etc.) after the file open command (EZ_FS_Open) is used. Each EZ_FS function requires the identification (*FileHandle*) to know which file to access (of the open files). Once a file is closed, the identification is no longer valid.

## File Transfer Tool

P-Series EZ LADDER Toolkit supports a built-in file transfer tool for reading, writing and erasing contents from an SD card in a P-Series based controller. This can be used to load ladder diagram update files (.hex), kernel files (.dat) or webserver files (multiple types).

### Acessing the File Transfer (tool)

The File Transfer tool is located in the Project Menu. You must first be connected to the hardware target (controller).

1. Open a program or create a simple program with the target info.

2. Change EZ LADDER to the Monitor mode by clicking the MON button.

3. Make sure the target is connected to the computer and click the button to connect EZ LADDER Toolkit to the hardware target.

4. From the menu at the top, select Project then select File Transfer. Refer to Figure 20-4.

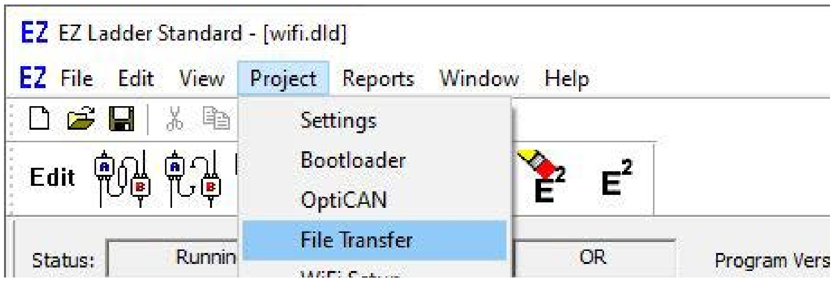
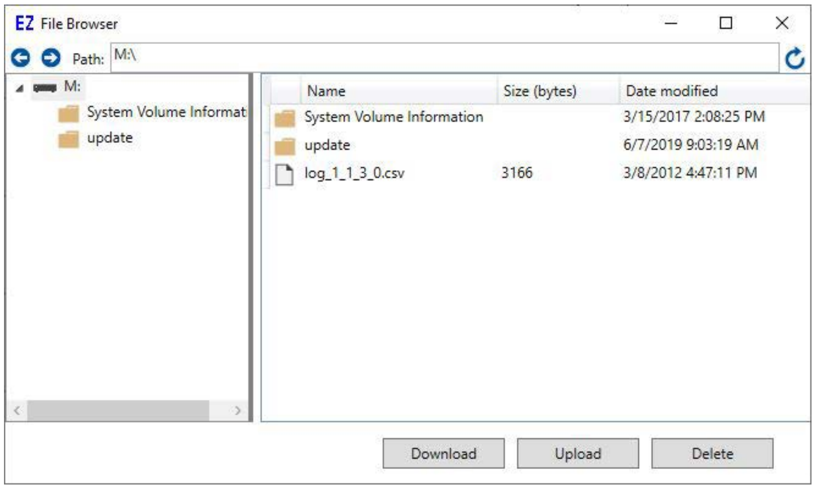


**Figure 20-4**

5. The ***File Browser*** window will open. An intermediate temporary dialog may be seen while the SD card is accessed. Refer to Figure 20-5.

If there is no SD card or a corrupt card in the target (controller), an error dialog will result. Ensure the SD card is installed before trying to use the File Transfer option.



| Name | Size (bytes) | Date modified |
|---|---|---|
| System Volume Information | | 3/15/2017 2:08:25 PM |
| update | | 6/7/2019 9:03:19 AM |
| log_1_1_3_0.csv | 3166 | 3/8/2012 4:47:11 PM |

**Figure 20-5**

6. The File Browser window shows the folder and file structure on the SD card. Use the **DOWNLOAD** button to download files from the SD card to the PC, use the upload button to **UPLOAD** files from the PC to the SC card or use the **DELETE** button to delete files from the SD card.

Note: depending on the file size, some file transfers may take extra time due to speed limitations of SD cards.

7. Click the **X** in the upper-right hand corner to close the File Browser window when all file activity has been completed.

# CHAPTER 21

# EZ LADDER Toolkit Reports

This chapter provides basic information to understand how to create and use EZ LADDER Toolkit project reports.

## Chapter Contents

# EZ LADDER Toolkit Reports

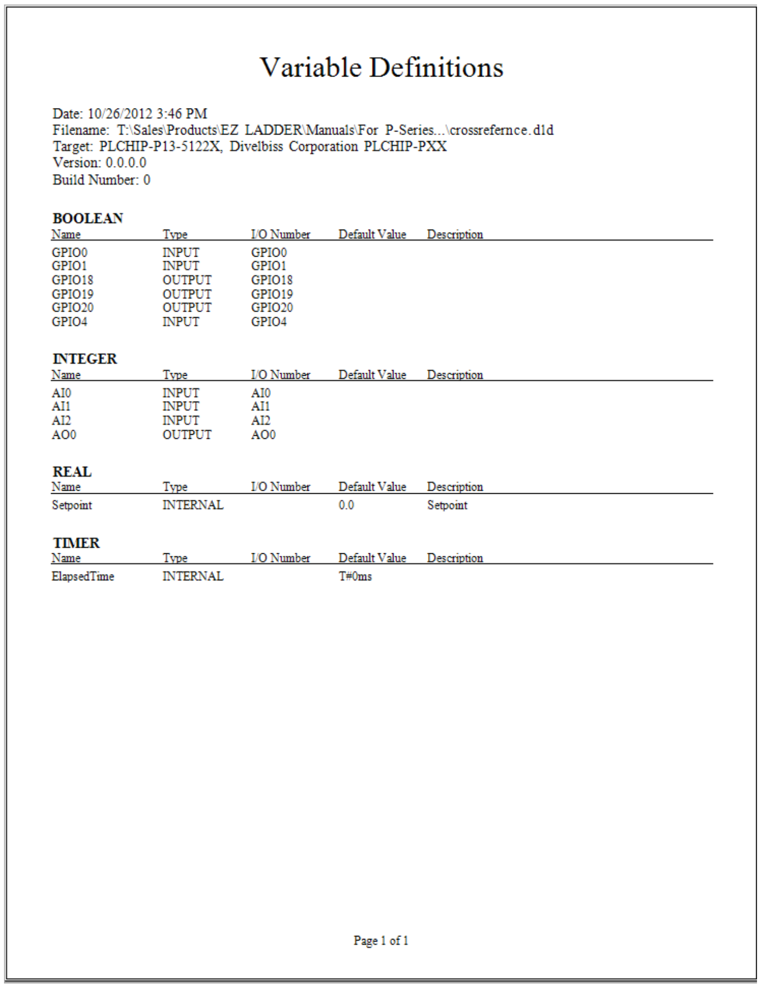
EZ LADDER Toolkit includes reporting features to aid in creation, troubleshooting and documenting ladder diagram projects.  Each report, when generated, is viewable and printable.

There are two basic reports that can be generated:  Variable Definitions and Cross References.

## Variable Definitions Report

The variable definitions report provides a summary of all of the variables in the ladder diagram project. These variables are sorted by type for easy reference.  For each variable, the report shows Name, Type, I/O Number, Default Value and its Description.

To generate and view this report, using the **Reports Menu**, select *Variable Definitions*.  A report window will open displaying the generated report. Controls are available to change pages and print.  Figure 20-1 is an example of the report that is seen or printed.



Variable Definitions

Date: 10/26/2012 3:46 PM
Filename:  T:\Sales\Products\EZ LADDER\Manuals\For P-Series...\crossrefernce.d1d
Target:  PLCHIP-P13-5122X,  Divelbiss Corporation PLCHIP-PXX
Version: 0.0.0.0
Build Number: 0

**BOOLEAN**

| Name | Type | I/O Number | Default Value | Description |
|---|---|---|---|---|
| GPIO0 | INPUT | GPIO0 | | |
| GPIO1 | INPUT | GPIO1 | | |
| GPIO18 | OUTPUT | GPIO18 | | |
| GPIO19 | OUTPUT | GPIO19 | | |
| GPIO20 | OUTPUT | GPIO20 | | |
| GPIO4 | INPUT | GPIO4 | | |

**INTEGER**

| Name | Type | I/O Number | Default Value | Description |
|---|---|---|---|---|
| AI0 | INPUT | AI0 | | |
| AI1 | INPUT | AI1 | | |
| AI2 | INPUT | AI2 | | |
| AO0 | OUTPUT | AO0 | | |

**REAL**

| Name | Type | I/O Number | Default Value | Description |
|---|---|---|---|---|
| Setpoint | INTERNAL | | 0.0 | Setpoint |

**TIMER**

| Name | Type | I/O Number | Default Value | Description |
|---|---|---|---|---|
| ElapsedTime | INTERNAL | | T#0ms | |

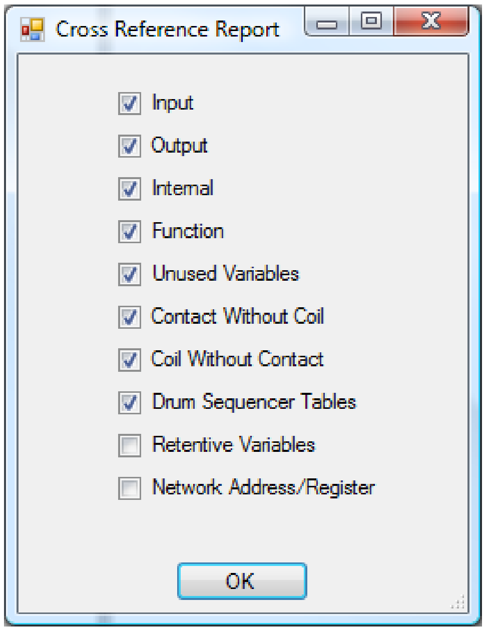
Page 1 of 1

**Figure 20-1**

## Cross References Report

The Cross Reference Report provides a summary of the objects that are in the ladder diagram project. The project objects are sorted by the type of object. The actual types of objects and data to view is selected prior to generating the Cross Reference Report.

To generate and view this report, using the **Reports Menu**, select *Cross References*. The Cross Reference Report dialog box will open with the choices to what objects to include in the report. See Figure 17-2.



- [✓] Input
- [✓] Output
- [✓] Internal
- [✓] Function
- [✓] Unused Variables
- [✓] Contact Without Coil
- [✓] Coil Without Contact
- [✓] Drum Sequencer Tables
- [ ] Retentive Variables
- [ ] Network Address/Register

**Figure 20-2**

Using the check boxes provided, select or de-select the items desired to be included on the Cross Reference Report. The items to select are:

| | |
|---|---|
| **Input**: | This will include all real world inputs on the report. |
| **Output**: | This will include all real world outputs on the report. |
| **Internal**: | This will include all *internal* contacts and coils on the report. |
| **Function**: | This will include all functions (function blocks) used on the report. |
| **Unused Variables**: | This will list any variables that are in the ladder diagram project, but are not actually used in the ladder diagram itself (created but not used). |
| **Contact without Coil**: | This will list all contacts that have been created and are used in the ladder diagram, but have no coil used in the ladder diagram, |
| **Coil without Contact**: | This will list all coils that have been created and are used in the ladder diagram, but have no contacts used in the ladder diagram. |

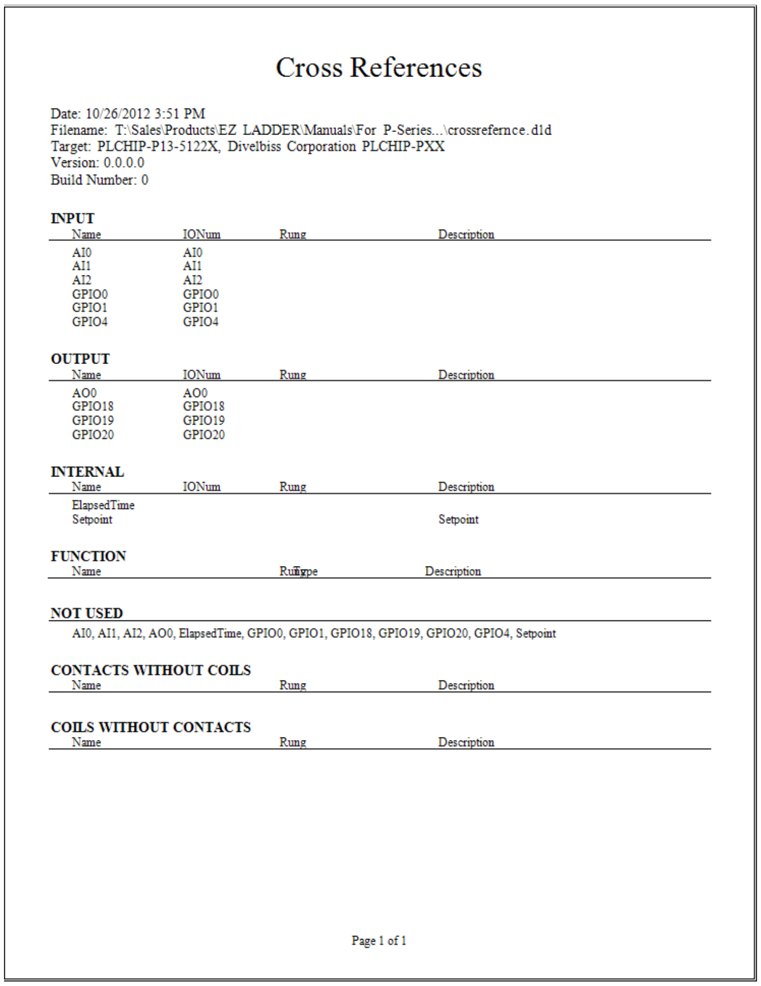
**Drum Sequencer Tables**: This will include all drum sequencer matrix tables on the report.

**Retentive Variables**: This lists all variables (all types) that are configured to be retentive.

**Network Address / Register**: This lists the variables and network addresses on the report (only variables with network addresses are listed).

For each option selected in the dialog box, the report is generated identifying the rung number, type and description for each item.

Click ok to generate the report. A report window will open displaying the generated report. Controls are provided to change pages and to print. Figure 20-3 represents the report that is viewed or printed.



Cross References

Date: 10/26/2012 3:51 PM
Filename: T:\Sales\Products\EZ LADDER\Manuals\For P-Series...\crossrefernce.dld
Target: PLCHIP-P13-5122X, Divelbiss Corporation PLCHIP-PXX
Version: 0.0.0.0
Build Number: 0

**INPUT**

| Name | IONum | Rung | Description |
|---|---|---|---|
| AI0 | AI0 | | |
| AI1 | AI1 | | |
| AI2 | AI2 | | |
| GPIO0 | GPIO0 | | |
| GPIO1 | GPIO1 | | |
| GPIO4 | GPIO4 | | |

**OUTPUT**

| Name | IONum | Rung | Description |
|---|---|---|---|
| AO0 | AO0 | | |
| GPIO18 | GPIO18 | | |
| GPIO19 | GPIO19 | | |
| GPIO20 | GPIO20 | | |

**INTERNAL**

| Name | IONum | Rung | Description |
|---|---|---|---|
| ElapsedTime | | | |
| Setpoint | | | Setpoint |

**FUNCTION**

| Name | Rung | Type | Description |
|---|---|---|---|

**NOT USED**

AI0, AI1, AI2, AO0, ElapsedTime, GPIO0, GPIO1, GPIO18, GPIO19, GPIO20, GPIO4, Setpoint

**CONTACTS WITHOUT COILS**

| Name | Rung | Description |
|---|---|---|

**COILS WITHOUT CONTACTS**

| Name | Rung | Description |
|---|---|---|

Page 1 of 1

**Figure 20-3**

# CHAPTER 22

## Troubleshooting

This chapter provides basic information to understand how to solve problems and to identify problems and common error message found using the EZ LADDER Toolkit.

## Chapter Contents

## Error Messages

The following is a list of error messages that may be encountered when using the EZ LADDER Toolkit. While you may experience any of these messages, many are rarely encountered. These error message may appear as pop-up dialog boxes or in the Output window.

A different program is running (Monitor Mode)
When connecting to a target,  the program running on the target is different than the program currently opened in EZ LADDER Toolkit.

Could not connect to target (Monitor Mode)
EZ LADDER Toolkit was not able to connect to a hardware target.

Could not get target version. Please connect first (Monitor Mode)
EZ LADDER Toolkit was unable to retrieve the target version when using the *target information* feature or button.

Could not open: COMX (Monitor Mode)
When connecting to a target, the selected Com Port does not exist or is in use.  This is typically caused when another application is using and locked the serial port as a resource.  Close the other application to correct this.

Error downloading file (Monitor Mode)
An unknown error occurred  while downloading the program to target.  Try downloading the program again.

ERROR downloading user program: invalid address (Monitor Mode)
An invalid address was detected in a communications packet while EZ LADDER Toolkit was connected to a target.

ERROR downloading user program: invalid record (Monitor Mode)
An invalid record was detected in a communications packet while EZ LADDER Toolkit was connected to a target.

ERROR downloading user program: checksum error (Monitor Mode)
An invalid checksum was detected in a communications packet while EZ LADDER Toolkit was connected to a target.

ERROR downloading user program: record to long (Monitor Mode)
An invalid record length was detected in a communications packet while EZ LADDER Toolkit was connected to a target.

ERROR putting target into bootloader (Monitor Mode)
An error occurred when EZ LADDER Toolkit was trying to access the target bootloader.  Verify the serial connections and settings, cables and the target.

Error, serial port not open (Monitor Mode)
The serial port that is configured in EZ LADDER Toolkit cannot be opened for use. This may caused when another application is using and locked the serial port as a resource.  Close the other application to correct this.

ERROR programming target (Monitor Mode)
EZ LADDER Toolkit detected an undefined error while attempting to store the project on the hardware target. Repeat the download (and store process) to correct this issue.

Error staring program. Program doesn't exist (Monitor Mode)
The program that is trying to start does not exist on the target. Download the program. This is typically caused by clicking the Go button before the ladder diagram project is loaded on the target.

Error starting program. Program could not be started (Monitor Mode)
The program cannot be started. Re-compile and download the program.

Error while receiving packet (Monitor Mode)
There was an error when receiving communications packets from the target.

File could not be opened (Monitor Mode)
When downloading the program to target, the file with the compiled code could not be opened. The file could have been moved or deleted. Compile the project and then download to the target.

Invalid Api (Monitor Mode)
When connecting to a target, the status box displays this message. The current EZ LADDER kernel and compiled program versions are not compatible. This occurs typically when updating kernel without erasing program or using a newer program with older kernel. Requires either changing kernel, EZ LADDER version or recompiling.

Invalid File (Editor Mode)
The file you are trying to open in EZ LADDER Toolkit is not a valid EZ LADDER Toolkit ladder diagram file.

Invalid HEX file (Monitor Mode)
When downloading to a target, the file used to store compiled code is invalid, or corrupt. Re-compile the ladder diagram project to correct this issue.

x is not supported by the current target (Editor Mode)
The object or function block that you are trying to use and place is not supported on target selected in the Project Settings. This can be caused if the hardware target is changes after a ladder diagram is created, then function blocks are edited. Either change the target or delete this function block / object.

Ladder program is not present (Monitor Mode)
No ladder diagram program was detected on the connected hardware target.

Link at: (x, y) had an invalid Grid point (Editor Mode)
The link is open or not connected at a grid point. Correct or re-draw the link.

Link is not valid (Editor Mode)
The link you are trying to create is not valid. This is typically caused when trying to link one type of variable (integer, real, etc) to a function block or object that does not support that type or all variables linked to the function block must be identical types and you are trying to link a variable that does not match the types already connected to the function block.

Nack or No acknowledgement sent from target (x) (Monitor Mode)
The target did not send a no acknowledgement during communications with EZ LADDER Toolkit. This error can occur occasionally based on many factors. Click ok to clear.

Object already there (Editor Mode)
An object already exists where you are trying to place another object. Select a new location to place the object.

Object type: X, not found Aborting load (Editor Mode)
Error loading program into EZ LADDER Toolkit. The ladder diagram file may be corrupt.

Packet contained a formatting ERROR (Monitor Mode)
An packet formatting error was detected in a packet during communication with a target.

Packet contained an invalid checksum (Monitor Mode)
An invalid checksum was detected in a packet during communication with a target.

Packet length was invalid (Monitor Mode)
An invalid communications packet length was detected during communications with the connected target.

Please save project before compiling (Editor Mode)
EZ LADDER Toolkit projects must be saved prior to allowing them to be compiled. Save the ladder diagram project.

Please select a target (Editor Mode)
A target has not been selected. You must select and configure a target in the Project Settings before placing any objects and function blocks.

Please select a target before compiling (Editor Mode)
Unable to compile because no target was selected. You must select and configure a target in the Project Settings before compiling.

Please select a target before verifying (Editor Mode)
Unable to run program verification because no target is selected. You must select and configure a target in the Project Settings before verifying.

Targets do not match (Monitor Mode)
When connecting to a target the target specified in the ladder diagram project does not match the actual detected hardware target connected to the serial port. Correct the target in the Project Settings.

Target does not support bootloader (Monitor Mode)
This specific target is too old to support any bootloader functions. Contact Support for options.

There is not enough room for the paste. Increase the number of rungs (Editor Mode)
There is not enough rung space to paste from the clipboard. Increase the number of rungs where the paste is to occur.

There is not enough room to the right of the paste point. (Editor Mode)
There is not enough room at the insertion point to paste objects from the clipboard. Paste the objects farther left.

This object must be place in the last column (Editor Mode)
The selected object can only be placed in the last column. All coils can only be placed in the last column.

Timeout ERROR. Entire packet was not received (Monitor Mode)
During communication with a target, part of a packet was lost or not received.

Timeout ERROR. Target didn't respond (Monitor Mode)
During communication with a target, the target did not respond. Check the cables, connections, target and Serial port settings in EZ LADDER Toolkit.

Undefined packet type (Monitor Mode)
EZ Ladder has detected a undefined communications packet during communications with the connected target.

## Structured Text Errors

The following is a list of structured text specific error messages that may be encountered when using the EZ LADDER Toolkit. These error message may appear as pop-up dialog boxes or in the Output window. For more details on structured text, refer to **Chapter 26 - Structured Text**.

### Errors During Structured Text Verifying

The following is a list of structured text specific error messages that may be encountered when structured text is verified. Verification may be manually initiated in the Structured Text Editor. It is also performed automatically as a prerequisite during a ladder diagram Compile.

This error message gets formatted as {POU Name} {Location}: ERROR STV{Error Num}: {Description}

| Member Name | Value | Description |
|---|---|---|
| Variable | 1000 | Generic variable error |
| Variable_NotDefined | 1001 | Variable is not defined message -> Variable {0} is not defined |
| Variable_NotModifiable | 1002 | Variable is not modifiable message -> |
| Variable_TypeMismatch | 1003 | Types don't match |
| Variable_NotArray | 1004 | Variable is not an array, don't use '[]' message -> Variable {0} is not an array |
| Variable_IsArray | 1005 | Variable is an array, make sure use '[]' message -> Variable {0} is an array, but subscript list is missing |
| Variable_ConstantNotInialized | 1006 | constant variable is not initialized message -> CONSTANT Variable {0} not initialized |
| Variable_ExternMustBeConstant | 1007 | Global Variable is defined as a constant, so external must be constant message -> Variable {0} is defined as CONSTANT |
| Variable_TooManyInit | 1008 | array has to many initializers message -> Variable {0} has too many initializers |
| Function | 2000 | Generic Function Error |
| Function_NotDefined | 2001 | Function is not defined message -> Function {0} is not defined |
| FunctionBlock | 3000 | Generic Function Block Error |
| FunctionBlock_InstanceUndefined | 3001 | Function Block Instance is not defined message -> Function Block instance is undefined: {0} |
| FunctionBlock_NotSupportedHere | 3002 | Function Block is not supported here message -> Invalid expression, Function Block name {0} not supported here |

| Member Name | Value | Description |
|---|---|---|
| TypeDeclarations | 4000 | Generic type definition errors |
| TypeDecl_NotDefined | 4001 | Type is not defined |
| EnumDeclarations | 5000 | Generic enumeration errors |
| EnumDecl_NotDefined | 5001 | Enumeration is not defined |

## Errors During Structured Text Code Generation

The following is a list of structured text specific error messages that may be encountered when structured text code is generated.

| Member Name | Value | Description |
|---|---|---|
| Variable | 1000 | Generic Variable ERROR |
| Variable_NotDefined | 1001 | Variable is not defined message -> Variable {0} is not defined |
| Func_GenCode | 2000 | Generic code generation error, message -> Failed generating code for {0} |
| Func_GenCode_TooManyParams | 2001 | In function call user specified too many function parameters message -> Too many parameters for function call: {0} |
| Func_GenCode_TooLittleParams | 2002 | In function call user didn't specify enough function parameters message -> Too little parameters for function call: {0} |
| FuncBlk_GenCode | 3000 | |

## Common Ladder Diagram Errors

When creating ladder diagram projects using EZ LADDER Toolkit, here are some of the common errors made during the creating process.

### Connecting Functions to Functions Errors

When connecting Variable outputs of one function to a variable input of another function, a variable must be placed between the two functions. Figure 22-1 illustrates the incorrect way of connecting functions to functions (variable inputs and outputs). Figure 22-2 illustrates the same ladder diagram project, but with the corrections made (a variable between the function blocks).

If a function's variable output is connected directly to another functions' variable input, the program will compile successfully, however; the program will not function as designed.
A variable must be placed between the output and the input for proper operation.

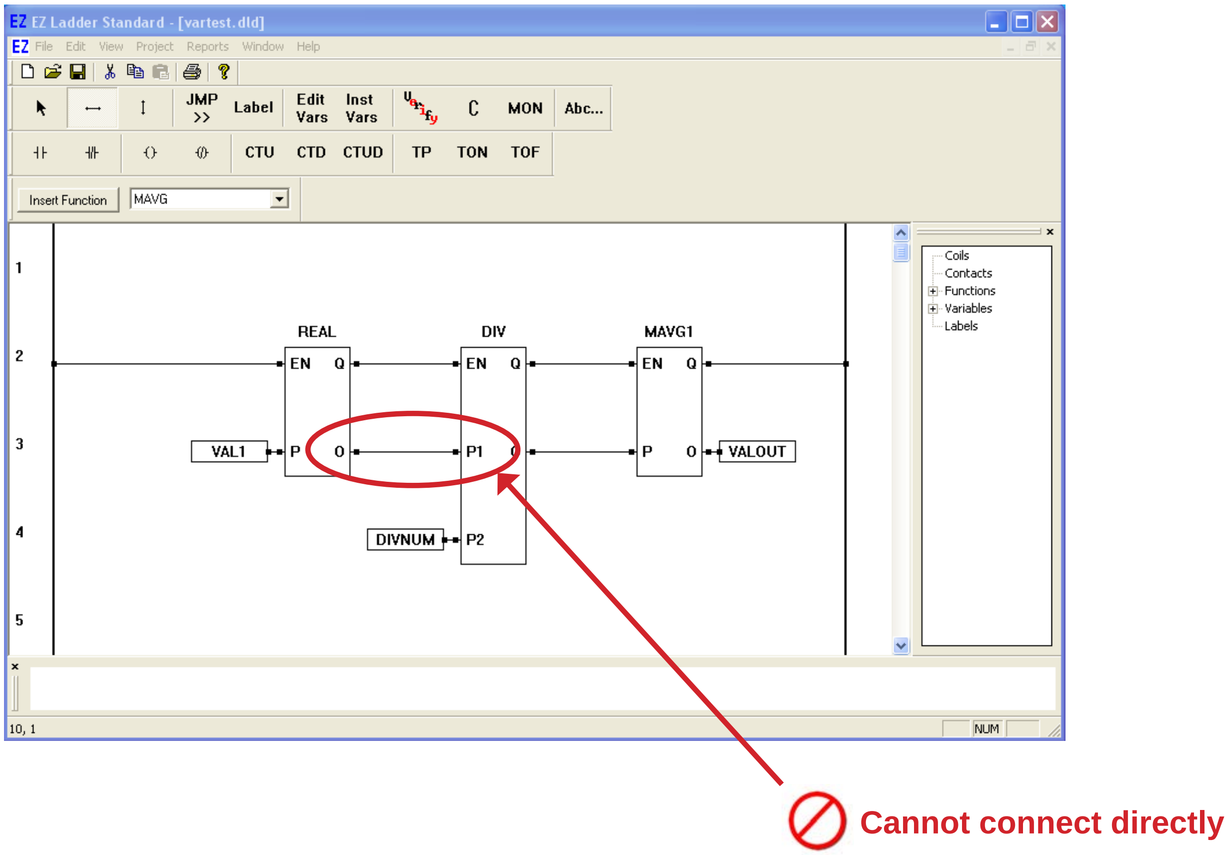


Figure 22-1

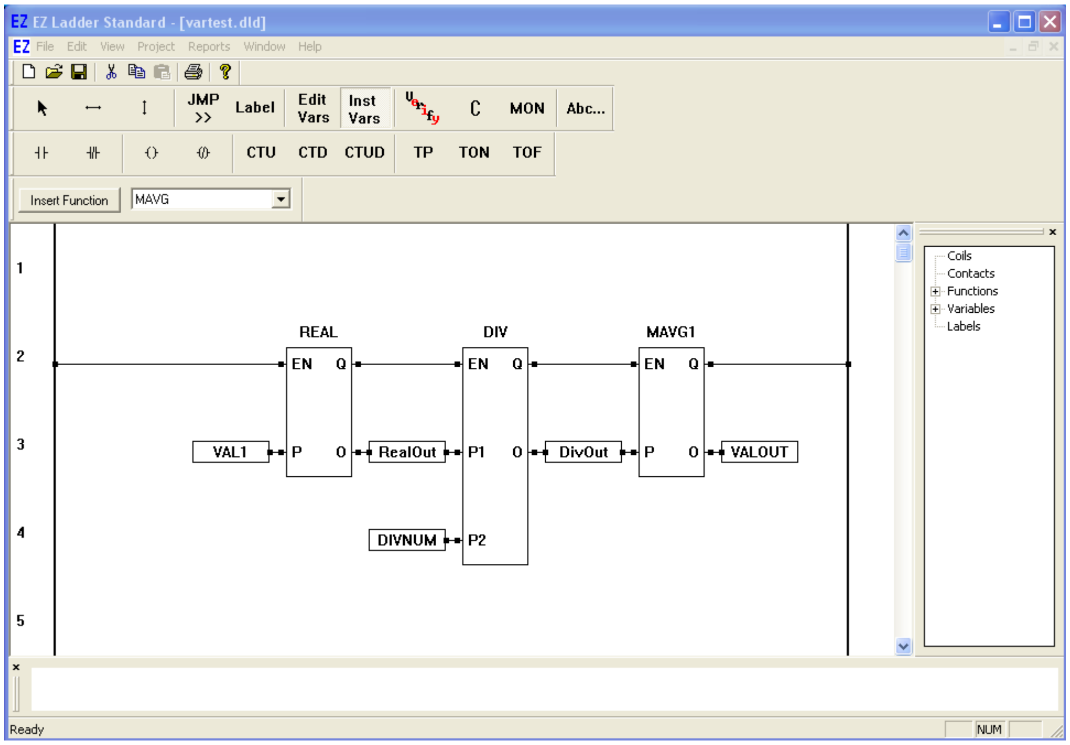


Figure 22-2

# CHAPTER 23

## Hardware Targets

This chapter provides detailed information for P-Series PLC on a Chip based hardware targets including supported functions and features for each as well as specific information needed to use hardware features.

## Chapter Contents

## P-Series PLC on a Chip™ Integrated Circuits

Each P-Series PLC on a Chip™ integrated circuit model supports different features and function blocks. Typically, the larger memory models support more features and function blocks. For all P-Series PLC on a Chip™ models, any feature listed must be individually installed using the Project Settings Menu.

### PLCHIP-P13-5122X

All listed features and function blocks listed are supported individually.  Using certain features or function blocks may limit the availability of other features and function blocks.

#### Features

On-Board Real Time Clock
Retentive Memory (using FM24xxx)
Up to 164 I/O Digital I/O
Up to 12 PWM Outputs
Quadrature Input
3 High Speed Counter / Timer Inputs
Up to 8 Analog Inputs, On-Board
Up to 1 Analog Output, On-Board
Up to 4 Serial Ports
Up to 2 CAN Ports
Ethernet Port / Wi-Fi
Up to 3 $I^2C$ Ports
Up to 2 SPI Ports
EEPROM Storage  (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet / Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
Character (CHR) LCD Support
Graphics (GFX) LCD Support
Keypad Support
Expandable Analog using SPI / I2C
SD Card Storage
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

#### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Ceiling (CEIL)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad (KEYPAD)
Keypad2 (KEYPAD2)
Label
Latching Coil (LATCH)
LCD Clear (LCD_CLEAR)
LCD Print (LCD_PRINT)
Limit (LIMIT)
LS7366R Quad Counter (CNTR_LS7366R)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish

MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## PLCHIP-P10-5122X

All listed features and function blocks listed are supported individually. Using certain features or function blocks may limit the availability of other features and function blocks.

### Features

On-Board Real Time Clock
Retentive Memory (using FM24xxx)
Up to 106 I/O Digital I/O
Up to 3 PWM Outputs
Quadrature Input
2 High Speed Counter / Timer Inputs
Up to 8 Analog Inputs, On-Board
Up to 1 Analog Output, On-Board
Up to 4 Serial Ports
Up to 2 CAN Ports
Ethernet Port / Wi-Fi
Up to 3 I$^2$C Ports
Up to 2 SPI Ports
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet / Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
Character (CHR) LCD Support
Graphics (GFX) LCD Support
Keypad Support
Expandable Analog using SPI / I2C
SD Card Storage
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Ceiling (CEIL)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)

Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad (KEYPAD)
Keypad2 (KEYPAD2)
Label
Latching Coil (LATCH)
LCD Clear (LCD_CLEAR)
LCD Print (LCD_PRINT)
Limit (LIMIT)
LS7366R Quad Counter (CNTR_LS7366R)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

# HEC-P6xxx Series

Each HEC-P6xxx model supports different features and function blocks based differences in the internal hardware. When any HEC-P6xxx model is selected in the Project Settings, some supported functions are automatically installed while others must be manually installed.

## HEC-P6000

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
14 Digital Inputs - Sink / Source (DC)
12 PWM / 12 On, Off Digital Outputs (DC)
2 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
4 Analog Inputs, Range/Type Field Adjustable
2 Analog Outputs, 0-10VDC
2 Serial Ports RS232/RS485
2 CAN Ports
Ethernet Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)

Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P6010

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
14 Digital Inputs - Sink / Source (DC)
12 PWM / 12 On, Off Digital Outputs (DC)
2 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
4 Analog Inputs, Range/Type Field Adjustable
2 Analog Outputs, 0-10VDC
2 Serial Ports RS232/RS485
2 CAN Ports
Ethernet Port
EEPROM Storage (3500 bytes)
GPS Module Port
Modbus Master / Slave
Modbus TCP over Ethernet
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)

Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P6100

### Features

Retentive Memory  (FRAM 480 bytes)
14 Digital Inputs - Sink / Source (DC)
12 PWM / 12 On, Off Digital Outputs (DC)
2 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
4 Analog Inputs, Range/Type Field Adjustable
2 Analog Outputs, 0-10VDC
2 CAN Ports
EEPROM Storage  (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)

Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
Bitwise XOR (XOR)

## HEC-P6110

### Features

Retentive Memory  (FRAM 480 bytes)
14 Digital Inputs - Sink / Source (DC)
12 PWM / 12 On, Off Digital Outputs (DC)
2 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
4 Analog Inputs, Range/Type Field Adjustable
2 Analog Outputs, 0-10VDC
2 CAN Ports
EEPROM Storage  (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)

Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

## HEC-P6200

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
14 Digital Inputs - Sink / Source (DC)
12 PWM / 12 On, Off Digital Outputs (DC)
2 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
4 Analog Inputs, Range/Type Field Adjustable
2 Analog Outputs, 0-10VDC
2 Serial Ports RS232/RS485
2 CAN Ports
GPS Module Port
Wi-Fi Connectivity
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)

PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P6210

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
14 Digital Inputs - Sink / Source (DC)
12 PWM / 12 On, Off Digital Outputs (DC)
2 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
4 Analog Inputs, Range/Type Field Adjustable
2 Analog Outputs, 0-10VDC
2 Serial Ports RS232/RS485
2 CAN Ports
GPS Module Port
Wi-Fi Connectivity
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)

Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

# HEC-P5xxx Series

Each HEC-P5xxx model supports different features and function blocks based differences in the internal hardware. When any HEC-P5xxx model is selected in the Project Settings, some supported functions are automatically installed while others must be manually installed.

## HEC-P5000

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
16 Digital Inputs - Sink / Source (DC)
12 PWM / On, Off Digital Outputs (DC)
4 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Analog Inputs, Range/Type Field Adjustable
2 Serial Ports RS232/RS485
2 CAN Ports
Ethernet Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)

Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P5010

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
16 Digital Inputs - Sink / Source (DC)
12 PWM / On, Off Digital Outputs (DC)
4 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Analog Inputs, Range/Type Field Adjustable
2 Serial Ports RS232/RS485
2 CAN Ports
Ethernet Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet
OptiCAN Networking
J1939 / NMEA 2000 Networking
512K SRAM
GPS Module Port
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Modulo (MOD)
Multiplication (MULT)

Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP) PLCHIP
Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P5100

### Features

Retentive Memory (FRAM 480 bytes)
16 Digital Inputs - Sink / Source (DC)
12 PWM / On, Off Digital Outputs (DC)
4 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Analog Inputs, Range/Type Field Adjustable
2 CAN Ports
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)

PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
Bitwise XOR (XOR)

## HEC-P5110

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
16 Digital Inputs - Sink / Source (DC)
12 PWM / On, Off Digital Outputs (DC)
4 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Analog Inputs, Range/Type Field Adjustable
2 CAN Ports
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
512K SRAM
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP) PLCHIP
Quad Counter Velocity (CNTR_PXX_VEL)

PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

## HEC-P5200

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
16 Digital Inputs - Sink / Source (DC)
12 PWM / On, Off Digital Outputs (DC)
4 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Analog Inputs, Range/Type Field Adjustable
2 Serial Ports RS232/RS485
2 CAN Ports
Wi-Fi Connectivity
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port
512K SRAM
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)

Minimum (MIN)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P5210

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
16 Digital Inputs - Sink / Source (DC)
12 PWM / On, Off Digital Outputs (DC)
4 On,Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Analog Inputs, Range/Type Field Adjustable
2 Serial Ports RS232/RS485
2 CAN Ports
Wi-Fi Connectivity
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
GPS Port
512K SRAM
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)

J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Moving Average (MAVG)
Maximum (MAX)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P2xxx Series

Each HEC-P2xxx model supports different features and function blocks based differences in the internal hardware. When any HEC-P2xxx model is selected in the Project Settings, some supported functions are automatically installed while others must be manually installed.

## HEC-P2000

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
1 Serial Port RS232
1 Serial Port RS485
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)

Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

## HEC-P2001

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
Wi-Fi Connectivity
SD Card Support
1 Serial Port RS232
1 Serial Port RS485
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)

Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-P2010

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
Quadrature Input
3 High Speed Counter / Timer Inputs - NPN/PNP
Wi-Fi Connectivity
SD Card Support
1 Serial Port RS232
1 Serial Port RS485
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Cellular Modem for Versacloud M2M+IoT Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)

PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

# HEC Gateway Series

Each HEC Gateway model (HEC-GW-X-X) supports different features and function blocks based differences in the internal hardware. When any HEC-GW-X-X model is selected in the Project Settings, some supported functions are automatically installed while others must be manually installed.

## HEC-GW-C-W

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
1 Digital Inputs - Sink / Source (DC)
1 PWM / On, Off Digital Outputs (DC)
1 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Serial Ports RS232
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port
Cellular Modem for Versacloud M2M+IoT Data
Wi-Fi Connectivity
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)

Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-GW-C-X

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
1 Digital Inputs - Sink / Source (DC)
1 PWM / On, Off Digital Outputs (DC)
1 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Serial Ports RS232
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port
Cellular Modem for Versacloud M2M+IoT Data
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)

Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

## HEC-GW-X-W

### Features

On-Board Real Time Clock
Retentive Memory  (FRAM 480 bytes)
1 Digital Inputs - Sink / Source (DC)
1 PWM / On, Off Digital Outputs (DC)
1 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Serial Ports RS232
1 CAN Port
EEPROM Storage  (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port
Wi-Fi Connectivity
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare  (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)

Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## HEC-GW-X-X

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
1 Digital Inputs - Sink / Source (DC)
1 PWM / On, Off Digital Outputs (DC)
1 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
2 Serial Ports RS232
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
OptiCAN Networking
J1939 / NMEA 2000 Networking
GPS Port

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Label
Latching Coil (LATCH)
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PLCHIP Quad Counter (CNTR_PXX_QEI)
PLCHIP Quad Counter Compare (CNTR_PXX_CMP)
PLCHIP Quad Counter Velocity (CNTR_PXX_VEL)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)

Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

# VB-2xxx Series

Each VB-2XXX model supports different features and function blocks based differences of the on-board hardware. When any VB-2XXX model is selected in the Project Settings, some supported functions are automtically installed while others must be manually installed. Additional features may become available via plug-in expansion boards (VB2X-X-X-X)

## VB-2000

### Features

Retentive Memory (FRAM 480 bytes)
12 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
7 Analog Inputs, Range/Type Field Adjustable
1 Analog Output, Range/type Field Adjustable
2 Serial Ports RS232/RS485
1 CAN Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
OptiCAN Networking
MQTT* IoT Communications
SNTP*(Simple Network Time Protocol)
J1939 / NMEA 2000 Networking
VBDSP-X Display Port
Keypad Port
Din Rail Mount
Accepts VB2X-X-X-X Expansion Boards
DCCoAP Communications*

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Print
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)

Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M* (VCLOUD)
Bitwise XOR (XOR)

* VersaCloud, DCCoAP, MQTT and SNTP Enabled using VB2X-X-X VersaCloud M2M Plug-in Expansion Board

## VB-2100

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
12 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
7 Analog Inputs, Range/Type Field Adjustable
1 Analog Output, Range/type Field Adjustable
2 Serial Ports RS232/RS485
1 CAN Port
Ethernet Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
OptiCAN Networking
J1939 / NMEA 2000 Networking
VBDSP-X Display Port
Keypad Port
Din Rail Mount
Accepts VB2X-X-X-X Expansion Boards
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)

Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VB-2120

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
12 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
7 Analog Inputs, Range/Type Field Adjustable
1 Analog Output, Range/type Field Adjustable
2 Serial Ports RS232/RS485
1 CAN Port
Wi-Fi Connectivity
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Wi-Fi
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
OptiCAN Networking
J1939 / NMEA 2000 Networking
VBDSP-X Display Port
Keypad Port
Din Rail Mount
Accepts VB2X-X-X-X Expansion Boards
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)

J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VB-2200

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
12 Digital Inputs - Sink / Source (DC)
8 PWM / On, Off Digital Outputs (DC)
3 High Speed Counter / Timer Inputs - NPN/PNP
SD Card Support
7 Analog Inputs, Range/Type Field Adjustable
1 Analog Output, Range/type Field Adjustable
2 Serial Ports RS232/RS485
1 CAN Port
Ethernet Port
EEPROM Storage (3500 bytes)
Modbus Master / Slave
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
OptiCAN Networking
J1939 / NMEA 2000 Networking
Standard LCD Display Port
Keypad Port
Din Rail Mount
Accepts VB2X-X-X-X Expansion Boards
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)

EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
PWM (PWM)
PWM Frequency (PWM_FREQ)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Serial Print (SERIAL_PRINT)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## P-Series Bear Bones Controllers

Each P-Series Bear Bones controller model supports different features and function blocks based differences of the on-board hardware. When any P-Series Bear Bones model is selected in the Project Settings, some supported functions are automtically installed while others must be manually installed. Additional features may be available using plug-in expansion boards (ICM-BBP13EXP-X-X-X)

### ICM-BB-P13-30

#### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (DC)
8 On, Off Digital Outputs (DC)
SD Card Support
Up to 8 Analog Inputs, Field Adjustable
Up to 1 Analog Output, Field Adjustable
1 CAN Port
Ethernet Port
EEPROM Storage (3500 bytes)
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
OptiCAN Networking
J1939 / NMEA 2000 Networking
Standard LCD Display Port
Keypad Port
Expandable I/O
Accepts ICM-PUI-01 Expansion Board
Accepts VersaCloud M2M Plug-in Expansion Boards
DCCoAP Communications

#### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)

Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## ICM-BB-P13-31

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (DC)
8 On, Off Digital Outputs (DC)
SD Card Support
Up to 8 Analog Inputs, Field Adjustable
Up to 1 Analog Output, Field Adjustable
1 CAN Port
EEPROM Storage (3500 bytes)
MQTT* IoT Communications
SNTP* (Simple Network Time Protocol)
OptiCAN Networking
J1939 / NMEA 2000 Networking
Standard LCD Display Port
Keypad Port
Expandable I/O
Accepts ICM-PUI-01 Expansion Board
Accepts VersaCloud M2M Plug-in Expansion Boards
DCCoAP Communications*

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)

Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

* VersaCloud, DCCoAP, MQTT and SNTP Enabled using VB2X-X-X VersaCloud M2M Plug-in Expansion Board

## ICM-BB-P13-40

### Features

On-Board Real Time Clock
Retentive Memory  (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (120VAC)
8 On, Off Digital Outputs (120VAC)
SD Card Support
Up to 8 Analog Inputs, Field Adjustable
Up to 1 Analog Output, Field Adjustable
1 CAN Port
Ethernet Port
EEPROM Storage  (3500 bytes)
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
OptiCAN Networking
J1939 / NMEA 2000 Networking
Standard LCD Display Port
Keypad Port
Accepts ICM-PUI-01 Expansion Board
Accepts VersaCloud M2M Plug-in Expansion Boards
Expandable I/O
Accepts ICM-PUI-01 Expander
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)

Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## ICM-BB-P13-41

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
8 Digital Inputs - Sink / Source (120VAC)
8 On, Off Digital Outputs (120VAC)
SD Card Support
Up to 8 Analog Inputs, Field Adjustable
Up to 1 Analog Output, Field Adjustable
1 CAN Port
EEPROM Storage (3500 bytes)
MQTT* IoT Communications
SNTP* (Simple Network Time Protocol)
OptiCAN Networking
J1939 / NMEA 2000 Networking
Standard LCD Display Port
Keypad Port
Expandable I/O
Accepts ICM-PUI-01 Expansion Board
Accepts VersaCloud M2M Plug-in Expansion Boards
DCCoAP Communications*

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)

Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Bitwise XOR (XOR)

* VersaCloud, DCCoAP, MQTT and SNTP Enabled using VB2X-X-X VersaCloud M2M Plug-in Expansion Board

# VersaGateway Programmable Communications Gateways

Each VersaGateway model supports different features and function blocks based differences of the on-board hardware. When any VersaGateway model is selected in the Project Settings, some supported functions are automtically installed while others must be manually installed.

## VCG-E-C-G

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Ethernet Port
Modbus Master / Slave
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
GPS Port
512K SRAM
Versacloud M2M+IoT Cellular Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)

Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-E-C-X

### Features

On-Board Real Time Clock
Retentive Memory  (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage  (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Ethernet Port
Modbus Master / Slave
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
512K SRAM
Versacloud M2M+IoT Cellular Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)

Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-E-X-G

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Ethernet Port
Modbus Master / Slave
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
512K SRAM
GPS Port
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)

Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-E-X-X

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Ethernet Port
Modbus Master / Slave
Modbus TCP over Ethernet
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
512K SRAM
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)

Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-W-C-G

### Features

On-Board Real Time Clock
Retentive Memory  (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage  (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Wi-Fi Connectivity
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Modbus Master / Slave
Modbus TCP over Wi-Fi
GPS
512K SRAM
Versacloud M2M+IoT Cellular Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)

Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-W-C-X

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Wi-Fi Connectivity
Modbus Master / Slave
Modbus TCP over Wi-Fi
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
512K SRAM
Versacloud M2M+IoT Cellular Data
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)
Bitwise OR (OR)
PID (PID)

Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-W-X-G

### Features

On-Board Real Time Clock
Retentive Memory  (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage  (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Wi-Fi Connectivity
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Modbus Master / Slave
Modbus TCP over Wi-Fi
GPS
512K SRAM
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)

Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

## VCG-W-X-X

### Features

On-Board Real Time Clock
Retentive Memory (FRAM 480 bytes)
Full Size SD Card Support
Up to 1 Analog Input, Battery Monitor
1 Standard CAN Port
1 NMEA 2000 Compliant CAN Port
EEPROM Storage (3500 bytes)
OptiCAN Networking
J1939 / NMEA 2000 Networking
2 Serial Ports (RS232/RS485)
Wi-Fi Connectivity
MQTT IoT Communications
SNTP (Simple Network Time Protocol)
Webserver
Modbus Master / Slave
Modbus TCP over Wi-Fi
512K SRAM
DCCoAP Communications

### Supported Function Blocks

Less Than (<)
Less Than Equal To (<=)
Not Equal To (<>)
Equal To (=)
Greater Than (>)
Greater Than Equal To (>=)
Absolute Value (ABS)
Arc Cosine (ACOS)
Addition (ADD)
Arc Sine (ASIN)
Arc Tangent (ATAN)
Bitwise AND (AND)
Average (AVG)
Bit Pack (BIT_PACK)
Bit Unpack (BIT_UNPACK)
Convert to Boolean (BOOLEAN)
Compare (CMP)
Cosine (COS)
Count Down (CTD)
Count Up (CTU)
Count Up / Down (CTUD)
Division (DIV)
Drum Sequencer (DRUM_SEQ)
EEprom Read (EEPROM_READ)
EEprom Write (EEPROM_Write)
Exponential (EXP)
Power Function (EXPT)
Falling Edge Detect (F_TRIG)
Floor (FLOOR)
Get Date (GETDATE)
Get Time (GETTIME
Hysteresis (HYSTER)
Convert to Integer (INTEGER)
J1939 Receive PGN (J1939_RX_PGN)
J1939 Transmit PGN (J1939_TX_PGN)
Jump (JMP)
Keypad
Keypad2
Label
Latching Coil (LATCH)
LCD Display
LCD Clear
Limit (LIMIT)
Natural Logarithm (LN)
Base-10 Logarithm (LOG)
Modbus Master (MODBUS_MASTER)
Modbus Master2 (MODBUS_MASTER2)
Modbus Master3 (MODBUS_MASTER3)
Moving Average (MAVG)
Maximum (MAX)
Minimum (MIN)
Modulo (MOD)
MQTT_Connect
MQTT_Publish
MQTT_Receive
Multiplication (MULT)
Multiplexer (MUX)
Bitwise NOT (NOT)
Optican Node Status (OPTICAN_NODESTATUS)
Optican Transmit Message (OPTICAN_TXNETMSG)

Bitwise OR (OR)
PID (PID)
Rising Edge Detect (R_TRIG)
Random (RANDOM)
Convert to Real (REAL)
Rotate Left (ROL)
Rotate Right (ROR)
Reset / Set -Reset Dominant (RS)
Seed Random (SEED)
Select (SEL)
Set Date (SETDATE)
Set Time (SETTIME)
Shift Left (SHL)
Shift Right (SHR)
Sine (SIN)
Square Root (SQRT)
Set / Reset -Set Dominant (SR)
Structured Text Function (ST_FUNC)
Structured Text Function Block (ST_FUNC_BLK)
Subtraction (SUB)
Tangent (TAN)
Convert to Timer (TIMER)
Time Delay Off (TOF)
Time Delay On (TON)
Timer Counter (TimerCounter)
Pulse Timer (TP)
Uart Set Property (UART_SET_PROPERTY)
Unlatching Coil (UNLATCH)
VersaCloud M2M (VCLOUD)
Webserver_Data
Bitwise XOR (XOR)

# CHAPTER 24

## Webserver

This chapter provides detailed information for using the Webserver feature with P-Series PLC on a Chip based targets.

## Chapter Contents

## Disclaimer

**This chapter is intended as an informational tool of how the Embedded Webserver works with some basic instruction how to install it as well as basic examples of coding to send and receive data from web pages (.html) on the webserver to / from the ladder diagram.**

**A strong understanding of web design including html, javascript and JSON is required based on the level of complexity of the final application. This chapter is not intended to instruct how to code html or javascript, but only provide the basic structure necessary for the transfer of data.**

## Webserver Overview

The P-Series Embedded Webserver is an embedded webserver on the P-Series PLC on a Chip products and is accessed from the target's ladder diagram making the ladder side programmable using the P-Series EZ LADDER Toolkit. For this functionality, the target (or product), must have an on-board SD card. As the title indicates, it is a webserver, it also must support a web enabled interface such as Ethernet or Wi-Fi (not cellular)

The webserver is capable of serving stored documents, html (web pages) and files on the SD card to web-enabled devices. The actual web pages (html) and other supporting files are stored on the SD card and the web enabled devices such as a PC or tablet will browse to the files with the webserver handling the interaction as any webserver serving webpages or files.

The P-Series Embedded webserver is an HTTP server that supports JSON formatted data confirming to the Divelbiss EZData API to pass data to and from the ladder diagram running on the hardware target (product).

| | |
|---|---|
| Supported HTTP version: | HTTP 1.0 (with some 1.1 featues) |
| Supported HTTP Method: | GET, POST |

## Webserver Resource Addressing

When using the embedded webserver, care must be taken when addressing resources. Resources can only be referenced using **relative** or **absolute** addresses, but relative addressing is only allowed in specific instances.

**Relative addressing may be used:**
When the the current page is referenceing resources (files or other pages) **AND** the current page is NOT the ***default/root page*** **AND** the resource is NOT a Divelbiss ***EZData resource*** (webserver block in the ladder diagram). The default/root page is defined in the webserver setup when the webserver is installed in the ladder diagram and would be the page pointed to when the browser address is “http://<IP Address/>”

In cases where relative addressing is not allowed or not desired, the following address scheme must be used. It is also permissible to embed the http://<IPADDRESS>/into the path, but it is not required.

| | |
|---|---|
| <Data or Web Tag>: | - For EZ Web data (JSON Data to/from ladder) use: ezdata<br>- For EZ Web file (web browser address bar) use : ezweb |
| <Drive Type>: | - For files stored on SD Card use: sd<br>(<Drive Type> only applicable for EZ Web tag) |
| <Drive Letter>: | - Drive Letter of SD Card that file is located on use: m<br>(<Drive Letter> only applicable for SD card Drive Types) |

## Addressing Examples

### Relative Addressing (Standard HTTP relative addressing is allowed):

For a file located in the same directory as the current page.

**<img src=”image1.png”>**

For a file located in the parent directory as the current page.

**<img src=”../image1.png”>**

### Absolute Addressing :

For a file located on the sd card at “/webserver/images/image1.png”

**<img src=”/ezweb/sd/m/images/image1.png”>**

To access a webserver ladder diagram block named “ParameterGroup1” via javascript

**<var ezWebAPI=”http://<IP Address>/ezdata/ParameterGroup1”>**

The complete path is required to browse to the default / root page and any images or links on this page must be absolute.

An easy solution is to provide a ‘re-direct’ on this page to a new main page on the sd card that will act as the *main* index page of the site. This new main page would allow relative addressing (except for accessing the ezdata / ladder diagram data transfer).

## IP Addressing, Wi-Fi and Ethernet

The IP address is determined by multiple factors. The actual target (product) with the webserver can be configured (see it’s configuration settings) with a static IP or use DHCP.

Static IP refers to an IP address that is manually entered into the target that will not change. This is the IP Address of the target and thus becomes the IP Address of the webserver (http://<IP Address>/).

When connected via Ethernet (static) if the network is configured to support Hostname, then the hostname may be used as the address.

When connected via Wi-Fi, the hostname is not supported, only IP Address may be used.

When configured for DHCP, the IP Address is not hard coded, but does follow certain guidelines. The guidelines are different depending upon which type of connection is used (Ethernet or Wi-Fi).

For Ethernet, when configured as DHCP, the target will acquire its IP Address from the network it is connected to. If the network is configured to support Hostname, then the hostname may be used as the address.

For Wi-Fi, when configured for DHCP and set to client mode, the target will aquire its IP address from the network it is connected to.

If the Wi-Fi is set to Host mode, the target will aquire its IP Address from the internal Wi-Fi module and will default to `192.168.0.101`

## Installing the Webserver in the Ladder Diagram Project

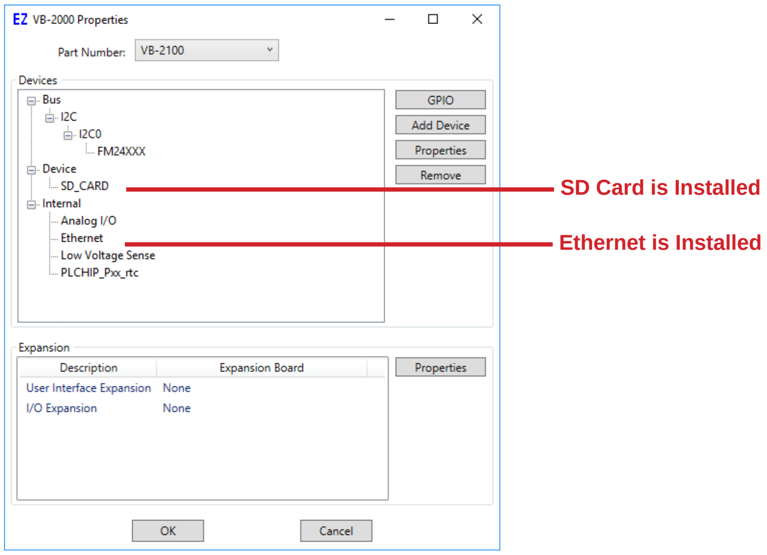
Before the webserver can be used and the webserver function blocks placed in the ladder diagram, the webserver must be installed in the ladder diagram project.

1. Open the ladder diagram to install the webserver in. It is only supported in P-Series targets and must also support the SD card and either Ethernet or Wi-Fi.

2. In EZ LADDER, use the Project.....Settings menu at the top. The Project Settings window will open.

3. Select the target and click the **PROPERTIES** button. The 'target' Properties window will open.

4. Verify the Wi-Fi or Ethernet device and the SD Card is already installed in the project. If not, install them before proceeding. See Figure 24-1. (Refer to the appropriate sections of this manual for details in installing Ethernet/Wi-Fi and SD Card).



Part Number: VB-2100

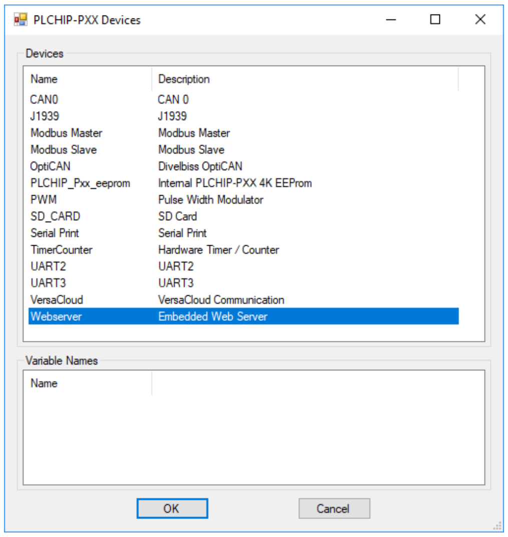
Devices
- Bus
  - I2C
    - I2C0
      - FM24XXX
- Device
  - SD_CARD
- Internal
  - Analog I/O
  - Ethernet
  - Low Voltage Sense
  - PLCHIP_Pxx_rtc

Expansion

| Description | Expansion Board |
|---|---|
| User Interface Expansion | None |
| I/O Expansion | None |

**Figure 24-1**

5. Click the **ADD DEVICE** button. The PLCHIP-PXX Devices window will open. Select the ***Webserver*** and click **OK**. See Figure 24-2.


PLCHIP-PXX Devices

Devices

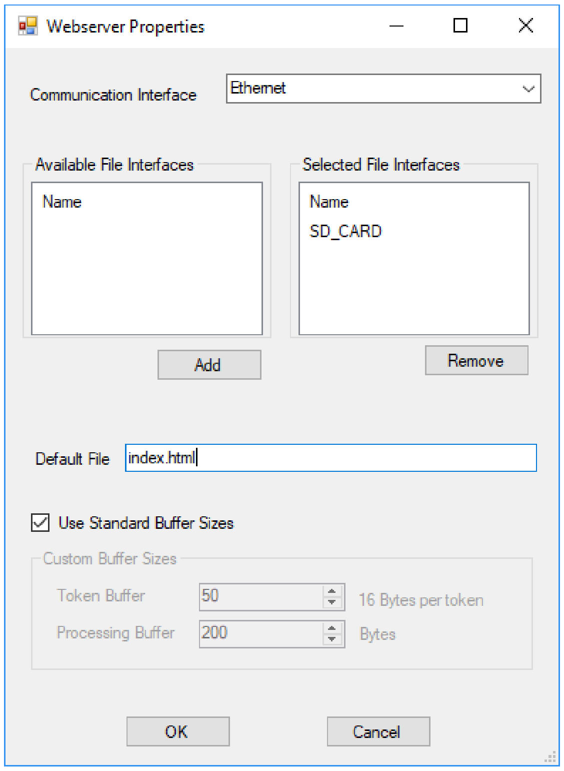
| Name | Description |
|---|---|
| CAN0 | CAN 0 |
| J1939 | J1939 |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |
| UART3 | UART3 |
| VersaCloud | VersaCloud Communication |
| Webserver | Embedded Web Server |

Variable Names

Name

**Figure 24-2**

6. The Webserver Properties window will open. See Figure 24-3.


Webserver Properties

Communication Interface: Ethernet

Available File Interfaces
Name

Selected File Interfaces
Name
SD_CARD

Default File: index.html

[✓] Use Standard Buffer Sizes

Custom Buffer Sizes
Token Buffer: 50 16 Bytes per token
Processing Buffer: 200 Bytes

**Figure 24-3**

7. Select the *Communication Interface* (Ethernet or Wi-Fi). This selects the interface.

8. Select the SD Card in the *Available File Interfaces* pane and click the **ADD** button to move the SD card to the *Selected File Interfaces* pane. This selects the SD card as the storage.

9. Enter the name of the default or root web page in the *Default File field.*

10. Leave the *Use Standard Buffer Sizes* box checked. Typically, these setting are sufficient.

11. Click the **OK** button. The Webserver Properties window will close.

12. Click the **OK** button. The 'target' Properties window will close.

The webserver is now installed in the ladder diagram project making the webserver function block available in the ladder diagram.

## The Ladder Diagram WEBSERVER_DATA Function Block

The WEBSERVER_DATA function block is used in the ladder diagram as the interface to communicate via the embedded webserver to web enabled devices accessing the stored web files on the sd card. This function block allows for variables to be selected that are to be sent / received via the webserver interface.

### To Place the WEBSERVER_DATA function block:

1. From the EZ LADDER Toolkit drop down function menu, select WEBSERVER_DATA.

2. Locate the position in the ladder diagram where the function is to be added and click that positiion. The Webserver Data Group properties window will open. See Figure 24-4.

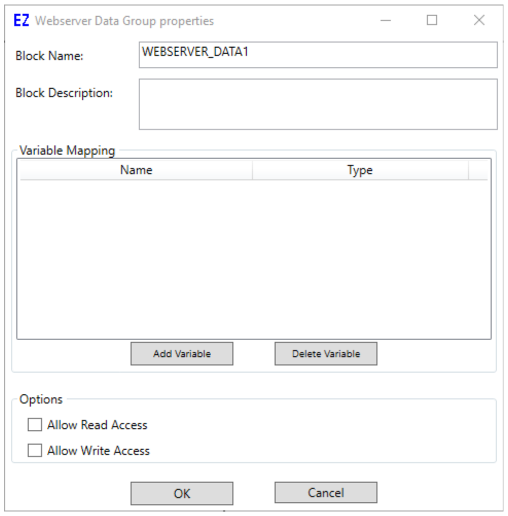


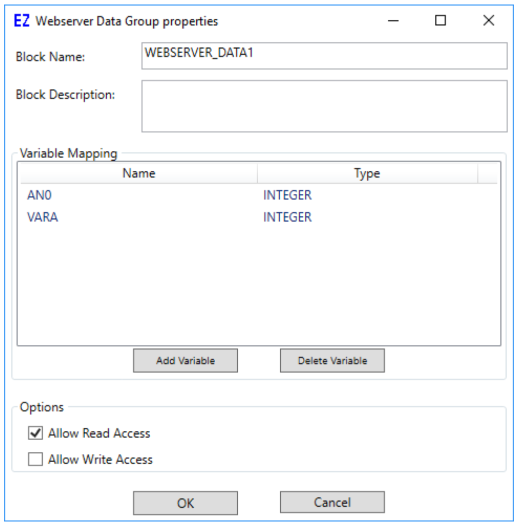
**Figure 24-4**

3. Enter data for the communications to the webserver:

| | |
|---|---|
| Block Name: | This unique name will be used in the web page / javascript / JSON for communications. It should be named uniquely to idenfity the purpose of the block. |
| Block Description: | Optional info for the block to describe what is is used for. |

| | |
|---|---|
| Variable Mapping: | Variables that are being sent / received will appear here. |
| Add Variable button: | Adds variables to be sent / received. The variable must already exist in the ladder diagram. It cannot be created in this dialog. |
| Delete Variable button: | Deletes variables to be sent / received from the block. |
| Allow Read Access: | Allows the webserver to read the variables in this block. |
| Allow Write Access | Allows the webserver to write the variables in this block. |

4. Add the variables and select the options as needed. Refer to Figure 24-5.


EZ Webserver Data Group properties

Block Name: WEBSERVER_DATA1

Block Description:

Variable Mapping

| Name | Type |
|---|---|
| AN0 | INTEGER |
| VARA | INTEGER |

Options

[✓] Allow Read Access

[ ] Allow Write Access

**Figure 24-5**

5. Click the **OK** button. The Webserver Data Group properties window will close and the block will be placed. See Figure 24-6.

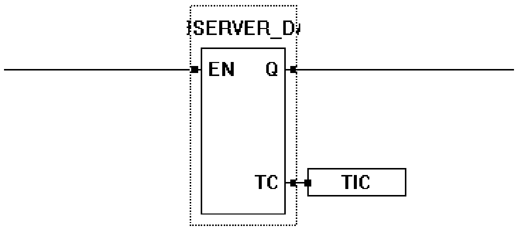


**Figure 24-6**

6. As with all function blocks, the EN (enable) input enables the function block and the Q output is true when the function is enabled. An integer variable must be connected to the TC port. This TC port is the current *tic count* (internal timing tic) and is updated each time this function block communicates to the webserver.

Multiple WEBSERVER_DATA blocks may be used in the ladder diagram.

## The WEBSERVER EZData API

The webserver EZDataAPI is used to send and receive data between the embedded application and the web enabled client (browser). As was previously shown in Chapter 1, all EZData requests are made with the base URL of: `/ezdata/` .

### Data Format

The Divelbiss EZData API uses JSON formatted data.

### Supported Variable Types

Integer, Real, Boolean, Structured Text global variables and arrays (no Structs).

## Pre-Defined Data Objects

There are pre-defined data objects that when used, the connected device will respond with known information that is independent of the WEBSERVER_DATA function blocks in the ladder diagram.

**Read Device Information**

Using GET and `/ezdata/Device_information`

The device will respond (if successful) with the following JSON data:

```
Status: 200 OK

{
 “Device_information”: {
 “Serial_number”: 123456789,
 “Bootloader_version”: “1.1.1.1”,
 “Kernel_version”: “1.2.2.2”,
 “Program_name”: “MyApplication1”,
 “Program_version”: “0.0.0.1”,
 “Program_build”: 50,
 }
}
```

The data will still have to be parsed and used on the web page.

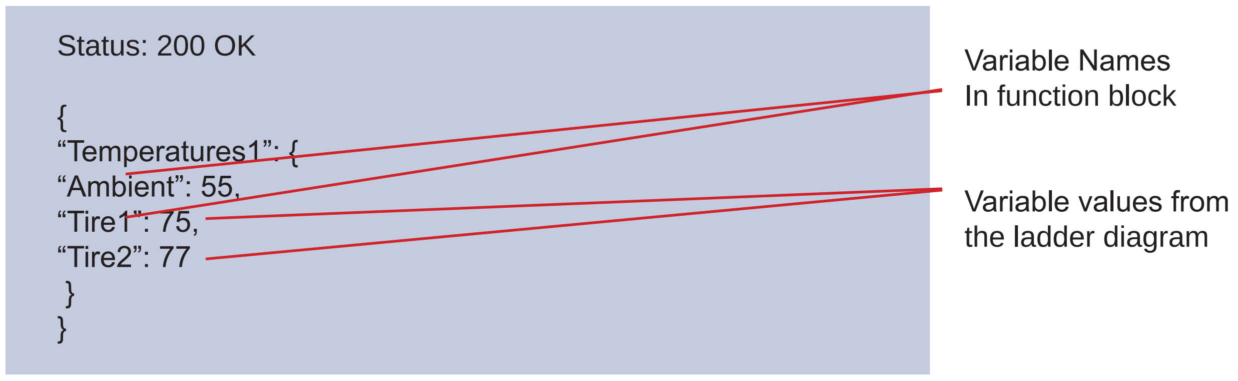
## WEBSERVER_DATA Function Block Data Objects

Web enabled devices communicated via the web page (html, javascript and JSON) to the device using the WEBSERVER_DATA function blocks shown earler. The communication is read and write accessible (using the GET and POST commands).

To read from a WEBSERVER_DATA function block named: `Temperatures1`

Using GET and `/ezdata/Temperatures1`

The device will respond (if successful) with the following JSON data:



```
Status: 200 OK

{
"Temperatures1": {
"Ambient": 55,
"Tire1": 75,
"Tire2": 77
 }
}
```

The data will still have to be parsed and used on the web page.

## Requesting, Sending & Displaying Data

The webserver EZDataAPI is used to send and receive data between the embedded application and the web enabled device as shown previously.

### Requesting & Displaying Data

The following is an simple example of the html (javascript and html) to communicate to the embedded device (webserver block), parse the data and the display the information received on a browser page. This exact example is the ***Simple Data Read No External Files*** example found in the ***webserver_examples.zip***.

This example uses a webserver block named ***sensor_data*** and will request and receive three data points: temperature, pressure and speed. As shown earlier, the variable names in the ladder diagram (webserver block) must be the same as the JSON/javascript to read the data successfully. After the data is read, it is parsed and then three html web page elements (DIV) tags with unique id's are updated with the values received.

**Javascript / JSON Code ( in web page)**

Causes function to run once when page loads

function (and name) to read data and update DIVs

```
<script>
    // Start periodic data download
    downloadData();

     function downloadData()
     {
         var ezWebAPI = "http://" + location.hostname + "/ezdata/sensor_data";

         var xmlhttp = new XMLHttpRequest();

         xmlhttp.onreadystatechange = function() {
              if (xmlhttp.readyState == XMLHttpRequest.DONE ) {
                  if (xmlhttp.status == 200) {
                     // Get JSON data from response
                     var data = JSON.parse(this.responseText);

                     // Update divs with new data
                     document.getElementById('temp').innerHTML = data.sensor_data.temp;
                     document.getElementById('press').innerHTML = data.sensor_data.press;
                     document.getElementById('speed').innerHTML = data.sensor_data.speed;

                     // Set Timer to get next set of data after 1000ms
                     window.setTimeout(downloadData, 1000);
                  }
              }
          };

         xmlhttp.open("GET", ezWebAPI, true);
         xmlhttp.send();
         }

</script>
```

ezweb location and webserver block name for API

DIV ids in web-page to update

webserver block name

Variable Names In function block

These 3 lines copy the values (data) and update the DIVs in the webpage to view

This line schedules this function to run again in 1000 milli-seconds

**HTML Code (in web page)**

```
 <!-- HTML PAGE DATA -->
        <h2>Device Data</h2>
        <table style="font-family: arial, sans-serif; border-collapse: collapse;">
                  <tr><th>Temperature</th><th>Pressure</th><th>Speed</th></tr>
                  <tr><td id="temp"></td><td id="press"></td><td id="speed"></td></tr>
        </table>
```

ids "temp", "press" and "speed" are updated from downloadData function with values from embedded device.

This example is shown using *document.getElementByID* and *innerHTML* as ways to update elements on the webpage; however, there are several web design elements that may be utilized to display and update on the webpage. The correct element to use is dependent upon the needs of the end application.

## Updating / Sending Data to the Embedded Application

The following is a simple example of the html (javascript and html) to communicate to the embedded device (webserver block) to take updated (input data) from the webpage and send the data to the embedded device (embeeded application). This exact example is the ***Simple Data Write No External Files*** example found in the ***webserver_examples.zip***.

This example uses a webserver block named ***sensor_data*** and one of the three data points at a time when the corresponding ***Update*** button is pressed: temperature, pressure and speed. As shown earlier, the variable names in the ladder diagram (webserver block) must be the same as the JSON/javascript to write the data successfully. This example does this a little differently as it receives the variable name from the button that was pressed (the name temp, press or speed is passed to the javascript when the actual Update button is pressed). The HTML uses 3 text form fields to enter data and in this example, there is no range or type checking.

**HTML Code (in web page)**

```
<!-- HTML PAGE DATA -->
            <h2>PLC on a Chip Device Information</h2>
               <table>
                   <tr>
                        <td width="125">Temperature</td>
                        <td  width="150"><input type="text" id="temp"></td>
                        <td><button onclick="sendData('temp')">Update</button></td>
                  </tr>
                  <tr>
                        <td>Pressure</td>
                        <td><input type="text" id="press"></td>
                        <td><button onclick="sendData('press')">Update</button></td>
                 </tr>
                 <tr>
                        <td>Speed</td>
                        <td><input type="text" id="speed"></td>
                        <td><button onclick="sendData('speed')">Update</button></td>
                </tr>
            </table>
```

When Temperature Update button is clicked, sendData function is called and 'passed' the value and name of *temp*.

When Pressure Update button is clicked, sendData function is called and 'passed' the value and name of *press*.

When Speed Update button is clicked, sendData function is called and 'passed' the value and name of *speed*.

**Javascript / JSON Code ( in web page)**

```
<script>

    function sendData(parameterName, parameterValue)
    {
            // Get form form value
            var data_obj = {};
            data_obj[parameterName] = document.getElementById(parameterName).value;
            var obj = {“sensor_data”:data_obj};

            var ezWebAPI = “http://” + location.hostname + “/ezdata/sensor_data”;

            var xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function() {
               if (xmlhttp.readyState == XMLHttpRequest.DONE ) {
                   if (xmlhttp.status == 200) {
                        // Success. Do something here if desired.
                   }
                   else{
                         alert(“Error sending data: ERROR “ + xmlhttp.status)
                   }
               }
            };

          xmlhttp.open(“POST”, ezWebAPI, true);
          xmlhttp.setRequestHeader(“Content-Type”, “application/json;”);
          xmlhttp.send(JSON.stringify(obj));

   }

</script>
```

Function name (passed parameter name or in this case variable name, parameter value in this case new value to send to embedded device.

Converts passed data into JSON object to send to embedded device using API.

Sets up API with webserver location and webserver block name

If send is successful, this code area is active.

If send fails, this code area is active.

## Other HTML / Javascript / JSON Information

The examples shown were for simple pages, one with a read from the embedded webserver and one with a write to the embedded webserver. Functionality may be combined as needed in the application to do both or kept as individual items as long as all the data transmission needs and formats are kept.

As with all HTML web design, the webpages can be developed using HTML, CSS and other languages to make colorful and dynamic layouts to fix the application needs.

## Resource Control

The embedded webserver must parse the JSON data and can only do so one request at a time. The embedded webserver can generally handle read requests from multiple sources provided there are not too many devices simultaneously requesting data or they are not requesting data too frequently  or there are not requesting too much data.  As the number of devices and amount of data is application specific, care must be taken to allow sufficient time for processing.

The Apple Safari browser sometimes sends too many requests, which may cause the embedded webserver to appear to hang until it can process all the requests. When using Safari, it is recommended to use resource locking in the webpage to allow only one javascript (function) to send or receive from the embedded webserver at a time.

## Web Browser Issues

As the embedded webserver relies on a web browser to receive and receive data using javascript and JSON, as well as display data, browser differences and known issues must be considered when developing an application.  It is recommended to test the application for the intended browser and audience.

Here is a list of some browser issues to consider and test for:

**Apple Safari**
If more than one webserver send/receive script on a webpage, Safari may send too many requests at one time to the webserver causing it to appear to lag or hang. Adding resource locking to the scripting to lockout so that only one script can be sending/receiving at time corrects this problem.

On mobile devices, there is no ‘debug’ information to help when developing the application.

**Microsoft Internet Explorer**
Internet Explorer may or may not work depending on the actual version and / or its settings. As the embedded webserver receives data to display, it relies on receiving this data on a consistent basis. Some versions and configurations of Internet Explorer have ***caching*** enabled. This caching will only allow the page to display the values on the first load, but not update values after. This is due to the page does not believe the page has changed. If possible, disable the caching. This is Internet Explorer version dependent. Internet Explorer can provide ‘debug’ information.

**Microsoft Edge**
Edge may or may not work depending on the actual version. As the embedded webserver receives data to display, it relies on receiving this data on a consistent basis. Some versions of Edge ***caching*** seems to work properly allowing for updates to come through while others do not. There are no settings to disable the caching on Edge. Edge can provide ‘debug’ information.

**Google Chrome**
On tested versions, Chrome did not cause have any issues with ***caching*** or resource control. Chrome can provide ‘debug’ information.

# CHAPTER 25

## Password Protection

This chapter provides information on securing ladder diagram projects and targets using EZ LADDER Toolkit native password protection.

## Chapter Contents

## Password Protection Overview

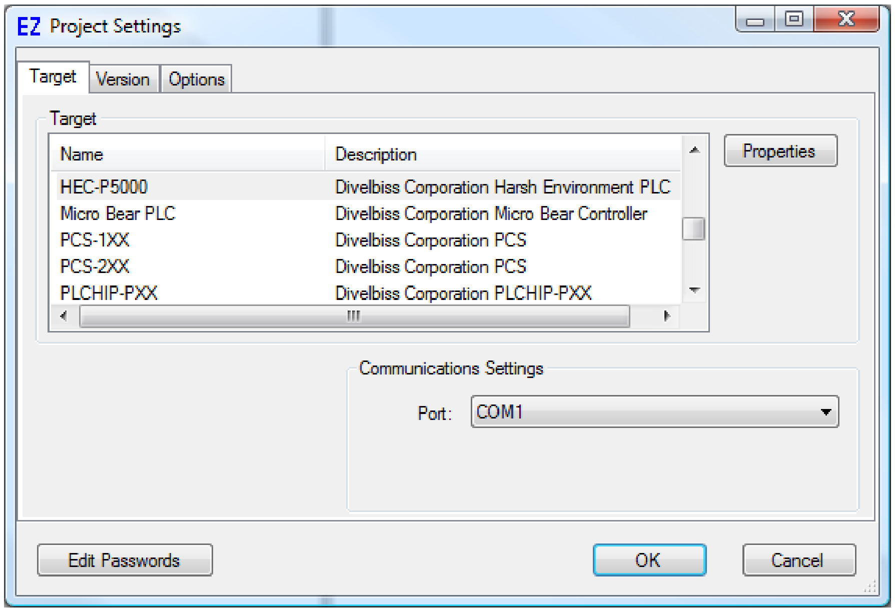
Password protection may be used on all P-Series PLC on a Chip™ targets. There may be multiple passwords for a ladder diagram project. When using Password protection, the Master password is the only pre-named password. Other lower-level passwords and descriptions may be created as needed to provide maximum versatility in allow certain individuals to certain ladder diagram project features.

If password protection is enabled in a ladder diagram project, a password will be required prior to the project opening and being viewed in the EZ LADDER Toolkit Workspace.

## Configuring Password Protection

By default, no passwords are set for a new ladder diagram project (.dld). If password protection is desired, the protection must be configured using the Project...Settings Menu.

Select the target (PLCHIP-PXX) and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target. Click **OK** to save the Target Properties and close the Target Properties Window. You are now back at the Project Settings Window. See Figure 25-1.



| Name | Description |
|---|---|
| HEC-P5000 | Divelbiss Corporation Harsh Environment PLC |
| Micro Bear PLC | Divelbiss Corporation Micro Bear Controller |
| PCS-1XX | Divelbiss Corporation PCS |
| PCS-2XX | Divelbiss Corporation PCS |
| PLCHIP-PXX | Divelbiss Corporation PLCHIP-PXX |

Port: COM1

**Figure 25-1**

In the bottom left corner of the Project Settings window, click the **EDIT PASSWORDS** button. The Password Setup window will open. This window is used to configure the passwords for the ladder diagram project. See Figure 25-2.

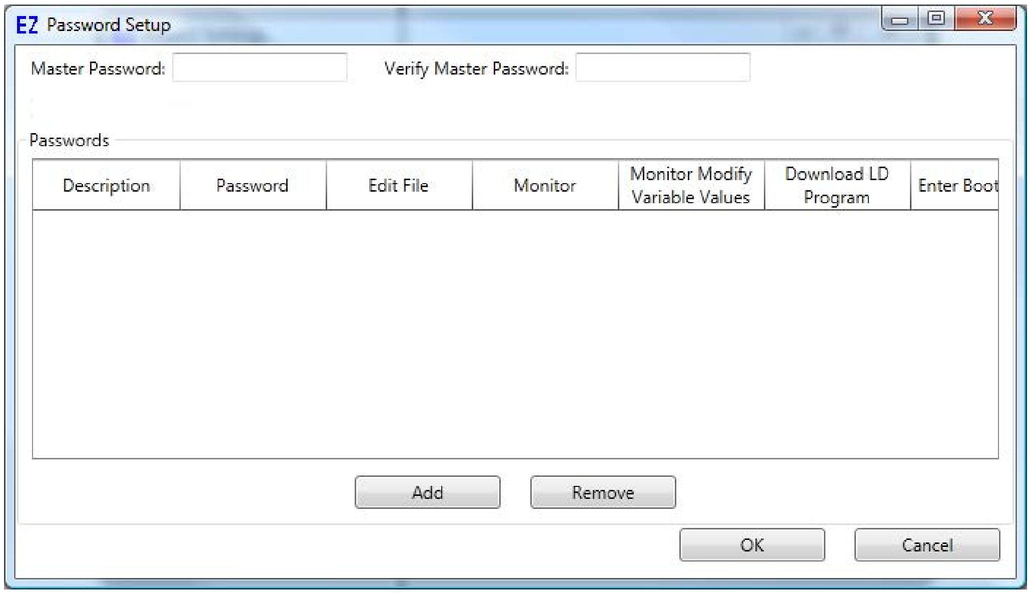


**Figure 25-2**

## Master Password

The Master Password is exactly as phrased. This password is the highest level password in the ladder diagram project. This password should be set and kept closely guarded. When this password is entered, it provides access to all EZ LADDER Toolkit features of the ladder diagram project and allows editing of other level passwords. Enter a master Password in the Master Password box. You will need to re-enter the same password in the Verify Master Password box.

Until the Master Password and Verify Master Password boxes are equal, no other window functions will be functional.

The Master Password should be recorded in a safe place. There is no user-provided way to remove or bypass the Master Password. If you have lost your Master Password, you will need to contact Divelbiss Corporation for options in recovering access to the program.

Once into the Password Setup window, the Master Password can be changed by entering a new password in both the Master Password and the Verify Master Password boxes. It may also be removed by completely clearing both boxes.

If the password Setup is closed by clicking OK, the passwords installed will take effect immediately. Verify you have recorded the Master Password.

## Creating Lower-Level Passwords

Additional passwords for lower-level functions may be created. Refer to Figure 25-2.

To add a new password, click the **ADD** button. A new Row in the Passwords section of the Password Setup window will appear. Refer to Figure 25-3

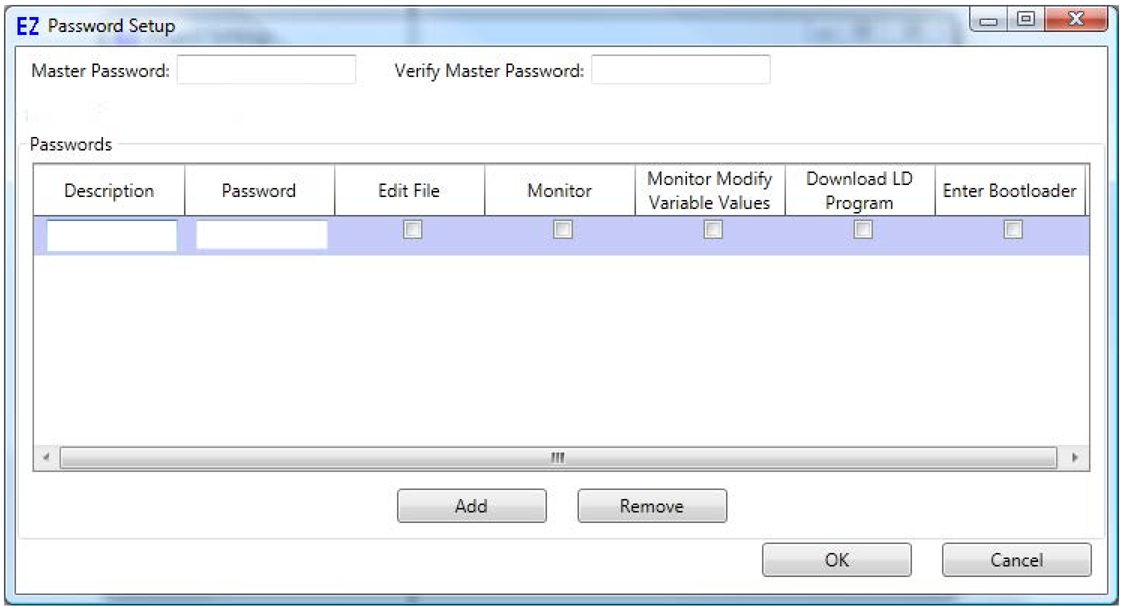


Figure 25-3

| | |
|---|---|
| **Description**: | Enter a description for the password's function. |
| **Password**: | Enter the password. |
| **Edit File**: | When checked, if this password is used, the user will be able to edit the ladder diagram project (ladder diagram). |
| **Monitor**: | When checked, if this password is used, the user will be able to monitor the program in EZ LADDER Toolkit (Run mode) and see the program operate in real time. |
| **Monitor Modify Variable Values**: | When checked, if this password is used, the user will be able to monitor the program in EZ LADDER Toolkit, see the program operate in real time and be able to modify variable values. |
| **Download LD Program**: | When checked, if this password is used, the user will be able to download the ladder diagram program to the hardware target (replacing the current program installed on the hardware target). |
| **Enter Bootloader**: | When checked, if this password is used, the user will be able to enter the Bootloader on the hardware target and will allow the user to make any changes normally allowed in the bootloader, including updating the kernel. |

## Editing Lower-Level Passwords

To edit lower-level passwords, the Master Password is required. When prompted for a password, enter the Master Password. The Password Setup window will be used. Refer to Figure 25-3.

## Removing Lower-Level Passwords

To remove lower-level passwords, the Master Password is required. When prompted for a password, enter the Master Password. The Password Setup window will be used. Refer to Figure 25-3.

Passwords may be deleted by highlighting the row in the window (click on the row) and clicking the **REMOVE** button.

# CHAPTER 26

# Structured Text

This chapter provides information on using Structured Text in EZ LADDER Toolkit.

## Chapter Contents

## Structured Text Overview

Structured Text is a high level textual language which is an IEC61131-3 language. Structured Text; also called ST is used in EZ LADDER to create custom functions and function blocks. Structured Text syntax resembles PASCAL as it is based on it.

Structured Text is a powerful programming tool to create custom functions and function blocks for items such as serial communication drivers, complex mathematical calculations and many more.

**This manual section provides basics on structured text support including supported statements, commands, EZ LADDER specific items and basic syntax and is not a structured text programming manual or tutorial. Other Structured Text sources should be used for a total understanding of how Structured Text is used to accomplish actions. Structured Text is not an intuitive or easy to use language; therefore, it should only be used by those with adequate Structured Text knowledge or experience.**

## Structured Text Technical Support

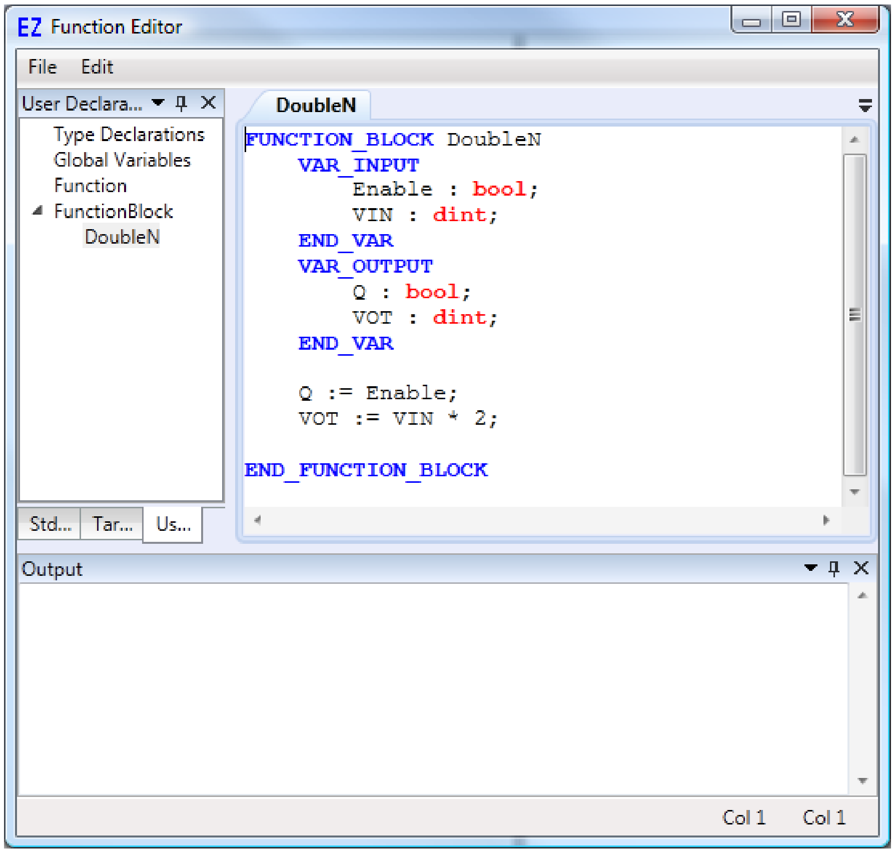
Structured Text technical support for EZ LADDER Toolkit in regards to help with how to create programs, functions and function blocks is limited to our Ticket Support System. Refer to https://www.divelbiss.com/tech.html for details on requirements for submitting a structured text support request via our Ticket Support System. Our Ticket Support System allows the support item to be evaluated and routed to the appropriate support personnel based on the content and needs to provide the quickest and best responses to questions and issues.

## Structured Text Introduction

Structured Text is one of the included languages of the IEC-61131 standard. Using Structured Text, functions and function blocks may be created to perform actions. These functions and function blocks are designed to operate with other programming languages and therefore may 'call' other functions or function blocks and may also be 'called' from the main ladder diagram, other function or function block. If you are familiar with other high level languages such as C, Structured Text (ST) programming will be similar.

Each program, function and function block begin and end with their respective statements. Between the statements, is where all the variables are declared and the actual actions are taken using other commands and statements such as For...Next, etc.  Figure 26-1 is a sample of a custom function block using structured text.

One of the important differences between other programming languages and Structured Text is program flow.  It is possible to program Structured Text so that it will loop indefinitely, giving undesired results and operation, possibly preventing the controller from scanning and operating the ladder diagram correctly.



```
FUNCTION_BLOCK DoubleN
    VAR_INPUT
        Enable : bool;
        VIN : dint;
    END_VAR
    VAR_OUTPUT
        Q : bool;
        VOT : dint;
    END_VAR

    Q := Enable;
    VOT := VIN * 2;

END_FUNCTION_BLOCK
```

**Figure 26-1 - ST Function Editor**

The example shown in Figure 26-1 is the ST code for a function block named ***DoubleN***. This block's purpose is to take an integer input, double it and then output the result.

As shown, the ST language is composed of statements separated by semicolons. These statements and subroutines are used to change variables defined. These defined variables may be defined values such as constants, internally stored variables, input and outputs. Spaces are used to separate variables from statements. Semicolons are used to signify the end of most code lines (as shown in Figure 26-1).

As shown in Figure 26-1, it is good form to indent variable declarations and subroutines. This allows easier reading and scanning of the line of code for debugging purposes.

## Structured Text Variables

Structured Text programming requires named variables to be defined. Variable names must begin with a letter with the remaining name being any combinations of letters, numbers and some symbols (such as '_'). While variable names are not case sensitive, the use of case can be helpful for readability.

Certain variable names are reserved and cannot be used. Names of functions, function blocks, statements and others items are not allowed as variable names. Examples of invalid variable names would be IF, EXP, etc. If an incorrect variable name is used, an error should occur during the code 'Verify'.

## Variable Declarations

The variable declaration statements in figure 26-2 may be used to declare variables and name them in a *callable* structured text function or function block. Certain declarations are limited based on the type of *callable* ST code (if function or function block). **Callable in this instance refers to if the function or function block will be called from the ladder diagram and only applies to the input / output variable declarations.** ST functions (function blocks) called from other ST functions (function blocks) do not have these limitations.

| Declaration | Description | Use with ST_FUNC | Use with ST_FUN_BLK |
|---|---|---|---|
| VAR | A general variable declaration. Used for internal function (block) declarations. Keeps variable values from call to call. | | X |
| VAR_INPUT | Defines variables that are treated as inputs to the function (block). These values are passed into the function (block) from an external source. | X | X |
| VAR_OUTPUT | Defines variables that are treated as outputs to other functions (blocks). These values are passed to other external items (function, etc). | X | X |
| VAR_EXTERNAL | Used to access variables from the ladder diagram or variables declared under the 'Global' user declarations. | | X |
| VAR_TEMP | A general variable declaration. Used for internal function (block) declarations. Does not keep variable values from call to call. | | X |
| VAR_IN_OUT | Defines variables that are treated as both inputs and outputs to other functions (blocks). These values are received from and passed to other external items (function, etc). | | X |

**Figure 26-2 - Variable Declarations as Input/Output of Callable Function or Function Block**

For the variable declarations listed in Figure 26-2, additional restrictions apply depending whether the ST code is a function or a function block. Figure 26-3 lists the type of actual variables that may be used in a ladder diagram *callable* function (function block) when used with VAR_IN, VAR_OUT or VAR_IN_OUT declarations (input / outputs to function or function block). Internal variable declarations are not limited.

| Variable Data Type | Description | Use with ST_FUNC | Use with ST_FUN_BLK |
|---|---|---|---|
| BOOL | Boolean value (0,1, True, False) | X | X |
| DINT | Double Integer (32 bit) | X | X |
| REAL | Float (real) number with decimal point | X | X |

**Figure 26-3 - Variable Data Type for Input/Output Declarations of Callable Functions & Function Blocks**

Only the variable data types listed in Figure 26-3 may be used in ladder diagram callable functions or function blocks as variable declaration types for input or output (VAR_IN, VAR_OUT, VAR_IN_OUT). The use of other data types for input or output variable declarations will result in the function or function block not appearing in the selection menu for inserting the function or function block into the ladder diagram. Other variable data types may be used internally to the function or function block.

As shown in 26-1, all variable declarations begin with the VAR (see Figure 26-2) statement to identify the beginning of the variable declaration and end with the VAR_END statement to identify where the variable declaration ends.

All the previous Figures and examples have illustrated the variable declarations and data types that are supported for the inputs and outputs of callable functions and function blocks. Other data types are supported for functions and function blocks provided they are internal to the function or function block and are not declared as an Input or Output.

## Supported Variable Data Types

The variable types listed in Figure 26-4 are supported in EZ LADDER Toolkit Structured Text. Some limitations apply depending upon the type of function or function block and the type of variable declaration.

| Keyword | Data Type | Bit Size | Internal Data Type | Input/Output Data Type |
|---|---|---|---|---|
| BOOL | Boolean | 1 | X | X |
| SINT | Short Integer | 8 | X | |
| INT | Integer | 16 | X | |
| DINT | Double Integer | 32 | X | X |
| LINT | Long Integer | 64 | X | |
| USINT | Unsigned Short Integer | 8 | X | |
| UINT | Unsigned Integer | 16 | X | |
| UDINT | Unsigned Double Integer | 32 | X | |
| ULINT | Unsigned Long Integer | 64 | X | |
| REAL | Real Numbers | 32 | X | X |
| LREAL | Long Real Numbers | 64 | X | |
| BYTE | Bit String of length 8 | 8 | X | |
| WORD | Bit String of length 16 | 16 | X | |
| DWORD | Bit String of length 32 | 32 | X | |
| LWORD | Bit String of length 64 | 64 | X | |
| ARRAY [..] OF | Array of Type | --- | X | |
| STRING [# of bytes] | ASCII String of [x] bytes in length | --- | X | |

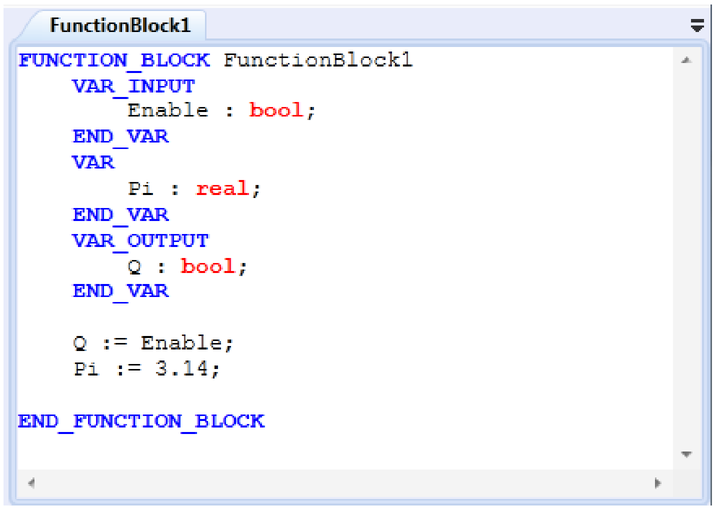
**Figure 26-4 Supported Data Types**

# Structured Text Language

Structured Text (ST) language is uses a combination of assignments, expressions, construct statements, variables and comments to form function or function blocks.

Assignments are used in ST to assign values to tags (variables). Assignments use the operator := (colon and equal signs).

In Structured Text, to set a Variable named Pi to 3.14, see Figure 26-5.



```
FUNCTION_BLOCK FunctionBlock1
    VAR_INPUT
        Enable : bool;
    END_VAR
    VAR
        Pi : real;
    END_VAR
    VAR_OUTPUT
        Q : bool;
    END_VAR

    Q := Enable;
    Pi := 3.14;

END_FUNCTION_BLOCK
```

**Figure 26-5**

Expressions may be part of an assignment or construct statement. An expression evaluates to a number (numerical expression, INT, REAL) or to a true or false condition (BOOL). An expression may contain variables, constants, operators and functions. Refer to Figure 26-6.

| Expression Contains | Description / Definition | Example |
|---|---|---|
| Variables | Variable where data is stored inside the function (BOOL, INT, DINT, REAL) | **Pi** |
| Constants | A hard coded constant value | **6** |
| Operators | A symbol that specifies a operation within an expression (+, -, *, /) | **Vb1 + Vb2** |
| Functions | When executed, a function call returns a single value. Parenthesis are used to contain the operand of a function call. Functions can be used in expressions while instructions cannot. | **functionname (vbl)** |

**Figure 26-6**

Instructions are standalone statements that when executed, uses one or more values that are part of it's data structure. Instructions are terminated with a semi colon (;). Instructions cannot be used in expressions.

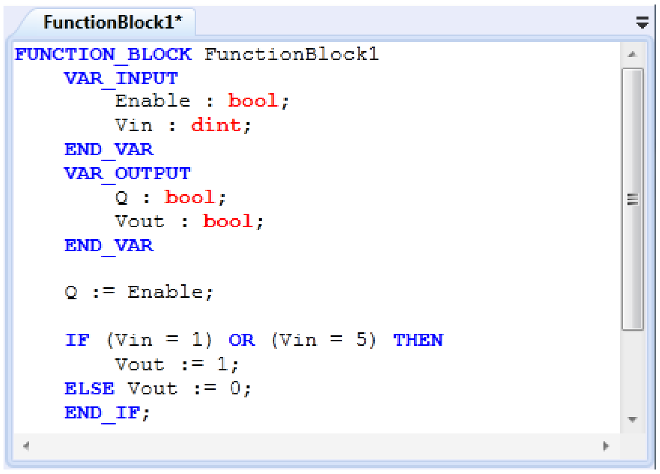
A typical instruction would appear as: *instruction (operand1, operand2...);*

Constructs are conditional statements used to trigger additional structure text code depending upon the evaluation of a condition. Conditional statements are terminated with a semi colon (;).

Typically used constructs:

| | | |
|---|---|---|
| IF...THEN | CASE | FOR...DO |
| WHILE...DO | REPEAT...UNTIL | EXIT |

Figure 26-7 is a sample of using a construct in a function block.

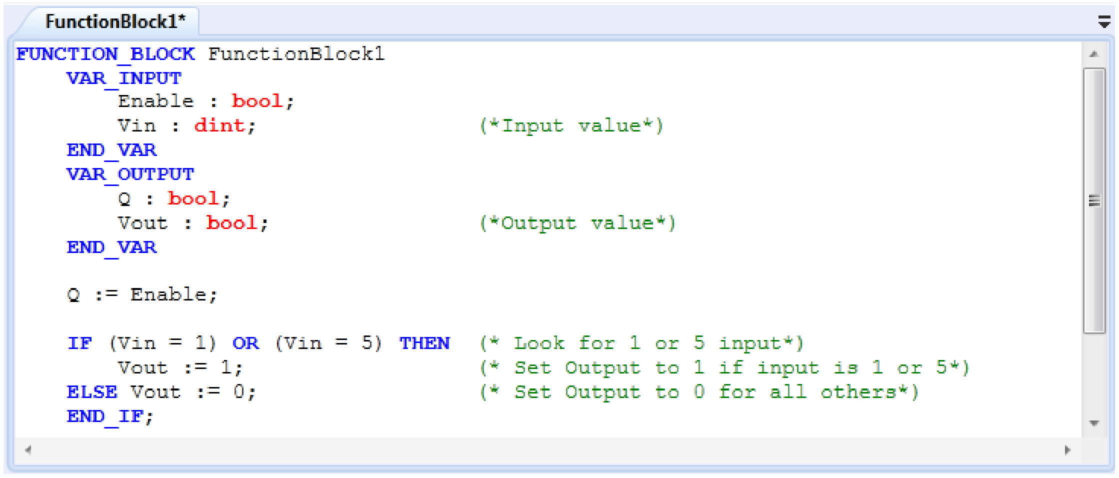


```
FUNCTION_BLOCK FunctionBlock1
    VAR_INPUT
        Enable : bool;
        Vin : dint;
    END_VAR
    VAR_OUTPUT
        Q : bool;
        Vout : bool;
    END_VAR

    Q := Enable;

    IF (Vin = 1) OR (Vin = 5) THEN
        Vout := 1;
    ELSE Vout := 0;
    END_IF;
```

**Figure 26-7**

Comments are used in a function or function block to explain, clarify and document what a section of structured text does and any information the programmer wants to include. Comments are helpful when revisiting code in a function or function block.

Comments may appear anywhere in the structured text and have no effect on the structured text execution. Comments are identified in structured text by the use of parenthesis and asterisks  (* or *).  Figure 26-8 is an example of adding comments to a function block.



```
FUNCTION_BLOCK FunctionBlock1
    VAR_INPUT
        Enable : bool;
        Vin : dint;                 (*Input value*)
    END_VAR
    VAR_OUTPUT
        Q : bool;
        Vout : bool;                (*Output value*)
    END_VAR

    Q := Enable;

    IF (Vin = 1) OR (Vin = 5) THEN  (* Look for 1 or 5 input*)
        Vout := 1;                  (* Set Output to 1 if input is 1 or 5*)
    ELSE Vout := 0;                 (* Set Output to 0 for all others*)
    END_IF;
```

**Figure 26-8**

## Arithmetic Operators & Functions

A major advantage to programming in structured text is the ability to make custom functions or function blocks that include complex calculations and mathematical features. Structured text supports many arithmetic operators and standard functions for creating these complex calculations.

Figure 26-9 lists the standard arithmetic operators supported by EZ LADDER Toolkit's function ST Editor. Although not shown, a semi colon (;) ends each operator line. Parenthesis should be used to control order of operation.

| To | Operator | Example |
|---|---|---|
| Add | + | X := A1 + A2 + A3.... |
| Subtract | - | X := A1 - A2 |
| Multiply | * | X := A1 * A2 * A3.... |
| Exponent ($x^y$) | EXPT | X := A1**A2 |
| Divide | / | X := A1 / A2 |
| Modulo-divide | MOD | X := A1 MOD A2 |

**Figure 26-9**

The arithmetic functions in Figure 26-10 are supported by EZ LADDER Toolkit's Structured Text.

| Function | Description | Syntax |
|---|---|---|
| ABS | Absolute Value of a numeric expression. | ABS (*numeric_expression*) |
| ACOS | Arc Cosine of a numeric expression. | ACOS (*numeric_expression*) |
| ASIN | Arc Sine of a numeric expression. | ASIN (*numeric_expression*) |
| ATAN | Arc Tangent of a numeric expression | ATAN (*numeric_expression*) |
| COS | Cosine of a numeric expression | COS (*numeric_expression*) |
| EXP | Exponential of a numeric expression. | EXP (*numeric_expression*) |
| EXPT | Exponent of two numerical expressions. | EXPT *(num_exp1, num_exp2)* |
| LN | Natural Log of a numeric expression | LN (*numeric_expression*) |
| LOG | Log Base 10 of a numeric expression | LOG (*numeric_expression*) |
| SIN | Sine of a numeric expression | SIN (*numeric_expression*) |
| SQRT | Square Root of a numeric expression | SQRT (*numeric_expression*) |
| TAN | Tangent of a numeric expression | TAN (*numeric_expression*) |
| TRUNC | Truncate a numerical expression. | TRUNC (*numeric_expression*) |

**Figure 26-10**

## Relational Operators

Relational operator compare two values (strings or numerical expressions) and provide a true or false result based on the comparison. The result of a relational operator is always a true or false (BOOL, 0 or 1).

Figure 26-11 lists the supported relational operators for EZ LADDER Toolkit's Structured Text.

| Operator | Description | Example |
|---|---|---|
| = | Equal to. | IF X = 1 THEN... |
| < | Less Than | IF X < 25 THEN.. |
| <= | Less Than or Equal to | IF X <=15 THEN.. |
| > | Greater Than | IF X > 5 THEN.. |
| >= | Greater Than or Equal to | IF X >= 100 THEN.. |
| <> | Not Equal to | IF X <> 6 THEN.. |

**Figure 26-11**

## Logical Operators

Logical operators are used to compare if multiple conditions are true or false. The result of a logical operator is always a true or false (BOOL, 0 or 1). These operators are helpful in determining a status of multiple items and performing an action based on that status. Figure 26-12 lists the supported logical operators in EZ LADDER Toolkit's Structured Text. Parenthesis should be used to control logical flow.

| Operator | Description | Example |
|---|---|---|
| AND | Logical AND | IF X = 1 AND Y = 2 THEN... |
| OR | Logical OR | IF X = 2 OR X=4 THEN.. |
| XOR | Logical Exclusive OR | IF XOR (A,B) = 1 THEN.. |
| NOT | Logical Complement | IF NOT(A) = 1 THEN.. |

**Figure 26-12**

The combination of arithmetic, relational and logic operators allow for complex control of multiple inputs and outputs in a control scheme.

## Bitwise Operators

Bitwise operators are used to manipulate bits within values based on two additional values. Different from logical operators, bitwise operators actually compare the bits of the number and not the entire number. The results will differ.

| Operator | Description | Example |
|---|---|---|
| AND | Bitwise AND | Z := X AND Y |
| OR | Bitwise OR | Z := X OR Y |
| XOR | Bitwise Exclusive OR | Z := A XOR B |
| NOT | Bitwise Complement | Z := NOT B |

**Figure 26-13**

## Order of Execution

Operation written into structured text expressions are performed in a prescribed order. This order may be from left to right, but not always, depending upon the operators and language used.

Operations that have equal order will be performed left to right.

When writing expressions with multiple operations and functions, it is ideal to use parenthesis to group the conditions. This will control the order (or flow) of execution and also makes it easier to read and understand the expression.

## Standard Functions

Figure 26-14 lists the supported EZ LADDER Toolkit's Structured Text standard functions. These functions are standard in structured text and conform to the IEC-61131-3 standard.

| Function | Function | Function | Function | Function |
|---|---|---|---|---|
| BYTE_TO_DINT | INT_TO_LINT | LSB_UDINT_TO_ARRAY | RIGHT | UDINT_TO_DWORD |
| BYTE_TO_DWORD | INT_TO_LREAL | LSB_UINT_TO_ARRAY | ROL | UDINT_TO_INT |
| BYTE_TO_INT | INT_TO_LWORD | LSB_ULINT_TO_ARRAY | ROR | UDINT_TO_LINT |
| BYTE_TO_LINT | INT_TO_REAL | LSB_WORD_TO_ARRAY | SEL | UDINT_TO_LREAL |
| BYTE_TO_LREAL | INT_TO_SINT | LWORD_TO_BYTE | SHL | UDINT_TO_LWORD |
| BYTE_TO_LWORD | INT_TO_UDINT | LWORD_TO_DINT | SHR | UDINT_TO_REAL |
| BYTE_TO_REAL | INT_TO_UINT | LWORD_TO_DWORD | SIN | UDINT_TO_SINT |
| BYTE_TO_SINT | INT_TO_ULINT | LWORD_TO_INT | SINT_TO_BYTE | UDINT_TO_UINT |
| BYTE_TO_UDINT | INT_TO_USINT | LWORD_TO_LINT | SINT_TO_DINT | UDINT_TO_ULINT |
| BYTE_TO_UINT | INT_TO_WORD | LWORD_TO_LREAL | SINT_TO_DWORD | UDINT_TO_USINT |
| BYTE_TO_ULINT | LEFT | LWORD_TO_REAL | SINT_TO_INT | UDINT_TO_WORD |
| BYTE_TO_USINT | LEN | LWORD_TO_SINT | SINT_TO_LINT | UINT_TO_BYTE |
| BYTE_TO_WORD | LIMIT | LWORD_TO_UDINT | SINT_TO_LREAL | UINT_TO_DINT |
| CONCAT | LINT_TO_BYTE | LWORD_TO_UINT | SINT_TO_LWORD | UINT_TO_DWORD |
| DELETE | LINT_TO_DINT | LWORD_TO_ULINT | SINT_TO_REAL | UINT_TO_INT |
| DINT_TO_BYTE | LINT_TO_DWORD | LWORD_TO_USINT | SINT_TO_UDINT | UINT_TO_LINT |
| DINT_TO_DWORD | LINT_TO_INT | LWORD_TO_WORD | SINT_TO_UINT | UINT_TO_LREAL |
| DINT_TO_INT | LINT_TO_LREAL | MAX | SINT_TO_ULINT | UINT_TO_LWORD |
| DINT_TO_LINT | LINT_TO_LWORD | MID | SINT_TO_USINT | UINT_TO_REAL |
| DINT_TO_LREAL | LINT_TO_REAL | MIN | SINT_TO_WORD | UINT_TO_SINT |
| DINT_TO_LWORD | LINT_TO_SINT | MSB_DINT_TO_ARRAY | TO_LSB_DINT | UINT_TO_UDINT |
| DINT_TO_REAL | LINT_TO_UDINT | MSB_DWORD_TO_ARRAY | TO_LSB_DWORD | UINT_TO_ULINT |
| DINT_TO_SINT | LINT_TO_UINT | MSB_INT_TO_ARRAY | TO_LSB_INT | UINT_TO_USINT |
| DINT_TO_UDINT | LINT_TO_ULINT | MSB_LINT_TO_ARRAY | TO_LSB_LINT | UINT_TO_WORD |
| DINT_TO_ULINT | LINT_TO_USINT | MSB_LREAL_TO_ARRAY | TO_LSB_LREAL | ULINT_TO_BYTE |
| DINT_TO_USINT | LINT_TO_WORD | MSB_LWORD_TO_ARRAY | TO_LSB_LWORD | ULINT_TO_DINT |
| DINT_TO_DWORD | LREAL_TO_BYTE | MSB_REAL_TO_ARRAY | TO_LSB_REAL | ULINT_TO_DWORD |
| DWORD_TO_BYTE | LREAL_TO_DINT | MSB_UDINT_TO_ARRAY | TO_LSB_UDINT | ULINT_TO_INT |
| DWORD_TO_DINT | LREAL_TO_DWORD | MSB_UINT_TO_ARRAY | TO_LSB_UINT | ULINT_TO_LINT |
| DWORD_TO_INT | LREAL_TO_INT | MSB_ULINT_TO_ARRAY | TO_LSB_ULINT | ULINT_TO_LREAL |
| DWORD_TO_LINT | LREAL_TO_LINT | MSB_WORD_TO_ARRAY | TO_LSB_WORD | ULINT_TO_LWORD |
| DWORD_TO_LREAL | LREAL_TO_LWORD | MUX | TO_MSB_DINT | ULINT_TO_REAL |
| DWORD_TO_LWORD | LREAL_TO_SINT | REAL_TO_BYTE | TO_MSB_DWORD | ULINT_TO_SINT |
| DWORD_TO_REAL | LREAL_TO_UDINT | REAL_TO_DINT | TO_MSB_INT | ULINT_TO_UDINT |
| DWORD_TO_SINT | LREAL_TO_UINT | REAL_TO_DWORD | TO_MSB_LINT | ULINT_TO_UINT |
| DWORD_TO_UDINT | LREAL_TO_ULINT | REAL_TO_INT | TO_MSB_LREAL | ULINT_TO_USINT |
| DWORD_TO_UINT | LREAL_TO_USINT | REAL_TO_LINT | TO_MSB_LWORD | ULINT_TO_WORD |
| DWORD_TO_ULINT | LREAL_TO_WORD | REAL_TO_LWORD | TO_MSB_REAL | USINT_TO_BYTE |
| DWORD_TO_USINT | LSB_DINT_TO_ARRAY | REAL_TO_SINT | TO_MSB_UDINT | USINT_TO_DINT |
| DWORD_TO_WORD | LSB_DWORD_TO_ARRAY | REAL_TO_UDINT | TO_MSB_UINT | USINT_TO_DWORD |
| FIND | LSB_INT_TO_ARRAY | REAL_TO_UINT | TO_MSB_ULINT | USINT_TO_INT |
| INSERT | LSB_LINT_TO_ARRAY | REAL_TO_ULINT | TO_MSB_WORD | USINT_TO_LINT |
| INT_TO_BYTE | LSB_LREAL_TO_ARRAY | REAL_TO_USINT | TRUNC | USINT_TO_LREAL |
| INT_TO_DINT | LSB_LWORD_TO_ARRAY | REAL_TO_WORD | UDINT_TO_BYTE | USINT_TO_LWORD |
| INT_TO_DWORD | LSB_REAL_TO_ARRAY | REPLACE | UDINT_TO_DINT | USINT_TO_REAL |

| Function | Function | Function | Function |
|---|---|---|---|
| USINT_TO_SINT | WORD_TO_BYTE | WORD_TO_LREAL | WORD_TO_UINT |
| USINT_TO_UDINT | WORD_TO_DINT | WORD_TO_LWORD | WORD_TO_ULINT |
| USINT_TO_UINT | WORD_TO_DWORD | WORD_TO_REAL | WORD_TO_USINT |
| USINT_TO_ULINT | WORD_TO_INT | WORD_TO_SINT | |
| USINT_TO_WORD | WORD_TO_LINT | WORD_TO_UDINT | |

**Figure 26-14**

Other functions such as *EZ_EEPromRead, EZ_EEPromReadArray, EZ_EEPromWrite, EZ_EEPromWriteArray, EZ_FormatString, EZ_TimeDateCalenderToUnix and EZ_TimeDateUnixToCalendar* functions may be listed in the **Std Function List** tab because they are supported on all P-Series PLC on a Chip targets. They are covered in detail in **Appendix B - Target Specific ST Function Reference** of this manual. Standard structured text functions are covered in detial in **Appendix C - Standard ST Function Reference.**

## Constructs & Statements

Figure 26-15 lists the supported EZ LADDER Toolkit's Structured Text constructs and statements.

| Construct / Statement | Structure |
|---|---|
| IF...THEN / ELSIF...THEN / ELSE | IF_statement ::=<br>'IF' expression 'THEN' statement_list<br>{'ELSIF' expression 'THEN' statement_list}<br>['ELSE' statement_list]<br>'END_IF' |
| CASE...OF / ELSE | CASE_statement ::=<br>'CASE' expression 'OF'<br>case_element<br>{case_element}<br>['ELSE' statement_list]<br>'END_CASE' |
| FOR / WHILE / REPEAT / EXIT | FOR_statement ::=<br>'FOR' control_variable ':=' for_list 'DO' statement_list 'END_FOR'<br>control_variable ::= identifier<br>FOR_LIST ::= expression 'TO' expression ['BY' expression]<br><br>WHILE_statement ::= 'WHILE' expression 'DO' statement_list 'END_WHILE'<br><br>REPEAT_statement ::=<br>'REPEAT' statement_list 'UNTIL' expression 'END_REPEAT'<br><br>EXIT_statement ::= 'EXIT' |

**Figure 26-15**

## Structured Text Function Blocks

EZ LADDER Toolkit provides two types of function blocks that use structured text: ST_FUNC and ST_FUN_BLK. These two function blocks are similar is some ways but different in others.

These blocks may be callable; referring to the function blocks may be inserted into a ladder diagram project and the ladder diagram *call* or cause the structured text to operate inside the function block based on the inputs to the block. Certain rules must be followed for a function block (either ST_FUNC or ST_FUNC_BLK) may be used as a callable function (block).

## ST_FUNC Function (Function Block)

The ST_FUNC function block is used to place a ST User-Defined function into a ladder diagram. When the ST_FUNC function block is placed, a drop-down menu is used to select the functions from the User-Defined Functions (created by you in the ST Editor).

To place a ST_FUNC, use the drop-down function block menu in the toolbar just as any other ladder function / function block. Select ST_FUNC and click to place it in the ladder where it is required.

Structured text user-defined functions differ from function blocks. When a function's instance is called, it executes and internal variables are used only for that instance and call. Variable values are not kept from one execution (or scan) to another while function blocks pass their variables from one execution (scan) to another making them ideal when variables would need to be passed from scan to scan such as a count.

For a user-defined function to be usable as a callable function (inserted in the ladder diagram), it must meet the following criteria:

- Return Type must be a BOOL.
- May only contain input/output variable declarations: VAR_INPUT, VAR_OUTPUT
- Variable Input/Output types may only be BOOL, DINT or REAL. Internal variables may be any supported type.

## ST_FUNC_BLK Function Block

The ST_FUNC_BLK function block is used to place a ST User-Defined functionblock into a ladder diagram. When the ST_FUNC_BLK function block is placed, a drop-down menu is used to select the functionblocks from the User-Defined Functionblocks (created by you in the ST Editor).

To place a ST_FUNC_BLK, use the drop-down function block menu in the toolbar just as any other ladder function / function block. Select ST_FUNC_BLK and click to place it in the ladder where it is required.

Structured text user-defined functions differ from function blocks. When a function's instance is called, it executes and internal variables are used only for that instance and call. Variable values are not kept from one execution (or scan) to another while function blocks pass their variables from one execution (scan) to another making them ideal when variables would need to be passed from scan to scan such as a count.

For a user-defined functionblock to be usable as a callable function (inserted in the ladder diagram), it must meet the following criteria:

- First Output type must be a BOOL.
- May only contain input / output variable declarations: VAR_INPUT, VAR_OUTPUT, VAR, VAR_EXTERNAL, VAR_TEMP
- Variable Input/Output types may only be BOOL, DINT or REAL. Internal variables may be any supported type.

## Target Specific ST Functions

Target specific ST Functions refers to functions in addition to the standard functions listed earlier that are required interact with specific features on the actual hardware target, such as Uarts, SPI ports, etc. These functions are provided by Divelbiss Corporation as part of EZ LADDER Toolkit and cannot be edited, deleted or added to except by Divelbiss. Refer to **Appendix B - Target Specific ST Function Reference** for details on implementing and using target specific functions.

All Target Specific ST function names begin with EZ_. This is an easy way to identify a target specific ST functions by name alone. For target specific functions to be listed in the Target Specific ST tab, the devices that functions would access must be installed in the Project...Settings first (Ethernet, UART, etc).

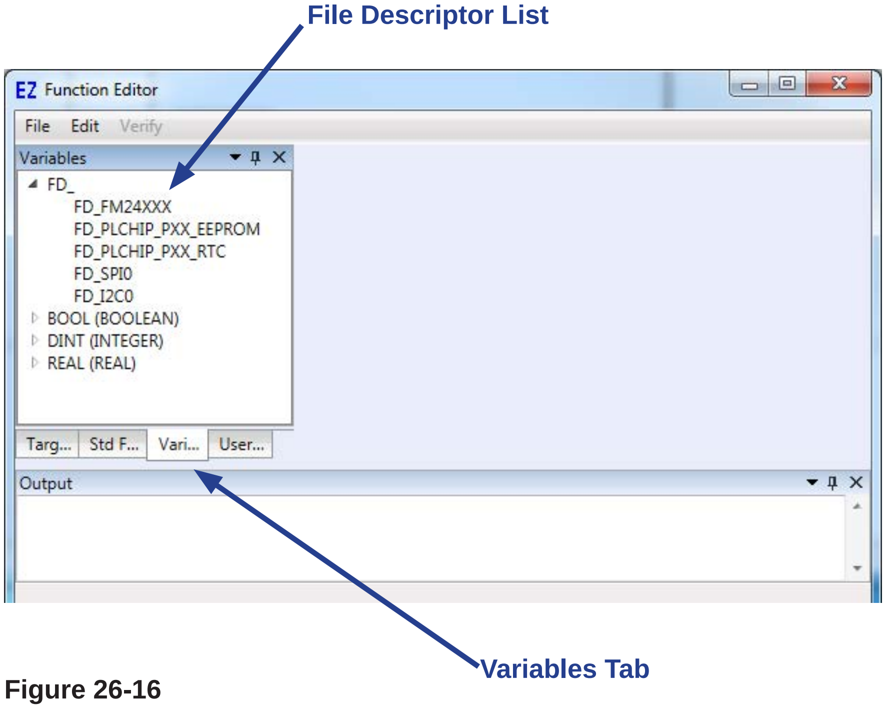
Other functions such as *EZ_EEPromRead, EZ_EEPromReadArray, EZ_EEPromWrite, EZ_EEPromWriteArray, EZ_FormatString, EZ_TimeDateCalenderToUnix and EZ_TimeDateUnixToCalendar* functions may be listed in the **Std Function List** tab because they are supported on all P-Series PLC on a Chip targets. They are covered in detail in **Appendix B - Target Specific ST Function Reference** of this manual. Standard structured text functions are covered in detial in **Appendix C - Standard ST Function Reference.**

### File Descriptors

Many of the target specific functions require a *File Descriptor* that acts as a device locator that the structured text uses to locate the specific resource. They listed in the table as *FD_x*. For example, to use the EZ_EE-PromReadArray, you must have the file descriptor for the actual EEPROM that is the device to read from.

The file descriptors for devices may be found in the Structured Text Editor. Click the Variables tab at the bottom left. All devices listed under FD_ are actual file descriptors. Refer to Figure 26-16. With an open function block to edit, double-clicking on any variable including file descriptors (FD_) will insert the it into the cursors location in the function block.

Only devices installed in the Project Settings will appear. If the device does not appear, check the Project Settings and install the device.



FD_
FD_FM24XXX
FD_PLCHIP_PXX_EEPROM
FD_PLCHIP_PXX_RTC
FD_SPI0
FD_I2C0
BOOL (BOOLEAN)
DINT (INTEGER)
REAL (REAL)

**Figure 26-16**

## EZ LADDER Structured Text Editor

To create structured text functions and functionblocks, EZ LADDER Toolkit has a built-in editor. To open the ST Editor, click the **EDIT ST FUNCTIONS** button located on the toolbar. Refer to Figure 26-17.

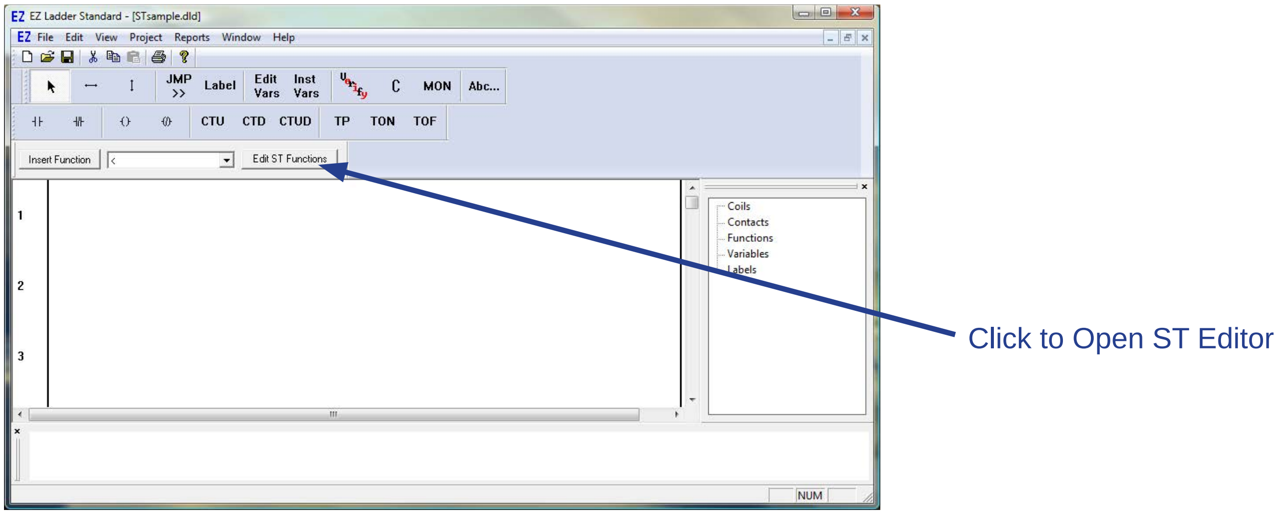
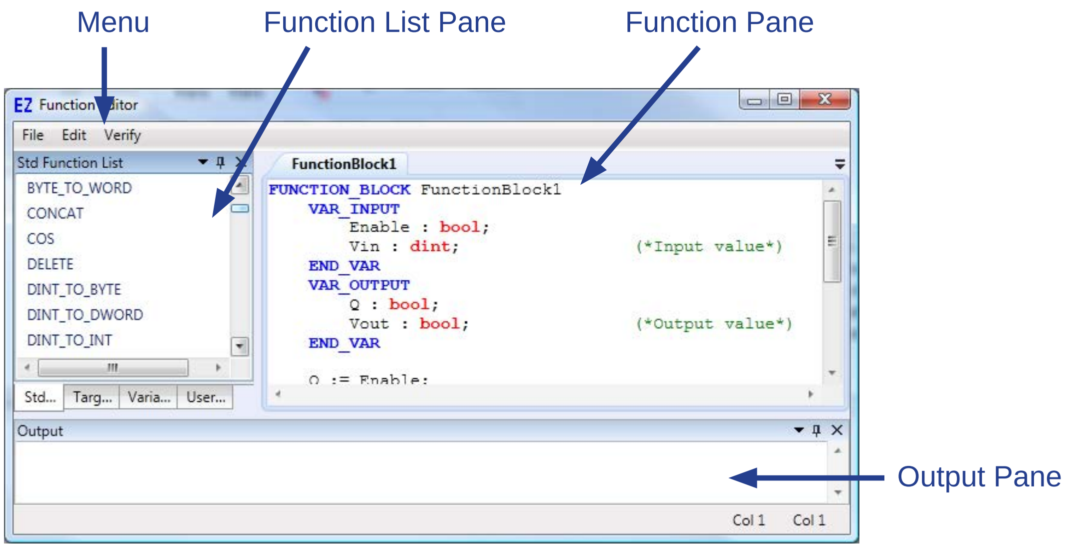


**Figure 26-17**

The ST Editor window (labeled Function Editor) will open It has it’s own menu and is divided into three distinct sections: Function List Pane, Function Pane and Output Pane. Each pane has it’s own purpose. Refer to Figure 26-18.



```
FUNCTION_BLOCK FunctionBlock1
    VAR_INPUT
        Enable : bool;
        Vin : dint;                    (*Input value*)
    END_VAR
    VAR_OUTPUT
        Q : bool;
        Vout : bool;                   (*Output value*)
    END_VAR

    Q := Enable;
```

**Figure 26-18**

## Function List Pane

The function list pane lists the functions and function blocks by type. Four tabs are located at the bottom of the pane. The first tab is for Standard functions such as ASIN and LIMIT. The second tab is for Target Specific functions (as previously mentioned). Only the target specific functions listed in this tab are available to your selected hardware target. The third tab list the variables. The variables listed are the variable file descriptors for accessing specific target hardware (and ladder diagram variables) from structured text. The fourth tab is for User-Declarations functions. This tab will hold any functions or function blocks that you have created. Selecting the tab changes the contents for the pane. Figure 26-19 is a sample of each tab contents.

Target specific functions will only appear in the second tab if the feature is supported on the hardware target and has been installed in the Project Settings. If the expected target specific function is not listed, check the availability of the feature in the hardware and/or verify it is installed in the Project Settings. Refer to **Chapter 4 - Configuring Targets** and other specific chapters based on features required for installation in the Project Settings.

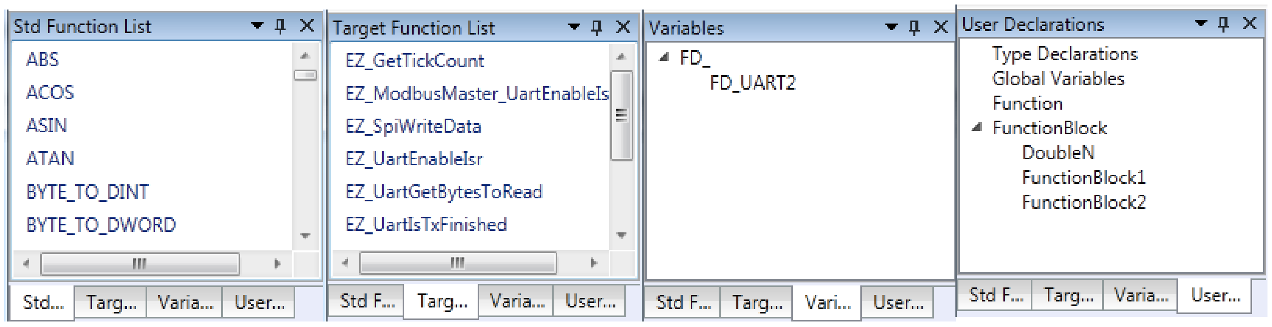
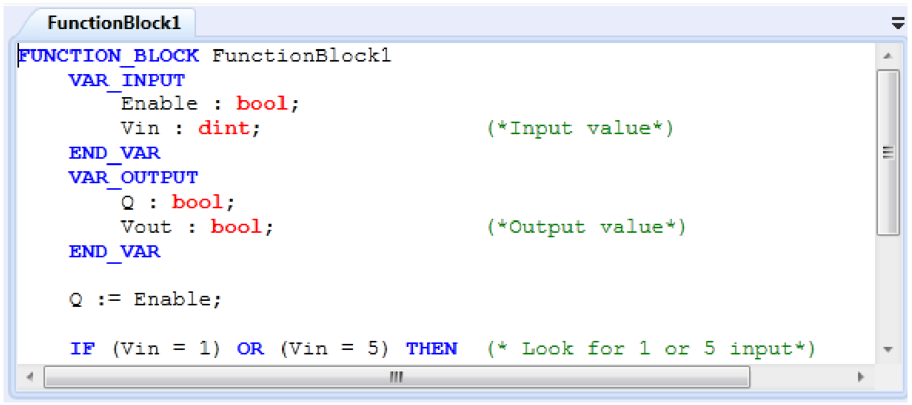


**Figure 26-19**

When creating and editing User-Defined functions, placing the cursor at a location in the function pane and then double-clicking a Standard function, Target Specific function or variable in the functions pane will place the item in your user-defined function.

In the User-Declaration tab, the user defined declarations, variables, functions and function blocks are sorted by type. Expanding the type will list any defined items. In Figure 26-19, the Function Block type is expanded showing three user-defined function blocks have been created.

The function pane is the area that user-defined functions and function blocks structured text code is edited. When a user-defined function is open, the window will act as a text editing window. The structured text code is entered and manipulated as in any other text editor. Figure 26-20 is a sample of the function pane with an open function block.



```
FUNCTION_BLOCK FunctionBlock1
    VAR_INPUT
        Enable : bool;
        Vin : dint;                (*Input value*)
    END_VAR
    VAR_OUTPUT
        Q : bool;
        Vout : bool;               (*Output value*)
    END_VAR

    Q := Enable;

    IF (Vin = 1) OR (Vin = 5) THEN  (* Look for 1 or 5 input*)
```

**Figure 26-20**

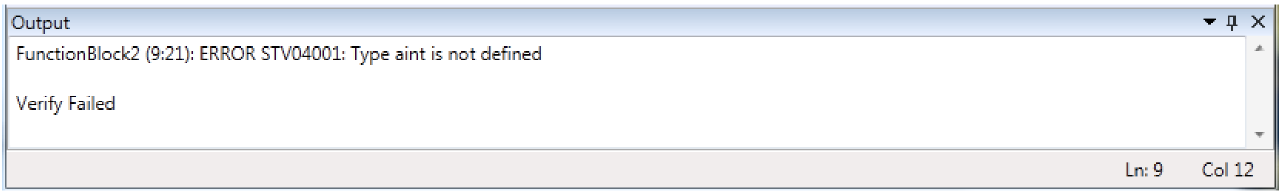
The function pane colors text based on what is detected. In Figure 26-20, constructs and statements are colored blue, variable types are colored red, comments are colored green and standard text is colored black. This coloring is automatic.

After a function has been changed in any way, is must be verified and saved before it may be used. See the *User-Defined Functions and Function blocks* section of this chapter.

## Output Pane

The Output Pane is where important messages and status about the open structured text user-defined functions and function blocks are listed including when any function or function block is verified for proper syntax and content. Figure 26-21 is an example of a function block that has failed verification with an error. This error would need to be corrected before the function block can be used. Note, the location (line and column) are shown in the lower right hand corner identifying the location where the error was detected.

The detected error line and column may be approximate as the verify may detect the error on the next line or column after the actual error occurs. The detection depends on the ST functions and commands being used as to how the error may be detected.  Double-clicking the error in the Output window will place the cursor at the location that the error is detected.



```
FunctionBlock2 (9:21): ERROR STV04001: Type aint is not defined

Verify Failed
```

**Figure 26-21**

## User-Defined Functions & Function Blocks

As previously discussed, custom structured text functionality is added using user-defined functions or function blocks. These functions (blocks) can be called from other ST functions (blocks) or from the ladder diagram. Restrictions apply depending how the user-defined function (block) is to be called. See earlier sections of this chapter.

### Creating a New User-defined Function or Function Block

1. To create a new user-defined function or function block, open the ST Editor using the **EDIT ST FUNCTIONS** button. The ST Editor will open.
2. Click the User-Declarations tab to view the user-defined items.
3. Single Click to highlight the Functionblock (or Function) heading in the User-Declarations tab.
4. Right click and select the ADD from the menu.
5. A new function block will be created and opened in the Function Pane. This function block will have all minimal requirements including the function block beginning and ending code required. This function block can be edited and added to as required.

To quickly add a standard function, Target-specific function or variable to the function (function block), place the cursor as the desired insertion point in the function (function block); click the appropriate tab and double-click the desired function to add. The item is added into the location selected in the user-defined function (function block).

The next step with your code added is to verify the user-defined function (function block). In the ST Editor window, click the **Verify** item from the menu. In the Output Pane, either an error message will be seen or the *Verify Passed* will be seen. If an error message is shown, the error must be corrected before proceeding. If the *Verify Passed* is shown, it is time to save the function (function block).

An unverified function or function block may be saved without verification but it will not be usable until it is verified and will not show in drop down menu to call the function or function block from the ladder diagram.

To save the user-defined function or function block, using the ST Editor menu, click the File...then Save option. The function or function block is now saved.

## Editing an Existing User-defined Function or Function Block

1. To edit a new user-defined function or function block, open the ST Editor using the **EDIT ST FUNCTIONS** button. The ST Editor will open.
2. Click the User-Declarations tab to view the user-defined items.
3. Single Click to expand the FunctionBlock (or Function) heading in the User-Declarations tab.
4. Double-click the desired function or function block to edit.
5. The function block (or function) will be opened in the Function Pane. This function can be edited by adding to or removing structured text as required.

To quickly add a standard function, Target-specific function or variable to the function (function block), place the cursor as the desired insertion point in the function (function block); click the appropriate tab and double-click the desired function to add. The item is added into the location selected in the user-defined function (function block).

The next step with your code added is to verify the user-defined function (function block). In the ST Editor window, click the **Verify** item from the menu. In the Output Pane, either an error message will be seen or the *Verify Passed* will be seen. If an error message is shown, the error must be corrected before proceeding. If the *Verify Passed* is shown, it is time to save the function (function block).

An unverified function or function block may be saved without verification but it will not be usable until it is verified and will not show in drop down menu to call the function or function block from the ladder diagram.

When verifying functions and function blocks, any function or function block with an error will be identified and displayed in the Output window. Ensure you are looking at the correct function or functionblock based on the name when finding errors.

To save the user-defined function or function block, using the ST Editor menu, click the File...then Save option. The function or function block is now saved.

## Copying / Pasting ST Functions & Function Blocks

To use EZ LADDER Toolkit Structured Text functions and function blocks in multiple EZ LADDER programs, the structured text functions must be copied from the source ladder diagram to the target ladder diagram or exported from the source program and imported into the target ladder diagram. Functions and function blocks created in EZ LADDER Toolkit are specific to the ladder diagram in which they are created.

Copying / Pasting the ladder diagram inserted function block (ladder diagram symbol of a structured text function block) will not properly copy the structured text function block correctly. All copy / paste actions must be completed as structured text source using the ST Editor. Failure to use the correct copy and paste method will result in Compile Errors.

To Copy / Paste a structured text function or function block:

1. Open the source ladder diagram program and the target ladder diagram program in EZ LADDER Toolkit.

2. With the source ladder diagram window selected (use the ***Window*** menu on the tool bar), click the **EDIT ST FUNCTIONS** button (located next to the Insert Function drop-down menu on the tool bar) to open the Structured Text Editor.

3. Click the User Defined Functions tab (see Figure 26-22) to view the list of user defined functions and function blocks. Select the function / function block to copy. You may need to expand the Function or FunctionBlock heading.

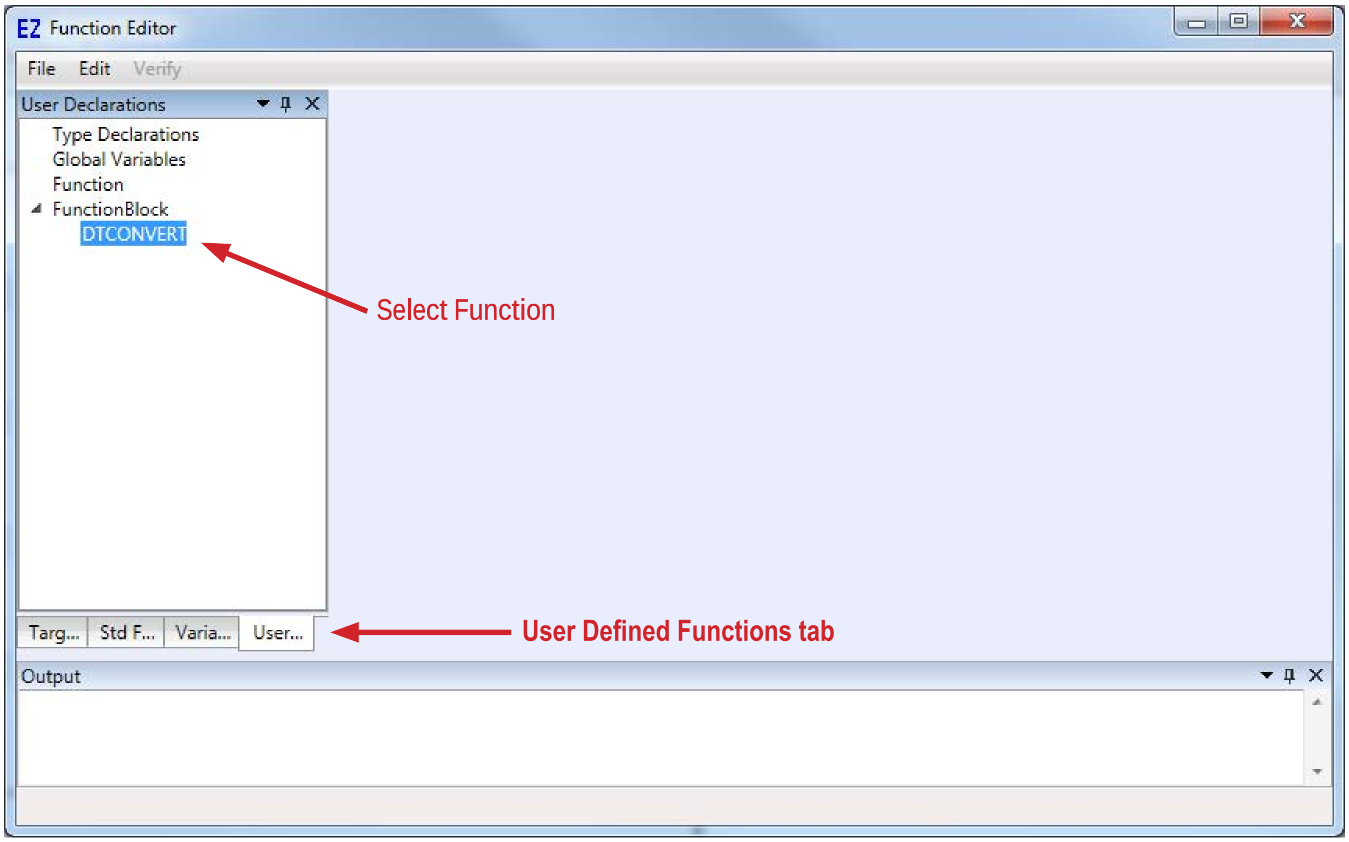
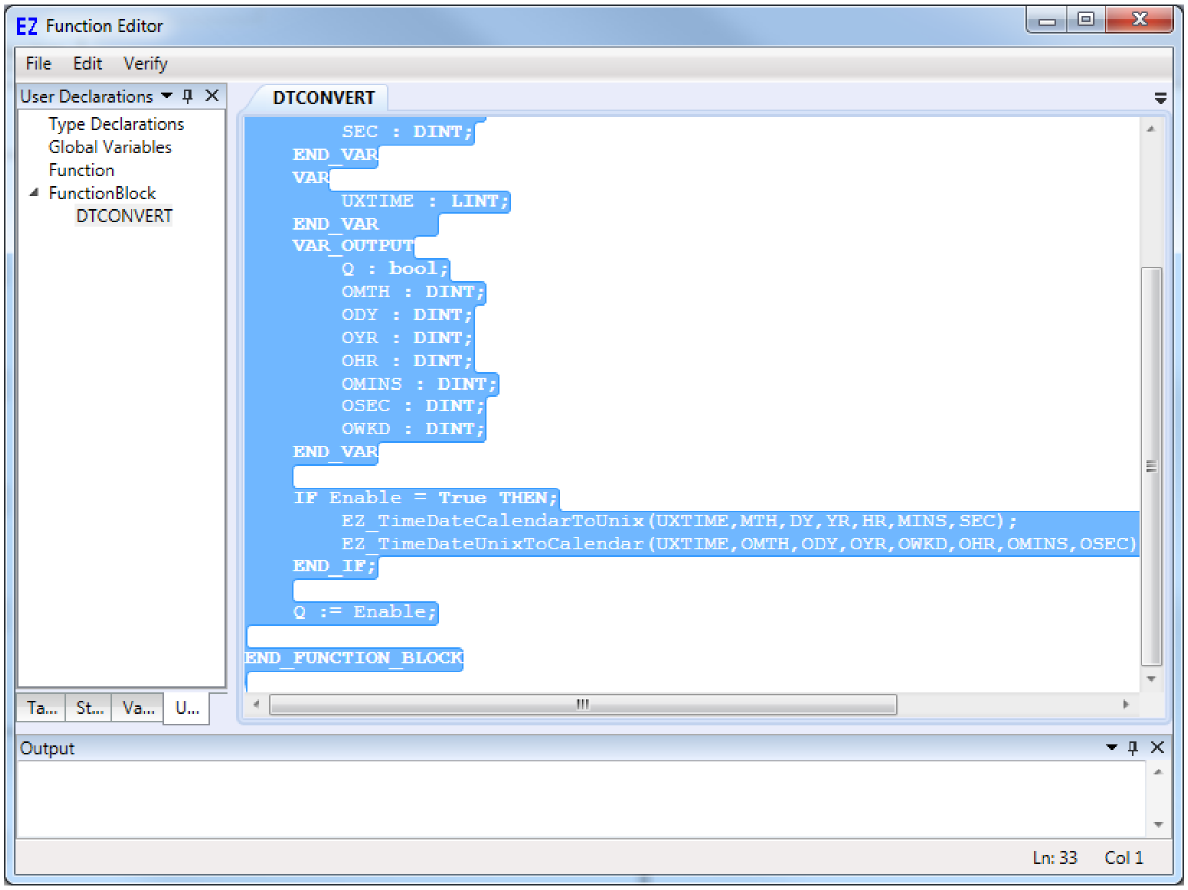


**Figure 26-22**

4. Double-click the selected function / function block to open the function / function block source code. In the source code window, select all (highlight) the text in the window. Verify all the text is selected (including text that is below the current viewable window without scrolling). Refer to Figure 26-23.



```
        SEC : DINT;
    END_VAR
    VAR
        UXTIME : LINT;
    END_VAR
    VAR_OUTPUT
        Q : bool;
        OMTH : DINT;
        ODY : DINT;
        OYR : DINT;
        OHR : DINT;
        OMINS : DINT;
        OSEC : DINT;
        OWKD : DINT;
    END_VAR

    IF Enable = True THEN;
        EZ_TimeDateCalendarToUnix(UXTIME,MTH,DY,YR,HR,MINS,SEC);
        EZ_TimeDateUnixToCalendar(UXTIME,OMTH,ODY,OYR,OWKD,OHR,OMINS,OSEC)
    END_IF;

    Q := Enable;

END_FUNCTION_BLOCK
```

**Figure 26-23**

5. Use CTRL-C or the menu and select ***Edit...Copy***. The function block source code is now stored in the Windows scratchpad. Close the Structured Text Editor.

6. Select the target ladder diagram (use the ***Window*** menu on the tool bar), click the **EDIT ST FUNCTIONS** button (located next to the Insert Function drop-down menu on the tool bar) to open the Structured Text Editor.

7. Click the User Defined Functions tab as before to view the list of user defined functions and function blocks. Select the appropriate type (function or function block) and right-click and select **Add**. Refer to Figure 26-24. A new function / function block will be created.

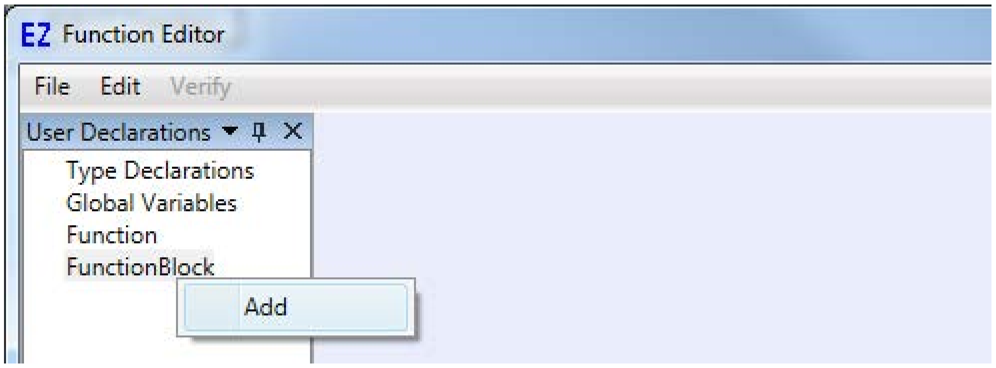
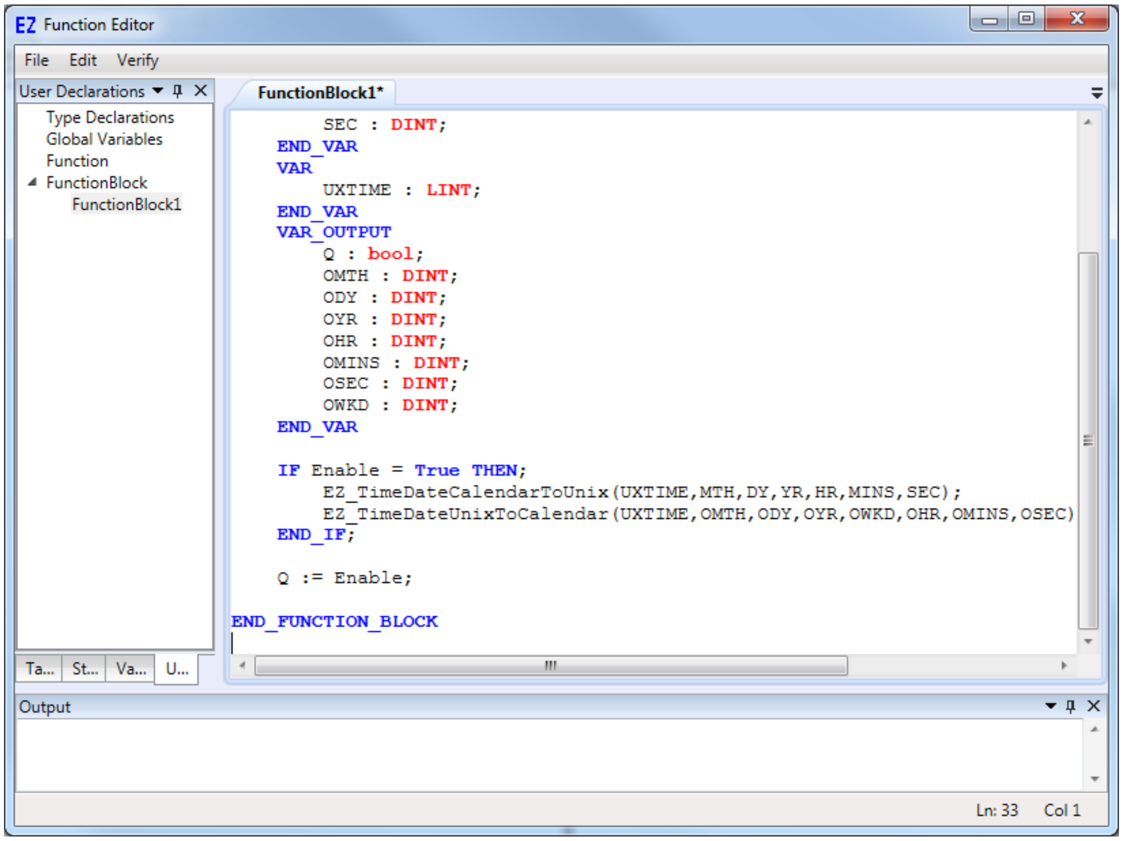


**Figure 26-24**

8. The new function has the basics required for a new function and is named *FunctionBlockx* by default. Select all (highlight) the text in the window. Verify all the text is selected (including text that is below the current viewable window without scrolling).

9. Use CTRL-V or the menu and select ***Edit...Paste***. The function block source will be over-written by the contents in the Windows scratchpad. Refer to Figure 26-25.



```
        SEC : DINT;
    END_VAR
    VAR
        UXTIME : LINT;
    END_VAR
    VAR_OUTPUT
        Q : bool;
        OMTH : DINT;
        ODY : DINT;
        OYR : DINT;
        OHR : DINT;
        OMINS : DINT;
        OSEC : DINT;
        OWKD : DINT;
    END_VAR

    IF Enable = True THEN;
        EZ_TimeDateCalendarToUnix(UXTIME,MTH,DY,YR,HR,MINS,SEC);
        EZ_TimeDateUnixToCalendar(UXTIME,OMTH,ODY,OYR,OWKD,OHR,OMINS,OSEC)
    END_IF;

    Q := Enable;

END_FUNCTION_BLOCK
```

**Figure 26-25**

10. Note the function code (text) has been pasted, but the function block name in the left pane is still FunctionBlockx. The name will not update in this pane until the code has been verified. Click **Verify** from the top menu of the editor. The code should verify and be identical if the entire function block was copied / pasted correctly. The name in the left pane is now the same as the source program would list.

11. Use the ***File..Save*** to save the function block. Close the ST Function Editor. The function / function block can now be inserted into the ladder diagram using the ST_FUNC or ST_FUNC_BLK functions listed in the Insert Function drop-down menu.

## Exporting / Importing ST Functions & Function Blocks

To use EZ LADDER Toolkit Structured Text functions and function blocks in multiple EZ LADDER programs, the structured text functions must be copied from the source ladder diagram to the target ladder diagram or exported from the source program and imported into the target ladder diagram. Functions and function blocks created in EZ LADDER Toolkit are specific to the ladder diagram in which they are created.

Copying / Pasting the ladder diagram inserted function block (ladder diagram symbol of a structured text function block) will not properly copy the structured text function block correctly. All copy / paste actions must be completed as structured text source using the ST Editor. Failure to use the correct copy and paste method will result in Compile Errors.

## Exporting ST Functions / Function Blocks

To Export a structured text function or function block:

1. Open the source ladder diagram program in EZ LADDER Toolkit.

2. Click the **EDIT ST FUNCTIONS** button (located next to the Insert Function drop-down menu on the tool bar) to open the Structured Text Editor.

3. Click the User Defined Functions tab (see Figure 26-26) to view the list of user defined functions and function blocks. Select the function / function block to export. You may need to expand the Function or FunctionBlock heading.

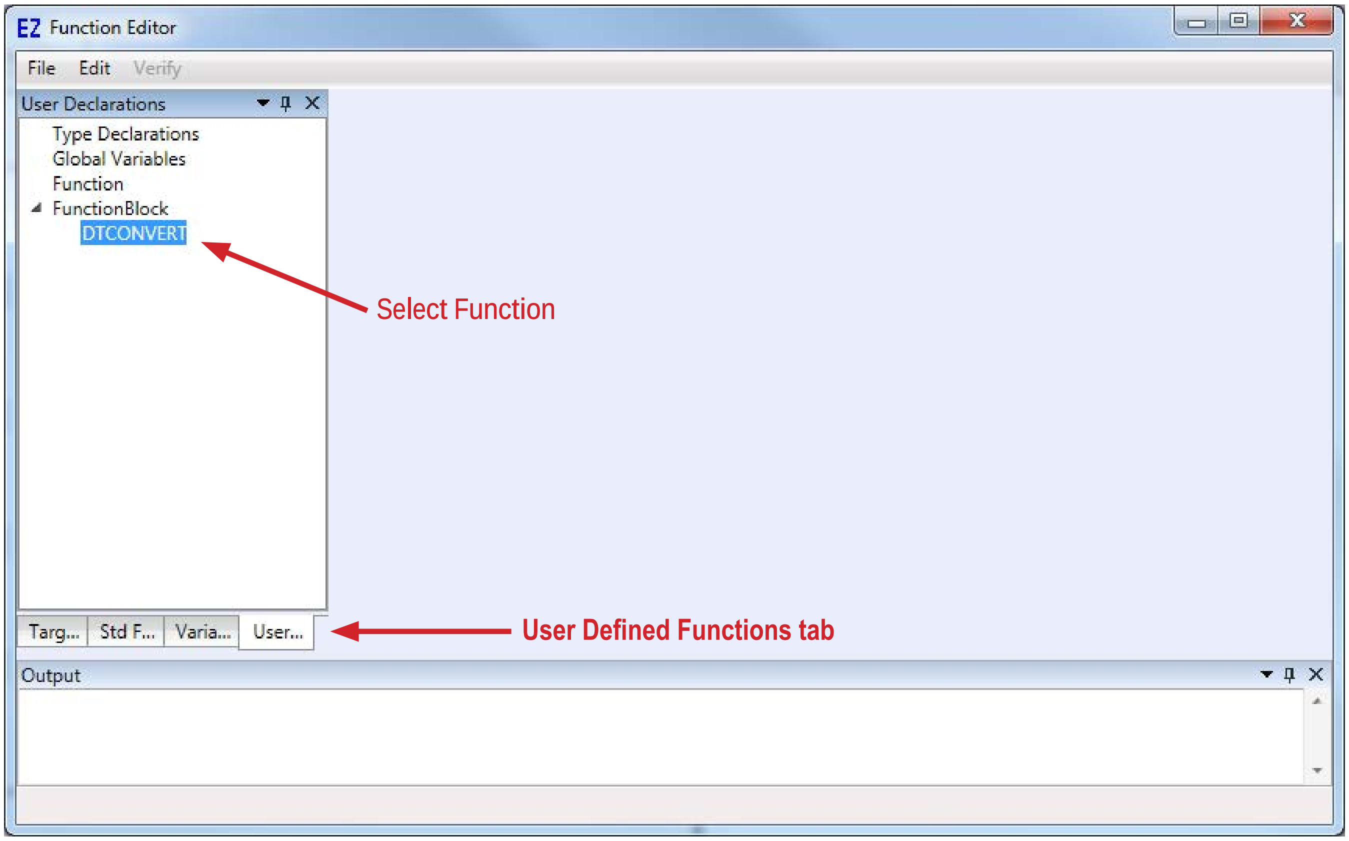


**Figure 26-26**

4. Double-click the selected function / function block to open the function / function block source code. Use the ***File...Export*** menu to export the currently opened function/function block. The function / function block will be stored as a text (.txt) file. Using the Save As window, locate the location to save the file and enter a name of the file. Click the **SAVE** button. Refer to Figure 26-27. The structured text function / function block source code is now saved as a text (.txt) file.

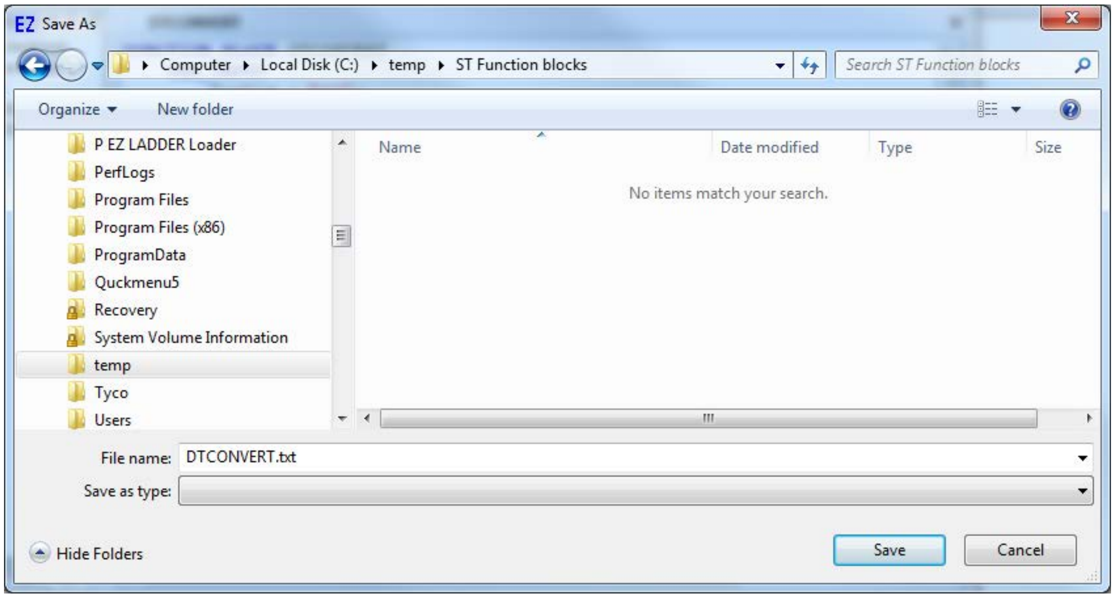


**Figure 26-27**

## Importing ST Functions / Function Blocks

To Import a structured text function or function block:

1. Open the target ladder diagram program in EZ LADDER Toolkit.

2. Click the **EDIT ST FUNCTIONS** button (located next to the Insert Function drop-down menu on the tool bar) to open the Structured Text Editor.

3. Click the User Defined Functions tab (see Figure 26-28) to view the list of user defined functions and function blocks. Select the type to import (function / function block). Right click and click **Add**. A new function block will be created with the default name *FunctionBlockx*.

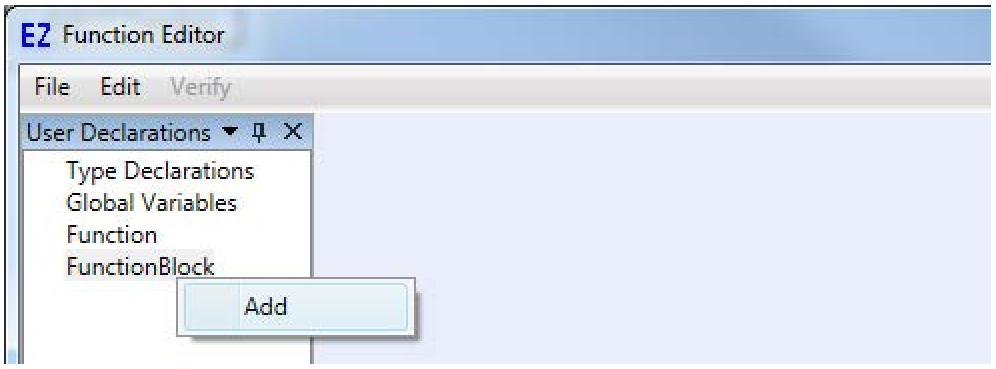


**Figure 26-28**

4. The new function has the basics required for a new function and is named *FunctionBlockx* by default. Use the ***File...Import*** menu to select and import a saved function / functionblock. Using the *Open* window, select a saved structured text function or function block that has been exported (text file, .txt) and click the **OPEN** button.

5. The saved function / function block source code will now overwrite the contents of the FunctionBlockx code window. Note the function code (text) has been updated, but the function block name in the left pane is still FunctionBlockx. The name will not update in this pane until the code has been verified. Click **Verify** from the top menu of the editor. The code should verify and be identical if the entire function block was exported / imported correctly. The name in the left pane is now the same as the function / function block code window name.

11. Use the ***File..Save*** to save the function block. Close the ST Function Editor. The function / function block can now be insterted into the ladder diagram using the ST_FUNC or ST_FUNC_BLK functions listed in the Insert Function drop-down menu.

## Viewing Structured Text Variables in Run Mode

The variable types supported in the ladder diagram (can be seen as variables in the actual ladder diagram) are limited to DINT, BOOL and REAL; but there are many other types of variables supported in structured text as shown earlier that may need to be monitored when running and debugging programs. These variables (as well as the DINT, BOOL and REAL) may be viewed in the RUN mode (real time) using the **Watchlist** in EZ LADDER Toolkit.

The watch appears at the bottom of the screen when EZ LADDER Toolkit is in the RUN mode (See Figure 26-29).

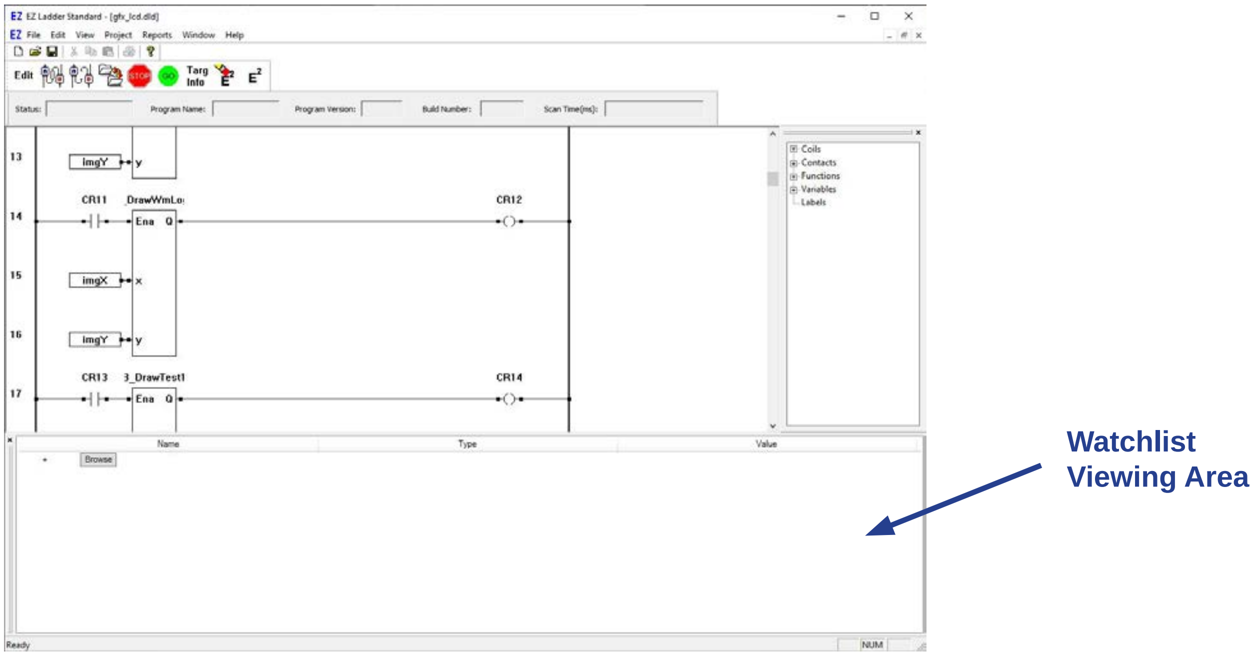


Figure 26-29

To view variables of a structured text function block placed in the ladder diagram, hover over the ST function block and **right-click**. A small optional pop-up will open with ***Add to Watchlist.*** Click on **Add to Watchlist*.*** See Figure 26-30. The function block will now be added to the Watchlist window.

For the Watchlist (and adding to to work correctly), the program in EZ LADDER Toolkit must be compiled and running on the actual hardware trget (RUN Mode).

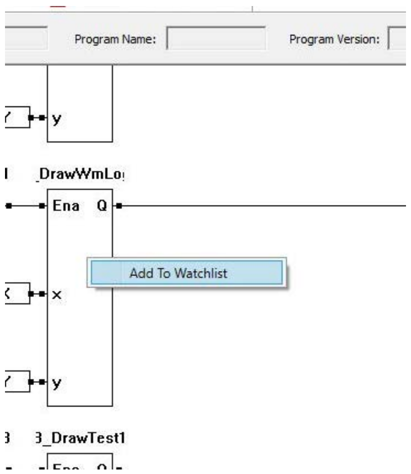


**Figure 26-30**

Clicking the little expand arrow next to the function block name in the Watchlist expands the function block and shows the variables, types and their values.

If the Type shows as invalid, verify the program has been compiled and downloaded to the target. All related files for the watchlist are created when the COMPILE process is done.

Variables for function blocks are viewed as shown above, but ST Global variables must be viewed differently. The **BROWSE** button allows for some direct viewing of variables in the ladder and GLOBAL variables in structured text.

# CHAPTER 27

## Cellular Connectivity

This chapter provides information on implementing and using hardware target supported Cellular data modems and connectivity.

## Chapter Contents

## Cellular Data Modem (CDM) Overview

P-Series PLC on a Chip based targets (model dependent) support a cellular data modem and cellular data connectivity. The cellular data modem and connectivity are design to interface directly to VersaCloud M2M+IoT Solutions or to Divelbiss customized data solutions.

Products that support cellular data modem (CDM) are shipped with the data modem pre-installed at the unit or expansion board level. These products may support multiple cellular data modem types and networks, but are all factory installed only. The typical installation CDM is a 4G cellular data modem.

## CDM Installation - Project Settings

P-Series PLC on a Chip based targets (model dependent) support a cellular data modem and cellular data connectivity. The cellular data modem and connectivity are design to interface directly to VersaCloud M2M+IoT solutions or to Divelbiss customized data solutions.

Before the CDM may be used in the ladder diagram / structured text, it must be installed in the ladder diagram Project Settings.

The following is the software requirements for installing CDM support for the target. Refer to the product's User Manual for details on installation of cellular data model (CDM) hardware and target Project Settings. For standard products, the cellular data model software support may automatically installed when the target is selected (or the target's cellular expansion board hardware option is selected in the Project Settings; therefore, no additional software configuration may be required.

PLC on a Chip integrated circuits and modules require additional setup before the cellular data modem (CDM) option may be used. Only cellular data modems provided by Divelbiss Corporation are supported in EZ LADDER Toolkit. Contact Divelbiss for purchasing cellular data modems, requirements for interfacing to PLC on a Chip and EZ LADDER target Project Settings configurations.

## Cellular Data Modem (CDM) Activation

The cellular data modem (CDM) supports multiple technologies and networks and requires either a Divelbiss Telematics contract which includes cellular data or you must provide your own SIM card and cellular data plan.

Technical Support for non-Divelbiss telematics (your own SIM card and plan) is a billable item is not covered under standard support. Divelbiss telematics plans include cellular technical support.

All products that support the Divelbiss cellular data modem (CDM) feature require an active SIM card to the cellular (carrier) network before they can be used. Current CDM products use 4G data modems and an activated SIM card for the carrier must be installed in the CDM. The SIM card / telematics plan may be purchased from Divelbiss or self provided. Technical support for cellular connection / modems is only provided at no charge for Divelbiss telematics and supplied SIM card plans.

**As each cellular data modem, technologies and coverage areas may have different activation needs, refer to additional documentation provided with the cellular data modem (CDM) or contact Divelbiss Corporation for activation steps.**

## Controlling the Cellular Data Modem

The cellular data modem is access and controlled as an interface using structured text and target specific structured text functions.

These functions include:

| | | |
|---|---|---|
| **EZ_Cell_ApplyPower** | **EZ_Cell_Connect** | **EZ_Cell_GetICCID** |
| **EZ_Cell_GetIIMEI** | **EZ_Cell_GetIPV4Addr** | **EZ_Cell_GetModelName** |
| **EZ_Cell_GetRegistration** | **EZ_Cell_GetSignalStrength** | **EZ_Cell_GetState** |

Each function is described and covered in detail in **Appendix B - Target Specific ST Function Reference**.

### General CDM Control Flow Chart

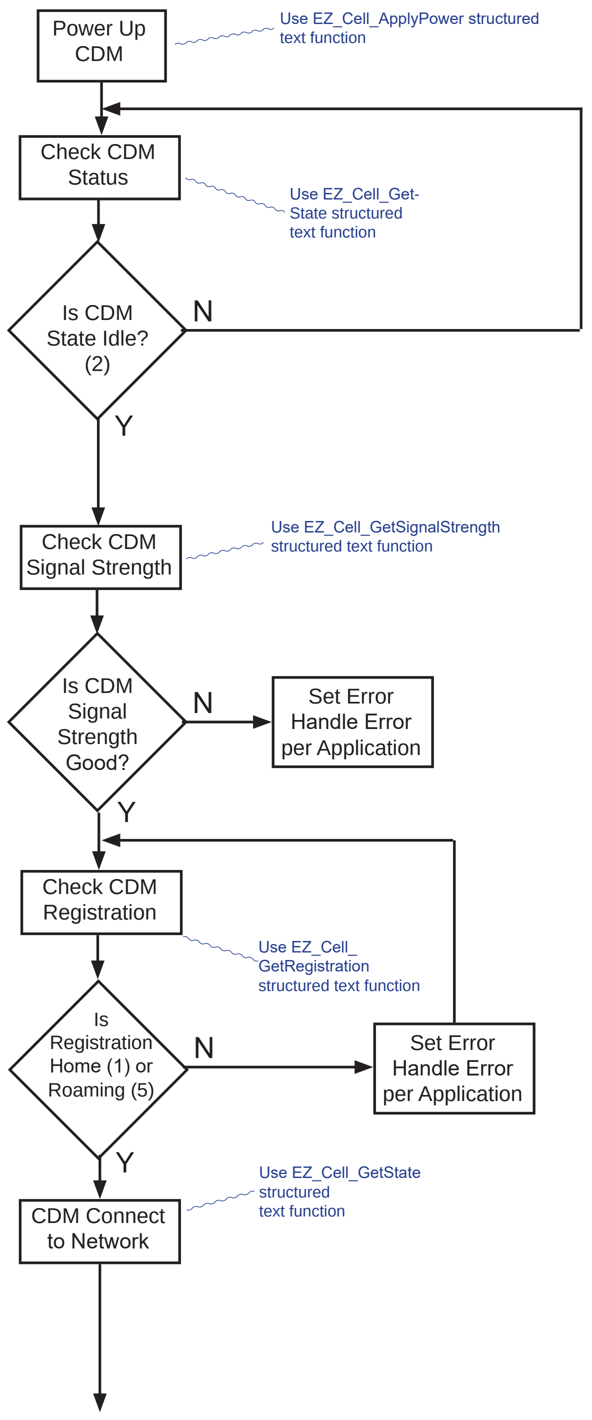
The flow chart provided is a general approach to the steps required to access and use the cellular data modem for communications. It is provided as a generic example and application requirements may dictate other or additional steps. CDM represents the cellular data modem in the flow chart.

The Flow chart shown is for general reference only. Care must be taken to monitor the state of the cellular data modem, the signal strength and other parameters in the application with enough frequency to detect connection issues.

Additionally, the flow chart shows only very basic error detection. Additonal error detection within the cellular modem control and the interaction to the MQTT function blocks in the ladder diagram is required as well as properly handling all shown and unknown potential errors according to the needs and requirements of the end application.

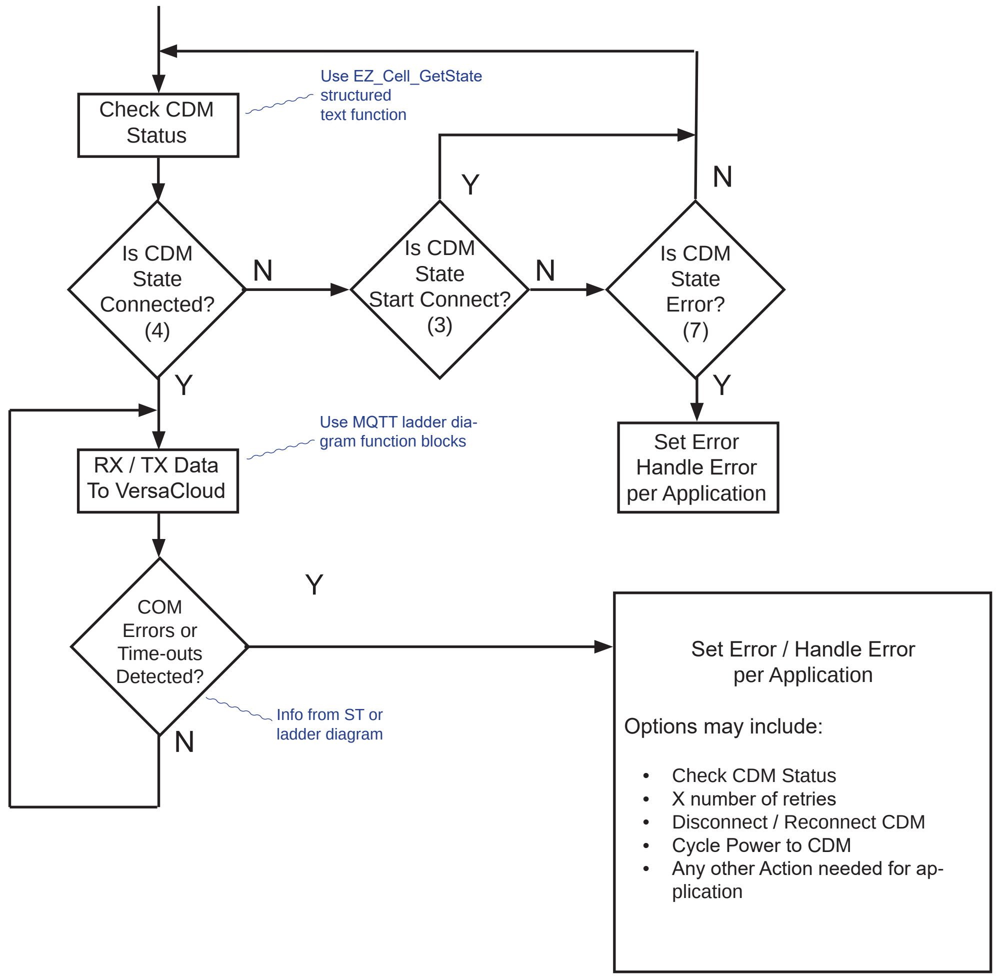
It is the responsibility of the application developer to properly control the cellular data modem, identify and handle all errors that may be encountered. Failure to do so may result in loss of communications, failure to make a connection and other data loss.

Items such as signal strength is dependent upon available cellular network coverage and actual application requirements for acceptable levels.


Power Up CDM
Use EZ_Cell_ApplyPower structured text function
Check CDM Status
Use EZ_Cell_Get-State structured text function
Is CDM State Idle? (2)
N
Y
Check CDM Signal Strength
Use EZ_Cell_GetSignalStrength structured text function
Is CDM Signal Strength Good?
N
Set Error Handle Error per Application
Y
Check CDM Registration
Use EZ_Cell_ GetRegistration structured text function
Is Registration Home (1) or Roaming (5)
N
Set Error Handle Error per Application
Y
Use EZ_Cell_GetState structured text function
CDM Connect to Network

*To Next Page*

*From Previous Page*


Check CDM Status
Use EZ_Cell_GetState structured text function
Is CDM State Connected? (4)
N
Is CDM State Start Connect? (3)
Y
N
Is CDM State Error? (7)
N
Y
Y
Set Error Handle Error per Application
Use MQTT ladder diagram function blocks
RX / TX Data To VersaCloud
COM Errors or Time-outs Detected?
Y
N
Info from ST or ladder diagram
Set Error / Handle Error per Application
Options may include:
Check CDM Status
X number of retries
Disconnect / Reconnect CDM
Cycle Power to CDM
Any other Action needed for application

In following sections several target specific functions are referenced from a generic view only. For details of each specific function, refer to **Appendix B - Target Specific ST Function Reference**.

## Cellular Modem Power Control

The hardware target (controller, gateway  or device) on-board cellular data modem's power must be controlled. By default, the cellular data modem is turned off (no power). The power is controlled using the Structured Text target specific function **EZ_Cell_ApplyPower.**

Before the cellular data modem can be used in any capacity, it's power must be turned on.

At specific times or specific conditions require such as power conservation or to hard reset the cellualr data mode,  it may be advantageous to turn off the power to the cellular modem. When the cellular data modem's power should be turned off is entirely based on the needs of an application.

## Cellular Modem Signal Level

The cellular signal level may be accessed using the Structured Text target specific function **EZ_Cell_GetSignalStrength.**

This function is only available when the cellular data modem is in the IDLE state (powered up but not communicating). While a connection is in use, this function is not valid. See **Cellular Data Modem Status** in this chapter.

## Cellular Modem Activation / Deactivation

As discussed earlier in this chapter:

**As each cellular data modem, technolognies and coverage areas may have different activation needs, refer to additional documentation provided with the cellular data modem (CDM) or contact Divelbiss Corporation for activation steps.**

For 4G CDM (cellular data modems), the SIM card must be activated with the cellular carrier and installed into the cellular data modem. The modem should register on first connection to the network.

The cellular data modem will not function without an activated / registered SIM card installed.

The cellular data modem (CDM) supports multiple technologies and networks and requires either a Divelbiss Telematics contract which includes cellular data or you must provide your own SIM card and cellular data plan.

Technical Support for non-Divelbiss telematics (your own SIM card and plan) is a billable item is not covered under standard support.  Divelbiss telematics plans include cellular technical support.

## Connect to Cellular Network

The connection to the cellular network is handled using the Structured Text target specific function **EZ_Cell_Connect.** This function causes the cellular modem to make a connection or to dis-connect from the cellular networks. This function can only be called to make a connection when the cellular data modem is in the IDLE state. The function can be called to disconnect the cellular data modem at any time (or any state).

The cellular data modem status must be IDLE to make a connection. See **Cellular Data Modem Status** in this chapter.

The cellular data modem registration must be REGISTERED HOME or REGISTERED ROAMING to make a connection. See **Cellular Data Modem Registration** in this chapter.

## Cellular Data Modem Status

The current status of the cellular data modem can be accessed using the Structured Text target specific function **EZ_Cell_GetState.** This function queries the on-board cellular data modem and returns a status code based on the current modem state.

The return states are:

| | |
|---|---|
| 0 | Off |
| 1 | Powering Up |
| 2 | Idle |
| 3 | Starting Connection |
| 4 | Connected |
| 5 | Stopping Connection |
| 6 | Activating |
| 7 | Error |

## Cellular Data Modem Registration

The current status of the cellular data modem registration (availability of connecting to a cellular network) can be accessed using the Structured Text target specific function **EZ_Cell_GetRegistration.** This function queries the on-board cellular data modem and returns a status code based on the current registration state.

The return states are:

| | |
|---|---|
| 0 | Not Registered |
| 1 | Registered Home |
| 2 | Not Registered - Searching |
| 3 | Registration Denied |
| 4 | Unknown |
| 5 | Registered Roaming |

## Retrieving Modem / Connection Information

### Cellular Modem (Network) IP Address

The cellular data modem's IP address can be retrieved for informational purposes using the Structured Text target specific function **EZ_Cell_GetIPV4Addr.**

The **EZ_Cell_GetIPV4Addr** function (and obtaining the IP address) is only valid when the cellular data modem (CDM) is Connected to the cellular network. See **Cellular Data Modem Status** in this chapter.

### Cellular Modem IMEI

The cellular data modem's IMEI is set and cannot be changed, but it can be retrieved for informational purposes using the Structured Text target specific function **EZ_Cell_GetIMEI.**

### Cellular Modem Model Name

The cellular data modem's Model Name can be retrieved for informational purposes using the Structured Text target specific function **EZ_Cell_GetModelName.**

### Cellular Sim Card ID

The cellular data modem's installed SIM Card ID can be retrieved for informational purposes using the Structured Text target specific function **EZ_Cell_GetICCID.**

## The Cellular Data Modem and VersaCloud M2M+IoT

The cellular data modem must have an active SIM Card and controlled using structured text. As it is controlled using structured text, it may accessed for VersaCloud M2M+IoT communication as a device using the MQTT_Connect, MQTT_Publish and MQTT_Recieve functions (ladder diagram function) and other structured text commands. Refer to **Appendix A - Function Reference** as the VCLOUD function is covered in detail. Refer to **Chapter 26 - Structured Text** and **Appendix B - Target Specific ST Function Reference** for details on using structured text.

Using the MQTT_Connect, MQTT_Publish and MQTT_Recieve  functions, structured text and selecting the **cellular** device as the communication method allows for transmitting and receiving variable data to/from the VersaCloud M2M+IoT Solutions.

The cellular data modem must be controlled using Structured Text and target specific functions before the MQTT function blocks or othe structured text functions may be used to communicate. Failure to properly control the cellular data modem will result in communications failure or loss.

# CHAPTER 28

# VersaCloud M2M+IoT Communications

This chapter provides information on implementing and VersaCloud M2M+IoT communications between VersaCloud M2M+IoT supported hardware targets and the VersaCloud M2M+IoT cloud solutions. Legacy COAP communications (previously known as VersaCloud M2M information is now located in Chapter 30 - COAP Communications.

## Chapter Contents

# VersaCloud M2M+IoT Communications Overview

VersaCloud M2M+IoT from Divelbiss is an end to end, machine to machine, IoT device to cloud solution. VersaCloud M2M+IoT solutions cover each area needed for remote control and / or monitoring of machinery and equipment, regardless of where the equipment is located; from the factory floor to remote sites. VersaCloud M2M+IoT solutions include interface hardware (PLCs and Gateways), communications links (Ethernet, WI-FI and cellular data including cellular data coverage plans) and cloud portal solutions that utilize Microsoft Azure IoT Central, IoT Hub as well as other custom cloud portal solutions.

VersaCloud M2M+IoT Solutions use MQTT communcations between devices and cloud solutions and portals. The original VersaCloud M2M used COAP. COAP communications are still supported to custom cloud solutions. For COAP communications information, see the **Chapter 30 - COAP Communications** of this manual.

EZ LADDER programmed VersaCloud enabled products support the communications from the device to VersaCloud M2M+IoT solution. Communications to the cloud may be handled via Ethernet, Wi-Fi or Cellular. For each of these types of communications, the product (target) must support the communication method to be used.

VersaCloud M2M+IoT communications via Cellular Data Modem requires a cellular data plan (provided by you with SIM card or a Telematics contract and cellular plan provided by Divelbiss. Additional charges apply for Divelbiss cellular data plans and technical support for non-Divelbisss cellular data plans.

VersaCloud M2M+IoT cloud solutions by Microsoft Azure IoT Central require account registration. Charges may apply based on the number of devices connected to the IoT Central portal.

For each communication method, EZ LADDER (per ladder diagram project) requires additional configuration and setup for the communications method itself as some communication methods may be used outside of VersaCloud M2M+IoT Portal communications. Refer to Chapter 19 - Ethernet / Wi-Fi for configuring Wi-Fi or Chapter 27 - Cellular Connectivity for configuring the Cellular Data Modem option.

Ethernet and Wi-Fi may be used to communicate via Modbus TCP without VersaCloud M2M+IoT solutions (local communications).

Cellular data modem may only be used for communications to VersaCloud M2M+IoT cloud solutions or to Divelbiss developed custom cloud solutions.

Regardless of the communications method, the product (target) must support the communication method to be used.

In addition to the communications method configuration, the VersaCloud M2M+IoT communications must be configured and enabled in EZ LADDER Toolkit (per ladder diagram project) before it can be used.

MQTT (VersaCloud M2M+IoT) requires the SNTP feature to be installed, configured and operating before the ladder diagram will compile successfully. The SNTP is required to sync the real time clock to UTC time.

## Communications using Ladder Diagram

The MQTT_Connect, MQTT_Publish and MQTT_Recieve function blocks within the ladder diagram are simple to use for communicating to and from the VersaCloud M2M+IoT cloud portal and is ideally suited for transmitting and receiving variables quickly and easily. The VCLOUD function block supports two variable types: REAL, BOOLEAN and INTEGER. Structured text may also be used to communicate to the Versa-cloud M2M+IoT cloud portal and may use other variable types as well.

Other variable types (TIMER) must be converted to either INTEGER, BOOLEAN or REAL variable types before they can be sent / received between the product (ladder diagram) and the VersaCloud M2M+IoT using the MQTT_Connect, MQTT_Publish and MQTT_Recieve function blocks.

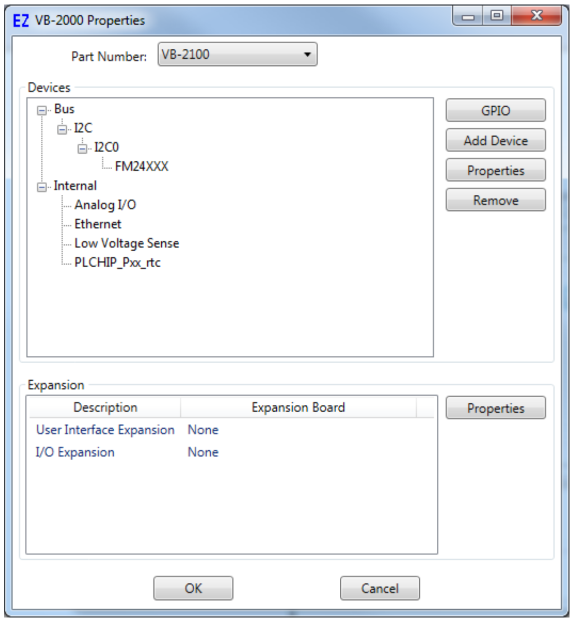
## Installing VersaCloud M2M+IoT (MQTT) in Project Settings

Before the MQTT_xxx function blocks and VersaCloud M2M+IoT communications may be used in the ladder diagram program, it must be installed in the Project Settings (for each ladder diagram project).

Verify the communication method is installed (Ethernet, Wi-Fi, Cellular) before proceeding. Refer to this manual's individual chapters for each of the methods for instructions for installation.

1. Using the EZ LADDER Toolkit menu, select **Project.....Settings**. The *Project Settings* dialog will open. The target should have already been pre-selected and configured and the communications method installed previously.

2. Click the **PROPERTIES** button. The *target's Properties* dialog will open. Refer to Figure 28-1.

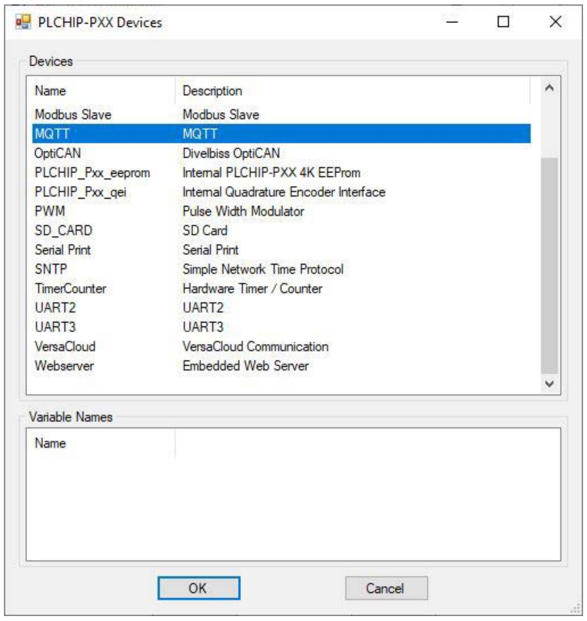


Part Number: VB-2100

Devices
- Bus
  - I2C
    - I2C0
      - FM24XXX
- Internal
  - Analog I/O
  - Ethernet
  - Low Voltage Sense
  - PLCHIP_Pxx_rtc

Expansion

| Description | Expansion Board |
|---|---|
| User Interface Expansion | None |
| I/O Expansion | None |



Devices

| Name | Description |
|---|---|
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SNTP | Simple Network Time Protocol |
| TimerCounter | Hardware Timer / Counter |
| UART2 | UART2 |
| UART3 | UART3 |
| VersaCloud | VersaCloud Communication |
| Webserver | Embedded Web Server |

Variable Names

Name

**Figure 28-1** **Figure 28-2**

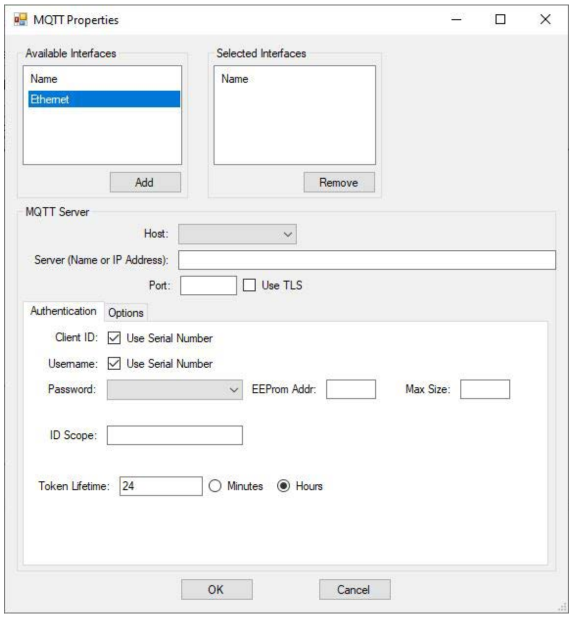
3. Click the **ADD DEVICE** button. The *PLCHIP-PXX Devices* dialog will open.

4. Find and select the **MQTT** option (highlight). Refer to Figure 28-2.

5. Click **OK.** The *MQTT Properties* dialog will open. This dialog is used to configure the MQTT (Versacloud

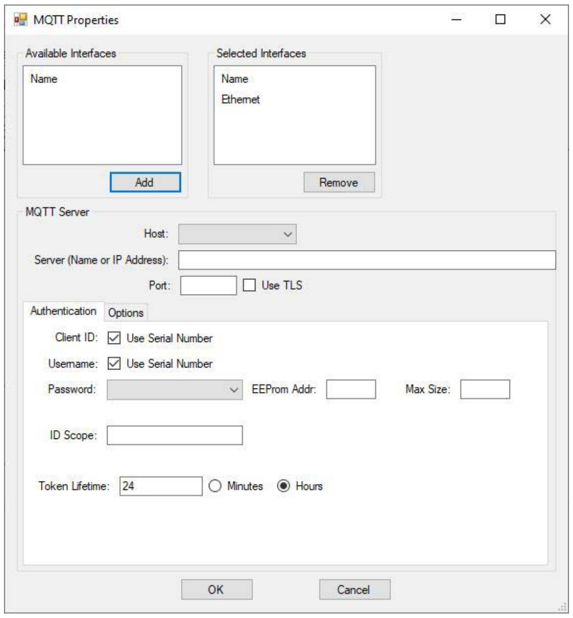
M2M+IoT) communications settings (what the MQTT function blocks use). Refer to Figure 28-3. The first section of this dialog is divided into two panes: ***Selected Interfaces*** and ***Available Interfaces***.

6. Select (highlight) the communications method in the Available Interfaces pane and click the **ADD** button. The selected method will now be moved from the ***Available Interfaces*** pane to the ***Selected Interfaces*** pane. Refer to Figure 28-4.



Available Interfaces: Name, Ethernet
Selected Interfaces: Name
MQTT Server
Host:
Server (Name or IP Address):
Port: ☐ Use TLS
Authentication | Options
Client ID: ☑ Use Serial Number
Username: ☑ Use Serial Number
Password: EEProm Addr: Max Size:
ID Scope:
Token Lifetime: 24 ○ Minutes ● Hours

**Figure 28-3**



Available Interfaces: Name
Selected Interfaces: Name, Ethernet
MQTT Server
Host:
Server (Name or IP Address):
Port: ☐ Use TLS
Authentication | Options
Client ID: ☑ Use Serial Number
Username: ☑ Use Serial Number
Password: EEProm Addr: Max Size:
ID Scope:
Token Lifetime: 24 ○ Minutes ● Hours

**Figure 28-4**

7. Set the **Host** of the Versacloud M2M+IoT solution using the drop down. The common would Azure IoT Central.

8. Depending upon the **Host** selected, some fields will change or will require additional configuration information.

## Azure Iot Central

a. The Server name is pre-set for Azure IoT Central by Divelbiss. The server port number is set automatically and uses TLS

b. Server Certificate tab. Refer to Figure 28-5A.

1. The Baltimer Cyber Trust Root certificate is selected by default.

2. If it is un-checked, a different certificate may be entered in the provided field.

3. The Disable Server Certificate check box allows for disabling part of the certificate security check. This option save some ladder diagram RAM space.

c. The Authentication tab requires additional configuration. Refer to Figure 28-5B.

1. The Device ID is by default configured to Use Serial Number. This can be changed if required using the check box and adding an EEPROM location and size where to find the ID on the device.

2. The Password is set to Symmetric key and you must provide an EEPROM starting address

to store this key on the device (EEProm Addr:) The maximum size is 46 bytes)

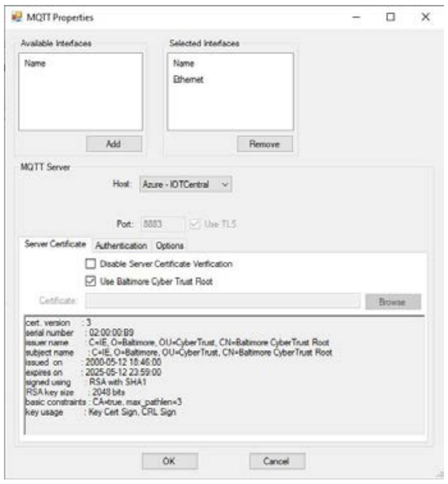
3. ID Scope: This field is provided by the Azure IoT Central portal. It will be stored in EEPROM also.

You can enter any value temporarily, but it must have the actual ID Scope from Azure IoT Central before any communcations can be established.

4. The Token Lifetime is set by default to 24 hours. This value can be changed any desired hours or minutes. This is the time limit for an active MQTT connection to time-out and automatically close the connection (required by Azure IoT Central). The connection will close and then re-connect automatically. This only applies if the MQTT connection has been connected for the entire duration with no disconnects.

d. Options Tab. Refer to Figure 28-5C.

1. Allows for adjusting the Number of buffered MQTT Messages. The default is set to 3.



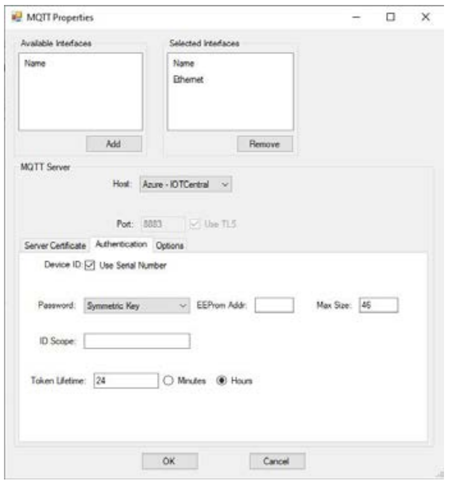
Selected Interfaces: Name: Ethernet

MQTT Server — Host: Azure - IOTCentral; Port: 8883; ☑ Use TLS

☐ Disable Server Certificate Verfication
☑ Use Baltimore Cyber Trust Root
Certificate:

```
cert. version     : 3
serial number     : 02:00:00:B9
issuer name       : C=IE, O=Baltimore, OU=CyberTrust, CN=Baltimore CyberTrust Root
subject name      : C=IE, O=Baltimore, OU=CyberTrust, CN=Baltimore CyberTrust Root
issued on         : 2000-05-12 18:46:00
expires on        : 2025-05-12 23:59:00
signed using      : RSA with SHA1
RSA key size      : 2048 bits
basic constraints : CA=true, max_pathlen=3
key usage         : Key Cert Sign, CRL Sign
```

Figure 28-5A

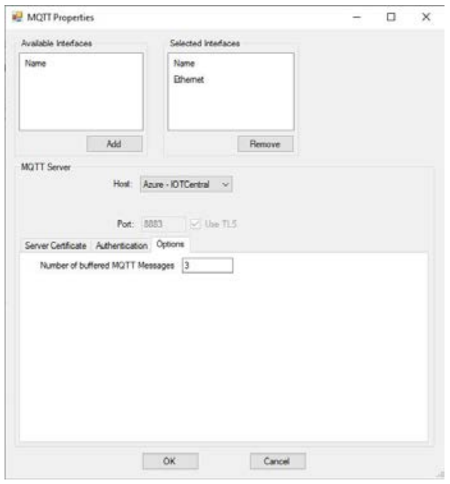


Selected Interfaces: Name: Ethernet

MQTT Server — Host: Azure - IOTCentral; Port: 8883; ☑ Use TLS

Device ID: ☑ Use Serial Number
Password: Symmetric Key   EEProm Addr:   Max Size: 46
ID Scope:
Token Lifetime: 24   ○ Minutes   ● Hours

Figure 28-5B



Selected Interfaces: Name: Ethernet

MQTT Server — Host: Azure - IOTCentral; Port: 8883; ☑ Use TLS

Number of buffered MQTT Messages: 3

Figure 28-5C

## Azure Iot Hub

a. The Server (Name or IP address) must be entered for the Azure IoT Hub. This address / name is custom per each application. 1. The server port number is set automatically and uses TLS

b. Server Certificate tab. Refer to Figure 28-6A.

1. The Baltimer Cyber Trust Root certificate is selected by default.

2. If it is un-checked, a different certificate may be entered in the provided field.

3. The Disable Server Certificate check box allows for disabling part of the certificate security check. This option save some ladder diagram RAM space.

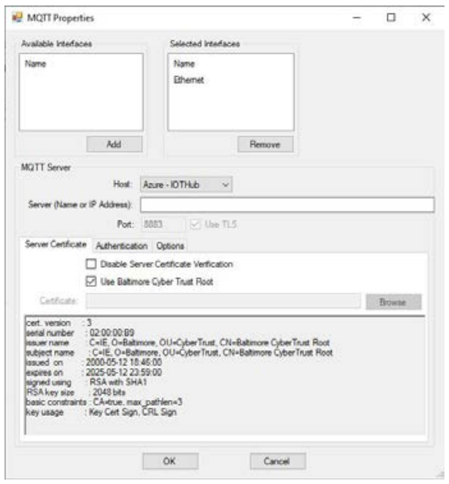
c. The Authentication tab requires additional configuration. Refer to Figure 28-6B.

1. The Device ID is by default configured to Use Serial Number. This can be changed if required using the check box and adding an EEPROM location and size where to find the ID on the device.
2. The Password is set to Symmetric key and you must provide an EEPROM starting address

to store this key on the device (EEProm Addr:) The maximum size is 46 bytes)

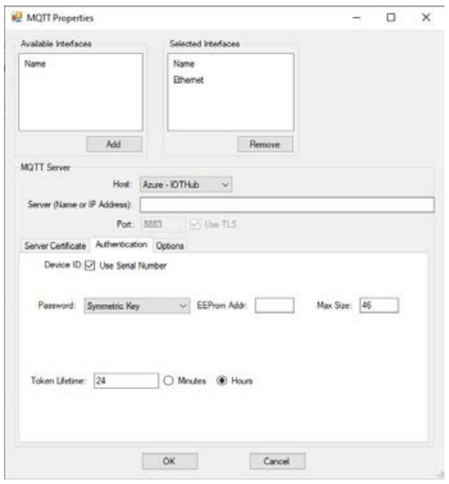
3. The Token Lifetime is set by default to 24 hours. This value can be changed any desired hours or minutes. This is the time limit for an active MQTT connection to time-out and automatically close the connection (required by Azure IoT Central). The connection will close and then re-connect automatically. This only applies if the MQTT connection has been connected for the entire duration with no disconnects.

d. Options Tab. Refer to Figure 28-6C.
1. Allows for adjusting the Number of buffered MQTT Messages. The default is set to 3.

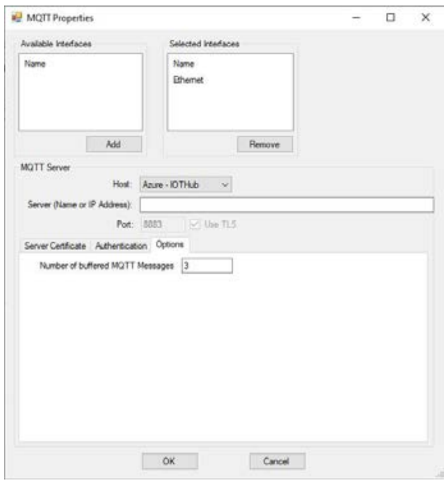


Host: Azure - IOTHub
Server (Name or IP Address):
Port: 8883 ☑ Use TLS
☐ Disable Server Certificate Verification
☑ Use Baltimore Cyber Trust Root
Certificate:

```
cert. version    : 3
serial number    : 02:00:00:B9
issuer name      : C=IE, O=Baltimore, OU=CyberTrust, CN=Baltimore CyberTrust Root
subject name     : C=IE, O=Baltimore, OU=CyberTrust, CN=Baltimore CyberTrust Root
issued on        : 2000-05-12 18:46:00
expires on       : 2025-05-12 23:59:00
signed using     : RSA with SHA1
RSA key size     : 2048 bits
basic constraints : CA=true, max_pathlen=3
key usage        : Key Cert Sign, CRL Sign
```



Host: Azure - IOTHub
Server (Name or IP Address):
Port: 8883 ☑ Use TLS
Device ID: ☑ Use Serial Number
Password: Symmetric Key EEProm Addr: Max Size: 46
Token Lifetime: 24 ○ Minutes ● Hours



Host: Azure - IOTHub
Server (Name or IP Address):
Port: 8883 ☑ Use TLS
Number of buffered MQTT Messages 3

**Figure 28-6A** **Figure 28-6B** **Figure 28-6C**

## Exosite - Murano

a. The Server (Name or IP address) must be entered for the Exosite-Murano portal. This address / name is custom per each application. The server port number is set automatically and uses TLS

b. Server Certificate tab. Refer to Figure 28-7A.
1. The Use Exosite Root CA is selected by default.

2. The option to select Use DigiCert Global Root CA is available.

3. Unchecking both boxes, allows for entering / selecting a different security certificate to use in the provided field.

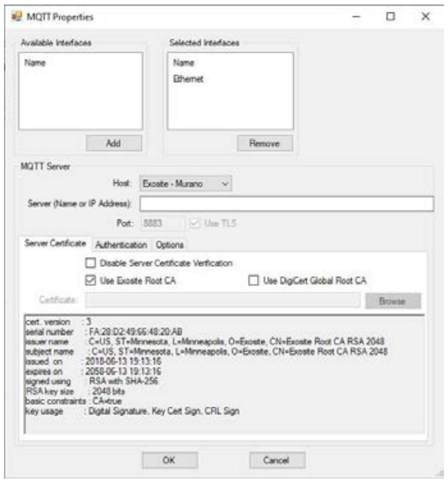
3. The Disable Server Certificate check box allows for disabling part of the certificate security check. This option save some ladder diagram RAM space.

c. The Authentication tab requires additional configuration. Refer to Figure 28-7B.
1. The Client ID is by default configured to Use Serial Number. This can be changed if required using the check box and adding an EEPROM location and size where to find the ID on the device.

2. The Username is by default configured to Use Serial Number. This can be changed if required using the check box and adding an EEPROM location and size where to find the Username on the device.

3. Password is set to Token. This cannot be changed.

d. Options Tab. Refer to Figure 28-7C.
1. Allows for adjusting the Number of buffered MQTT Messages. The default is set to 3.


MQTT Properties
Available Interfaces: Name
Selected Interfaces: Name, Ethernet
MQTT Server
Host: Exosite - Murano
Server (Name or IP Address):
Port: 8883 ☑ Use TLS
Server Certificate | Authentication | Options
☐ Disable Server Certificate Verification
☑ Use Exosite Root CA ☐ Use DigiCert Global Root CA
Certificate:

```
cert. version     : 3
serial number     : FA:28:D2:49:66:48:20:AB
issuer name       : C=US, ST=Minnesota, L=Minneapolis, O=Exosite, CN=Exosite Root CA RSA 2048
subject name      : C=US, ST=Minnesota, L=Minneapolis, O=Exosite, CN=Exosite Root CA RSA 2048
issued  on        : 2018-06-13 19:13:16
expires on        : 2058-06-13 19:13:16
signed using      : RSA with SHA-256
RSA key size      : 2048 bits
basic constraints : CA=true
key usage         : Digital Signature, Key Cert Sign, CRL Sign
```

Figure 28-7A

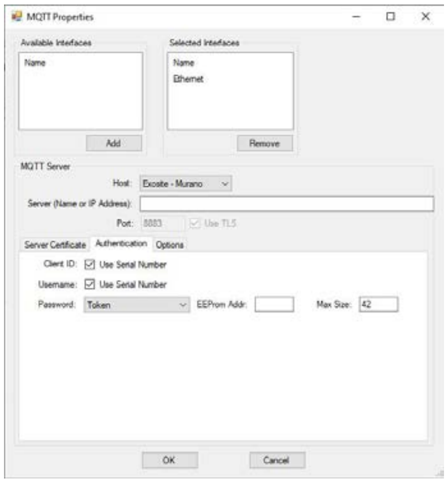

MQTT Properties
Available Interfaces: Name
Selected Interfaces: Name, Ethernet
MQTT Server
Host: Exosite - Murano
Server (Name or IP Address):
Port: 8883 ☑ Use TLS
Server Certificate | Authentication | Options
Client ID: ☑ Use Serial Number
Username: ☑ Use Serial Number
Password: Token EEProm Addr: Max Size: 42

Figure 28-7B

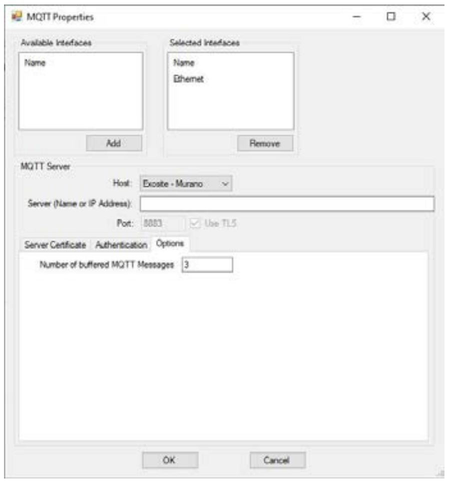

MQTT Properties
Available Interfaces: Name
Selected Interfaces: Name, Ethernet
MQTT Server
Host: Exosite - Murano
Server (Name or IP Address):
Port: 8883 ☑ Use TLS
Server Certificate | Authentication | Options
Number of buffered MQTT Messages: 3

Figure 28-7C

9. Click the **OK** button. The *MQTT Properties* dialog will close returning to the *target's Properties* dialog.

10. Click the **OK** button. The T*arget's Properties* dialog will close returning to the *Project Settings* dialog.

11. Click the **OK** button. The *Project Settings* dialog will close. The MQTT Versacloud M2M+IoT communications feature is now installed in the ladder diagram project.

Be sure to save the ladder diagram project using the ***Save*** or ***Save As*** from the EZ LADDER Toolkit menu.

**Custom**

This setting allows for a custom MQTT host. The settings to be entered will depend on the host used. Consult the Host documentation for information on possible configuration settings.

The Use TLS checkbox is optional when using custom. When the checkbox is enabled, MQTT_TLS memory will be reserved and when disabled, the memory will not be reserved (more memory available).

## MQTT Function Blocks Overview

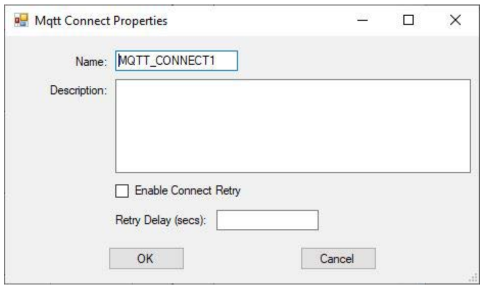
The MQTT function blocks are inserted into the ladder diagram by selecting it from the Insert Function drop-down menu next to the EDIT ST FUNCTIONS button on the tool bar. There are 2 MQTT function blocks used for communications to VersaCloud M2M+IoT solutions: MQTT_CONNECT and MQTT_PUBLISH.

Locate the position and click to insert the function block (as any other function block). The *MQTT Function block's Properties* dialog will open. See each function block for more details.

## MQTT _CONNECT Function Block

The MQTT_CONNECT function block opens the connection to the VersaCloud M2M+IoT cloud server (IoT Central, IoT Hub or Exosite Murano) selected in the Project Settings.

Select the MQTT_CONNECT function block from the list and insert it into the ladder diagram. The *Mqtt Connect Properties* dialog will open. When placing the MQTT_CONNECT function block, an optional checkbox is provided for it to automatically try reconnecting should the connection be lost as well as a retry delay (in seconds) between retries. Refer to Figure 28-8.


Mqtt Connect Properties

Name: MQTT_CONNECT1

Description:

☐ Enable Connect Retry

Retry Delay (secs):

**Figure 28-8**

The function block operates like all other function blocks with an **EN**able input to trigger. When the ENable input is true, the MQTT_CONNECT block will attempt to connect (and stay connected) to the VersaCloud M2M+IoT cloud solution selected in the Project Settings. When the **EN**able input goes false, the MQTT_CONNECT block will disconnect from the Versacloud M2M+Iot cloud solution selected in the Project Settings.

The **Q** output is true when the EN input is true.

There are three other outputs from the MQTT_CONNECT function block for identifying the current status of the connection. These outputs require INTEGER variables be connected to them. The variables will hold the values output by the function block. Refer to Figure 28-9.

**ST**: MQTT State of operation. The values listed identify the current state.

| | |
|---|---|
| 1 | Connecting |
| 2 | Provisioning |
| 3 | Provision Sent |
| 100 | Connect Sent |
| 101 | Connected |
| 200 | Connect Error |
| 300 | TLS Handshaking Failed |
| 301 | Authentication Read Error |
| 302 | Authentication Needs User Password |
| 303 | Authentication Invalid User Password |
| 304 | Authentication Not Authorized (Bad Username or Password) |
| 305 | Connection Refused (Invalid Protocol Version) |
| 306 | Connection Refused (Client ID Rejected) |
| 307 | Connection Refused (Server Unavailable) |

**ER**: MQTT Network Errors. The values listed identify the error.

| | |
|---|---|
| 0 | No Error |
| -1 | Out of Memory Error |
| -2 | Buffer Error |
| -3 | Timeout Error |
| -4 | Reverved (Should not see) |
| -5 | Operation Currently in Progress |
| -6 | Illegal Value |
| -7 | Reverved (Should not see) |
| -8 | Address in Use |
| -9 | Already Connecting |
| -10 | Connection Already Established |
| -11 | Not Connected |
| -12 | Network Interface Error (Cellular, Ethernet, etc) |
| -13 | Connection Aborted |
| -14 | Connection Reset |
| -15 | Connection Closed |
| -16 | Illegal Argument |

Additional errors below -8192 are specific to the SSL layter. For example, -9984 is a Certificate failed verification (CRL, CA or signature check failed. This can be caused by the certificate not matching, if there is not enough free memory (RAM) in the controller or if the certificate date is not vaild.

**RT**: The number of retries that have occurred. This number is cumulative (until target reset or power cycle). The larger the number, the more times the block has tried to connect to the selected cloud solution (some may be successful, others may not).

If the number of retries is increasing, identifying a problem connecting (once a know good program and connection have been established), it may be required to reset the controller or target using structured text, forcing a reboot to see if the problem is corrected.

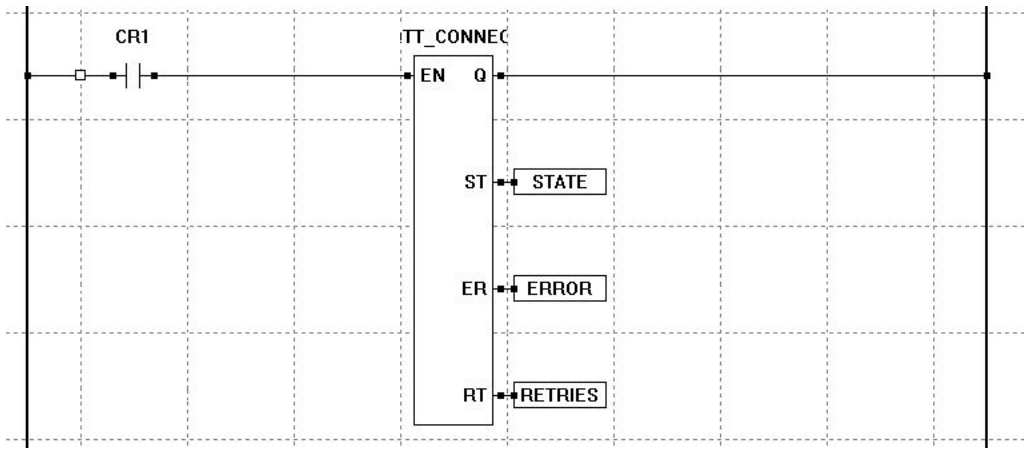


**Figure 28-9**

## MQTT _PUBLISH Function Block

The MQTT_PUBLISH function block sends data the VersaCloud M2M+IoT cloud server (IoT Central, IoT Hub or Exosite Murano) selected in the Project Settings.

Before the MQTT_PUBLISH block can send data, a connection must be established to the Versa Cloud M2M+IoT cloud solution using the MQTT_CONNECT block.

Select the MQTT_PUBLISH function block from the list and insert it into the ladder diagram. When placing the MQTT_PUBLISH function block, the *Mqtt Publish Properties* dialog will open. Refer to Figure 28-10.

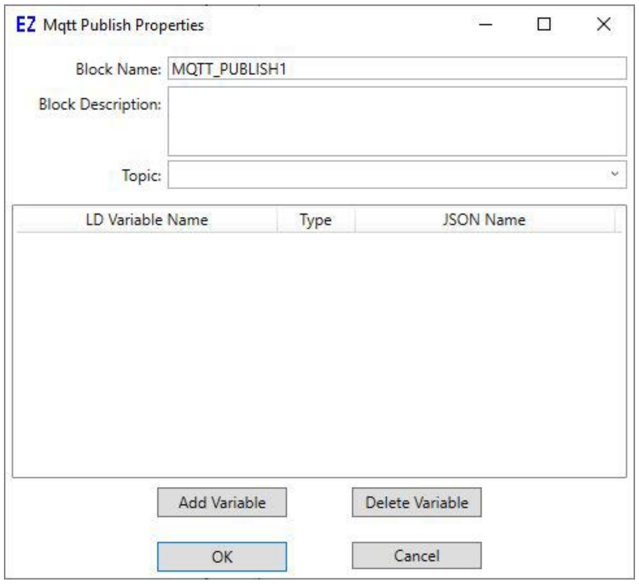
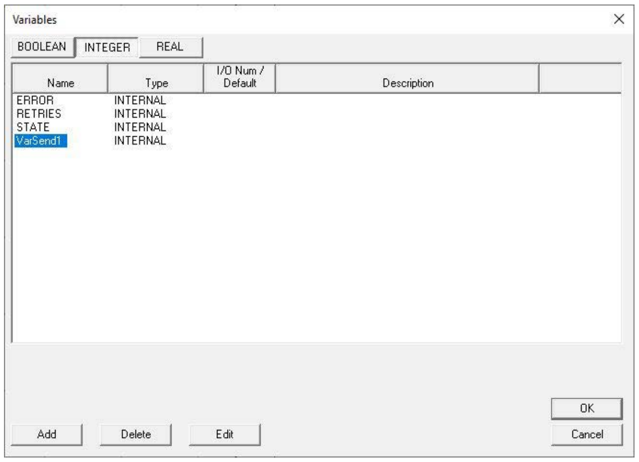


Figure 28-10



| Name | Type | I/O Num / Default | Description |
|---|---|---|---|
| ERROR | INTERNAL | | |
| RETRIES | INTERNAL | | |
| STATE | INTERNAL | | |
| VarSend1 | INTERNAL | | |

Figure 28-11

Thee dialog (shown in Figure 28-10), is where variables from the ladder diagram are selected to be sent to the Verscloud M2M+IoT cloud solution (chosen in the Project Settings).

The variables selected / added to the MQTT_PUBLISH block should already exist, or they may be added in the variable add dialog when selecting the variable (See variables in the **Chapter 5 - Ladder Diagram Projects**).

Variable types used with the MQTT_PUBLISH block include BOOLEAN, REAL and INTEGER.

Click the **ADD VARIABLE** button to add a variable to the MQTT_PUBLISH block control. Select (or add a new variable) to use and click **OK**. See Figure 28-11. The variable will now be populated in the Mqtt Publish Properies dialog.

The JSON Name will now by default be populated with the name of the variable. This JSON Name can be edited / changed, but this JSON Name entered is the name used in the cloud portal side to map the data. See Figure 28-12.

The JSON Name entered in this field must match the name of the variable on the cloud portal for data to be sent to and received on the cloud portal side. If these names do not match, data will not be received on the cloud portal.

In the **Topic** drop down menu, select: ***devices/{DeviceID}/messages/events/***

Click **OK** when all the variables have been added to send to the VersaCloud M2M+IoT cloud solution (portal).

Variables can be deleted using the **DELETE** button in the Mqtt Publish Properties dialog.

Multiple MQTT_PUBLISH blocks may be used in a single program.

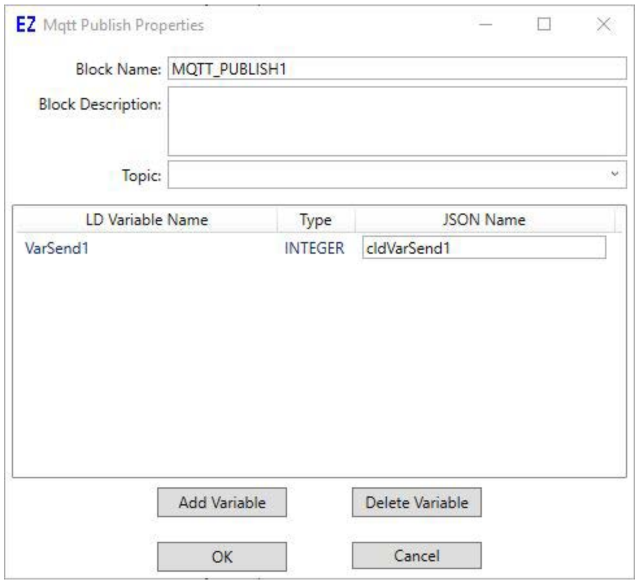


**Figure 28-12**

The function block operates like all other function blocks with an **EN**able input to trigger. The **EN**able input is rising edge sensitve; so when the ENable goes from false to true, the MQTT_PUBLISH block will attempt send data to the VersaCloud M2M+IoT cloud solution selected in the Project Settings.

For data to send regularly, the ENable input must be controlled (toggled) low to high and back to low (after successful send). This can be done using a timer or other control methods.

The **Q** output goes true for one ladder scan (the scan the send happened on), provided there was no errors.

There is an additional output from the MQTT_PUBLISH function block for identifying any error that may have occurred during the send. This outputs requires an INTEGER variable be connected to it. The variable will hold the value output by the function block. Refer to Figure 28-13.

**ER**: MQTT Network Errors. The values listed identify the error.

| | |
|---|---|
| 0 | No Error |
| -1 | Out of Memory Error |
| -2 | Buffer Error |
| -3 | Timeout Error |
| -4 | Reverved (Should not see) |
| -5 | Operation Currently in Progress |
| -6 | Illegal Value |
| -7 | Reverved (Should not see) |
| -8 | Address in Use |
| -9 | Already Connecting |

| | |
|---|---|
| -10 | Connection Already Established |
| -11 | Not Connected |
| -12 | Network Interface Error (Cellular, Ethernet, etc) |
| -13 | Connection Aborted |
| -14 | Connection Reset |
| -15 | Connection Closed |
| -16 | Illegal Argument |

Additional errors below -8192 are specific to the SSL layter. For example, -9984 is a Certificate failed verification (CRL, CA or signature check failed. This can be caused by the certificate not matching, if there is not enough free memory (RAM) in the controller or if the certificate date is not vaild.

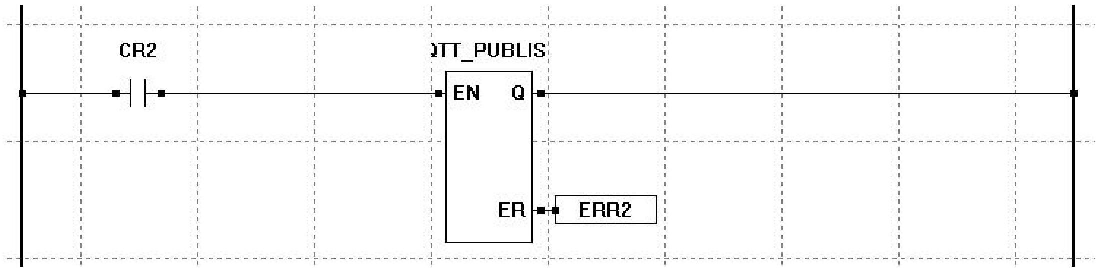


**Figure 28-13**

## Receiving Data from VersaCloud M2M+IoT

At this time, to receive data from VersaCloud M2M+IoT solutions and portals you must use structured text. Future addtions to EZ LADDER may provide a standard function block.

## VersaCloud M2M+IoT using Structured Text

The MQTT function blocks within the ladder diagram is simple to use for communicating to the VersaCloud M2M+IoT solutions (portals) and is ideally suited for transmitting variables quickly and easily. Communications to and receiving data from the portal and additional control features are available using structured text.

Structured text provides more flexibility in sending (such as sending strings) and receiving data (different variable types) that is not supported using the MQTT function blocks. This greater flexibility requires that the communications and it's handling inside an application are written using the target specific and standard structured text functions and requires a greater knowledge of the structured text programming language.

The following target specific structured text function blocks are used for VersaCloud M2M+IoT Portal communications:

**EZ_MQTT_Connect** **EZ_MQTT_Receive**
**EZ_MQTT_Publish**

For details on each of the target specific functions listed, refer to **Appendix B - Target Specific ST Function Reference**.

For details on using structure text, refer to **Chapter 26 - Structured Text**.

# CHAPTER 29

## GPS Support

This chapter provides information on accessing and using the Global Positioning Satellite (GPS) functions on P-Series based PLC on a Chip products using EZ LADDER Toolkit.

## Chapter Contents

## GPS Overview

P-Series PLC on a Chip based targets that support a GPS feature may be used to access NMEA GPS satellite data including location, date/time, movement and precision.

GPS functionality is available only on GPS enabled products and is dependent upon GPS satellite signal availability.

GPS functionality is supported only in Structured Text using the EZ_GPS_GetDateTimeUTC, EZ_GPS_GetMovement, EZ_GPS_GetPosition and EZ_GPS_GetPrecision target specific functions. These functions access and return GPS satellite information in data formats supported by structured text, VersaCloud M2M+IoT portals, COAP communications, SD Card Data-logging, LCD Display and Serial Ports only.

The ladder diagram cannot access most GPS data as the data format is not supported in the ladder diagram.

For details regarding Structured Text, refer to **Chapter 26 - Structured Text**. For details on using ST functions including structured text GPS functions, refer to **Appendix B - Target Specific ST Function Reference**.

## GPS Installation

Before GPS may be used in the ladder diagram / structured text, it must be installed in the ladder diagram Project Settings.

The following is the software requirements for installing GPS support for the target. Refer to the product's User Manual for details on installation of GPS hardware. For many standard products, the GPS software support is automatically installed when the target is selected (or the target's GPS hardware option is selected in the Project Settings; therefore, no additional software configuration is required.

PLC on a Chip integrated circuits and modules do require additional setup. Refer to the following steps.

1. Select the target and click the **PROPERTIES** button. The *Target Properties* window will open. From the drop-down menu (DCPN), select the model / part number of the target.

2. Several devices are required for installing the GPS on a PLC on a Chip (IC or module) target. The GPS requires the installation in Project Setting software for options: GPS (installs under Device) and UART (installs under Bus). If these devices were already installed, they would be listed. See Figure 29-1.

3. Click the **ADD DEVICE** button. The Target's ***Devices*** window will open. All the available devices and features for the target are shown in the Devices section. Scroll down and select GPS and the UART (UARTx where x represents the UART to use for the GPS communications). Figure 29-2 shows the Target's Devices window.

4. Click the **OK** button. Additional windows will open. The UART must be configured for 9600, No Parity, 8 Data bits, 1 Stop bit and RS232 in the *UARTx Properties* dialog for Divelbiss standard GPS interface modules. The same UART must be selected in the *GPS Properties* dialog.

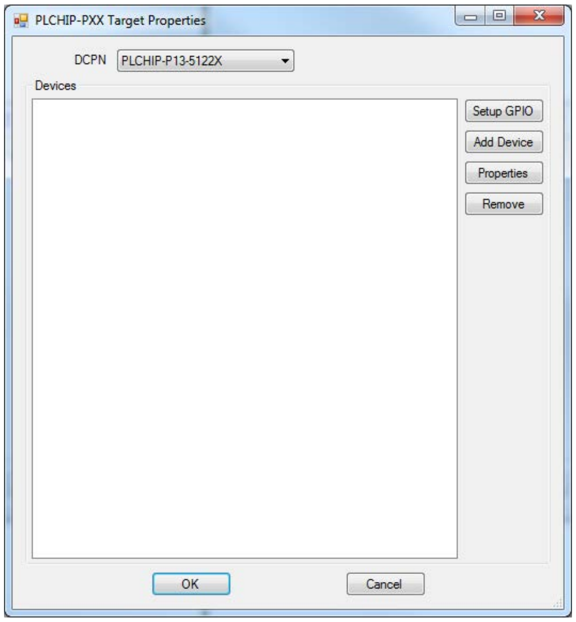
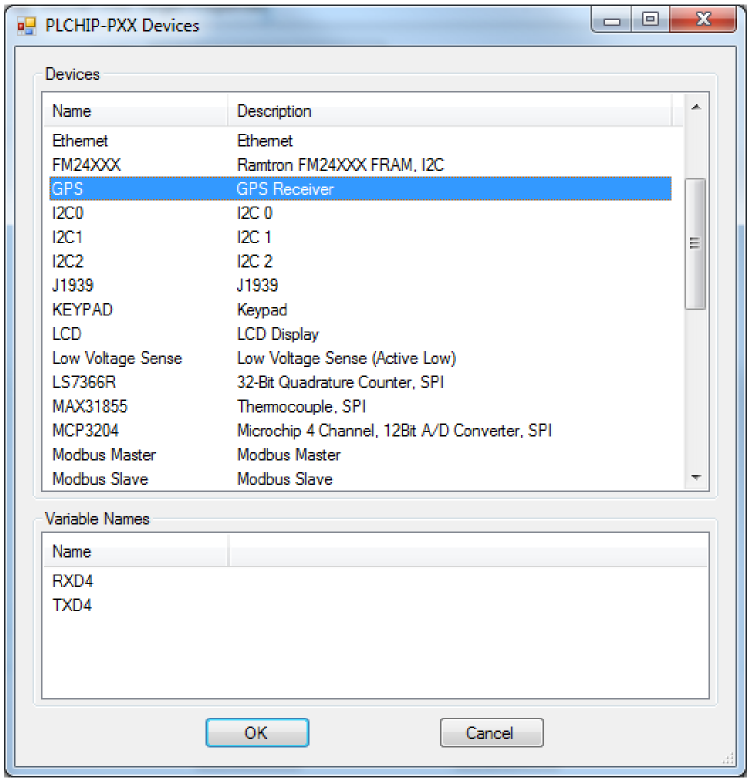


**Figure 29-1**


PLCHIP-PXX Devices

Devices

| Name | Description |
|---|---|
| Ethernet | Ethernet |
| FM24XXX | Ramtron FM24XXX FRAM, I2C |
| GPS | GPS Receiver |
| I2C0 | I2C 0 |
| I2C1 | I2C 1 |
| I2C2 | I2C 2 |
| J1939 | J1939 |
| KEYPAD | Keypad |
| LCD | LCD Display |
| Low Voltage Sense | Low Voltage Sense (Active Low) |
| LS7366R | 32-Bit Quadrature Counter, SPI |
| MAX31855 | Thermocouple, SPI |
| MCP3204 | Microchip 4 Channel, 12Bit A/D Converter, SPI |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |

Variable Names

| Name |
|---|
| RXD4 |
| TXD4 |

**Figure 29-2**

For other GPS modules may require different serial port (UART) settings. Refer to the GPS manufacturer for interface details.

## Using GPS

With the GPS installed (as hardware and in the Project Settings), provided there is a good GPS satellite signal, the GPS functions may be used to access satellite data.

GPS functionality is supported only in EZ LADDER using Structured Text with the structured text (target specific) functions EZ_GPS_GetDateTimeUTC, EZ_GPS_GetMovement, EZ_GPS_GetPosition and EZ_GPS_GetPrecision.

The ladder diagram cannot access most GPS data as the data format is not supported in the ladder diagram. Most GPS interface and decision making must be handled in structured text.

The structured text functions listed can access data received by the GPS receiver and return values that include speed, course, satellites used, latitude,longitude, altitude, month, day, year, hours, minutes and seconds. Additional precision parameters include Mode, VDOP, PDOP and HDOP. Refer to GPS and NMEA definitions and standards for more details regarding these parameters. For details on using structured text, refer to **Chapter 26 - Structured Text** and **Appendix B - Target Specific ST Function Reference**.

The GPS structured text functions access and return GPS satellite data that can be used by structured text, and exported (sent) to VersaCloud M2M+IoT portals, COAP Communications, SD Card Data-logging, LCD Display and Serial Ports as strings directly. To export satellite data from structured text to the ladder diagram, data would need to be converted and formatted.

# CHAPTER 30

## DCCoAP Communications

This chapter provides information on implementing and using DCCoAP communications (formerly VersaCloud M2M) between supported hardware targets and the custom DC-CoAP cloud services.

## Chapter Contents

## DCCoAP Communications Overview

DCCoAP Communications from Divelbiss (formerly known as VersaCloud M2M) solutions cover each area needed for remote control and / or monitoring of machinery and equipment, regardless of where the equipment is located; from the factory floor to remote sites. DCCoAP solutions include interface hardware (PLCs and Gateways) and communications links (Ethernet, WI-FI and cellular data including cellular data coverage plans).

DCCoAP communications require a custom DCCoAP cloud server solution to receive, send and process data.  VersaCloud M2M+IoT supports only MQTT (not DCCoAP). For VersaCloud M2M+IoT Solutions using MQTT, see **Chapter 28 - VersaCloud M2M+IoT Communications**.

EZ LADDER programmed DCCoAP enabled products support the communications from the device to custom DCCoAP servers, portals (and dash boards). Communications to the DCCoAP server may be handled via Ethernet, Wi-Fi or Cellular. For each of these types of communications, the product (target) must support the communication method to be used.

DCCoAP communications via Cellular Data Modem requires a Telematics contract with Divelbiss or a contract with a cellular carrier. Additional charges apply.

All DCCoAP cloud server solutions are custom. Consult Divelbiss for custom COAP cloud server solutions.

For each communication method, EZ LADDER (per ladder diagram project) requires additional configuration and setup for the communications method itself as some communication methods may be used outside of DCCoAP communications. Refer to **Chapter 19 - Ethernet / Wi-Fi** for configuring Wi-Fi or **Chapter 27 - Cellular Connectivity** for configuring the Cellular Data Modem option.

Ethernet and Wi-Fi may be used to communicate via Modbus TCP without a DCCoAP communications (local communications).

Cellular data modem may only be used for communications to custom developed DCCoAP communications solutions and do not provide internet access for the device (browsing, etc).

Regardless of the communications method, the product (target) must support the communication method to be used.

In addition to the communications method configuration, the DCCoAP communications must be configured and enabled in EZ LADDER Toolkit (per ladder diagram project) before it can be used.

## Communications using Ladder Diagram

The DCCoAP function block within the ladder diagram is simple to use for communicating to and from the DCCoAP cloud servers and is ideally suited for transmitting and receiving variables quickly and easily. The DCCoAP function block supports two variable types: REAL and INTEGER.

Other variable types (TIMER or BOOLEAN) must be converted to either INTEGER or REAL variable types before they can be sent / received between the product (ladder diagram) and the DCCoAP cloud server.

## Installing DCCoAP in Project Settings

Before the DCCoAP function block and DCCoAP communications may be used in the ladder diagram program, it must be installed in the Project Settings (for each ladder diagram project).

Verify the communication method is installed (Ethernet, Wi-Fi, Cellular) before proceeding. Refer to this manual's individual chapters for each of the methods for instructions for installation.

1. Using the EZ LADDER Toolkit menu, select **Project.....Settings**. The *Project Settings* dialog will open. The target should have already been pre-selected and configured and the communications method installed previously.

2. Click the **PROPERTIES** button. The *target's Properties* dialog will open. Refer to Figure 30-1.

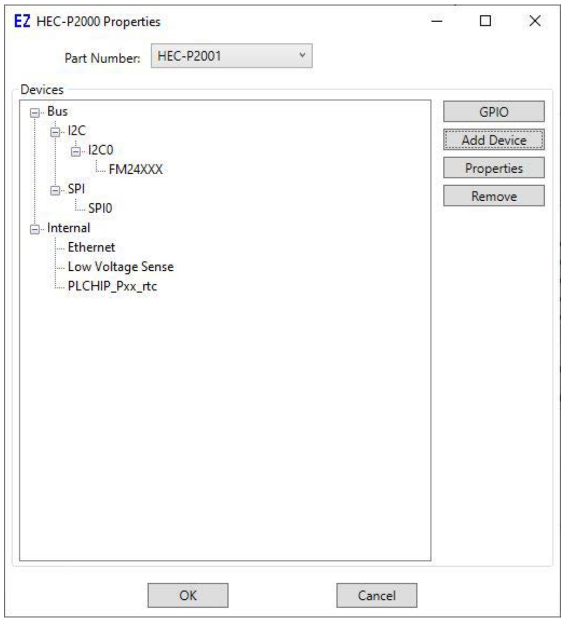
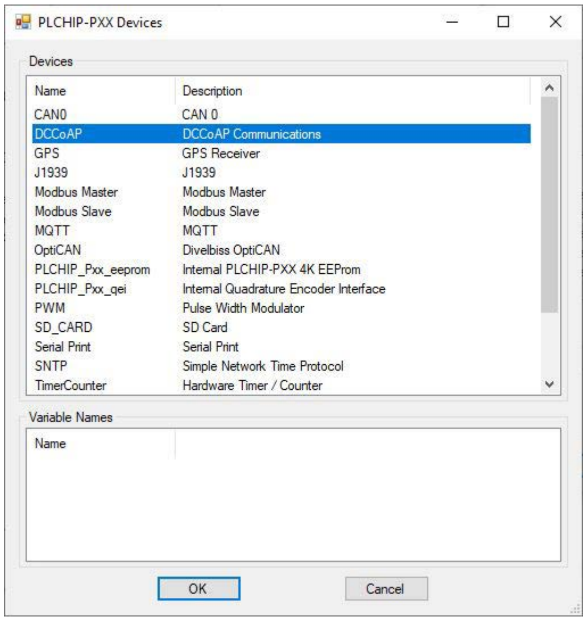


Figure 30-1



| Name | Description |
|---|---|
| CAN0 | CAN 0 |
| DCCoAP | DCCoAP Communications |
| GPS | GPS Receiver |
| J1939 | J1939 |
| Modbus Master | Modbus Master |
| Modbus Slave | Modbus Slave |
| MQTT | MQTT |
| OptiCAN | Divelbiss OptiCAN |
| PLCHIP_Pxx_eeprom | Internal PLCHIP-PXX 4K EEProm |
| PLCHIP_Pxx_qei | Internal Quadrature Encoder Interface |
| PWM | Pulse Width Modulator |
| SD_CARD | SD Card |
| Serial Print | Serial Print |
| SNTP | Simple Network Time Protocol |
| TimerCounter | Hardware Timer / Counter |

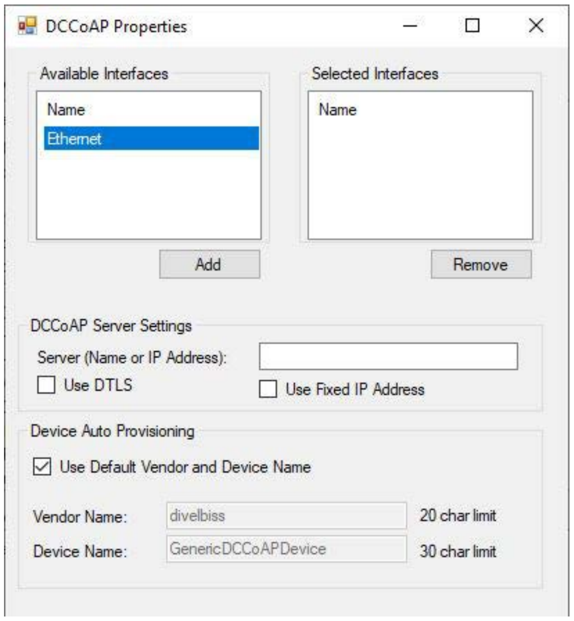
Figure 30-2

3. Click the **ADD DEVICE** button. The *PLCHIP-PXX Devices* dialog will open.

4. Find and select the **DCCoAP** option (highlight). Refer to Figure 30-2.

5. Click **OK.** The *DCCoAP Properties* dialog will open. This dialog is used to configure the DCCoAP communications settings (what the DCCoAP_CLOUD function block uses). Refer to Figure 30-3. The first section of this dialog is divided into two panes: ***Selected Interfaces*** and ***Available Interfaces***.

6. Select (highlight) the communications method in the Available Interfaces pane and click the **ADD** button. The selected method will now be moved from the ***Available Interfaces*** pane to the ***Selected Interfaces*** pane. Refer to Figure 30-4.


DCCoAP Properties

Available Interfaces
Name
Ethernet

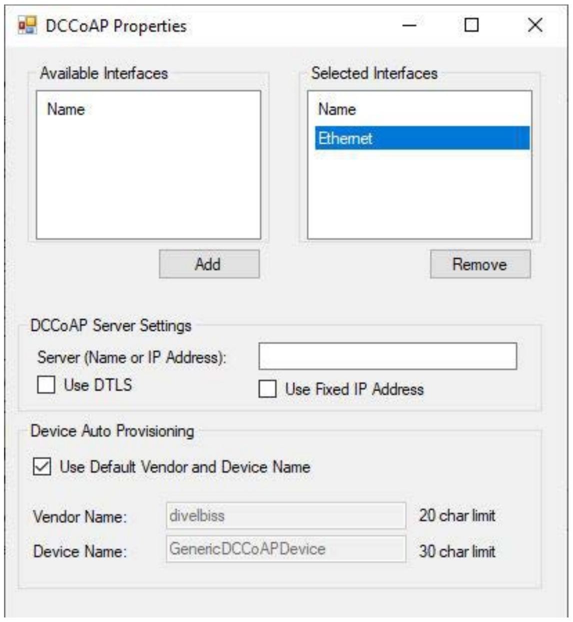
Selected Interfaces
Name

Add Remove

DCCoAP Server Settings
Server (Name or IP Address):
☐ Use DTLS ☐ Use Fixed IP Address

Device Auto Provisioning
☑ Use Default Vendor and Device Name
Vendor Name: divelbiss 20 char limit
Device Name: GenericDCCoAPDevice 30 char limit

**Figure 30-3**


DCCoAP Properties

Available Interfaces
Name

Selected Interfaces
Name
Ethernet

Add Remove

DCCoAP Server Settings
Server (Name or IP Address):
☐ Use DTLS ☐ Use Fixed IP Address

Device Auto Provisioning
☑ Use Default Vendor and Device Name
Vendor Name: divelbiss 20 char limit
Device Name: GenericDCCoAPDevice 30 char limit

**Figure 30-4**

7. Enter the **Server (Name or IP Address)** and other server information for the DCCoAP custom server. This information is provided by the DCCoAP cloud server manager.

8. Set the **Device Auto Provisioning** fields and settings per the requirements of the custom DCCoAP server mapped in item 7. This is specific to how the DCCoAP server is implemented.

9. Click the **OK** button. The *DCCoAP Properties* and *PLCHIP-PXX Devices* dialog will close returning to the *target's Properties* dialog.

10. Click the **OK** button. The T*arget's Properties* dialog will close returning to the *Project Settings* dialog.

11. Click the **OK** button. The *Project Settings* dialog will close. The DCCoAP communications feature is now installed in the ladder diagram project.

Be sure to save the ladder diagram project using the ***Save*** or ***Save As*** from the EZ LADDER Toolkit menu.

## Using the DCCoAP Function Block

The DCCoAP function block is inserted into the ladder diagram by selecting it from the Insert Function drop-down menu next to the **EDIT ST FUNCTIONS** button on the tool bar. Locate the position and click to insert the function block (as any other function block). The *DCCoAP Communications Properties* dialog will open.

This dialog is used to configure the variables that are sent to the DCCoAP cloud server (in the Send Variables pane) and the variables that will be received from the DCCoAP cloud server (in the Receive Variables pane). Refer to Figure 30-5.

Variables to be sent and received must already exist in the ladder diagram or be created using the Insert Variables (**INST VARS**) or Edit Variables (**EDIT VARS**) buttons before they can be added to the *DCCoAP Communications Properties* dialog.

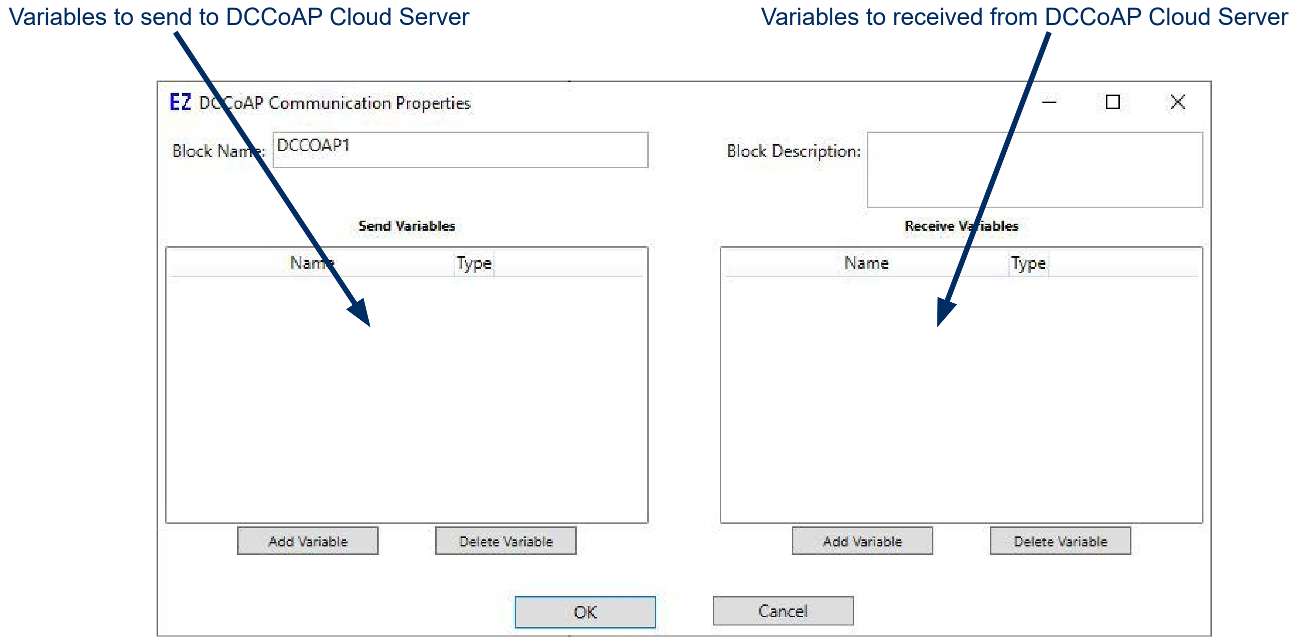


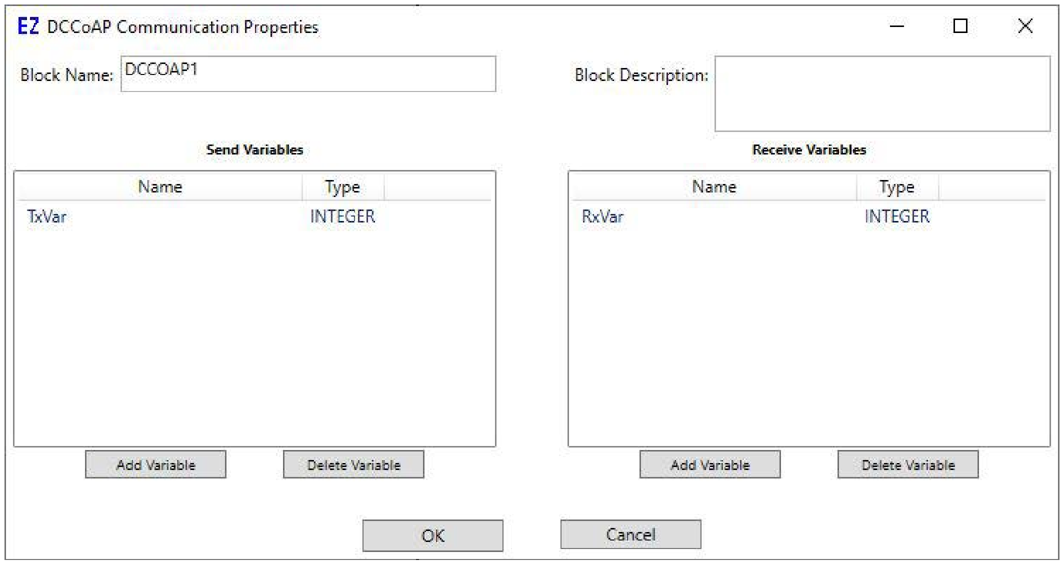
**Figure 30-5**

The DCCoAP (DCCoAP communications) supports REAL and INTEGER variable types only.

Other variable types (TIMER or BOOLEAN) must be converted to either INTEGER or REAL variable types before they can be sent / received between the product (ladder diagram) and the DCCoAP cloud server.

Variables are added to each pane by their corresponding **ADD VARIABLE** buttons. When the **ADD VARIABLE** button is clicked, the *Variables* dialog opens and the variable can be selected from the existing list (from the INTEGER and REAL tabs). Figure 30-6 is an example with one Send variable and one Receive variable.

It is good practice to keep variable names as short as possible as the variable names can affect the transmit and receive data packet sizes. Shorter names will reduce packet size which can be important depending upon the communications method such as cellular where charges are based on data.



Block Name: DCCOAP1

Send Variables

| Name | Type |
|---|---|
| TxVar | INTEGER |

Receive Variables

| Name | Type |
|---|---|
| RxVar | INTEGER |

**Figure 30-6**

Variables can be deleted from the Send and Receive panes by selecting the variable from the list in the appropriate pane and click the **DELETE VARIABLE** button.

When all the variables have been added, click **OK**.

The DCCoAP function is now placed and is ready to be connected in the ladder diagram.

Based on the DCCoAP server, the product (target) may be required be pre-registered as a device on your personalized DCCoAP cloud server. This action must be handled before any communications can take place.

In addition device being registration, the actual product (target) may require activation (based on the needs of the DCCoaP custom cloud server). If the activation is required, but completed or is not completed successfully, communications between the target and the DCCoAP cloud server will not function (See error codes).

The variable names entered in the DCCoAP function block will need to match specific configuration names on the actual DCCoAP cloud server. The specific requirements and implementation is custom for each DC-CoAP server implementation.

The DCCoAP function block has one logic input: EN. The EN (enable) input is used to enable the function block and EN is **rising edge sensitive**. It will only function once (send and receive to / from the DCCoAP cloud server) for each time a rising edge is seen on the EN input. This edge sensitivity is helpful for controlling send and receive operations.

For a schedule send and receive based on time, the function's EN input can be controlled using an internal control relay (CR) that is being controlled using a timer. Figure 30-7 illustrates such ladder diagram circuit.

Based on latency and communications times, the quickest any values can be sent / received and updated is about 1 / second.

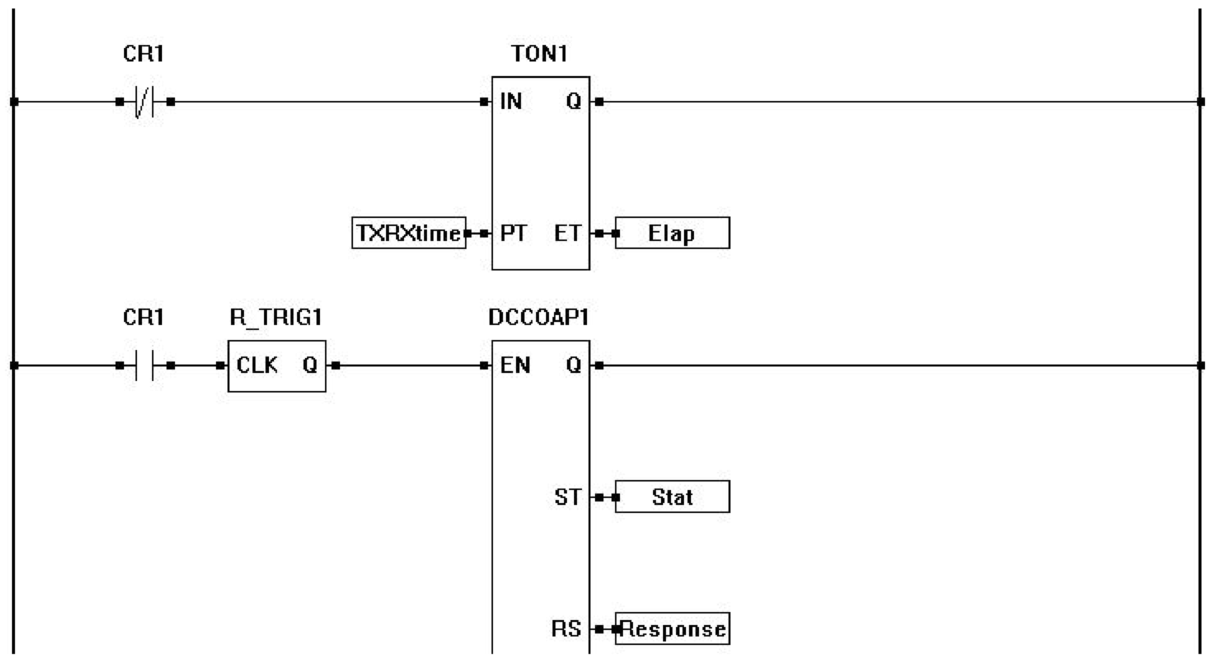


**Figure 30-7**

The DCCoAP function block has 3 logic outputs: Q, ST and RS.

The Q output is true when the function block's EN input is true.

The ST output pin provides the current function blocks status. This should be connected to an Integer variable. The status codes are as follows:

| | |
|---|---|
| 3 = Completed | 0 = Not Used |
| 2 = Sending | -1 = Error |
| 1 = Queued to Send | |

The RS output pin is the Response Code provided by the COAP cloud server during communications. This should be connected to an Integer variable.

If the DCCoAP function block encounters an error, it will return one of these error codes on the RS pin. The common codes are shown in red.

| Error | Error Codes | Description |
|---|---|---|
| None | 0 | No Error |
| LWIP | -1 | Contact Divelbiss Support for this error. |
| No Connection | -3 | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| Already Activated | -4 | Device has already been activated on the DCCoAP cloud server. Contact the COAP cloud server support for more information. |
| Activation Failed | -5 | The activation for this device failed on the DCCoAP cloud server. Contact the COAP cloud server support for more information. |
| Null Pointer | -6 | Contact Divelbiss Support for this error. |
| Invalid String | -7 | Contact Divelbiss Support for this error. |
| Queue is Full | -8 | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| Not Idle | -9 | Contact Divelbiss Support for this error. |
| Other Error | -10 | Other unspecified error. Contact Divelbiss Support for this error. |
| Not Activated | -11 | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| Time-out | -20 | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

When the DCCoAP function block has completed without an error, it will return one of the responses on the RS pin. This code returned is built with 8 total bits. These bits are divided with the 3 upper bits being the response code before the '.' shown below and the 5 lower bits representing the two digits after the '.' shown below. A conversion would be necessary to use this response code.
Typical response codes would be 69 or 129 (shown in red). Contact Divelbiss support for additional codes.

| Response | Converted Code | Description |
|---|---|---|
| 65 | 2.01 | Created |
| 68 | 2.04 | Changed |
| 69 | 2.05 | Content was received. |
| 95 | 2.31 | Continue |
| 129 | 4.01 | Unauthorized - The CIK couldn't be used to authenticate. |

| | | |
|---|---|---|
| 130 | 4.02 | Bad Option |
| 131 | 4.03 | Forbidden |
| 132 | 4.04 | Not Found |
| 136 | 4.08 | Request Entity Incomplete |
| 140 | 4.12 | Precondition Failed |

## Activating the Device on a DCCoAP Cloud Server

Each product (device), depending on the DCCoAP Cloud server, most likely will be required to be activated on the COAP cloud server before it can use the DCCoAP (or structured text) for communications (sending / receiving data). This activation is usually only required once per device (unless the activation is lost or deleted from the DCCoAP cloud server).

When activation is required, before any communication may occur, the product (target) must be pre-registered as a device on DCCoAP cloud server. This action is handled the DCCoAP cloud manager or other entity.

Activating a product (target that has been pre-registered on the DCCoAP cloud server requires the use of structured text functions, specifically **EZ_DCCoAP_Activate**. This function connects to the DCCoAP cloud server and handles creating a link from the product to the cloud server (portal). This is typically only needed to be done one time.

The activation process reads and writes a CIK identifier to the target's EEPROM memory. This is used for communications to the DCCoAP cloud server (identifier).

The CIK may be written and erased using target specific structured text functions. This should be done under direction of Divelbiss personnel. Erasing or modifying the CIK may cause loss of communications to the DCCoAP cloud server and require the device to be added to the DCCoAP cloud server and re-activated.

For more details regarding communications codes for the EZ_DCCoAP_Activate structured text function, refer to the target specific function **EZ_DCCoAP_Activate** (see **Appendix B- Structured Text Function Reference**)

Once activation is successful, it does not need to be repeated as the CIK identification and other parameters are stored.

As it needs only be activated one time, there is flexibility in allowing for a stand-alone activation program or implementing the activation needs into a larger application.

## DCCoAP Communications to Cloud Solutions

The DCCoAP function block within the ladder diagram is simple to use for communicating to and from DCCoAP cloud servers / portals and is ideally suited for transmitting and receiving variables quickly and easily. Communications to and receiving data from the portal, activation and additional control features are available using structured text.

Structured text provides more flexibility in sending and receiving data (such as sending and receiving strings, etc) that is not supported using the DCCoAP function block. This greater flexibility requires that the communications and it's handling inside an application are written using the target specific and standard structured text functions and requires a greater knowledge of the structured text programming language.

The following target specific structured text function blocks are used for DCCoAP Cloud communications:

**EZ_DCCoAP_Activate**
**EZ_DCCoAP_EnableInterface**
**EZ_DCCoAP_EraseCIK**
**EZ_DCCoAP_GetServerTime**
**EZ_DCCoAP_GetState**
**EZ_DCCoAP_ReadCIK**
**EZ_DCCoAP_SendData**
**EZ_DCCoAP_SendDataRecord**
**EZ_DCCoAP_WriteCIK**

For details on each of the target specific functions listed, refer to **Appendix B - Target Specific ST Function Reference**.

For details on using structure text, refer to **Chapter 26 - Structured Text**.

# Appendix A

## Function Reference

This chapter provides detailed information for each function block and object found in the EZ LADDER Toolkit (for P-Series Targets). For each function block and object, the following is provided: type, inputs, outputs and other special instructions needed to use them.

## Chapter Contents

## Object and Function Block Basics

This chapter provides information on using each of the EZ LADDER Toolkit function blocks and objects. For each object or function, the symbol diagram, information on the inputs and outputs and a description of the function or block operation is provided. When applicable, truth tables, timing diagrams or other functions details are provided.

This information is to provide basic practices of how each function or object works and is not intended to provide complete applications or uses.

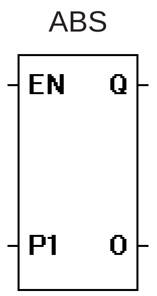
As this chapter is a reference, providing function block and object details on ALL functions available EZ LADDER Toolkit, the presence of a function does not guarantee availability of the function on all hardware targets.

Availability of functions and objects is determined by the hardware target that is configured for the ladder diagram projects. Some functions and objects are not available on some targets. Refer to the actual hardware target's data sheet / manual or **Chapter 22 - Hardware Targets** for a list of supported functions and objects based on target selection.

It is important to formulate which function blocks may be used in a ladder diagram project and then verify and select the target that supports the desired features and function blocks.

All objects and function blocks described in this chapter are organized in alphabetical order.

## ABS



### Description:

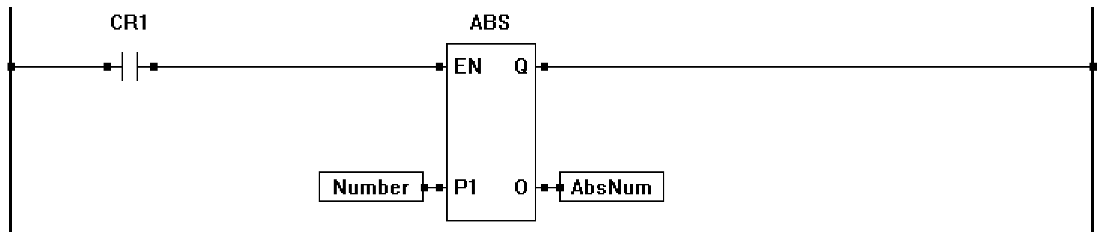
The ABS function provides an absolute value output (O) from the input value (P1). The enable (EN) must be true for the ABS function block to be enabled. The Q output is true when the ABS function is enabled.

### Input / Output Connections:

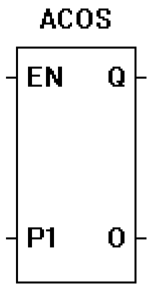
The ABS function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | X | | | | |
| Q | Output | | | X | | | |
| O | Output | X | X | | | | |

### Example Circuit:



## ACOS



### Description:

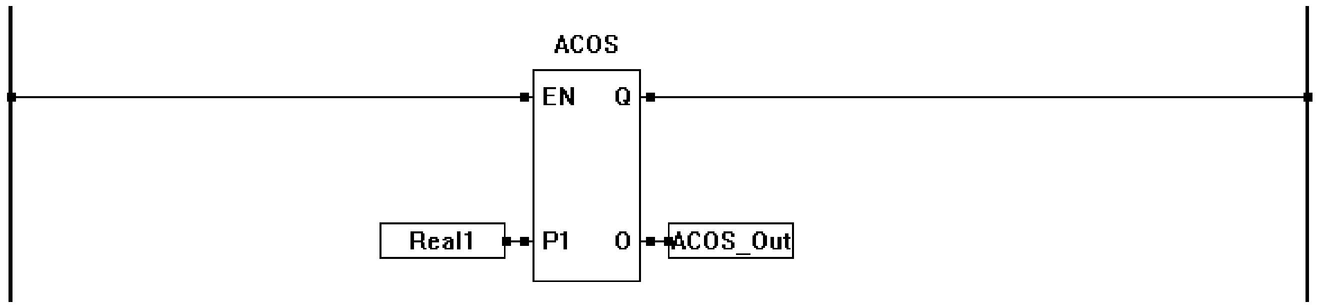
The ACOS function provides an Arc cosine (O) from the input value (P1). The enable (EN) must be true for the ACOS function to be enabled. The Q output is true when the ACOS function is enabled.

### Input / Output Connections:

The ACOS function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

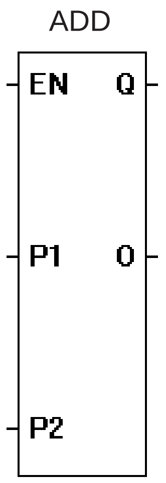
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Arc Cosine of P1 Base Number |

### Example Circuit:



**Related Functions:** ASIN, ATAN, COS, SIN, TAN

## ADD



### Description:

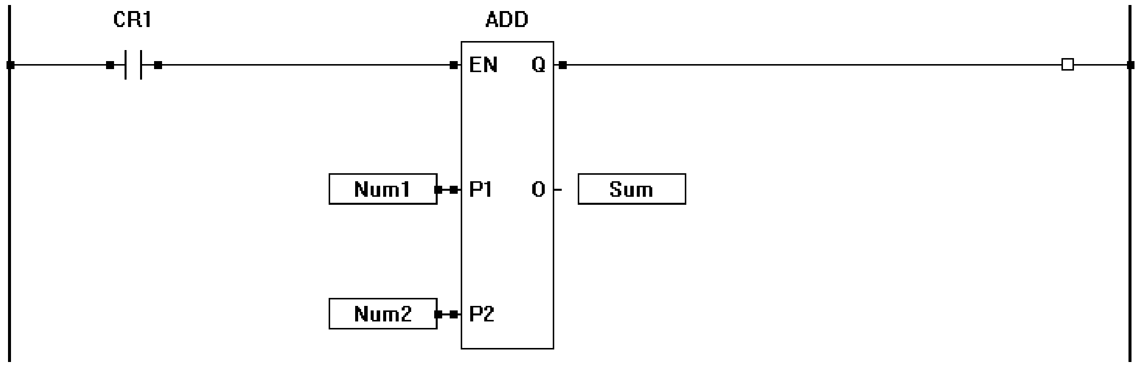
The ADD functions sums all the inputs (Px) together and outputs this number (O). The number of inputs is specified when the function is placed in the program. The enable (EN) must be true for the ADD function to be enabled. The Q output is true when the ADD function is enabled.

### Input / Output Connections:

The ADD function block placement requires connections of at least three input pins (EN, P1, P2) and two output pins (Q, O).The number of inputs is specified when the function is inserted. The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

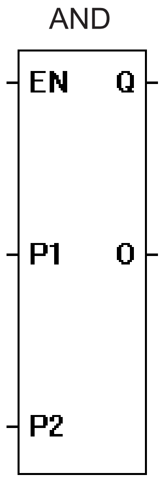
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| Px | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |
| O | Output | X | X | | | | |

### Example Circuit:



**Related Functions:** SUB, MULT, DIV

## AND



### Description:

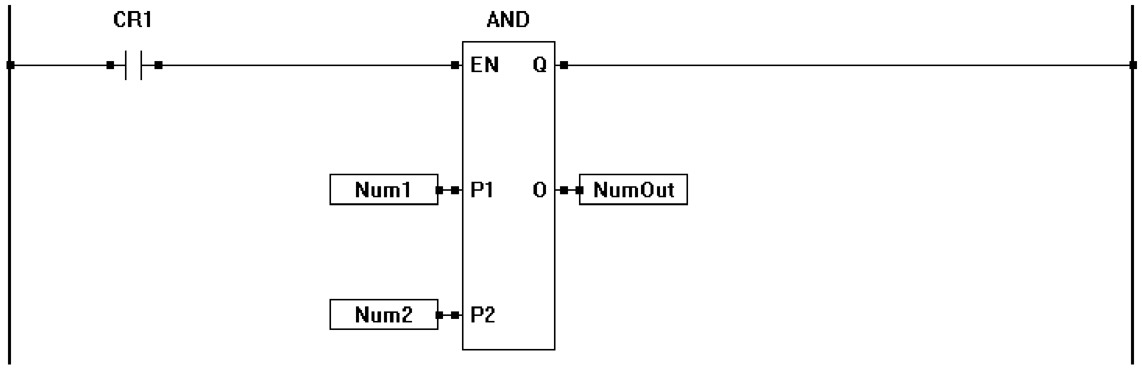
The AND functions provides a bitwise AND function of the P1 and P2 inputs. The enable (EN) must be true for the AND function to be enabled. The Q output is true when the AND function is enabled.

### Input / Output Connections:

The AND function block placement requires connections of three input pins (EN, P1, P2) and two output pins (Q, O).

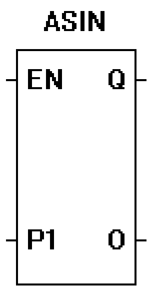
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| Q | Output | | | X | | | |
| O | Output | X | | | | | |

### Example Circuit:



**Related Functions:** OR, NOT, XOR

## ASIN



### Description:

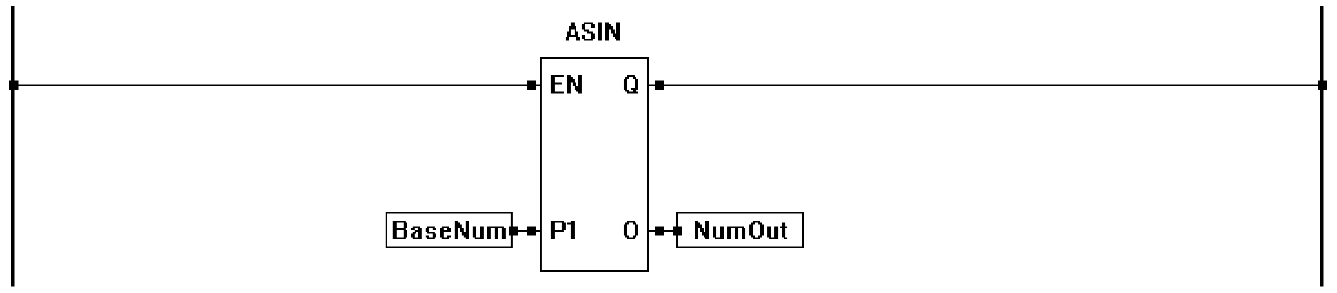
The ASIN function provides an Arcsine (O) from the input value (P1). The enable (EN) must be true for the ASIN function to be enabled. The Q output is true when the ASIN function is enabled.

### Input / Output Connections:

The ASIN function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

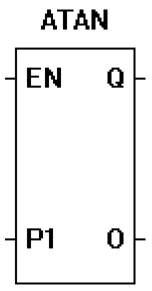
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Arc Sine of P1 Base Number |

### Example Circuit:



**Related Functions:** ACOS, ATAN, COS, SIN, TAN

## ATAN

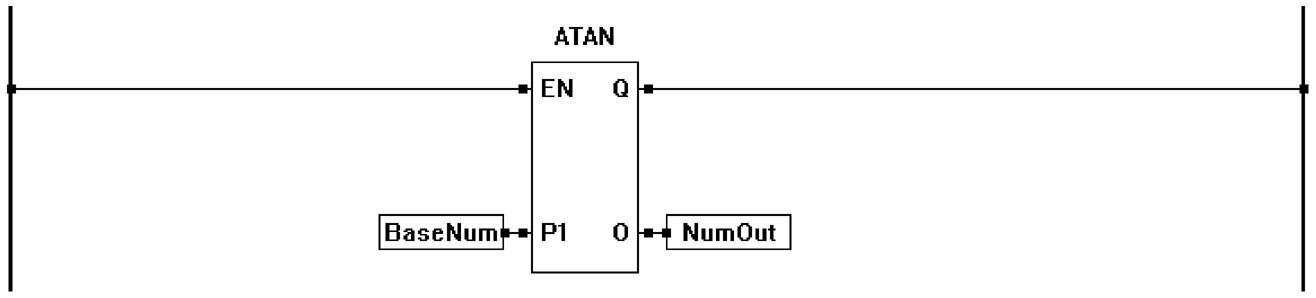


### Description:
The ATAN function provides an Arctangent (O) from the input value (P1). The enable (EN) must be true for the ATAN function to be enabled. The Q output is true when the ATAN function is enabled.

### Input / Output Connections:
The ATAN function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

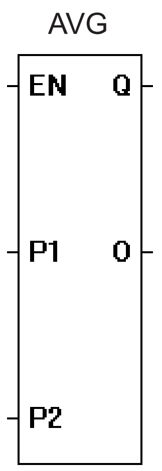
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Arc Tangent of P1 Base Number |

### Example Circuit:



**Related Functions:** ACOS, ASIN, COS, SIN, TAN

## AVG



### Description:

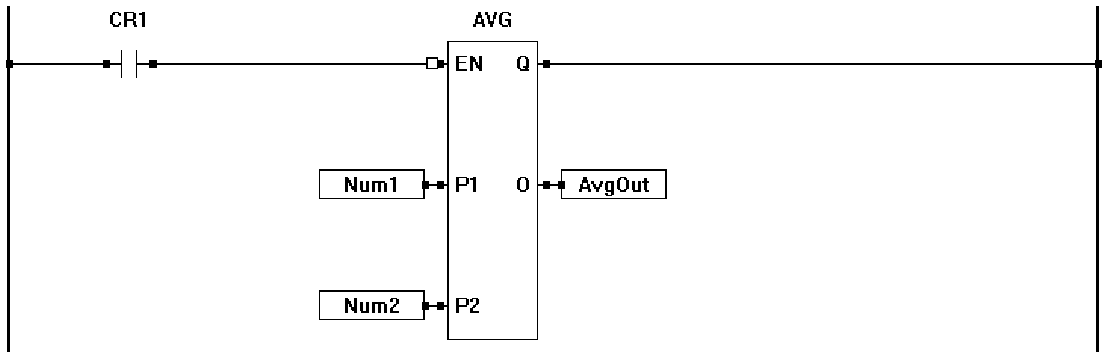
The AVG function averages all the inputs (Px) together and outputs this number (O). The number of inputs is specified when the function is placed in the program. The enable (EN) must be true for the AVG function to be enabled. The Q output is true when the AVG function is enabled.

### Input / Output Connections:

The AVG function block placement requires connections of a minimum of three input pins (EN, P1, P2) and two output pins (Q, O). The number of inputs is specified when the function is inserted. The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

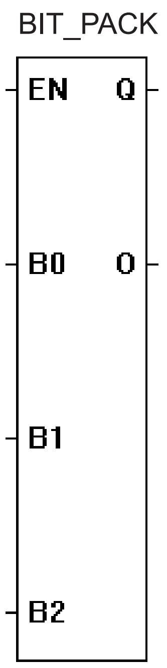
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| Px | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |
| O | Output | X | X | | | | |

### Example Circuit:



**Related Functions:** MAVG

## BIT_PACK



### Description:

The BIT_PACK is a configurable function that will convert the inputs bits (from binary) to a single 32 bit integer number. The Bx inputs are the bits to pack, the EN enables the function when true. The Q output is true when the function is enabled. The output O is the 32-bit integer result of the packed inputs.

The **number of bits** must be identified when the function is placed in the ladder diagram (1-32 bits). Only boolean variables or contacts may be used as bit inputs.

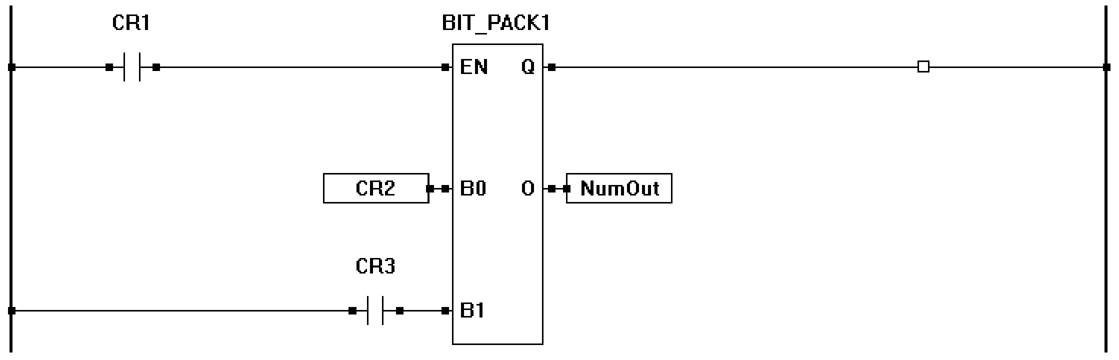
Included in the configuration is the **bit offset**. The bit offset allows the programmer to use multiple BIT_PACK functions and have a single 32 bit output integer by offsetting the bit range for each function block. Note the number of bits + offset bits must be less than or equal to 32.

### Input / Output Connections:

The BIT_PACK function block placement requires connections of a minimum of two input pins (EN, B0) and two output pins (Q, O). The number of bits is specified when the function is inserted. The EN is not considered a bit to pack and is not included in the number of bits to pack when placing the function block.

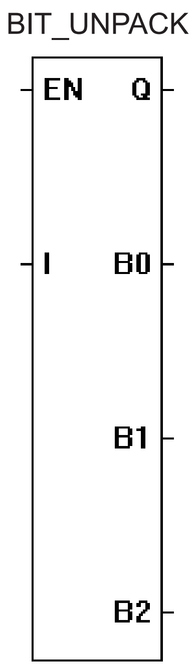
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| Bx | Input | | | X | | | Number of Bits is dynamic |
| Q | Output | | | X | | | |
| O | Output | X | | | | | |

### Example Circuit:



**Related Functions:** BIT_UNPACK

## BIT_UNPACK



### Description:

The BIT_UNPACK is a configurable function that will convert a 32 bit integer into up to 32 individual boolean outputs (bits). The I input is the 32 bit integer input, the EN enables the function when true. The Q output is true when the function is enabled. The Bx outputs are the result of the integer being converted to bit outputs (binary equivalent).

The **number of bits** must be identified when the function is placed in the ladder diagram (1-32 bits). Only boolean variables may be used as bit outputs.

Included in the configuration is the **bit offset**. The bit offset allows the programmer to use multiple BIT_UNPACK functions and have a single 32 bit input integer by offsetting the bit range for each function block. Note the number of bits + offset bits must be less than or equal to 32.

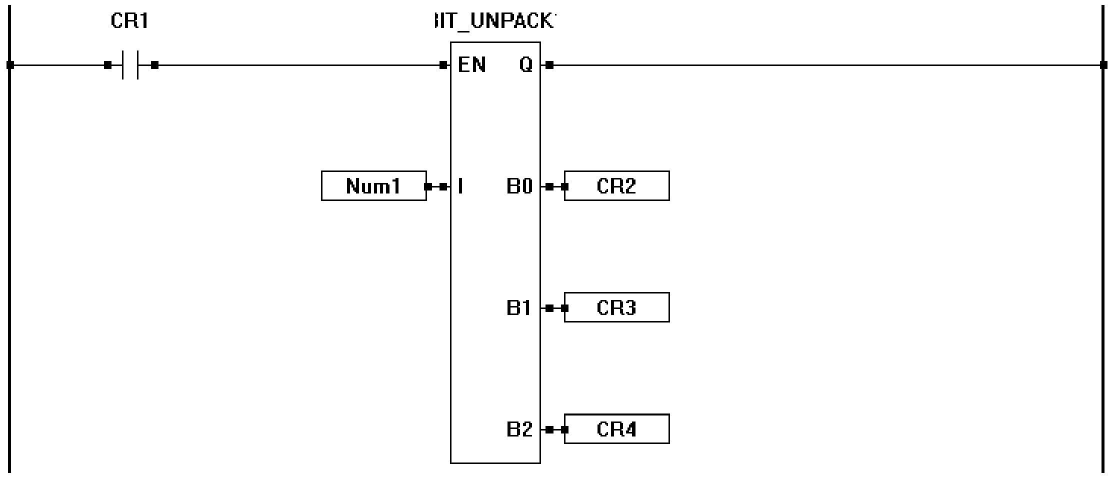
### Input / Output Connections:

The BIT_UNPACK function block placement requires connections of two input pins (EN, I) and a minimum of two output pins (Q, B*x*). The number of bits is specified when the function is inserted. The EN is not considered a bit to unpack and is not included in the number of bits to unpack when placing the function block.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| I | Input | X | | | | | |
| Q | Output | | | X | | | |
| Bx | Output | | | X | | | Number of Bits is dynamic |

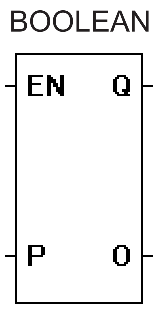
Although the output type for the B*x* bit outputs is boolean, boolean variables must be connected to the B*x* outputs. It is not allowed to connect coils.

### Example Circuit:



**Related Functions:** BIT_PACK

## BOOLEAN



### Description:

The BOOLEAN function converts the input (P) into a boolean (zero or non-zero) output (O). The enable (EN) must be true for the BOOLEAN function to be enabled. The Q output is true when the BOOLEAN function is enabled.

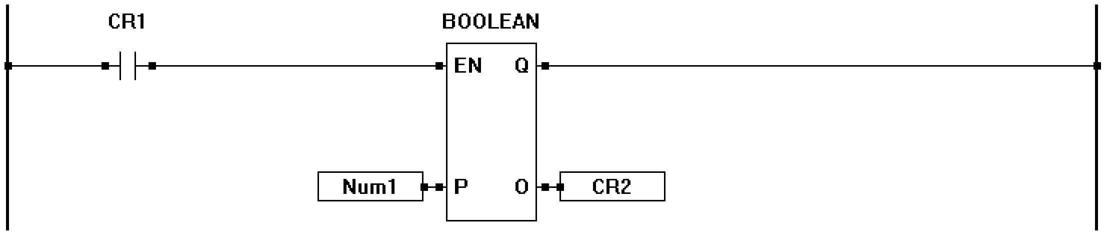
### Input / Output Connections:

The BOOLEAN function block placement requires connections of two input pins (EN, P) and two output pins (Q, O).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | X | X | | | |
| Q | Output | | | X | | | |
| O | Output | | | X | | | |

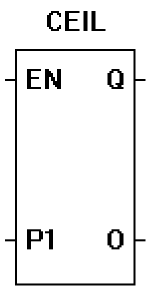
Although the output type for the O output is boolean, a boolean variable must be connected to the O output. A coil may be connected, but compile errors will result.

### Example Circuit:



**Related Functions:** INTEGER, REAL, TIMER

## CEIL



### Description:

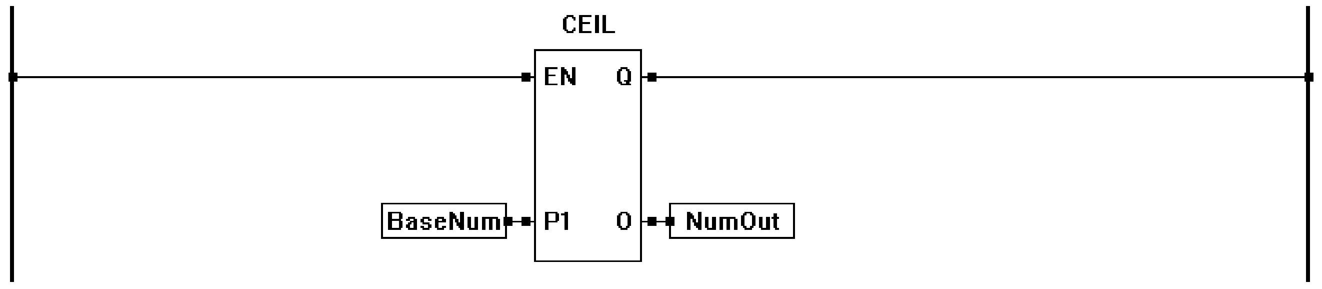
The CEIL function provides a rounded-up result of the P1 Input and outputs this number (O). The enable (EN) must be true for the CEIL function to be enabled. The Q output is true when the CEIL function is enabled.

### Input / Output Connections:

The CEIL function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

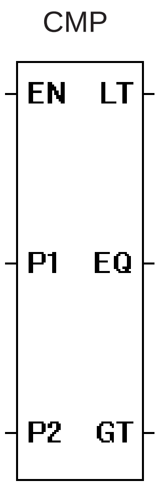
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Rounded Up P1 Base Number |

### Example Circuit:



**Related Functions:** FLOOR

## CMP



### Description:

The CMP function compares the P1 and P2 inputs. LT is true when the P1 input is less than the P2 input. EQ is true when the P1 input equals the P2 input. GT is true when the P1 input is greater than the P2 input. The enable (EN) must be true for the CMP function to be enabled. When the function is disabled, all outputs (LT, GT and EQ) are off.

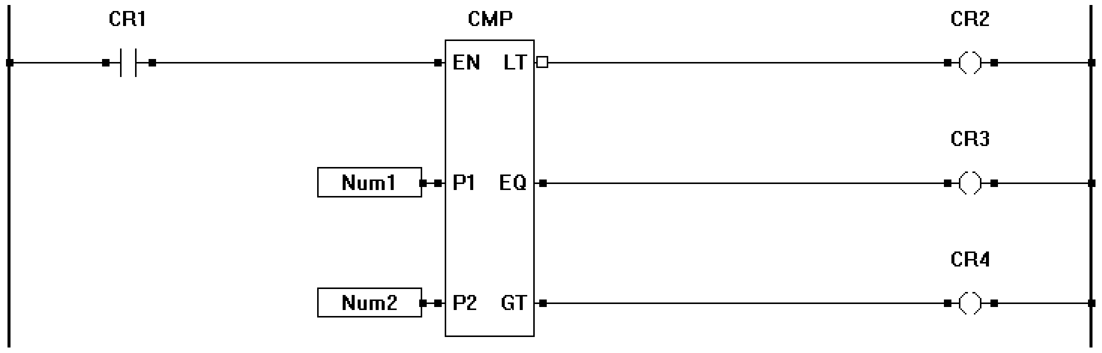
### Input / Output Connections:

The CMP function block placement requires connections of three input pins (EN, P1, P2) and three output pins (LT,EQ, GT). There is no Q output on the CMP function block

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | X | | | | |
| P2 | Input | X | X | | | | |
| LT | Output | | | X | | | |
| EQ | Output | | | X | | | |
| GT | Output | | | X | | | |

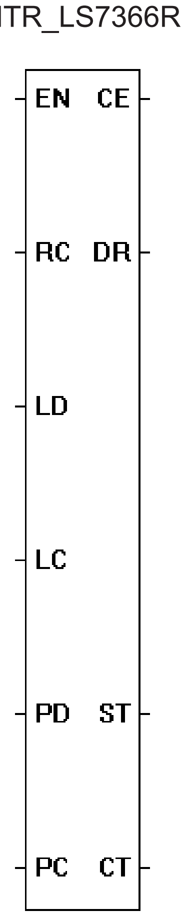
Although the output type for the LT,EQ and GT outputs is boolean, coils must be connected to the them. A boolean variable may be connected and it will compile, but it will not function on the target.

### Example Circuit:



**Related Functions:** LIMIT, HYSTER

## CNTR_LS7366R



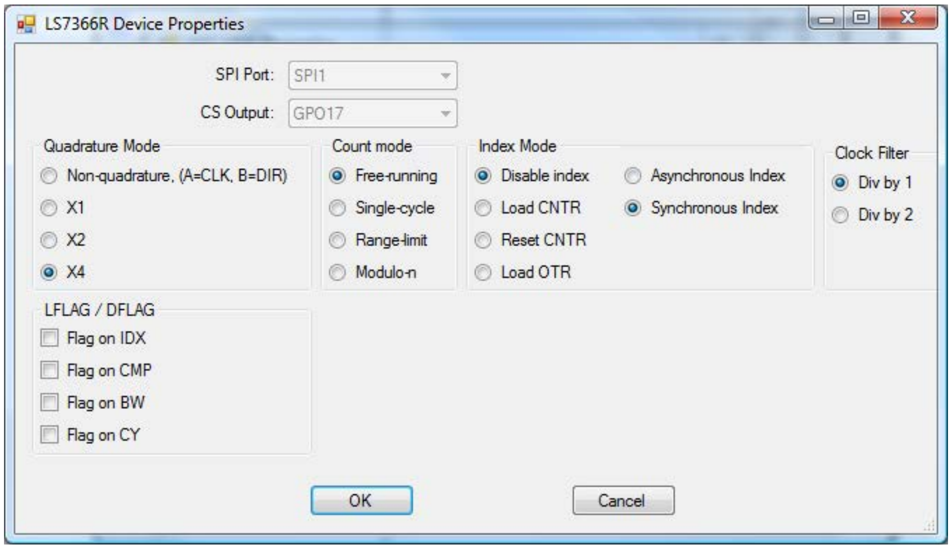
### Description:

The CNTR_LS7366R function block is used to read and write to the LS7366R counter integrated circuit. The LS7366R is an integrated circuit that operates as a high speed counter that supports counting up, down and also quadrature. In addition to this function block, the LS7366R must be configured for the application. It is configured in the *Project Settings*. A description of the configuration will follow later in this function block explanation.

The LS7366R operates using internal registers. There are three registers in the LS7366R. OTR, DTR and Actual Count. Per the design of the LS7366R, the actual count register can never be directly read or written to; therefore, the other registers must be used to read and write to the actual count. As an example, when the function block Read Count (RC) input is true, the actual count is copied to the OTR register and then the OTR registers is output at the function blocks count (CT) output. DTR is used to set the count value and may be used as a comparison (see LFLAG/DFLAG).

The first step is to configure the LS7366R. While targets may differ slightly, this configuration is found by clicking the ***Menu...then Project Settings***. Look for a button **LS7366R Properties**. Clicking this button will open the LS7366R Device Properties Window. In this window, configure the LS7366R for the type of application (type of counting along with optional settings).

### LS7366R Device Properties Window:


LS7366R Device Properties

SPI Port: SPI1

CS Output: GPO17

Quadrature Mode: Non-quadrature, (A=CLK, B=DIR); X1; X2; [✓ X4]

Count mode: [✓ Free-running]; Single-cycle; Range-limit; Modulo-n

Index Mode: [✓ Disable index]; Load CNTR; Reset CNTR; Load OTR; Asynchronous Index; [✓ Synchronous Index]

Clock Filter: [✓ Div by 1]; Div by 2

LFLAG / DFLAG: Flag on IDX; Flag on CMP; Flag on BW; Flag on CY

Continued Next Page------»

## Quadrature Mode

**Non-quadrature**: Counter input B sets the direction of counting (increase or decrease), and a pulse on input A causes the counter to count by 1.
**X1 quadrature**: Counter operates in X1 quadrature mode.
**X2 quadrature**: Counter operates in X2 quadrature mode.
**X4 quadrature**: Counter operates in X4 quadrature mode.

## Count Mode

**Free-Running**: Free running mode. Counter will wrap in either direction if maximum or minimum value is reached.
**Single-cycle**: Counter will count until maximum value is reached and then stop counting. Used with CY Flag. Counter must be reset to continue counting.
**Range-limit**: Counter will only count between zero and the value loaded in the DTR register.
**Modulo-n**: Actual count will equal number of pulses divided by value of the DTR register + 1.

## Index Mode

**Disable Index**: Index input is disabled and will not cause any action on the actual count register.
**Load CNTR**: When the index input is active, the actual count register is loaded with the value of the DTR register. The DTR register is loaded using PD and LD on the function block.
**Reset CNTR**: When the Index input is active, the actual count register is reset to zero.
**Load OTR**: When the index input is active, the OTR register is loaded with the actual count register value. The OTR register is used to read the current count.
**Asynchronous Index**: In quadrature mode, if index is active, it is applied (acted on) regardless of its phase relationship to inputs A and B.
**Synchronous Index**: In quadrature mode, If index is active, it must meed the phase relationship of inputs A and B before it can be applied (acted on).

## Clock Filter

**Div by 1**: Filter Frequency divided by 1. This is based on the input frequency of A and B inputs.
**Div by 2**: Filter Frequency divided by 2. This is based on the input frequency of A and B inputs.

## LFLAG/DFLAG

**Flag on IDX**: When index is true, the LFLAG will set and latch, while DFLAG will be set only while the condition is maintained.
**Flag on CMP**: When actual count value = value of the DTR register, the LFLAG will set and latch, while DFLAG will be set only while the condition is maintained.
**Flag on BW**: When enabled, when counter wraps from zero to maximum, the LFLAG will set and latch, while DFLAG will be set only while the condition is maintained.
**Flag on CY**: When enabled, when counter wraps from maximum to zero, the LFLAG will set and latch, while DFLAG will be set only while the condition is maintained.

The DFLAG and LFLAG is typically read using digital inputs that are specific to each target. Refer to the target's User Manual.

In addition to the hardware inputs that control the LS7366R, the CNTR_LS7366R function block is used in EZ LADDER to read the count, reset the count and control the registers.

Continued Next Page------»

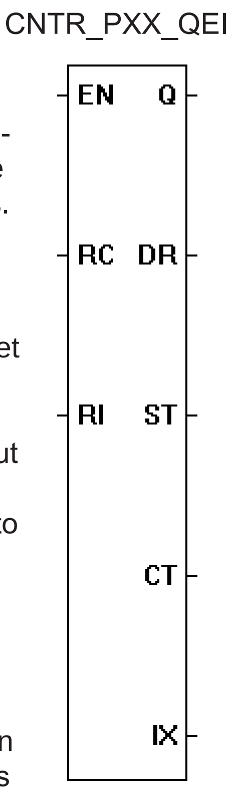
## Function Block Inputs:

| | |
|---|---|
| EN: | Function block enable (Boolean). When true, the function block is enabled. |
| RC: | Read Count Input (Boolean). When true, the actual count is internally copied to OTR and then OTR is output at the count output (CT). When false, the OTR is output at the count output (CT) without copying the actual count. |
| LD: | Load DTR input (Boolean). When true, the DTR register is loaded with the value of the variable connected to the PD input. When using LC and LD, LC has a higher priority and will execute first before LD. |
| LC: | Load Counter input (Boolean). When true, the value of PC is loaded into the DTR register and then the DTR register is copied to the actual count. When using LC and LD, LC has a higher priority and will execute first before LD. |
| PD: | Value (Integer) to be loaded into DTR when LD input is true. |
| PC: | Value (Integer) to be loaded into DTR and then actual count when LC is true. |

## Function Block Outputs:

| | |
|---|---|
| CE: | Output (Boolean) is true when the function block is enabled and no errors are present. |
| DR: | Direction output (Boolean). Identifies the current count direction (0 or 1). |
| ST: | Status output (Integer). The output provides a numeric represtation of the status of the LS7366R current function. Consult factory if more information is required. |
| CT: | Current Count (Integer). |

## CNTR_PXX_QEI



### Description:

The CNTR_PXX_QEI function block is used to read the current position of a quadrature encoder connected to the quadrature encoder inputs. For more details on how the quadrature counter inputs and function blocks operate together, see **Chapter 18 - Counters & Timers**.

The enable (EN) input must be true for the CNTR_PXX_QEI function block to operate. The Q output is true when the function block is enabled. The reset position counter input (RC) will reset the internal position counter when true. The reset index counter input (RI) will reset the internal index counter when true.

The direction output (DR) identifies the direction of encoder travel (0 or 1). The status output (ST) identifies the current status of the encoder. The position counter value output (CT) will be equal to the current encoder position. The index counter value output (IX) will be equal to the current index counter (number of time the encoder has passed its maximum position).

IX is incremented and decremented as needed when the current encoder position either passes its maximum position or its minimum position.

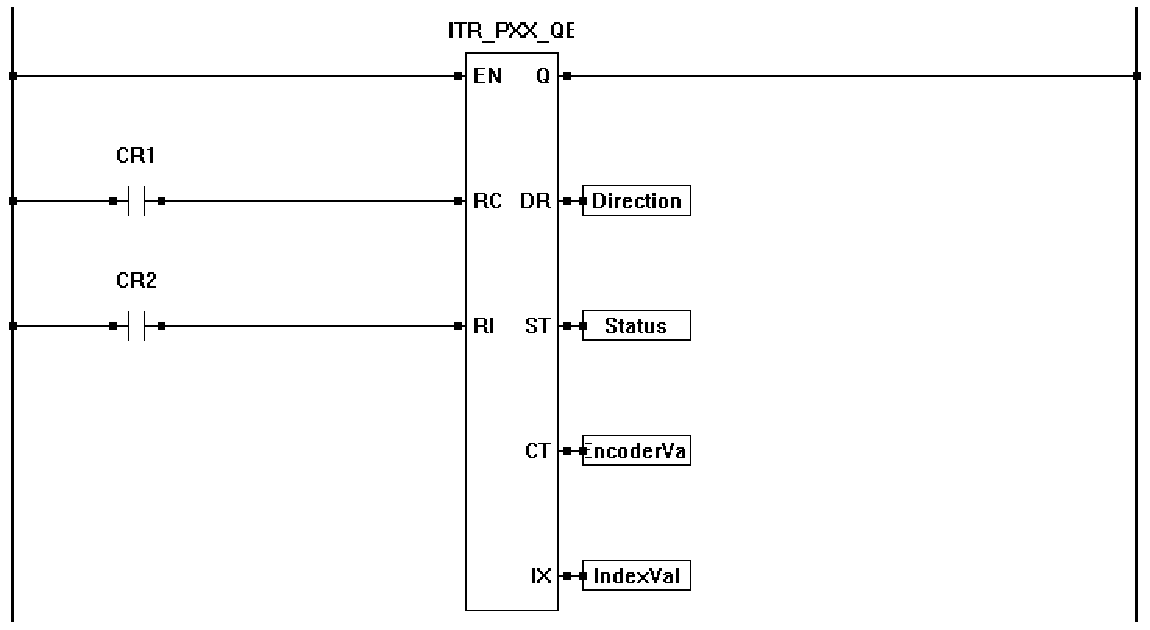
The quadrature encoder must be installed in the Project Settings before this function block may be placed. Other parameter configurations must be completed during this installation. Refer to **Chapter 18 - Counters & Timers**.

Based on the qudarature encoder input configuration in the Project Settings menu, as the encoder moves in the forward direction, when the position exceeds the maximum value, the index counter is incremented and the position is set to zero. As the encoder moves in a reverse direction, when the position reaches 0, the index counter is decremented and the position is set to the maximum.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| RC | Input | | | X | | Active True | Resets Position Counter (CT) |
| RI | Input | | | X | | Active True | Resets Index Counter (IX) |
| DR | Output | | | X | | | Direction of Encoder Travel (0 or 1) |
| ST | Output | X | | | | | Status of Encoder |
| CT | Output | X | | | | | Current Value Encoder Position |
| IX | Output | X | | | | | Current Value of Idex Counter |
| Q | Output | | | X | | | |

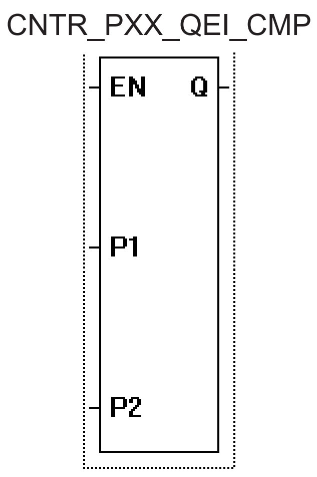
Continued Next Page------»

## Example Circuit:



**Related Functions:** CNTR_PXX_QEI_CMP, CNTR_PXX_QEI_VEL

## CNTR_PXX_QEI_CMP



### Description:

The CNTR_PXX_QEI_CMP function block is used to load values into compare registers (3 available for Position Counter and 3 avaliable for Index Counter). For more details on how the quadrature counter inputs and function blocks operate together, see **Chapter 18 - Counters & Timers**.

When the function block is placed, a dialog box is automatically displayed (See next page). The Compare drop-down menu is used to select the type of compare to use (load and monitor).

The enable (EN) input must be true for the CNTR_PXX_QEI_CMP function block to operate.

The Q Output is based on the type of compare and the actual count value.

The quadrature encoder must be installed in the Project Settings before this function block may be placed. Other parameter configurations must be completed during this installation. Refer to **Chapter 18 - Counters & Timers**.

Three hardware compare registers are provided for the Position Counter compare. These are listed as CMP0, CMP1 and CMP2 in the drop-down menu in the dialog. When a CMPx is selected, the value connected to the P1 input is compared to the Position Counter and the Q is true if these values are equal. As three compare registers are provided, using three different instances of this function block with different CMPx selected will provide three individual compares that may be made to the Position Counter.

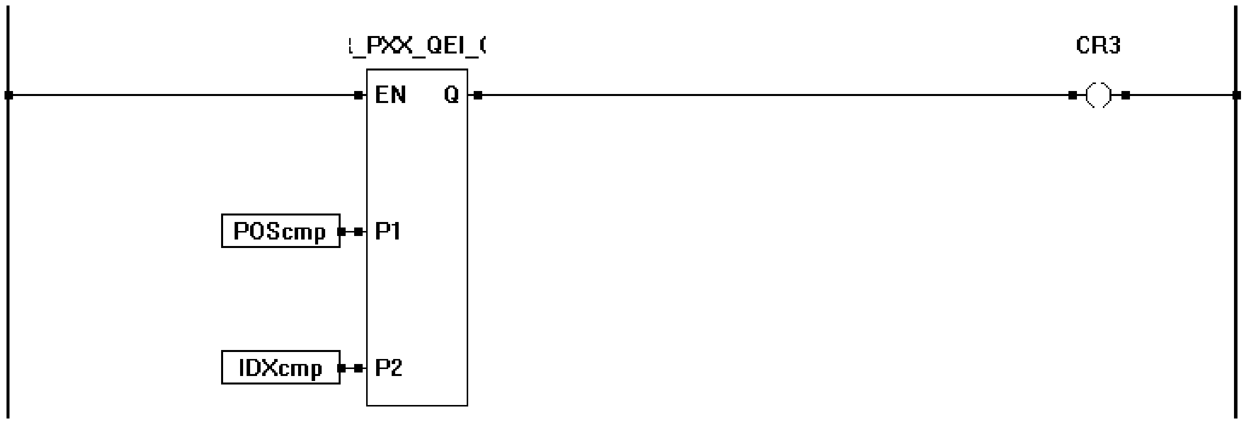
Three hardware compare registers are provided for the Index Counter compare. These are listed as IDX0, IDX1 and IDX2 in the drop-down menu in the dialog. When a IDXx is selected, the value connected to the P1 input is compared to the Index Counter and the Q is true if these values are equal. As three compare registers are provided, using three different instances of this function block with different IDXx selected will provide three individual compares that may be made to the Index Counter.

Additionally, these two features (Position Counter Compare and Index Counter Compare) may be combined by this function block. The combination requires the postion counter and index counter each equal their respective Px inputs to the function block. These items are listed as CMP_IDX0, CMP_IDX1, CMP_IDX2. When using the CMP_IDXx item, the P1 input is the compare value for the Position Counter and the P2 input is the compare value for the Index Counter. The Q Output is true when both the Position Counter equals the P1 input and the Index Counter equals the P2 input. As three compare registers are provided, using three different instances of this function block with different CMP_IDXx selected will provide three individual compares that may be made to the both the Position and Index Counters.

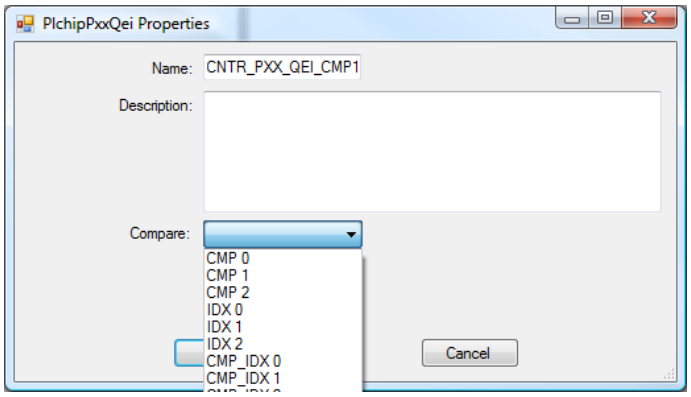
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | Compare Value for Position Counter |
| P2 | Input | X | | | | | Compare Value for Index Counter |
| Q | Output | | | X | | | True when Compare values are true |

Continued Next Page------»

## Example Circuit:



## Dialog Box:



Name: CNTR_PXX_QEI_CMP1

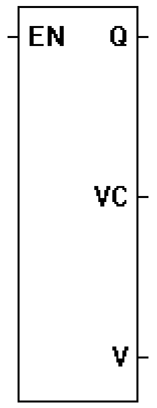
Description:

Compare:

**Related Functions:** CNTR_PXX_QEI, CNTR_PXX_QEI_VEL

## CNTR_PXX_QEI_VEL

CNTR_PXX_QEI_VEL



### Description:

The CNTR_PXX_QEI_VEL function block is used to convert quadrature input counts into a velocity based on engineering units.

When the function block is placed, a dialog box is automatically displayed (See next page). The Sample Period in seconds and the Pulses per Revolution is set using the dialog box.

The quadrature encoder must be installed in the Project Settings before this function block may be placed. Other parameter configurations must be completed during this installation. Refer to **Chapter 18 - Counters & Timers**.

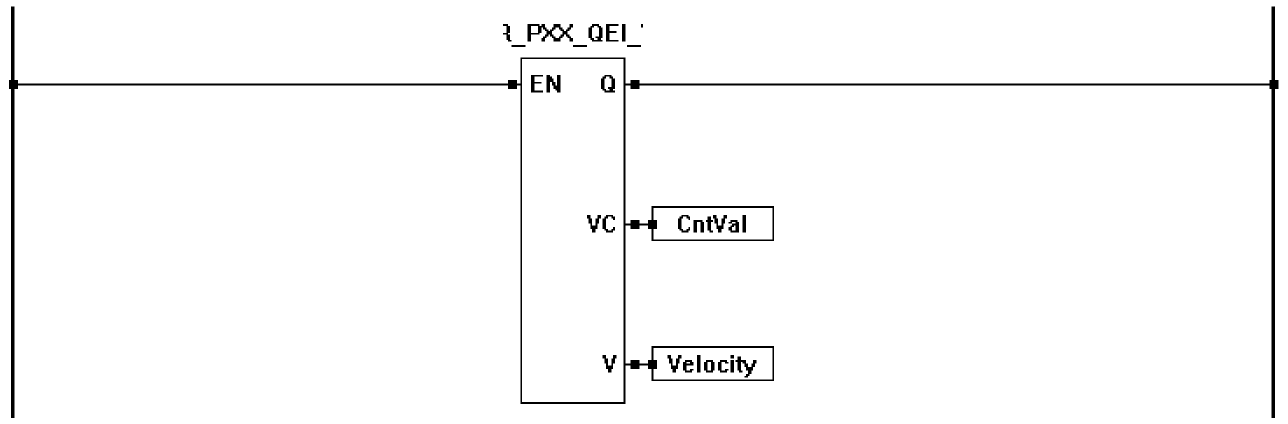
The enable (EN) input must be true for the CNTR_PXX_QEI_VEL function block to operate. The function block counts pulses on the quadrature inputs over the time set by the Sample Period in seconds. It uses the pulses per revolution, the Sample Period and the actual counts to calculate the velocity.

The actual velocity (in engineering units) is output on the V output. The VC output is the number of pulses counted in the sample period. The Q output is true only for the ladder diagram *scan* when the velocity is calculated and updated on the V output.

If the Pulses per Revolution is 1, then the V output is equal to the counter frequency. If not set to 1, then the V output will be revolutions per second.

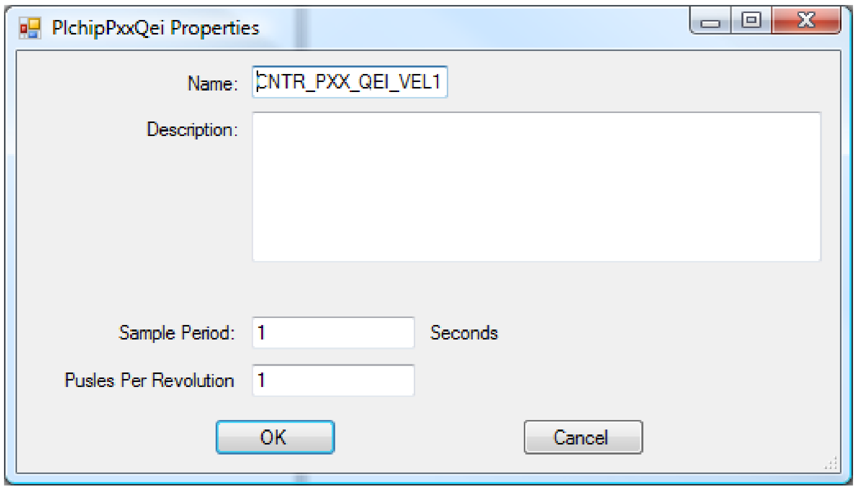
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| VC | Output | X | | | | | Number of Pulses in Sample Period |
| V | Output | | X | | | | Velocity in Revolutions per second |
| Q | Output | | | X | | | True for Scan when Velocity is updated |

### Example Circuit:



Continued Next Page------»

## Dialog Box:



Name: CNTR_PXX_QEI_VEL1

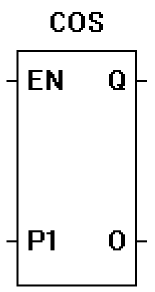
Description:

Sample Period: 1 Seconds

Pusles Per Revolution 1

**Related Functions:** CNTR_PXX_QEI, CNTR_PXX_QEI_CMP

## COS



### Description:

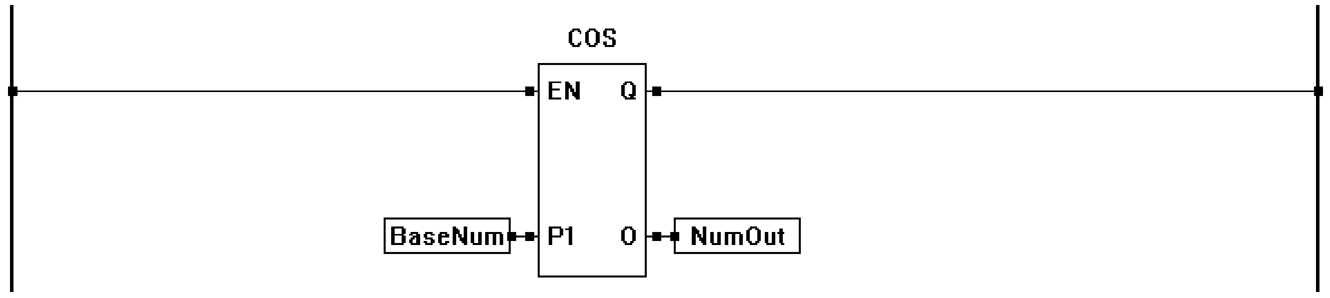
The COS function provides a cosine (O) from the input value (P1). The enable (EN) must be true for the COS function to be enabled. The Q output is true when the COS function is enabled.

### Input / Output Connections:

The COS function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

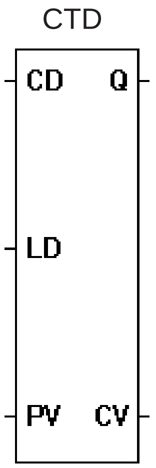
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Cosine Output of P1 Base Number |

### Example Circuit:



**Related Functions:** ACOS, ASIN, ATAN, SIN, TAN

## CTD



### Description:

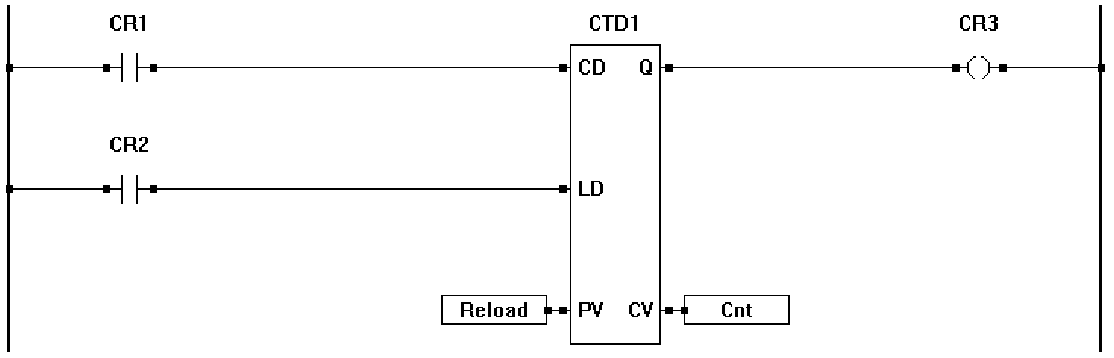
The CTD function is a programmable software down counter. A true on CD will cause the counter to decrement by one. Once the counter (CV) equals zero, the Q output will be true. A true on LD will cause the counter to load the PV as the current (CV) count and reset the Q output. The down counter triggers on a false to true transition on the CD input.

### Input / Output Connections:

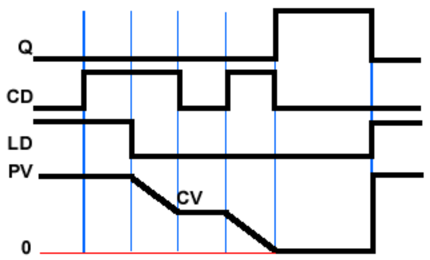
The CTD function block placement requires connections of three input pins (CD, LD, PV) and two output pins (Q,CV).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| CD | Input | | | X | | Rising Edge | |
| LD | Input | | | X | | | |
| PV | Input | X | | | | | |
| Q | Output | | | X | | | True when CV=0 |
| CV | Output | X | | | | | |

### Example Circuit:

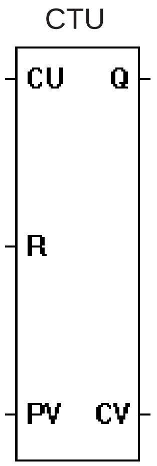


### Timing Diagram:



### Related Functions: CTU, CTUD

## CTU



### Description:

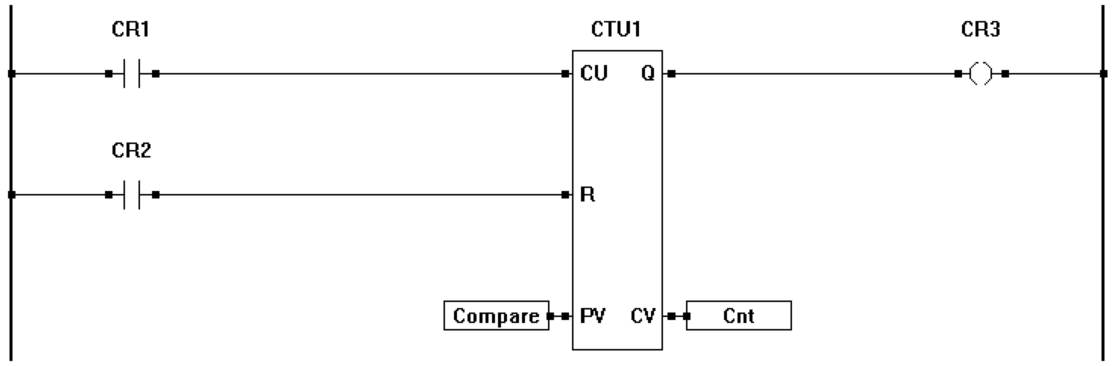
The CTU function is a programmable software up counter. A true on CU will cause the counter to increment by one. Once the counter (CV) equals the preset value (PV), the Q output will be true. A true on reset (R) will cause the counter reset to zero and reset the Q output. The down counter triggers on a false to true transition on the CU input.

### Input / Output Connections:

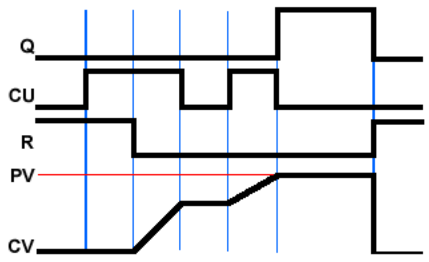
The CTU function block placement requires connections of three input pins (CU, R, PV) and two output pins (Q,CV).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| CU | Input | | | X | | Rising Edge | |
| R | Input | | | X | | | |
| PV | Input | X | | | | | |
| Q | Output | | | X | | | True when CV=PV |
| CV | Output | X | | | | | |

### Example Circuit:

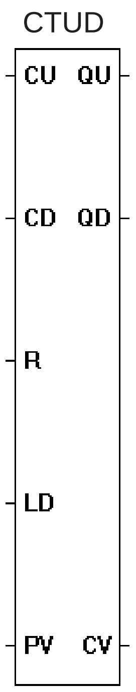


### Timing Diagram:



### Related Functions: CTD, CTUD

## CTUD



### Description:

The CTUD function is programmable software up and down counter. This counter is a combination of the CTU and CTD; therefore, it can count up and down based on the count inputs as well as the Reset and Load inputs.

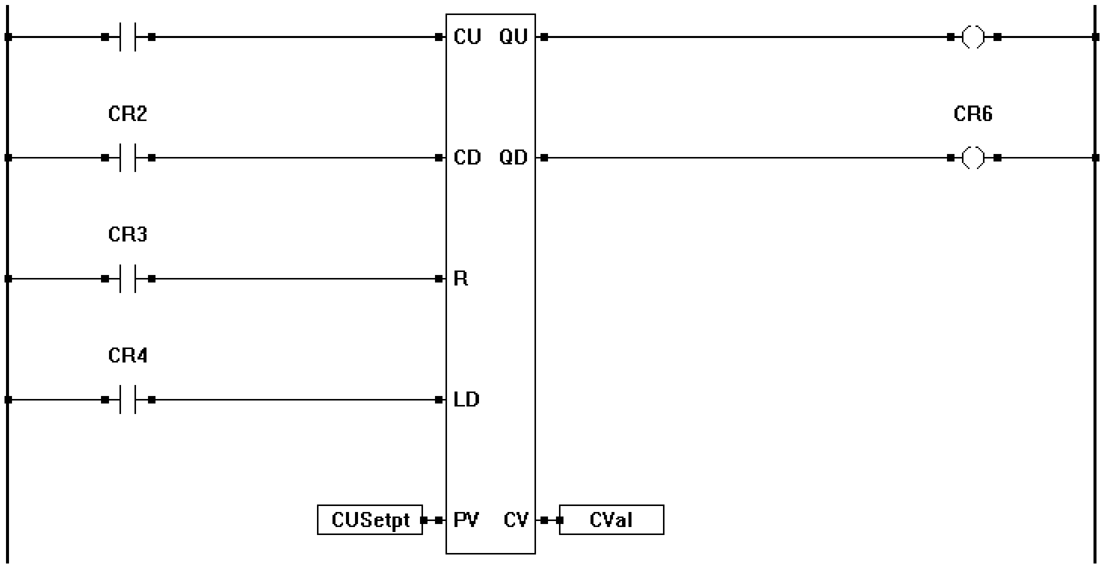
With reset (R) not active, a true on input (CU) will increment the current (CV) count by one, while a true on input (CD) will cause the current count (CV) to decrement by one. When the CV = PV, the output (QU) will be true. When the CV = 0, the output (QD) will be true. A true on the reset (R) will cause CV = 0, QU to go false and QD to go true. A true on the load (LD) will cause CV = PV, QU to go true and QD to go false. The reset (R) is dominant and takes priority over all inputs. The counter inputs trigger on a false to true transition on CU or CD.

### Input / Output Connections:

The CTUD function block placement requires connections of five input pins (CU, CD, R, LD, PV) and three output pins (QU, QD, CV).

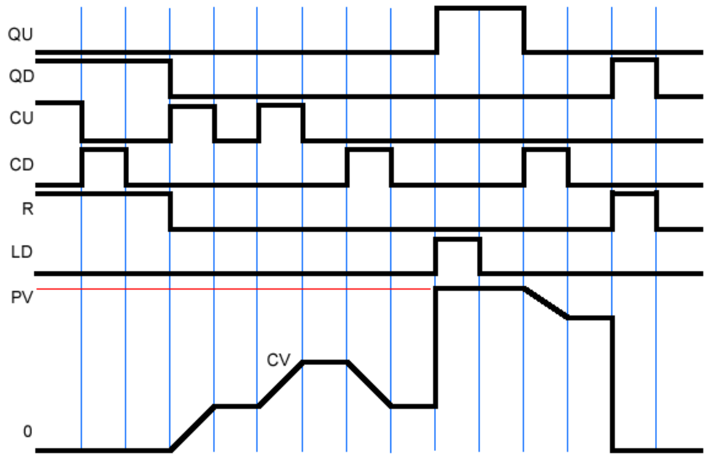
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| CU | Input | | | X | | Rising Edge | |
| R | Input | | | X | | | Reset is dominant |
| CD | Input | | | X | | Rising Edge | |
| LD | Input | | | X | | | |
| PV | Input | X | | | | | |
| QU | Output | | | X | | | True when CV=PV |
| QD | Output | | | X | | | True when CV=0 |
| CV | Output | X | | | | | |

### Example Circuit:



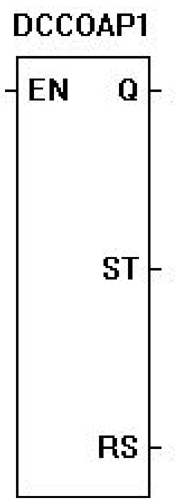
Continued Next Page------»

## Timing Diagram:



**Related Functions:** CTU, CTD

## DCCoAP



### Description:

The **DCCoAP** (formerly VCLOUD) is used to transmit data (variables) to and receive data (variables) from a DCCoAP Cloud server (formerly VersaCloud M2M cloud portal). This function can be configured to use Ethernet, Wi-Fi and Cellular (based on available target hardware) as the communications method to the cloud. The function block is rising edge sensitive to allow for control of transmit and receive.

As multiple **DCCoAP** function blocks operate, data may be queued for transmit and receive.

### Input / Output Connections:

The **DCCoAP** function block has 1 input port pin (EN) and 3 output port pins (Q, ST, RS).

**EN:** The EN (enable) pin enables the function block. When a rising edge on EN is seen the function block attempts to communicate to (transmit and receive data) to the DCCoAP cloud server based on configuration items that are chosen when the function block is placed in the ladder diagram. This should be connected to a boolean contact.

**Q:** The Q output pin is true when the ENable input pin is true. This should be connected to a boolean coil.

**ST:** The ST output pin provides the current function blocks status. This should be connected to an Integer variable. The status codes are as follows:

| | |
|---|---|
| 3 = Completed | 0 = Not Used |
| 2 = Sending | -1 = Error |
| 1 = Queued to Send | |

**RS:** The RS output pin is the Response Code provided by the DCCoAP cloud server during communications. This should be connected to an Integer variable.

If the **DCCoAP** function block encounters an error, it will return one of these error codes on the RS pin. The common codes are shown in red.

| Error | Error Codes | Description |
|---|---|---|
| None | 0 | No Error |
| LWIP | -1 | Contact Divelbiss Support for this error. |
| No Connection | -3 | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| Already Activated | -4 | Device has already been activated on the DCCoAP Cloud server. Contact Divelbiss support for more information. |
| Activation Failed | -5 | The activation for this device failed on the DCCoAP Cloud server. Contact Divelbiss support for more information. |
| Null Pointer | -6 | Contact Divelbiss Support for this error. |
| Invalid String | -7 | Contact Divelbiss Support for this error. |
| Queue is Full | -8 | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP Cloud server. |
| Not Idle | -9 | Contact Divelbiss Support for this error. |
| Other Error | -10 | Other unspecified error. Contact Divelbiss Support for this error. |

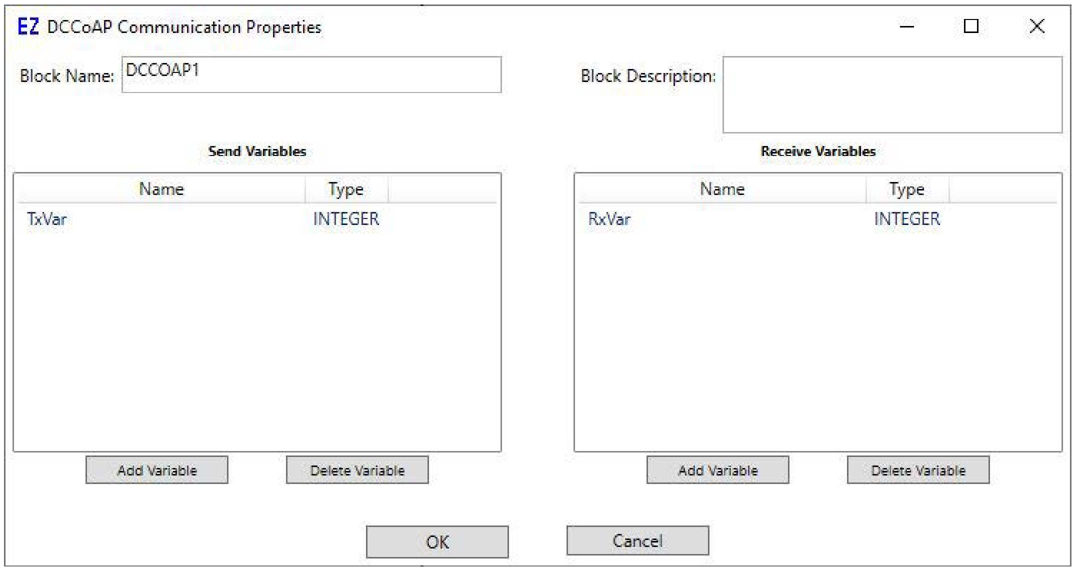
Continued Next Page------»

| | | |
|---|---|---|
| Not Activated | -11 | The device is trying to communicate to the DCCoAP Cloud server but has not been activated on the server. |
| Time-out | -20 | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

When the **DCCoAP** function block has completed without an error, it will return one of the responses on the RS pin. This code returned is built with 8 total bits. These bits are divided with the 3 upper bits being the response code before the '.' shown below and the 5 lower bits representing the two digits after the '.' shown below. A conversion would be necessary to use this response code. Typical response codes would be 69 or 129 (shown in red). Contact Divelbiss support for addtional codes.

| Response | Converted Code | Description |
|---|---|---|
| 65 | 2.01 | Created |
| 68 | 2.04 | Changed |
| 69 | 2.05 | Content was received. |
| 95 | 2.31 | Continue |
| 129 | 4.01 | Unauthorized - The CIK couldn't be used to authenticate. |
| 130 | 4.02 | Bad Option |
| 131 | 4.03 | Forbidden |
| 132 | 4.04 | Not Found |
| 136 | 4.08 | Request Entity Incomplete |
| 140 | 4.12 | Precondition Failed |

When placing the **DCCoAP** function block, the ***DCCoAP Communications Properties*** window will automatically open. This dialog is where the variables to transmit to the cloud and the variables to be received from the cloud server are specifed.



Block Name: DCCOAP1

Send Variables

| Name | Type |
|---|---|
| TxVar | INTEGER |

Receive Variables

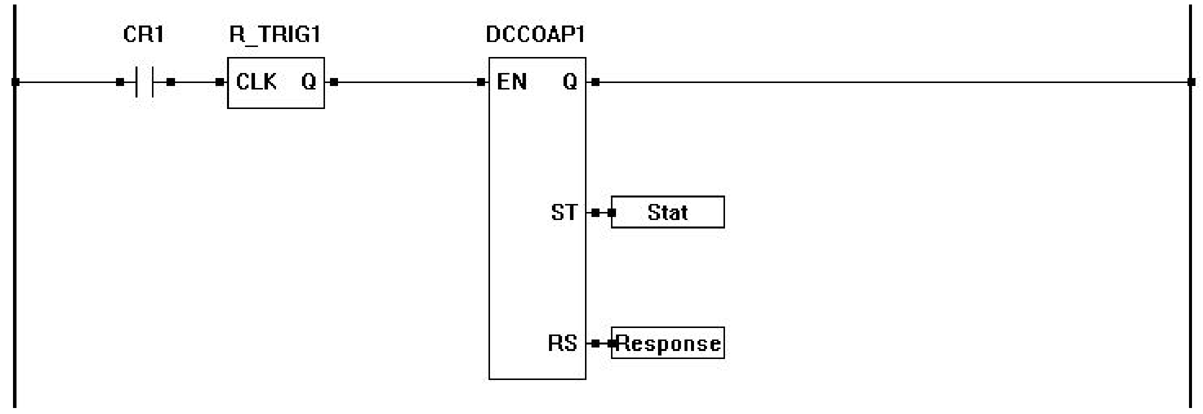
| Name | Type |
|---|---|
| RxVar | INTEGER |

The block is divided into two sections, the Send Variables and the Receive Variables. Variables may be added to either side (integer or real) by using the side's **ADD VARIABLE** button as well as removed from either side by using the side's **DELETE VARIABLE** button. Variables to be added in this dialog must already exist in the ladder diagram project.

Continued Next Page------»

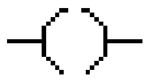
It is good practice to keep variable names as short as possible as the variable names can affect the transmit and receive data packet sizes. Shorter names will reduce packet size which can be important depending upon the communications method such as cellular where charges are based on data.

## Example Circuit:



## DIRECT COIL

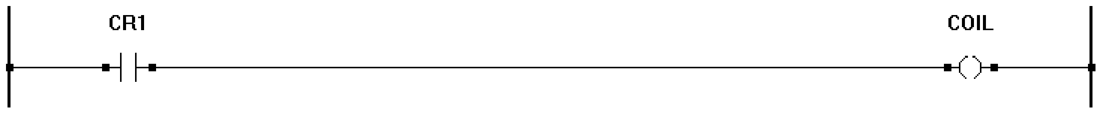
DIRECT COIL



### Description:

The DIRECT COIL is a representation of an internal boolean variable output (coil) or an actual hardware (real world) output. Its normal state is false or normally de-energized. An internal DIRECT COIL may also be referred to as a *control relay* (CR). If there is *power flow* to the DIRECT COIL, then it will be true (on). If there is no *power flow* to the DIRECT COIL, then it will be false (off). The DIRECT COIL may only be placed in the last column.

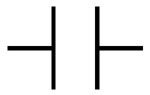
### Example Circuit:



**Related Functions:** INVERTED COIL, DIRECT CONTACT, INVERTED CONTACT

## DIRECT CONTACT

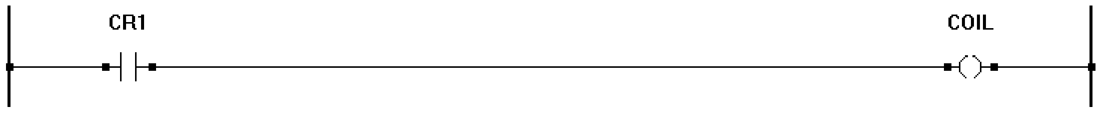
DIRECT CONTACT



### Description:

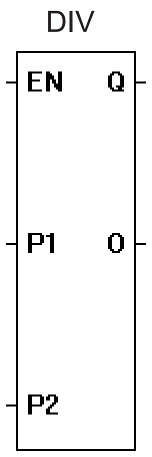
The DIRECT CONTACT is a representation of an internal boolean variable input or an actual hardware (real world) input. Its normal state is false or normally de-energized. An internal DIRECT CONTACT may also be referred to as a *control relay* (CR). A true condition on the input (if internal coil is true for internal contacts or real world input is true), then the contact will allow *power flow* and devices located to the right of the DIRECT CONTACT may operate.

### Example Circuit:



**Related Functions:** DIRECT COIL, INVERTED COIL, INVERTED CONTACT

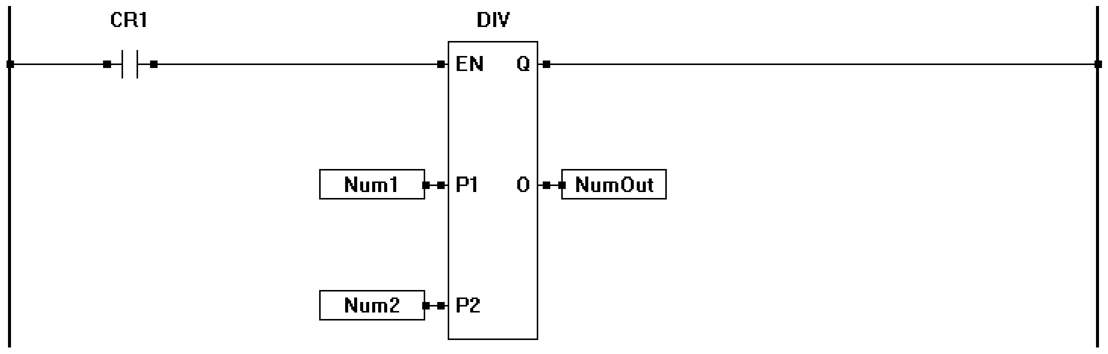
## DIV



### Description:
The DIV function divides the P1 input by the P2 input and outputs the result (O). The enable (EN) must be true for the DIV function to be enabled. The Q output is true when the DIV function is enabled. The result (O) is the whole number quotient only. No remainder is provided.

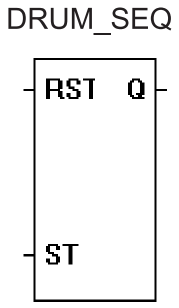
### Input / Output Connections:
The DIV function block placement requires connections of three input pins (EN, P1, P2) and two output pins (Q,O).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | X | | | | |
| P2 | Input | X | X | | | | |
| Q | Output | | | X | | | |
| O | Output | X | X | | | | O=P1/P2 |

### Example Circuit:


**Related Functions:** ADD, SUB, MULT

## DRUM_SEQ



### Description:

The DRUM_SEQ function is comprised of a matrix table of steps (rows of table) and the channels (columns of table). For each channel, a boolean variable (to be used as a contact) is automatically created. A DRUM_SEQ always starts in step 1. Each false to true transition on the ST input will cause the step to increment to the next. The DRUM_SEQ will wrap to step 1 after the last step. A true on RST will reset the DRUM_SEQ to step 1. RST is dominant and will not allow the DRUM_SEQ to step when true.

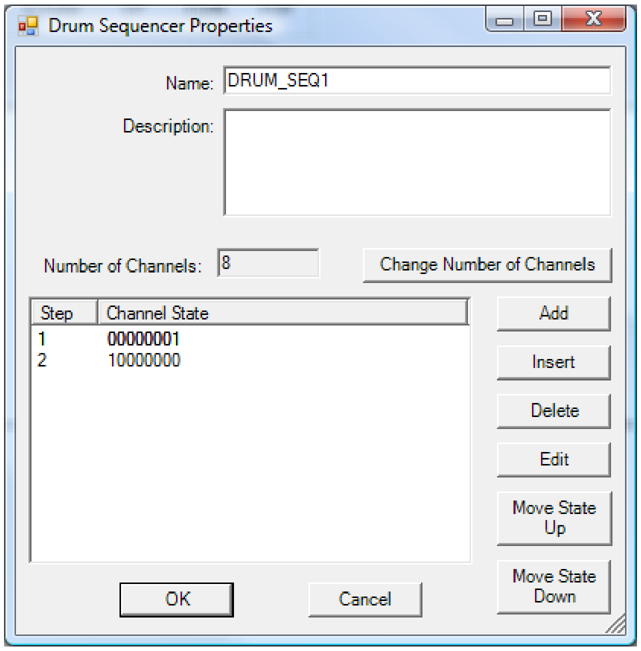
Each step stores a unique setting for each channel. This setting can be set as on or off (true, false). As a contact is created to represent each channel, when a DRUM_SEQ changes steps, each channel is automatically set to the state the channel in that step.

### Input / Output Connections:

The DRUM_SEQ function block placement requires connections of two input pins (RST, ST) and one output pin (Q). In addition, a boolean variable to be used as a contact is created for each channel. A maximum of 32 channels is permitted per DRUM_SEQ. The matrix table is completed when the function is placed.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| RST | Input | | | X | | | RST is dominant |
| ST | Input | | | X | | | |
| Q | Output | | | X | | | |

Configuring the number of channels, setting channel states and adding steps is handled using the DRUM Sequencer Properties dialog box. This box is displayed when placing a DRUM_SEQ function block. Use the buttons provided to add, insert, delete and edit steps. The order of steps may also be changed.


Drum Sequencer Properties

Name: DRUM_SEQ1

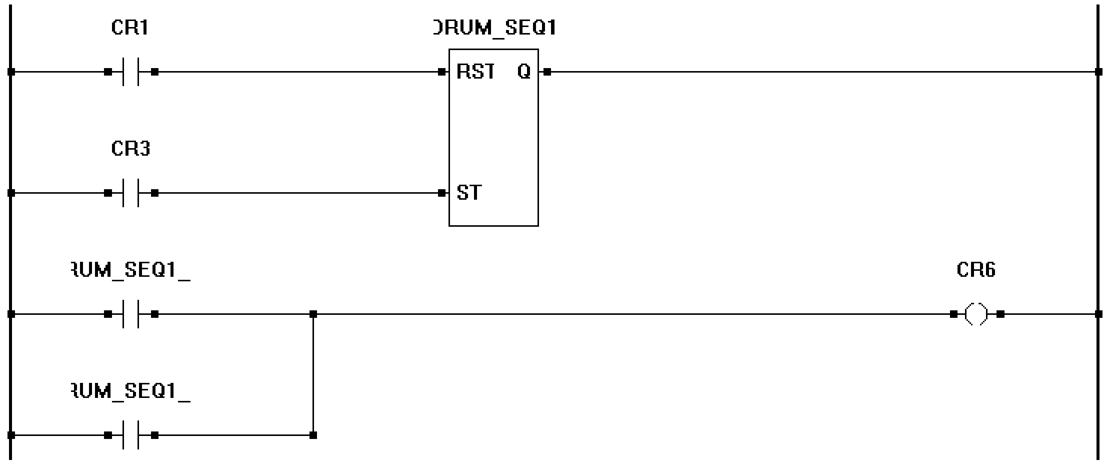
Description:

Number of Channels: 8

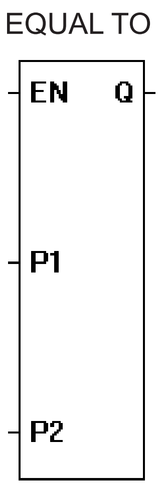
| Step | Channel State |
|---|---|
| 1 | 00000001 |
| 2 | 10000000 |

Continued Next Page------»

## Example Circuit:



## EQUAL TO (=)



### Description:

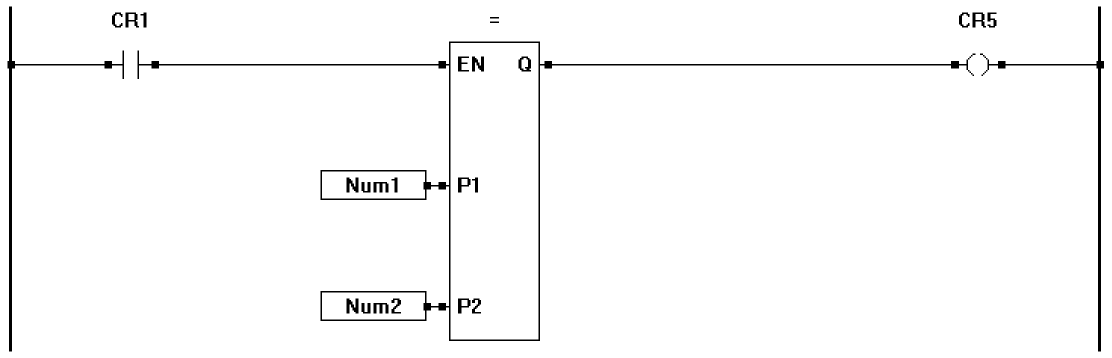
The EQUAL TO function provides an equal to comparison for the Px inputs. The number of inputs is specified when the object is placed. The Q output is true if all the Px inputs are equal. The Enable must be true for the EQUAL TO function to be enabled.

### Input / Output Connections:

The EQUAL TO function block placement requires connections of at least three input pins (EN, P1, P2) and one output pin (Q). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

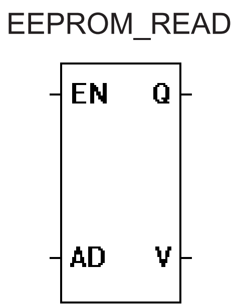
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | True when all P**x** are equal |

### Example Circuit:



**Related Functions:** <>, <, >, <=, >=

## EEPROM_READ



### Description:

The EEPROM_READ recalls variables  stored in non-volatile memory (EEPROM).  The function is enabled when a false to true is seen on EN.  AD provides the actual address to read from and V is the actual value that is read from the EEPROM.  Q is true when the read cycle has completed.

The same variable type that writes to the EEPROM location should be used to read the EEPROM location. A memory map is recommended for organizing variables stored in EEPROM.

Each EEPROM address is absolute and is one byte in size.  Boolean variables fill two bytes while all other variable types fill four bytes of EEPROM. When reading variables from EEPROM storage, it is important that use the exact address location for the variable only (taking into account variable types and sizes).

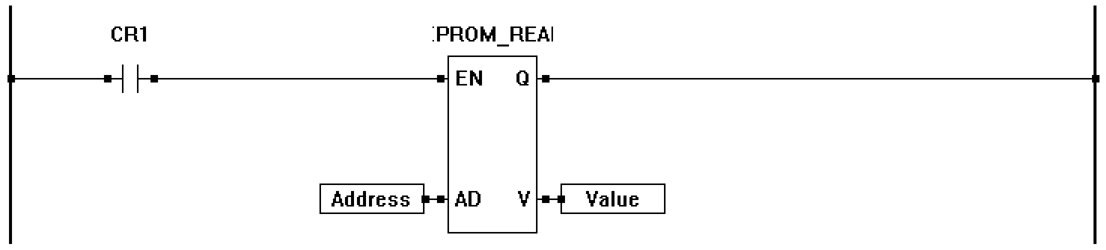
See EEPROM_WRITE for more on how variables are written to EEPROM storage.

### Input / Output Connections:

The EEPROM_READ function block placement requires connections of two input pins (EN, AD) and two output pins (Q, V).

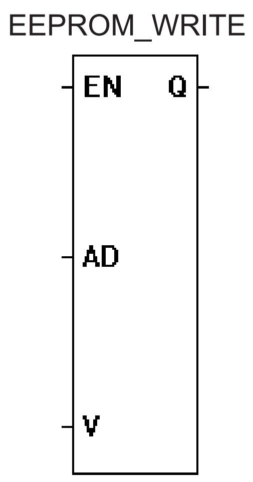
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Rising Edge | |
| AD | Input | X | | | | | |
| Q | Output | | | X | | | |
| V | Output | X | X | X | X | | |

### Example Circuit:



**Related Functions:** EEPROM_WRITE

## EEPROM_WRITE



### Description:

The EEPROM_WRITE function allows variables to be stored in non-volatile memory (EEPROM). The function is enabled when the EN sees a false to true transition. AD provides the actual address to write to EEPROM and V is the actual value that is written. Q is true when the write cycle has completed without error.

The same variable type that writes to the EEPROM location should be used to read the EEPROM location. A memory map is recommended for organizing variables stored in EEPROM.

Writing to EEPROM is a relatively slow operation and this must be considered when creating the ladder diagram project as scan time can be affected during a write.

EEPROM storage area has a limited number of write cycles; therefore it shouldn't be used to store data which changes often and must be re-written often. Writing often to the same location can cause the location to fail.

### Input / Output Connections:

The EEPROM_WRITE function block placement requires connections of three input pins (EN, AD, V) and one output pin (Q).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Rising Edge | |
| AD | Input | X | | | | | |
| V | Input | X | X | X | X | | |
| Q | Output | | | X | | | |

Each EEPROM address is absolute and is one byte in size. Boolean variables fill two bytes while all other variable types fill four bytes of EEPROM. When writing a boolean to address 0, the actual variable will use addresses 0 and 1 (two bytes). Should you write an integer variable into address 0, then it would use addresses 0-3. A memory map should be created and used to assign variable types and addresses prior to coding to ensure that variable size and types are accounted for.

**Variable 1 Address - Boolean (2 bytes) uses location 0 and 1.**

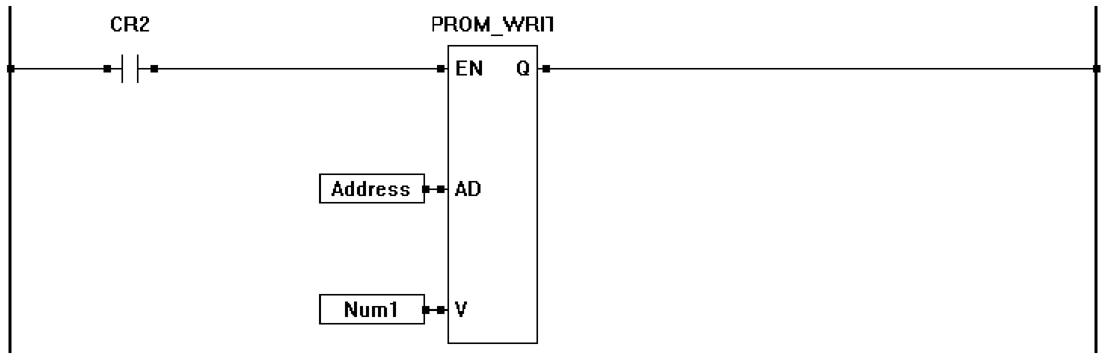
**Variable 2 Address - Integer (4 bytes) uses location 2,3,4 and 5.**

**Variable 3 Address - Boolean (2 bytes) uses location 6 and 7.**

| | EEPROM ADDRESS LOCATION | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Variable & Type** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| Variable 1 (Boolean) | ■ | ■ | | | | | | | | |
| Variable 2 (Integer) | | | ■ | ■ | ■ | ■ | | | | |
| Variable 3 (Boolean) | | | | | | | ■ | ■ | | |

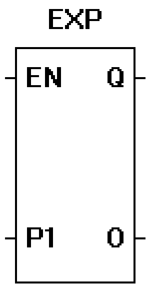
Continued Next Page------»

## Example Circuit:



**Related Functions:** EEPROM_READ

## EXP



### Description:

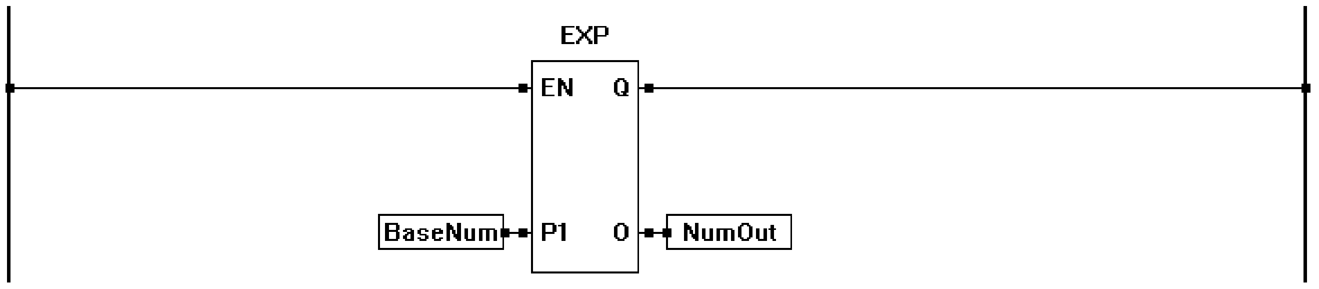
The EXP function provides the natural exponential of the P1 input. The output (O) is the natural exponential of the P1 input. The enable (EN) must be true for the EXP function to be enabled.

### Input / Output Connections:

The EXP function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

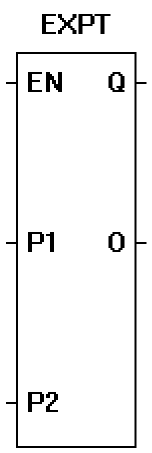
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Natural Exponential of P1 |

### Example Circuit:



**Related Functions:** EXPT, LN, SQRT. LOG

## EXPT

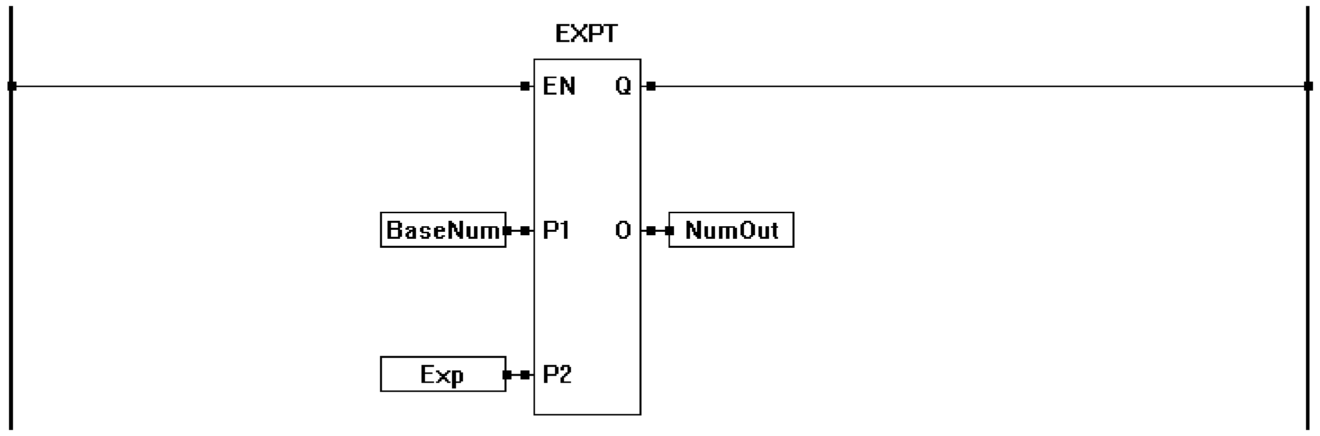


### Description:
The EXPT function provides the exponentiation of the P1 and P2 inputs. The output (O) is the is the result of the exponentiation (P1P2). The enable (EN) must be true for the EXPT function to be enabled.

### Input / Output Connections:
The EXPT function block placement requires connections of three input pins (EN, P1, P2) and two output pins (Q, O).

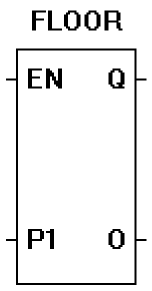
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| P2 | Input | | X | | | | Exponent Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Result of Exponentiation |

### Example Circuit:



**Related Functions:** EXP, LN, SQRT, LOG

## FLOOR



### Description:

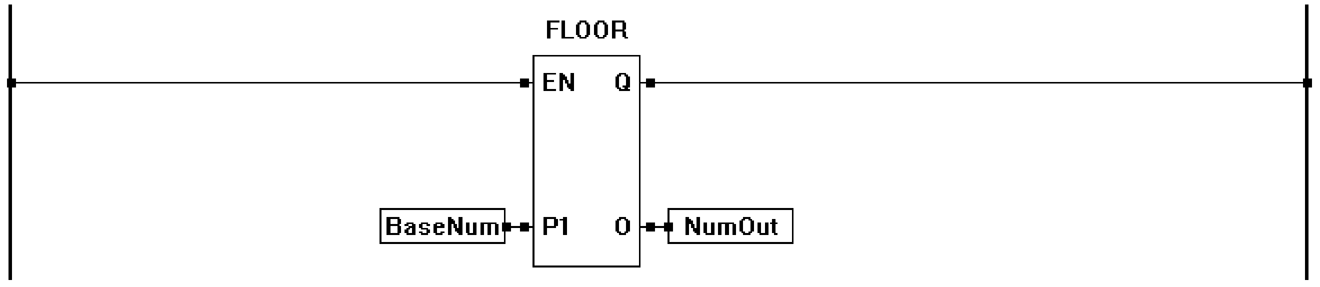
The FLOOR function provides a rouned-down output of P1 input. The output (O) is the is the rounded-down number. The enable (EN) must be true for the FLOOR function to be enabled.

### Input / Output Connections:

The FLOOR function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

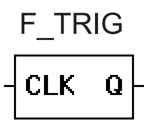
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Rounded Down P1 Base Number |

### Example Circuit:



**Related Functions:** CEIL

## F_TRIG



### Description:

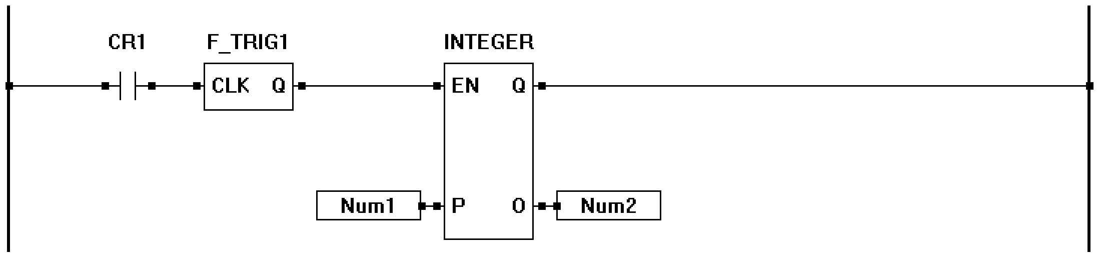
The F_TRIG is a function that may be used to trigger another function on the falling edge of a transition. When the CLK detects a true to false transition, the output (Q) is energized for one scan of the program only.

### Input / Output Connections:

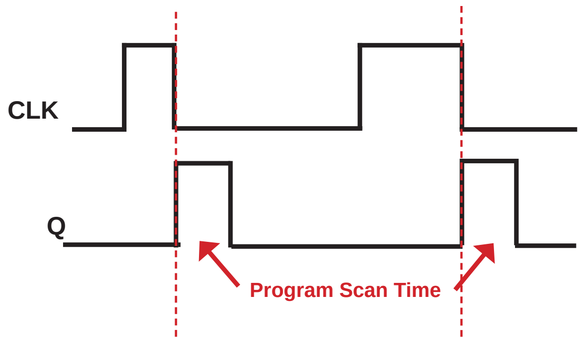
The F_TRIG function block placement requires connections of one input pin (CLK) and one output pin (Q).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| CLK | Input | | | X | | Falling Edge | |
| Q | Output | | | X | | | True for only one scan |

### Example Circuit:

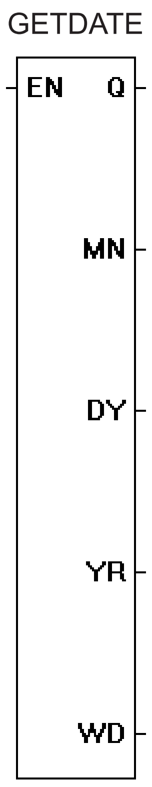


### Timing Diagram:



**Related Functions:** R_TRIG

## GETDATE



### Description:

The GETDATE function reads the current date from the hardware real time clock. The values of the date are stored into the integer variables on the output pins. The enable (EN) must be true for the GETDATE function to be enabled. The Q output is true when the function is enabled. The MN output returns the month (1-12), the DY output returns the day of the month (1-31) , the YR output returns the current year (last two digits) and the WD returns the day of the week (1-7, 1=Sunday). The MN, DY, YR and WD outputs must be connected to Integer variables.

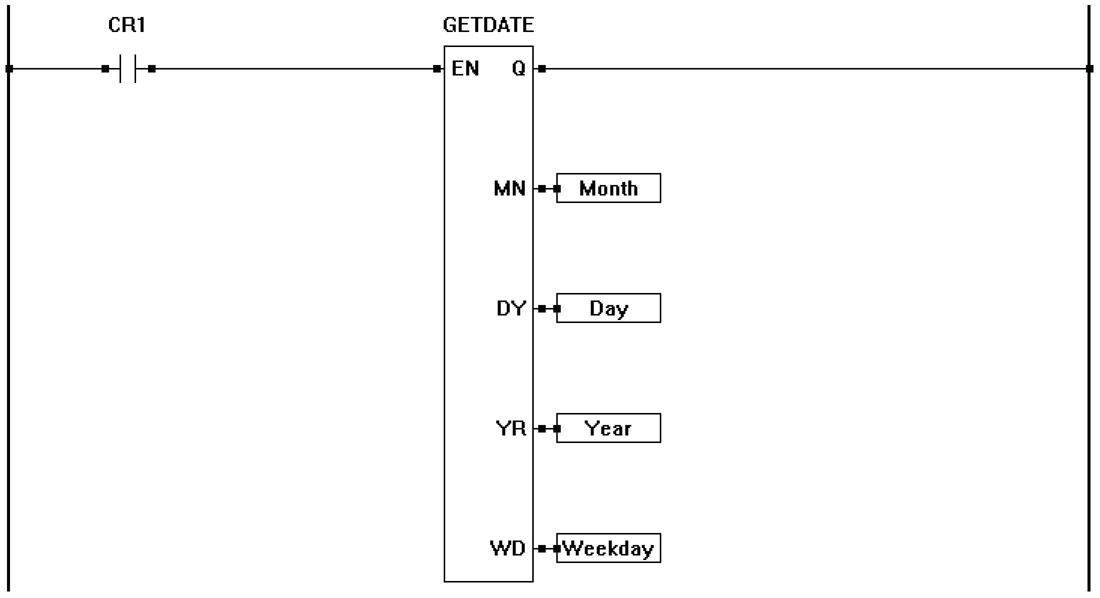
The GETDATE function will return what ever the real time clock returns. Typically these values are for local date. If the real time clock is using UTC time, then the values will be different (not local time).

### Input / Output Connections:

The GETDATE function block placement requires connections of one input pin (EN) and five output pins (Q, MN, DY, YR, WD).

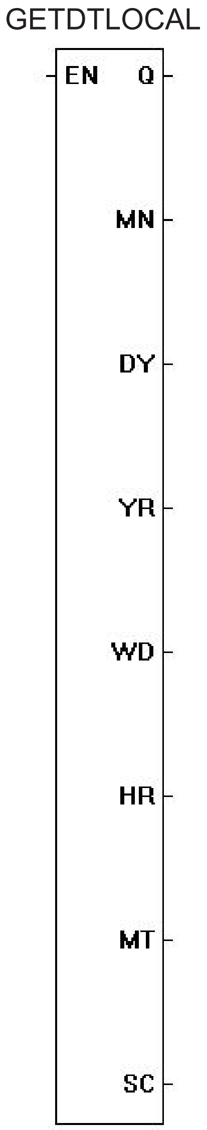
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| Q | Output | | | X | | |
| MN | Output | X | | | | |
| DY | Output | X | | | | |
| YR | Output | X | | | | |
| WD | Output | X | | | | |

### Example Circuit:



**Related Functions:** GETTIME, SETTIME, SETDATE, SETDTLOCAL, GETDTLOCAL, SETTZOFF

## GETDTLOCAL



### Description:
The GETDTLOCAL function block reads the current date and time from the hardware real time clock (stored as UTC time). The UTC time is converted using the time zone offset set by the SETTZOFF function block and returns the local date and time.

The values of the date and time are stored into the integer variables on the output pins. The enable (EN) must be true for the GETDTLOCAL function to be enabled. The Q output is true when the function is enabled. The MN output returns the month (1-12), the DY output returns the day of the month (1-31), the YR output returns the current year (either two for four digits based on how the real time clock was set) and the WD returns the day of the week (1-7, 1=Sunday).  The MN, DY, YR and WD outputs must be connected to Integer variables. The HR output returns the hours (0-23), the MT output returns the minutes (0-59) and the SC output returns the seconds (0-59) The HR, MT and SC outputs must be connected to Integer variables.

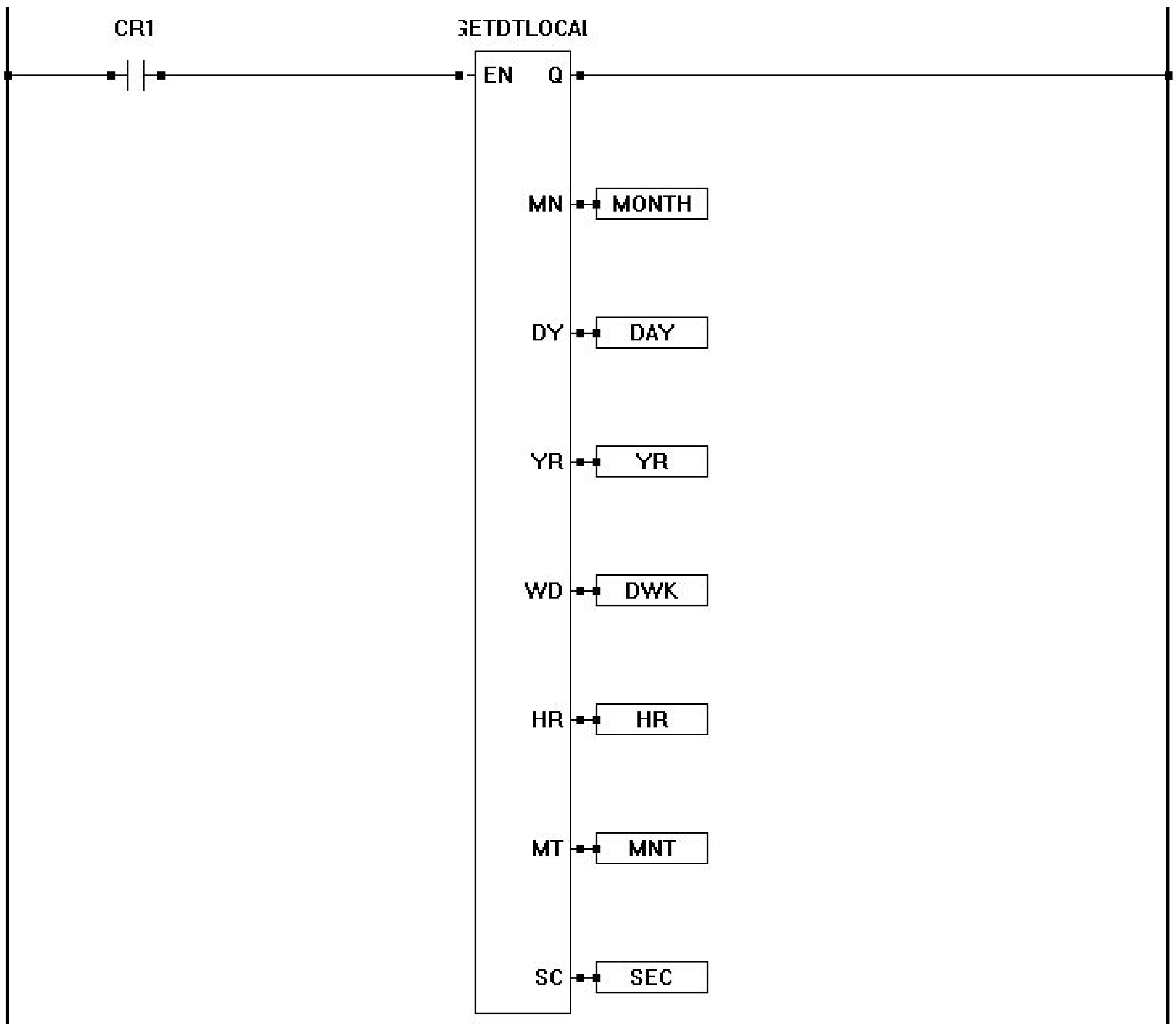
The SETTZOFF function block must be ran each time the hardware target is started (power cycled) to set the time zone offset with it's value. This offset is not stored or kept in the controller in the event of a power loss or power cycle. Failure to run the SETTZOFF function block will result in incorrect time/date when using the SETDTLOCAL and GETDTLOCAL function blocks.

### Input / Output Connections:
The GETDTLOCAL function block placement requires connections of one input pin (EN) and eight output pins (Q, MN, DY, YR, WD, HR, MT, SC).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| Q | Output | | | X | | |
| MN | Output | X | | | | |
| DY | Output | X | | | | |
| YR | Output | X | | | | |
| WD | Output | X | | | | |
| HR | Output | X | | | | |
| MT | Output | X | | | | |
| SC | Output | X | | | | |

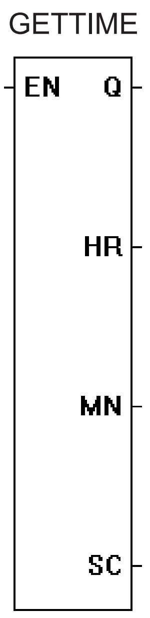
Continued Next Page------»

## Example Circuit:



**Related Functions:** GETTIME, SETTIME, SETDATE, SETDTLOCAL, SETTZOFF

## GETTIME



### Description:

The GETTIME function reads the current time from the hardware real time clock. The values of the time are stored into the integer variables on the output pins. The enable (EN) must be true for the GETTIME function to be enabled.

The Q output is true when the function is enabled. The HR output returns the hour of the day (0-23) , the MN output returns the minutes and the SC returns the seconds. The HR, MN and SEC outputs must be connected to Integer variables.

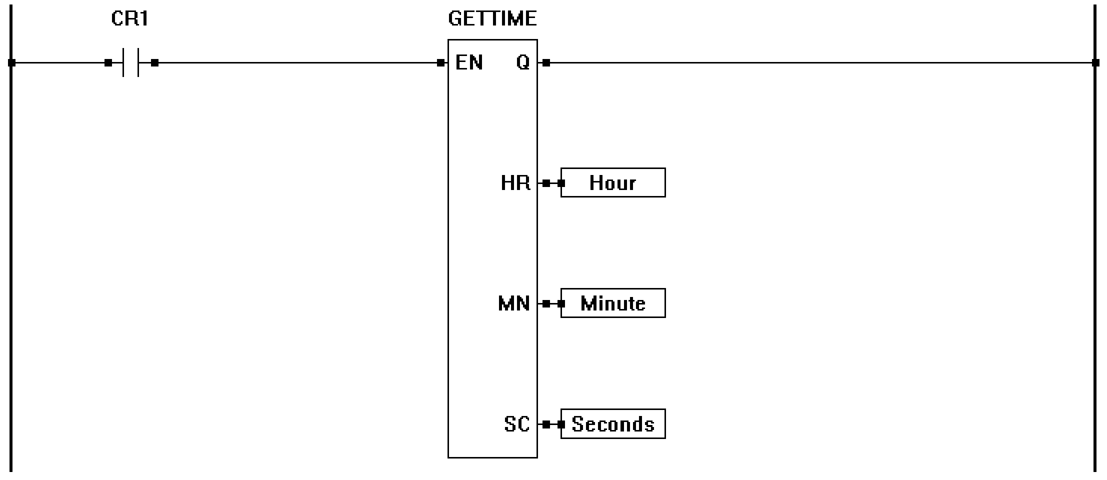
The GETTIME function will return what ever the real time clock returns. Typically these values are for local time. If the real time clock is using UTC time, then the values will be different (not local time).

### Input / Output Connections:

The GETTIME function block placement requires connections of one input pin (EN) and four output pins (Q, HR, MN, SEC).

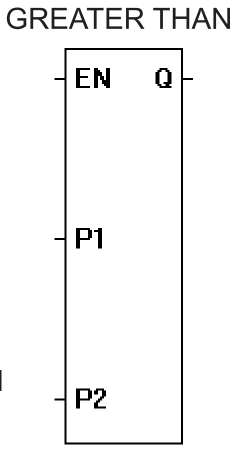
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| Q | Output | | | X | | |
| HR | Output | X | | | | |
| MN | Output | X | | | | |
| SC | Output | X | | | | |

### Example Circuit:



**Related Functions:** GETDATE, SETTIME, SETDATE, SETDTLOCAL, GETDTLOCAL, SETTZOFF

## GREATER THAN (>)



### Description:

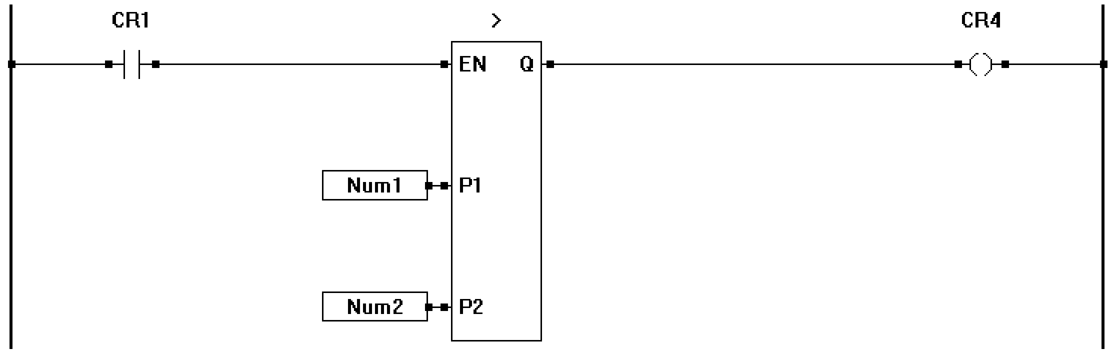
The GREATER THAN provides an if greater than comparison for the P*x* inputs. The number of inputs is specified when the object is placed. The output (Q) is true if P1 is greater than P2 and P2 is greater than P3 and so on. The enable (EN) must be true for the GREATER THAN function to be enabled.

### Input / Output Connections:

The GREATER THAN function block placement requires connections of at least 3 input pins (EN, P1, P2) and one output pin (Q). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |

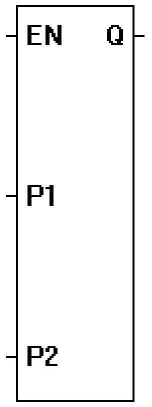
### Example Circuit:



**Related Functions:** >, <, <=, <>, =

## GREATER THAN OR EQUAL TO (>=)

GREATER THAN
OR EQUAL TO



### Description:

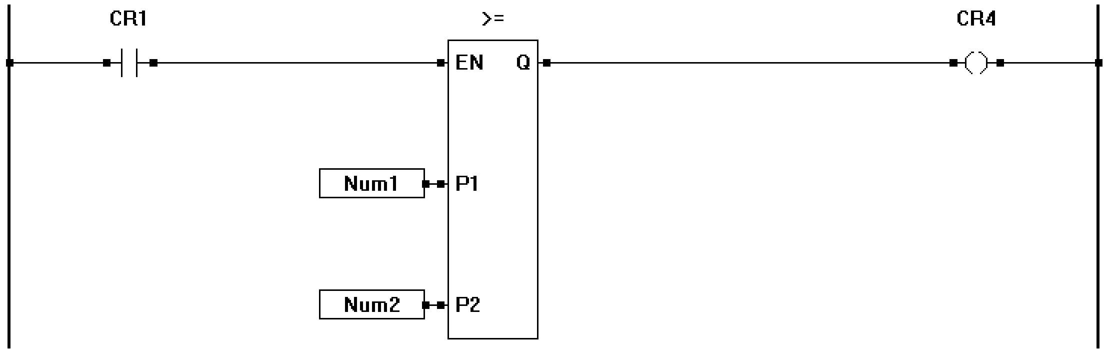
The GREATER THAN OR EQUAL TO provides an if greater than or equal to comparison for the Px inputs. The number of inputs is specified when the object is placed. The output (Q) is true if P1 is greater than or equal to P2 and P2 is greater than or equal to P3 and so on. The enable (EN) must be true for the GREATER THAN OR EQUAL TO function to be enabled.

### Input / Output Connections:

The GREATER THAN OR EQUAL TO function block placement requires connections of at least 3 input pins (EN, P1, P2) and one output pin (Q). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

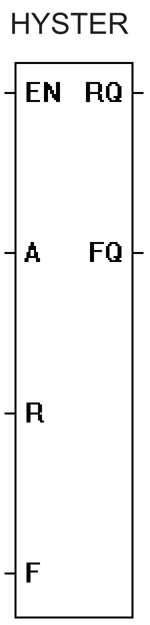
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** >, < , <=, <>, =

## HYSTER



### Description:

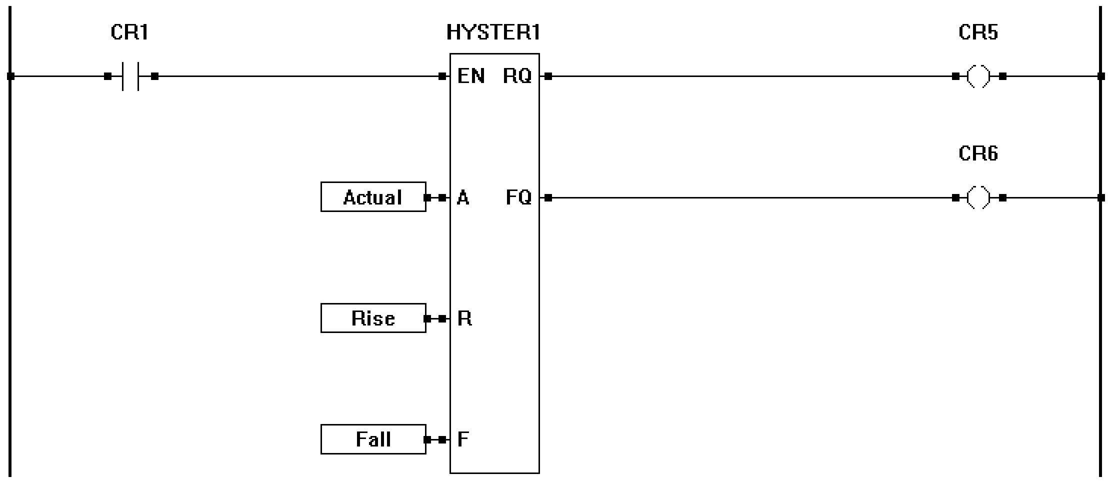
The HYSTER provides hysteresis into a control loop. When the actual (A) is greater than the rise (R), then output RQ is true and FQ is false. When actual (A) is less than fall (F), the output FQ is true and RQ is false. The enable (EN) must be true for the HYSTER function to be enabled.

### Input / Output Connections:

The HYSTER function block placement requires connections of four input pins (EN, A, R, F) and two output pins (RQ, FQ).

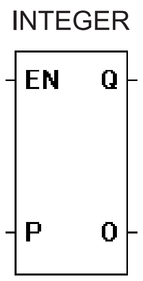
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| A | Input | | X | | | | |
| R | Input | | X | | | | |
| F | Input | | X | | | | |
| RQ | Output | | | X | | | |
| FQ | Output | | | X | | | |

### Example Circuit:



**Related Functions:** LIMIT

## INTEGER



### Description:

The INTEGER function converts the input (P) into an integer output (O). The enable (EN) must be true for the INTEGER function to be enabled. The Q output is true when the INTEGER function is enabled.

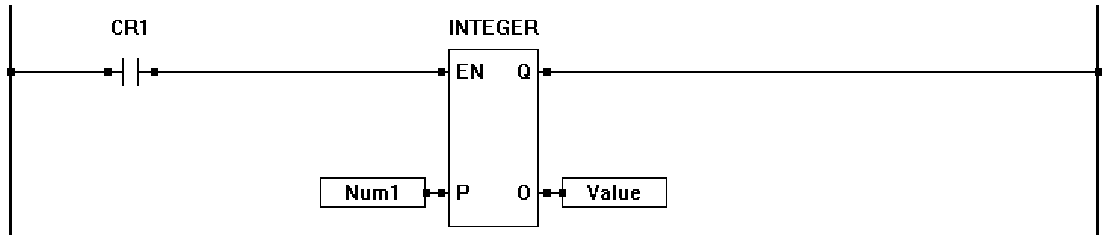
In addition to converting a Boolean, Timer or Real to an integer, the INTEGER function block can be used to copy one integer to another.

### Input / Output Connections:

The INTEGER function block placement requires connections of two input pins (EN, P) and two output pins (Q, O).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | X | X | X | | |
| O | | X | | | | | |
| Q | Output | | | X | | | |

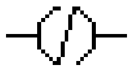
### Example Circuit:



**Related Functions:** REAL, BOOLEAN, TIMER

## INVERTED COIL

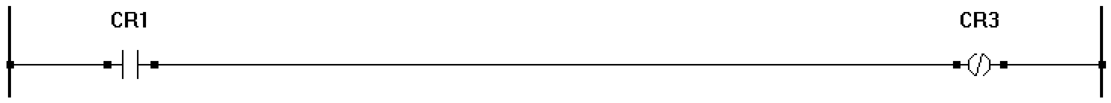
INVERTED COIL



### Description:

The INVERTED COIL is a representation of an internal boolean variable output (coil) or an actual hardware (real world) output. Its normal state is true or normally energized. An internal INVERTED COIL may also be referred to as a *control relay* (CR). If there is *power flow* to the INVERTED COIL, then it will be false (off). If there is no *power flow* to the INVERTED COIL, then it will be true (on). The INVERTED COIL may only be placed in the last column.

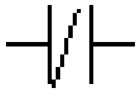
### Example Circuit:



**Related Functions:** DIRECT COIL, DIRECT CONTACT, INVERTED CONTACT

## INVERTED CONTACT

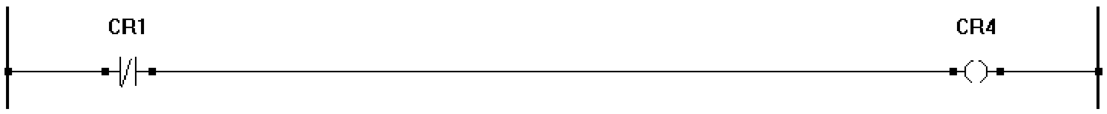
INVERTED CONTACT



### Description:

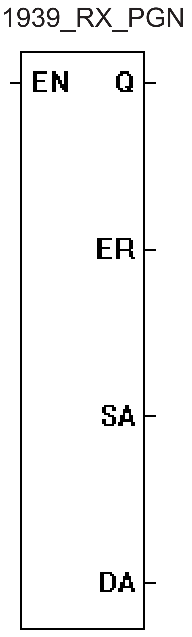
The INVERTED CONTACT is a representation of an internal boolean variable input or an actual hardware (real world) input. Its normal state is true or normally energized. An internal INVERTED CONTACT may also be referred to as a *control relay* (CR). A false condition on the input (if internal coil is true for internal contacts or real world input is false), then the contact will allow *power flow* and devices located to the right of the DIRECT CONTACT may operate. A true on it's coil or real world input will result in it's contacts to not allow *power flow*.

### Example Circuit:



**Related Functions:** DIRECT CONTACT, DIRECT COIL, INVERTED COIL

## J1939_RX_PGN



### Description:

The J1939_RX_PGN function block is used to receive data over the J1939 / NMEA 2000 CAN network. There are several configurable items for receiving data, refer to **Chapter 14 - CAN Networking (J1939/NMEA 2000).**

The J1939_RX_PGN function block receives data and stores it in variables located in the ladder diagram program. These variables are mapped when the J1939_RX_PGN function block is placed. Each function block insertion (instance) is treated individually in the ladder diagram program.

### Input / Output Connections:

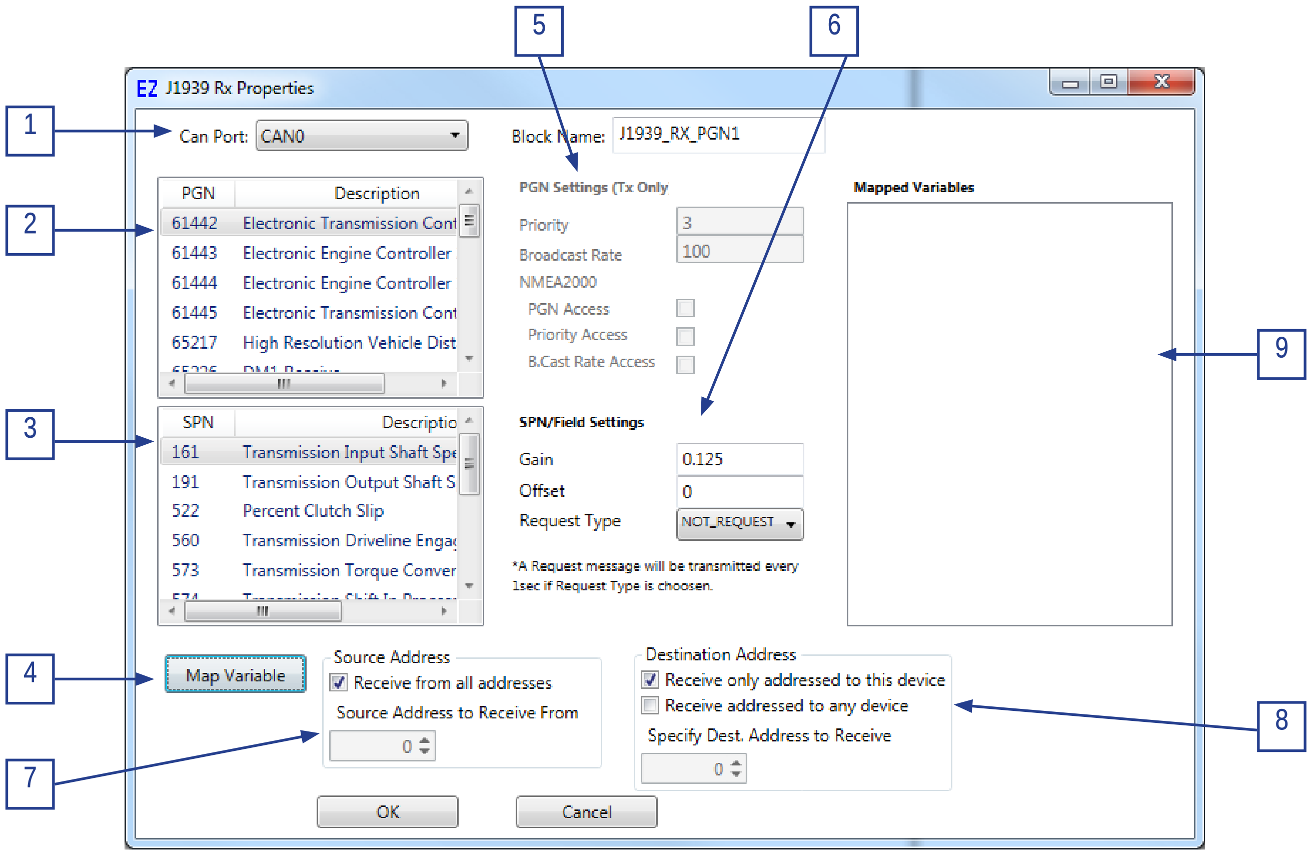
When the **EN** (ENABLE) is true, the function block is active and will receive J1939/NMEA 2000 data as configured. The data is stored in the mapped variables.

The **Q** Output is true for one ladder diagram scan when data is received.

The **ER** (ERROR) output stores the current status of the data received from the PGN.

The **SA** is the Source Address output. It must be connected to an integer variable. This variable will store the Source Address of the last receive (which address this PGN was received from).

The **DA** is the Destination Address output. It must be connected to an integer variable. This variable will store the Destination Address of the last receive if the PGN had a specific destination in the broadcast packet. If the broadcast was a global broadcast, then the DA does not apply.



Can Port: CAN0

Block Name: J1939_RX_PGN1

| PGN | Description |
|---|---|
| 61442 | Electronic Transmission Cont |
| 61443 | Electronic Engine Controller |
| 61444 | Electronic Engine Controller |
| 61445 | Electronic Transmission Cont |
| 65217 | High Resolution Vehicle Dist |

| SPN | Descriptio |
|---|---|
| 161 | Transmission Input Shaft Spe |
| 191 | Transmission Output Shaft S |
| 522 | Percent Clutch Slip |
| 560 | Transmission Driveline Engag |
| 573 | Transmission Torque Conver |

PGN Settings (Tx Only

| | |
|---|---|
| Priority | 3 |
| Broadcast Rate | 100 |
| NMEA2000 | |
| PGN Access | ☐ |
| Priority Access | ☐ |
| B.Cast Rate Access | ☐ |

SPN/Field Settings

| | |
|---|---|
| Gain | 0.125 |
| Offset | 0 |
| Request Type | NOT_REQUEST |

*A Request message will be transmitted every 1sec if Request Type is choosen.

Mapped Variables

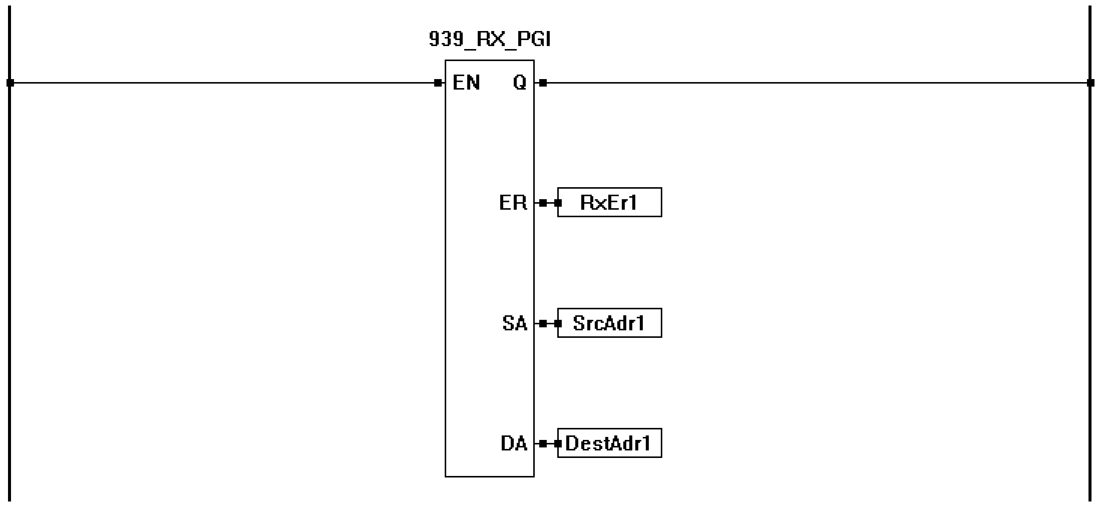
Source Address
☑ Receive from all addresses
Source Address to Receive From: 0

Destination Address
☑ Receive only addressed to this device
☐ Receive addressed to any device
Specify Dest. Address to Receive: 0

Continued Next Page------»

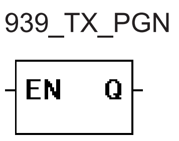
| | |
|---|---|
| 1. CAN Port Select | Drop down box to select the CAN port (network) to be used for this J1939 receive function block. |
| 2. PGN Pane | Select the PGN to receive with this receive function block. Only one PGN per function block is allowed. |
| 3. SPN Pane | Select the SPN(s) to receive and store in variables of the selected PGN. |
| 4. Map Variable Button | Clicking this button opens the map variable dialog to map the selected SPN data to variables. The data variable is required, but the status variable is optional. Double-c licking the SPN in the SPN pane will also open the map variable dialog. |
| 5. PGN Settings | The PGN settings will update as different PGNs are selected. These values are based on the PGN settings in the J1939 database. |
| 6. SPN /Field Settings | The SPN settings will update as different SPNs are selected. These values are based on the SPN settings in the J1939 database. The Gain and Offset may be overridden by entering new values. Values changed will only affect this function block, not the J1939 database. The Request Type may be configured to NOT_REQUEST which is standard or J1939_REQUEST to request a PGN/SPN on J1939 or NMEA_REQUEST to request data on an NMEA 2000 network. |
| 7. Source Address Area | This area determines the allowed source address of the network to receive data from (allowed device ID). Data may be received from all addresses or from specific address. |
| 8. Destination Address Area | This area determines the allowed destination addresses (in the broadcast packet) from which to receive data. Optionally, the receive function block can receive data that is specifically addressed to this device (controller), transmitted to any device or transmitted specifically to a device (but not this one). This allows to receive data that is specifcally transmitted to another device on the network. |
| 9. Mapped Variables Area | As variables are mapped to SPNs, they will be listed in this area. |

## Example Circuit:



**Related Functions:** J1939_TX_PGN

## J1939_TX_PGN



### Description:

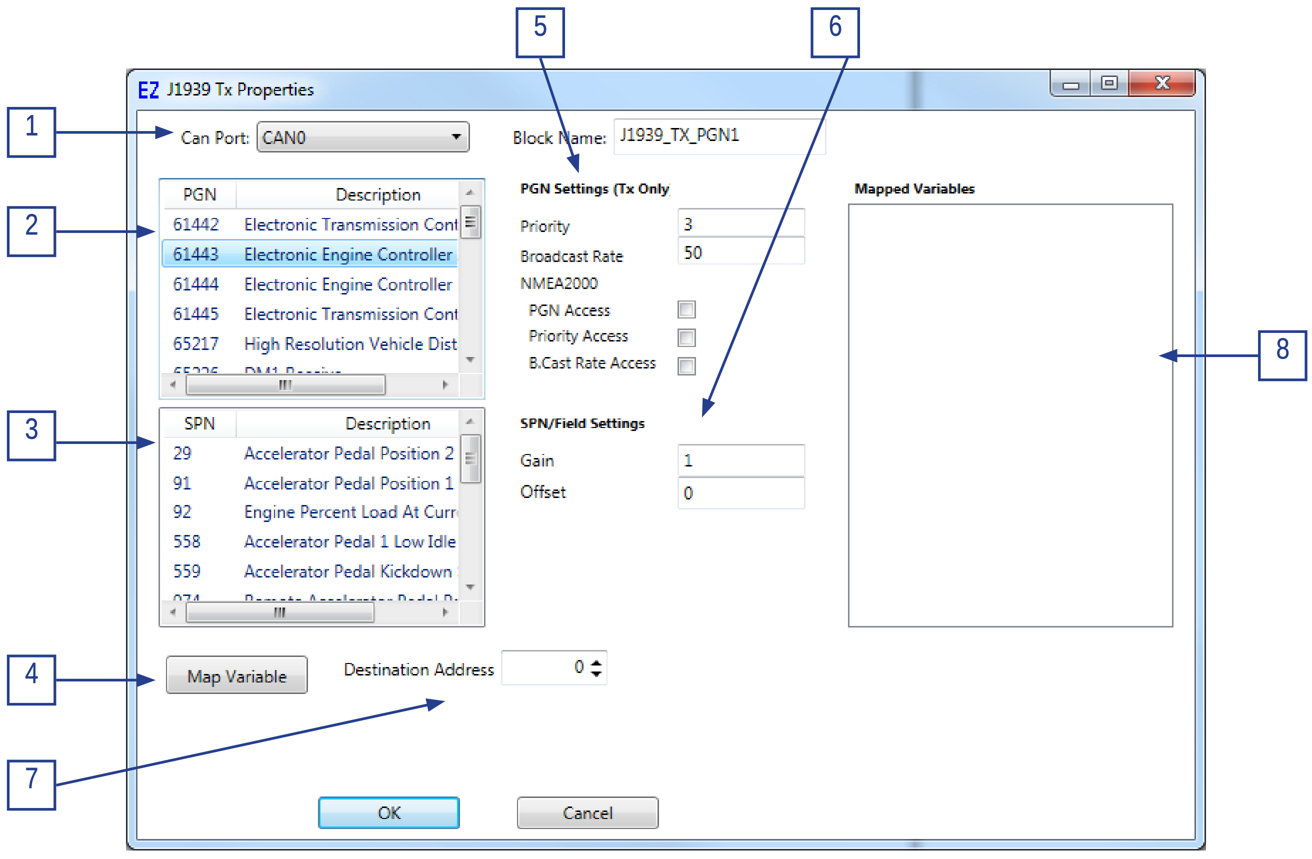
The J1939_TX_PGN function block is used to transmit / broadcast data over the J1939 / NMEA 2000 CAN network. There are several configurable items for transmitting data, refer to **Chapter 14 - CAN Networking (J1939/NMEA 2000).**

The J1939_TX_PGN function block transmits data that is stored in variables located in the ladder diagram program. These variables are mapped when the J1939_TX_PGN function block is placed. Each function block insertion (instance) is treated individually in the ladder diagram program.

### Input / Output Connections:

When the **EN** (ENABLE) is true, the function block is active and will transmit J1939/NMEA 2000 data as configured (based on broadcast rates). The transmitted data is read from the mapped variables.

The **Q** Output is true when the EN is true.



Can Port: CAN0

Block Name: J1939_TX_PGN1

| PGN | Description |
|---|---|
| 61442 | Electronic Transmission Cont |
| 61443 | Electronic Engine Controller |
| 61444 | Electronic Engine Controller |
| 61445 | Electronic Transmission Cont |
| 65217 | High Resolution Vehicle Dist |

| SPN | Description |
|---|---|
| 29 | Accelerator Pedal Position 2 |
| 91 | Accelerator Pedal Position 1 |
| 92 | Engine Percent Load At Curr |
| 558 | Accelerator Pedal 1 Low Idle |
| 559 | Accelerator Pedal Kickdown |

PGN Settings (Tx Only

Priority: 3
Broadcast Rate: 50
NMEA2000
PGN Access ☐
Priority Access ☐
B.Cast Rate Access ☐

SPN/Field Settings

Gain: 1
Offset: 0

Mapped Variables

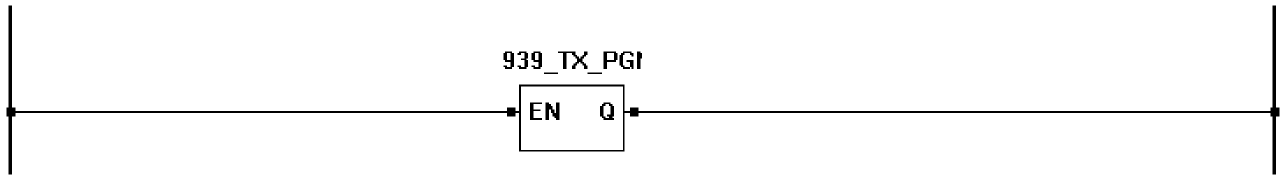
Destination Address: 0

| | |
|---|---|
| 1. CAN Port Select | Drop down box to select the CAN port (network) to be used for this J1939 transmit function block. |
| 2. PGN Pane | Select the PGN to transmit with this receive function block. Only one PGN per function block is allowed. |

Continued Next Page------»

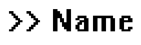
| | |
|---|---|
| 3. SPN Pane | Select the SPN(s) to transmit from variables as the selected PGN / SPN. |
| 4. Map Variable Button | Clicking this button opens the map variable dialog to map the selected SPN to variables storing the data. Double-clicking the SPN in the SPN pane will also open the map variable dialog. |
| 5. PGN Settings | The PGN settings will update as different PGNs are selected. These values are based on the PGN settings in the J1939 database. The Priority and Broadcast rate can be overridden by entering new values. Values changed will only affect this function block, not the J1939 database. For NMEA 2000, optional checkboxes determine specific parameters for NMEA 2000. Refer to the NMEA 2000 specification for details. |
| 6. SPN /Field Settings | The SPN settings will update as different SPNs are selected. These values are based on the SPN settings in the J1939 database. The Gain and Offset may be overridden by entering new values. Values changed will only affect this function block not the J1939 database. |
| 7. Destination Address Area | This area determines the destination address (in the broadcast packet) to send data to. If the destination address is set to 255, then the broadcast will be global. If set to a number other than 255, it is specifically addressed to that ID. The availability of setting the destination address depends on the PGN/SPN selected. Global may the only type supported based on the actual PGN/SPN. |
| 8. Mapped Variables Area | As variables are mapped to SPNs, they will be listed in this area. |

## Example Circuit:



**Related Functions:** J1939_RX_PGN

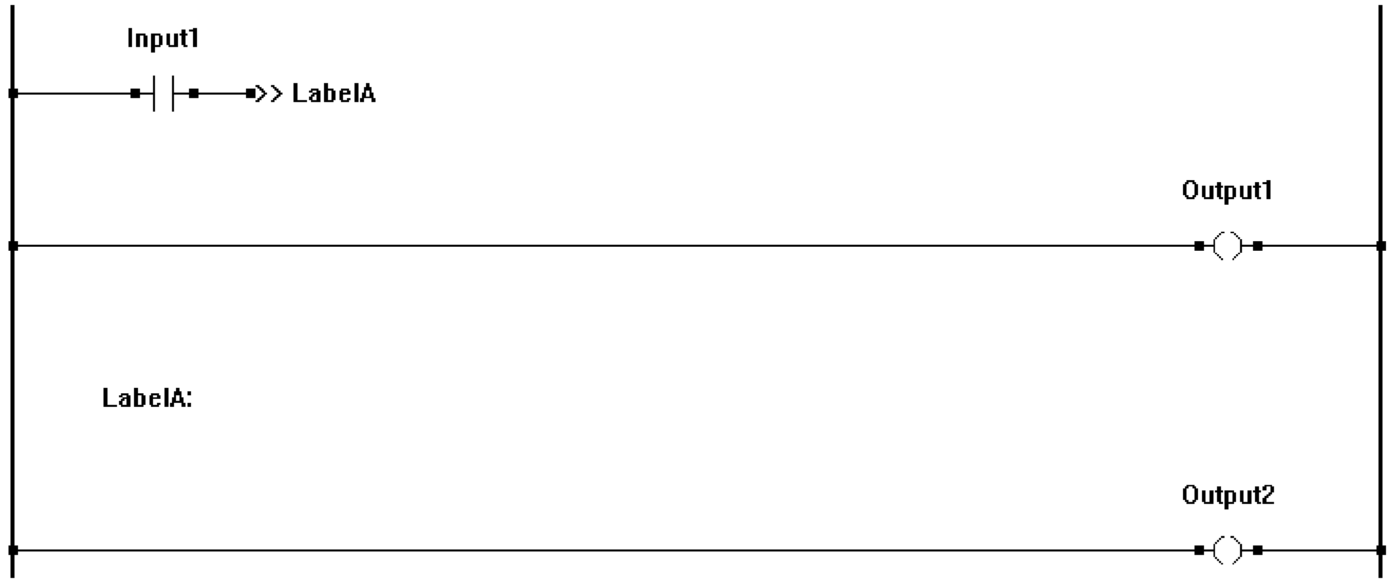
## JMP



### Description:
The JMP function allows sections of ladder to be skipped by “jumping” to another section. A LABEL must first be placed before the JMP is inserted. When the condition is true to trigger the jump, the program scan jumps to the label, skipping any ladder between the jump and label.

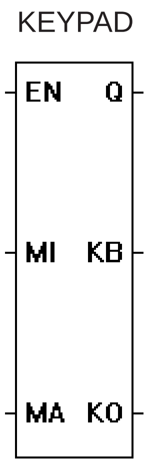
### Input / Output Connections:
There are no Input or Output Connections

### Example Circuit:



**Related Functions:** LABEL

## KEYPAD



### Description:

The KEYPAD function is used to allow users to input data. This function requires the Keypad feature be installed on the target's hardware and target settings.

The keypad may be used in two ways. The first is using the KEYPAD function. This is useful for allowing a user to input numeric data. The second is reading individual button presses as a digital input. This is useful for menus. **See Chapter 10 - Keypad Support** for details on keypad use.

***Using the KEYPAD for Numeric Input (Keypad Function Block)***

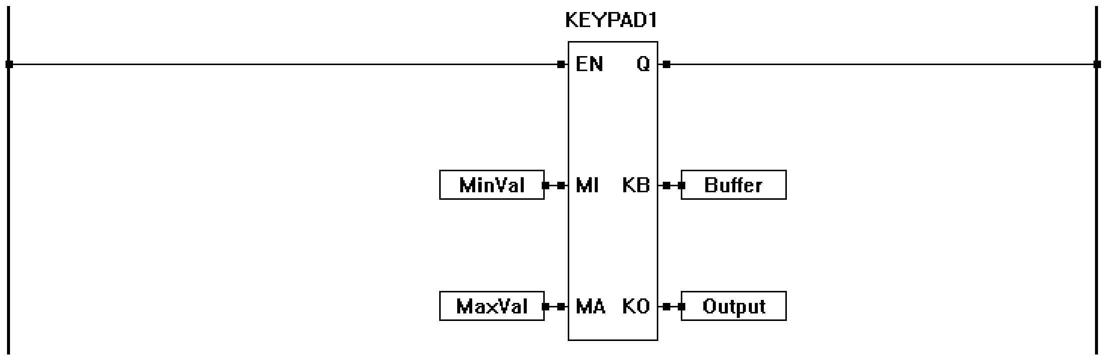
When EN is true, the function is enabled. Data is entered using numeric keypad buttons. These numeric buttons are temporarily stored in the keypad buffer KB. When **ENTER** is pressed, the KB is transferred stored in the variable connected to the output (KO). The output Q is true for the ladder diagram scan in which the **ENTER** was pressed. Pressing the clear button on the keypad erases the buffer (KB). The MI input specifies the minimum value allowed to be entered on the keypad while the MA input specifies the maximum value allow to be entered on the keypad.

### Input / Output Connections:

The KEYPAD function block placement requires connections of three input pins (EN, MI, MA) and three output pins (Q, KB, KO).

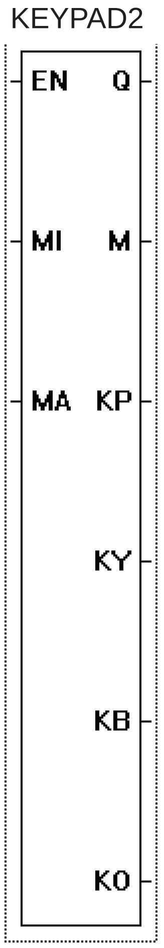
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| MI | Input | X | X | | | | Minimum Allowed Value |
| MA | Input | X | X | | | | Maximum Allowed Value |
| KB | Output | X | X | | | | Keypad Buffer |
| KO | Output | X | X | | | | Keypad Entered Value |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** KEYPAD2

## KEYPAD2



### Description:

The KEYPAD2 function is used to allow users to input data. It operates similar to the KEY-PAD function. This function requires the Keypad feature be installed on the target's hard-ware and target settings. **See Chapter 10 - Keypad Support** for details on keypad use.

The Keypad2 function block provides additional features over the Keypad function block. These featues allow menus and Discrete key press menu items to be combined, allowing for a more powerful and easier to implement menu.

***Using the KEYPAD for Numeric Input (Keypad2 Function Block)***

When EN is true, the function is enabled. Data is entered using numeric keypad buttons. These numeric buttons are temporarily stored in the keypad buffer KB. When **ENTER** is pressed, the KB is transferred stored in the variable connected to the output (KO). The output Q is true for the ladder diagram scan in which the **ENTER** was pressed. Pressing the clear button on the keypad erases the buffer (KB). The MI input specifies the minimum value allowed to be entered on the keypad while the MA input specifies the maximum value allow to be entered on the keypad.

The M output is a boolean that is set to true when any number (0-9), + or . is pressed. Pressing the **ENTER** or **CLEAR** will reset the M output to false. The KP output is a boolean output that is true only for the single scan that a key was pressed. The KY output is an integer output of the actual ASCII value of the key that was pressed.

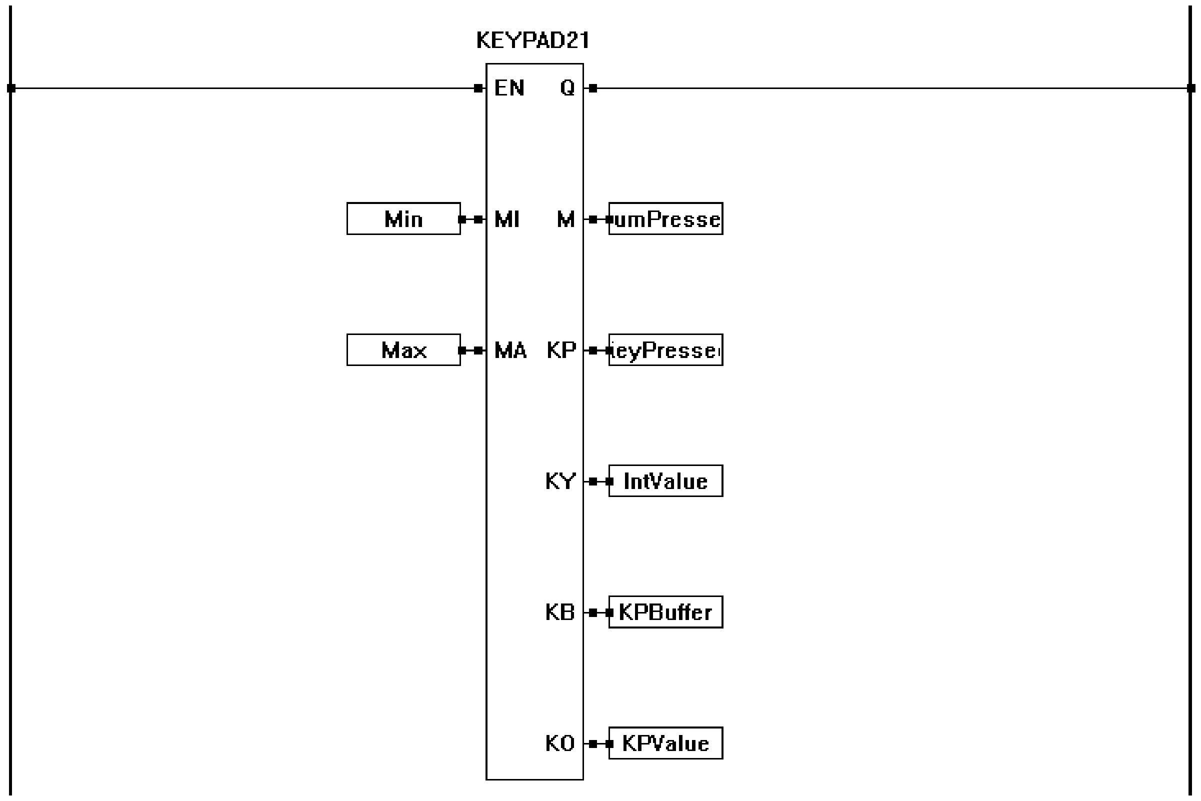
### Input / Output Connections:

The KEYPAD2 function block placement requires connections of three input pins (EN, MI, MA) and six output pins (Q, KB, KO, M, KY, KP).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| MI | Input | X | X | | | | Minimum Allowed Value |
| MA | Input | X | X | | | | Maximum Allowed Value |
| KB | Output | X | X | | | | Keypad Buffer |
| KO | Output | X | X | | | | Keypad Entered Value |
| M | Output | | | X | | | True when 0-9, + or - Pressed |
| KY | Output | X | | | | | Integer Output of Ascii Value Pressed |
| KP | Output | | | X | | True One Scan | True one scan when key is pressed |
| Q | Output | | | X | | | |

Continued Next Page------»

## Example Circuit:



**Related Functions:** KEYPAD

## LABEL

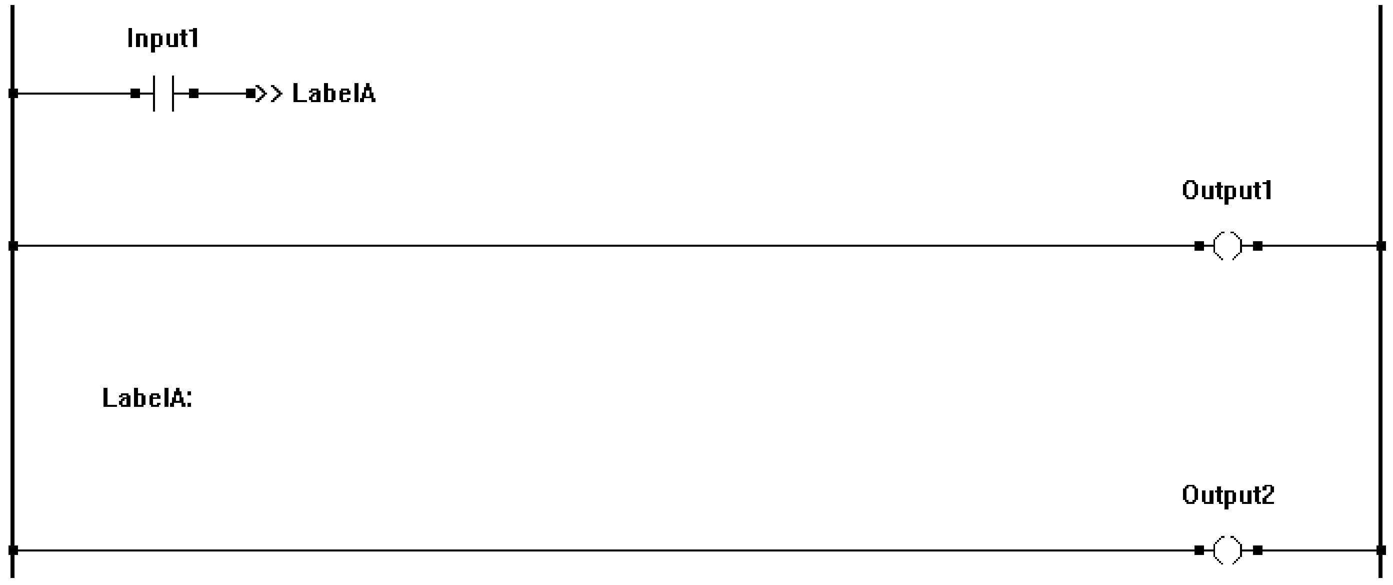
### Description:

The LABEL function works with the JMP function to skip ladder diagram sections.  A LABEL must be placed first, then the JMP inserted.  When the condition is true to trigger the jump, the program scan “jumps” to the LABEL, skipping any ladder between the jump and label.

### Input / Output Connections:

There are no Input or Output Connections

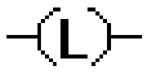
### Example Circuit:



**Related Functions:** JMP

## LATCH (COIL)

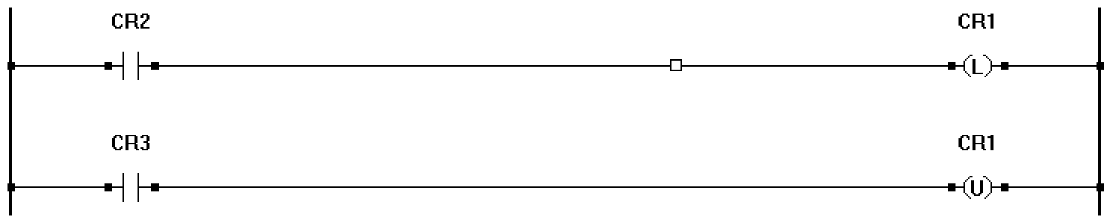
LATCH COIL



### Description:

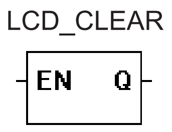
The LATCH coil operates similar to the DIRECT COIL except when true (energized), it will remain energized until a true is seen on the UNLATCH coil. LATCH and UNLATCH coils work as pairs. Any boolean variable can be used as a LATCH / UNLATCH coil.

### Example Circuit:



**Related Functions:** UNLATCH, DIRECT COIL, INVERTED COIL

## LCD_CLEAR



### Description:

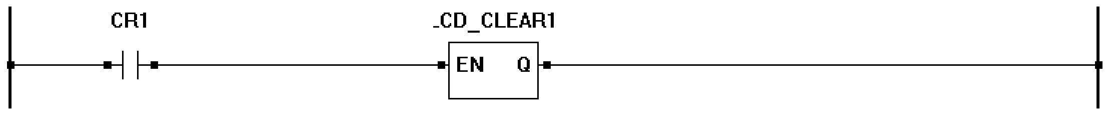
The LCD_CLR function block is used to clear the LCD display. When the EN input detects a rising edge, the LCD Display is set to be cleared. The LCD display is cleared and updated at the END of the ladder scan.

### Input / Output Connections:

The LCD_CLR function block placement requires connections of one input pin (EN) and one output pin (Q).

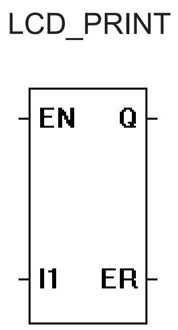
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Rising Edge | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** LCD_PRINT

## LCD_PRINT



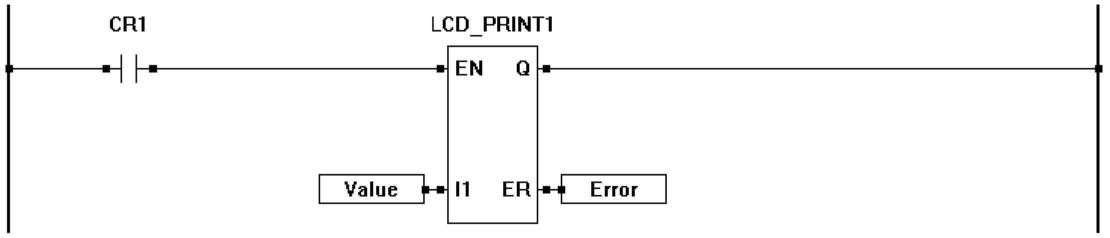
### Description:
The LCD_PRINT function is used for printing data to the LCD Display.

When then EN input senses a rising edge, the block prepares the text that was provided when the LCD_PRINT function was placed and marks it to update at the end of the current ladder scan. The Q output is set true when the print is completed. The ER output is set to non-zero if the printed data is larger than the LCD will display. At the end of the ladder scan, the display is updated. See **Chapter 9 - LCD Display Support**.

### Input / Output Connections:
The LCD_PRINT function block placement requires connections of at least one input pin (EN) and two output pins (Q, ER). Additional inputs are based on variables in printing text.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Rising Edge | |
| ERR | Output | X | | | | | Set to non-zero if error |
| I*x* | Input | X | X | X | | | Dynamic Inputs |
| Q | Output | | | X | | | True when print is completed |

### Example Circuit:



### Text / Message Formatting:
The LCD_PRINT function text formatted per ANSI C “printf”. Variables as well as text may be printed. These variables must be formatted correctly. As variables are added to the *text,* the function block will automatically add the appropriate input for the variables.

<u>**Text**</u>
Text is entered exactly as the message is intended.

Printing text longer than the display will support will result in truncated printing. It is ideal to structure printing based on column and row and to verify length of the printing.

Continued Next Page------»

## Variables

Variables are placed in the text using flags and print specification fields. The following is the configuration for adding variables to the text.

%*flag* width .precision    Example Text: OIL PSI %-3d

% - identifies the beginning of a variable or other type of text entry
*flag* - This flag is optional. Use the following flags to change the way data is transmitted.

| Flag | Description |
|---|---|
| - | Left align the variable within the specified *width*. Default is align right. |
| 0 | If width is prefixed with 0, leading zeros are added until the minimum width is reached. If 0 and - are used together, the 0 is ignored. If 0 is specified in an integer format, the 0 is ignored. |

| | |
|---|---|
| *width -* | This flag is optional. Width is the number of characters that will be printed (total). |
| *.precision -* | This flag is optional. The precision is the number of digits after the decimal point when using REAL variables. |

## Variable Formats

Variables are formatted based on the variable type. The following are supported variable types and their format.

| | | | |
|---|---|---|---|
| %d | Signed Integer | %X | Upper Case Hexadecimal |
| %u | Unsigned Integer | %f | Real or Float Variable |
| %x | Lower Case Hexadecimal | %b | binary |
| %o | Octal | | |

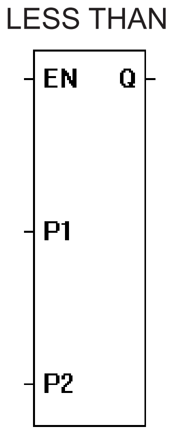
## Other Special Characters and Formats

| To Print | Use | To Print | Use |
|---|---|---|---|
| % | %% | OFF / ON | %O |
| Boolean 0 or 1 | %d | FALSE / TRUE | %T |

| Examples: | Format | Result | Format | Result |
|---|---|---|---|---|
| | OIL: %d | OIL: 25 | OIL: %04d | OIL: 0025 |
| | LS1: %T | LS1: TRUE | LS1: %O | LS1: OFF |
| | TEMP: %6.2f | TEMP: 234.12 | TEMP: %3.f | TEMP: 234 |

**Related Functions:** LCD_CLEAR

## LESS THAN (<)



### Description:

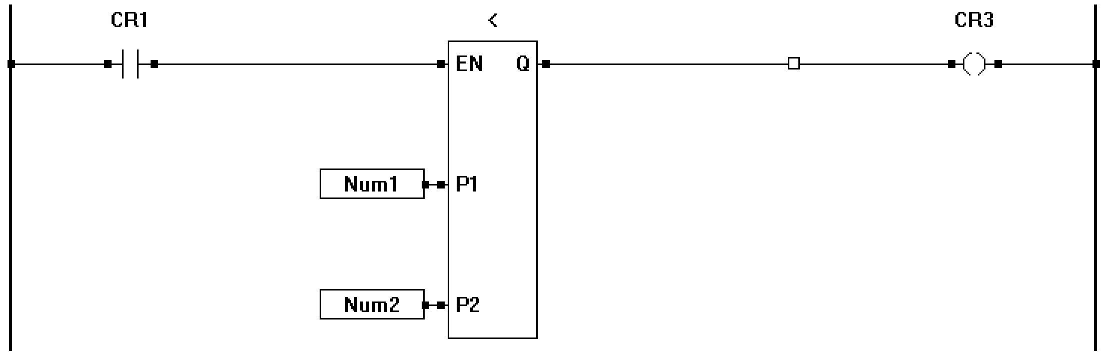
The LESS THAN provides an if less than comparison for the P*x* inputs. The number of inputs is specified when the object is placed. The output (Q) is true if P1 is less than P2 and P2 is less than P3 and so on. The enable (EN) must be true for the LESS THAN function to be enabled.

### Input / Output Connections:

The LESS THAN function block placement requires connections of at least 3 input pins (EN, P1, P2) and one output pin (Q). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

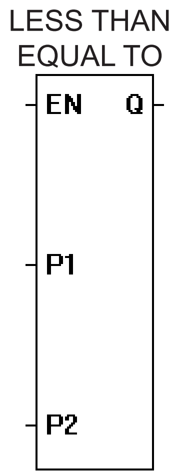
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** <=, >, >=, <>, =

## LESS THAN OR EQUAL TO (=<)



### Description:

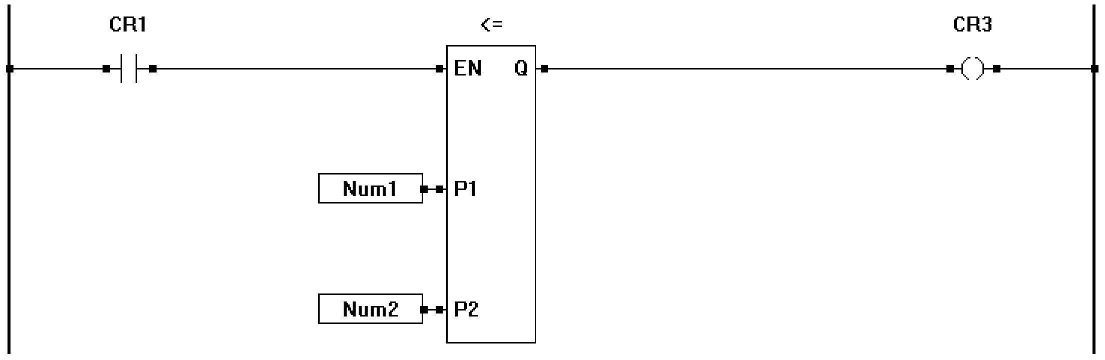
The LESS THAN or EQUAL TO provides an if less than or equal to comparison for the P*x* inputs. The number of inputs is specified when the object is placed. The output (Q) is true if P1 is less than or equal to P2 and P2 is less than or equal to P3 and so on. The enable (EN) must be true for the LESS THAN or EQUAL TO function to be enabled.

### Input / Output Connections:

The LESS THAN or EQUAL TO function block placement requires connections of at least 3 input pins (EN, P1, P2) and one output pin (Q). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

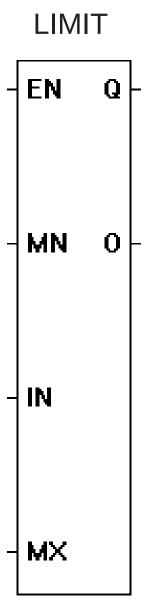
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** <, >, >=, <>, =

## LIMIT



### Description:

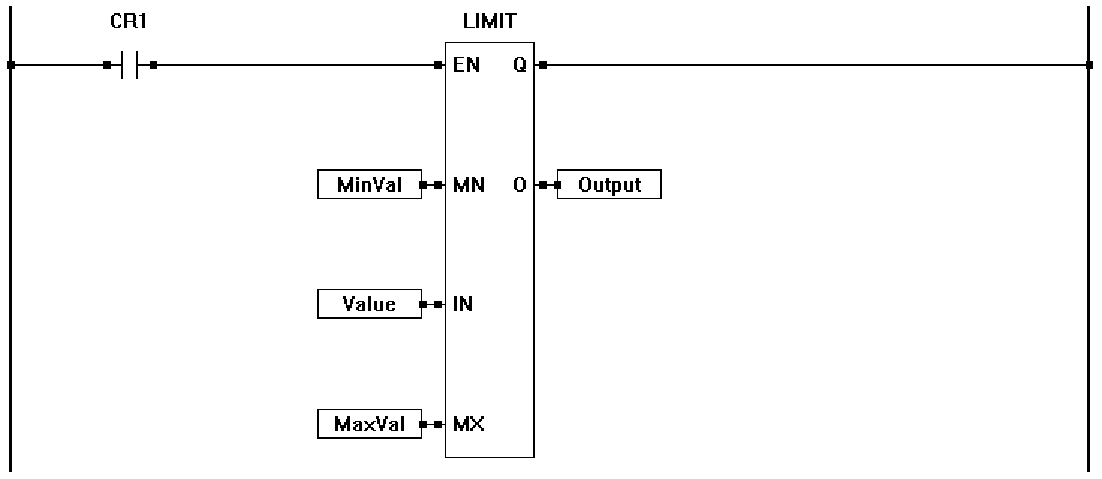
The LIMIT function provides minimum and maximum limited output for the input (IN). The function compares the input (IN). If it is greater that the maximum (MX), then the output (O) is equal to the maximum (MX). If it is less than the minimum (MN) then the output (O) is equal to the minimum (MN). If it is in between the maximum and minimum, then the output (O) is equal to the actual input (IN). The enable (EN) must be true for the LIMIT function to be enabled.

### Input / Output Connections:

The LIMIT function block placement requires connections of four input pins (EN, MN, IN, MX) and two output pins (Q, O).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| MN | Input | X | X | | | | |
| IN | Input | X | X | | | | |
| MX | Input | X | X | | | | |
| Q | Output | | | X | | | |
| O | Output | X | X | | | | |

### Example Circuit:

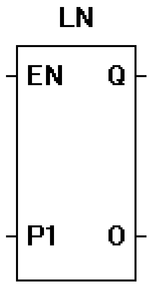


**Related Functions:** CMP, HYSTER

## LN

### Description:

The LN function provides the natural logarithm of the P1 input. The output (O) is the natural logarithm of the P1 input. The enable (EN) must be true for the LN function to be enabled.

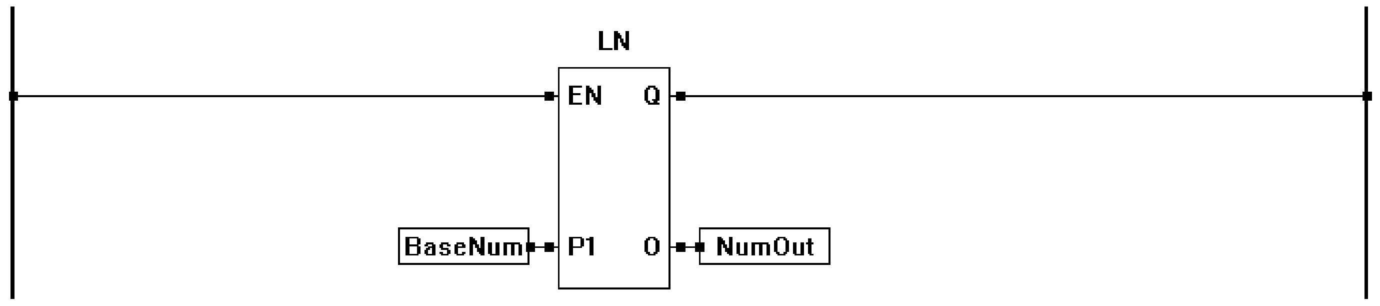


### Input / Output Connections:

The LN function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

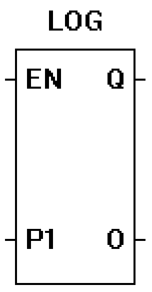
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | | |
| P1 | Input | | X | | | | Base Number |
| O | Output | | X | | | | Natural Logarithm of P1 Base Number |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** EXP, EXPT, SQRT, LOG

## LOG



### Description:

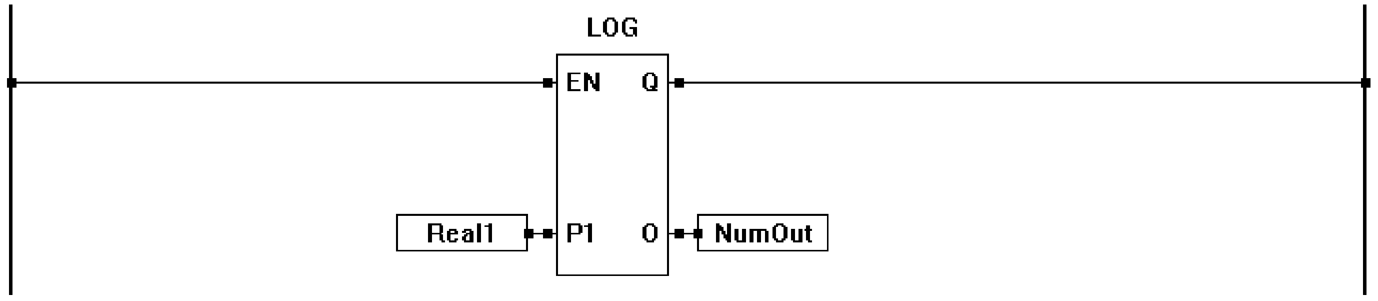
The LOG function calculates the logarithm (base 10) of the P1 input. The output (O) is the cacluated base 10 (logarithm) value of the P1 input. The enable (EN) must be true for the LOG function to be enabled.

### Input / Output Connections:

The LOG function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

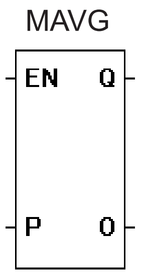
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | | |
| P1 | Input | | X | | | | Base Number |
| O | Output | | X | | | | Logarithm of P1 Number |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** EXP, EXPT, SQRT, LN

## MAVG



### Description:

The MAVG function calculates the moving average of the P input. The number of samples is specified when the object is placed. The output (O) is the calculated moving average value of the P input. The enable (EN) must be true for the MAVG function to be enabled. When EN is true, the output is the moving average. When EN is false, the output is equal to the P input.

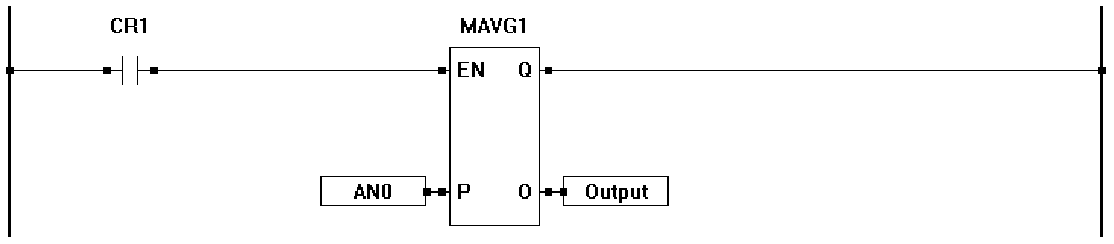
The larger the number of samples, the more RAM is used and the slower the reaction time of the block output to input changes. Size the number of samples to give the best suited reaction time and to use the least amount of RAM needed accomplish to meet the operation specifications.

### Input / Output Connections:

The MAVG function block placement requires connections of two input pins (EN, P) and two output pins (Q, O).

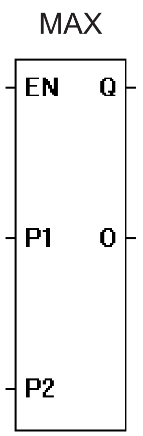
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | X | | | | |
| O | Output | X | X | | | | Moving Average of P |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** AVG

## MAX



### Description:

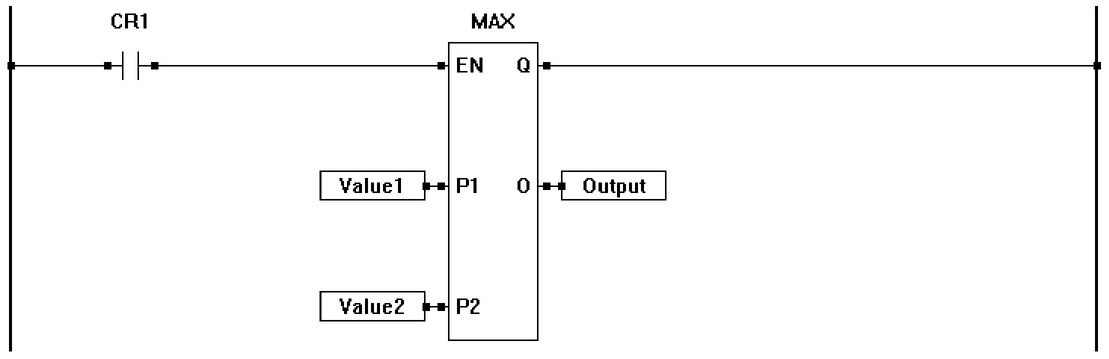
The MAX function compares all the Px input values and outputs the largest of them on the O Output. The number of inputs is specified when the object is placed. The enable (EN) must be true for the MAX function to be enabled.

### Input / Output Connections:

The MAX function block placement requires connections of at least 3 input pins (EN, P1, P2) and two output pins (Q, O). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

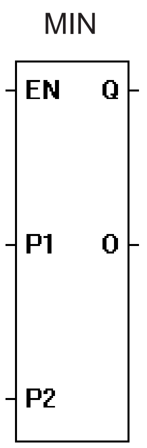
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| O | Output | X | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** MIN

## MIN



### Description:

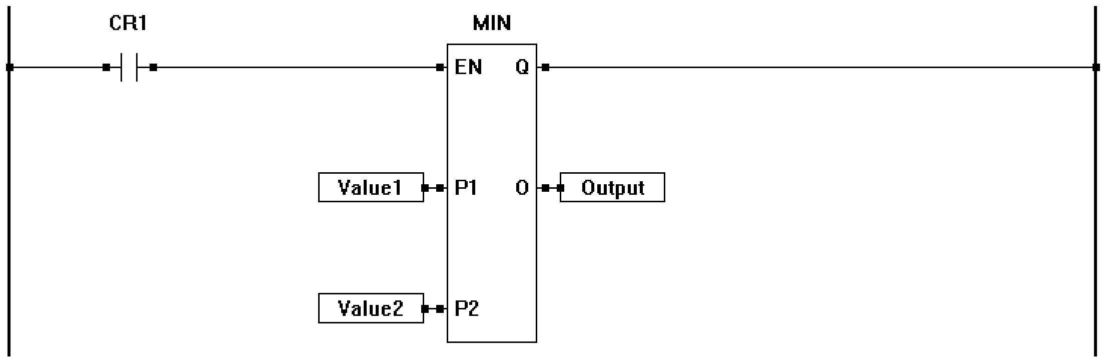
The MIN function compares all the Px input values and outputs the smalled of them on the O Output. The number of inputs is specified when the object is placed. The enable (EN) must be true for the MAX function to be enabled.

### Input / Output Connections:

The MIN function block placement requires connections of at least 3 input pins (EN, P1, P2) and two output pins (Q, O). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

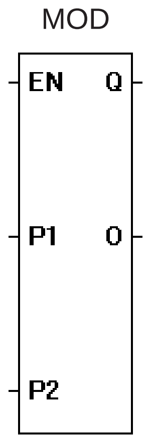
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| O | Output | X | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** MAX

## MOD



### Description:

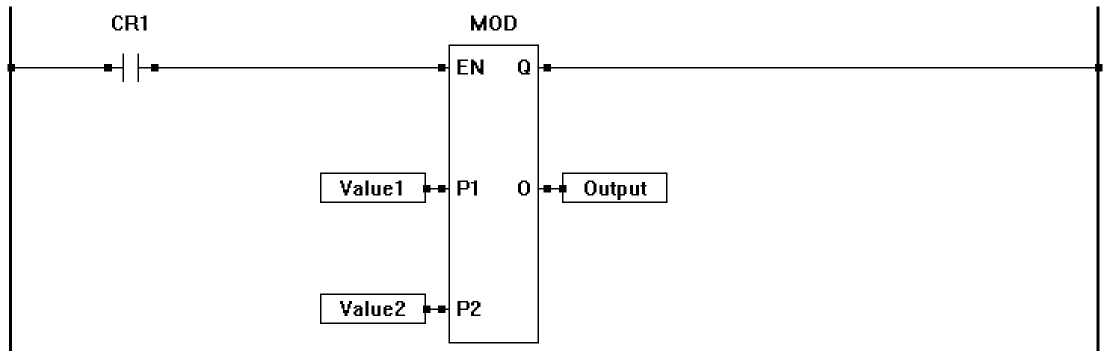
The MOD function calculates the modulo (remainder) of the division using the inputs P1 and P2. The P2 number should be greater than zero (zero or less than zero will cause the function to return invalid data the output). The enable (EN) must be true for the MOD function to be enabled.

### Input / Output Connections:

The MOD function block placement requires connections of three input pins (EN, P1, P2) and two output pins (Q, O).

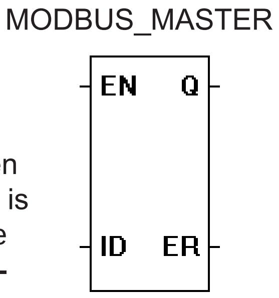
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | Dividend |
| P2 | Input | X | | | | | Divisor |
| O | Output | X | | | | | Remainder |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** DIV

## MODBUS_MASTER



### Description:

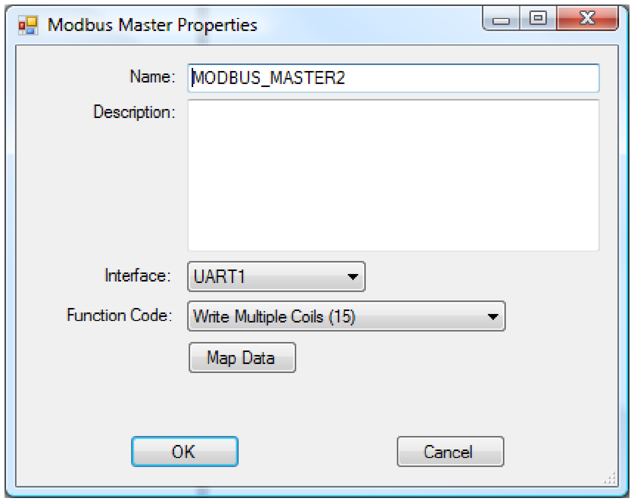
The MODBUS_MASTER function block is used to communicate to slave devices on a Modbus Network The MODBUS_MASTER function block only uses **TCP Port 502**. When the MODBUS_MASTER function block is placed, additional information is provided that is required for communications including the interface (UART/Ethernet) and Function Code (Type of communication). For additional details and information, see **Chapter 13 - Modbus Networking.**

### Input / Output Connections:

The MODBUS_MASTER function block placement requires connections of two input pins (EN, ID) and two output pins (Q, ER).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| ID | Input | X | | | | | ID Number / IP Address of Slave |
| ER | Output | X | | | | | Communication Status / Error Number |
| Q | Output | | | X | | | |

### MASTER Properties:



Name: MODBUS_MASTER2
Interface: UART1
Function Code: Write Multiple Coils (15)

The Modbus Master Properties Box shown left is automatically displayed when the MODBUS_MASTER function is placed in the ladder diagram.

Select the UART or Ethernet from the Interface Drop-down Box.

Select the Function Code from the Drop-down Box to identify the type of communication to the slave.

Click the Map Data button to map registers to variables during the communication process.

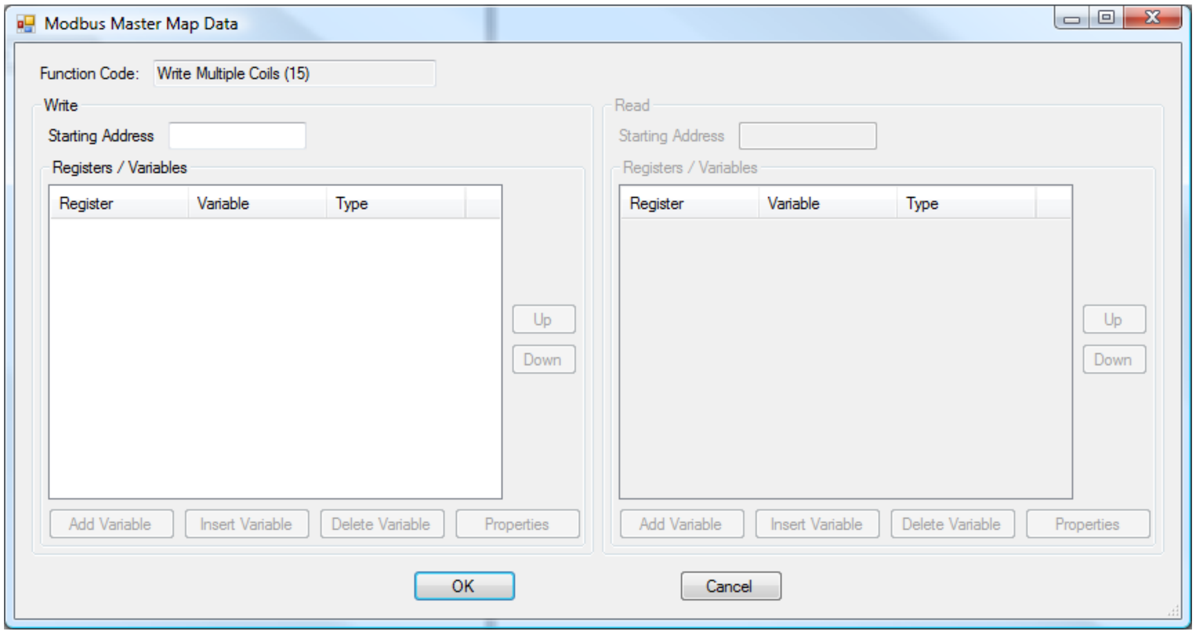
### Supported Function Codes:

| | | |
|---|---|---|
| Read / Write Multiple Registers (23) | Write Single Register (6) | Read Holding Registers (3) |
| Write Multiple Registers (16) | Write Single Coil (5) | Read Discrete Inputs (2) |
| Write Multiple Coils (15) | Read Input Registers (4) | Read Coils (1) |

### Map Data:

For the MODBUS_MASTER function block to read or write data (to/from slaves), variables and register assignments must be configured using the Map Data button. Using the Modbus Master Map Data window, variables may be assigned to transmit from and receive to. See window example shown.
For additional details and information, see **Chapter 13 - Modbus Networking.**

Continued Next Page------»


Modbus Master Map Data

Function Code: Write Multiple Coils (15)

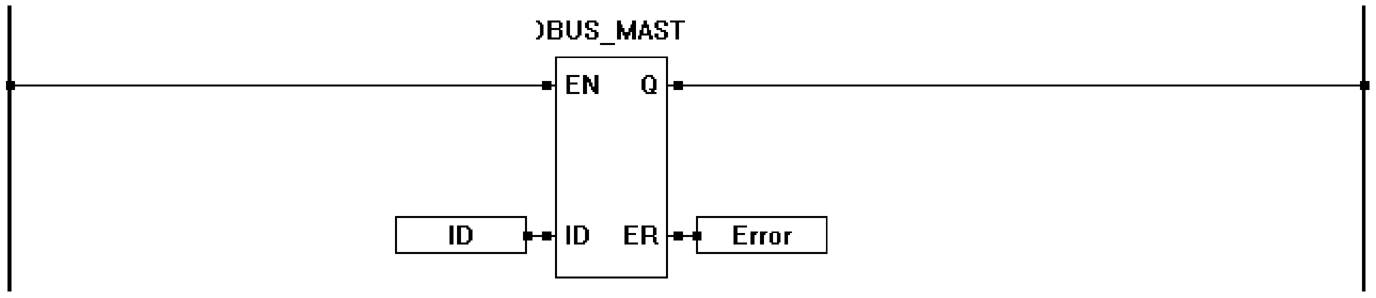
Write
Starting Address
Registers / Variables

| Register | Variable | Type |
|---|---|---|

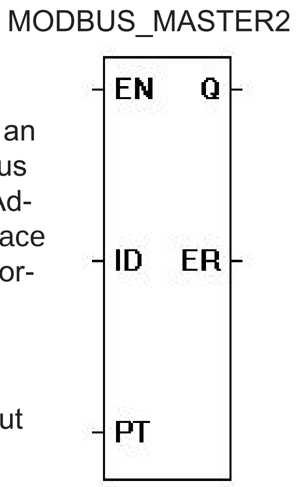
Read
Starting Address
Registers / Variables

| Register | Variable | Type |
|---|---|---|

## Example Circuit:



## MODBUS_MASTER2

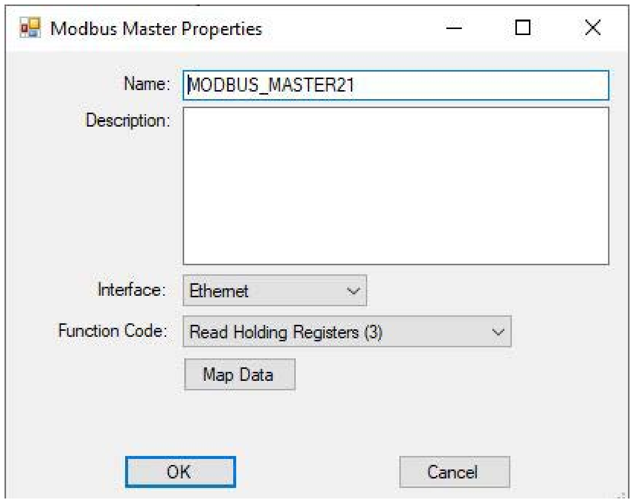


### Description:
The MODBUS_MASTER2 function block is used to communicate to slave devices on an Ethernet Modbus Network. The MODBUS_MASTER2 function block allows the modbus TCP Port to be set when thWhen the MODBUS_MASTER2 function block is placed. Additional information is provided that is required for communications including the interface (Ethernet) and Function Code (Type of communication). For additional details and information, see **Chapter 13 - Modbus Networking.**

### Input / Output Connections:
The MODBUS_MASTER2 function block placement requires connections of three input pins (EN, ID, PT) and two output pins (Q, ER).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| ID | Input | X | | | | | IP Address of Slave |
| PT | Input | X | | | | | TCP Port for Communications |
| ER | Output | X | | | | | Communication Status / Error Number |
| Q | Output | | | X | | | |

### MASTER Properties:



Name: MODBUS_MASTER21
Description:
Interface: Ethernet
Function Code: Read Holding Registers (3)

The Modbus Master Properties Box shown left is automatically displayed when the MODBUS_MASTER2 function is placed in the ladder diagram.

Select the Ethernet from the Interface Drop-down Box.

Select the Function Code from the Drop-down Box to identify the type of communication to the slave.

Click the Map Data button to map registers to variables during the communication process.

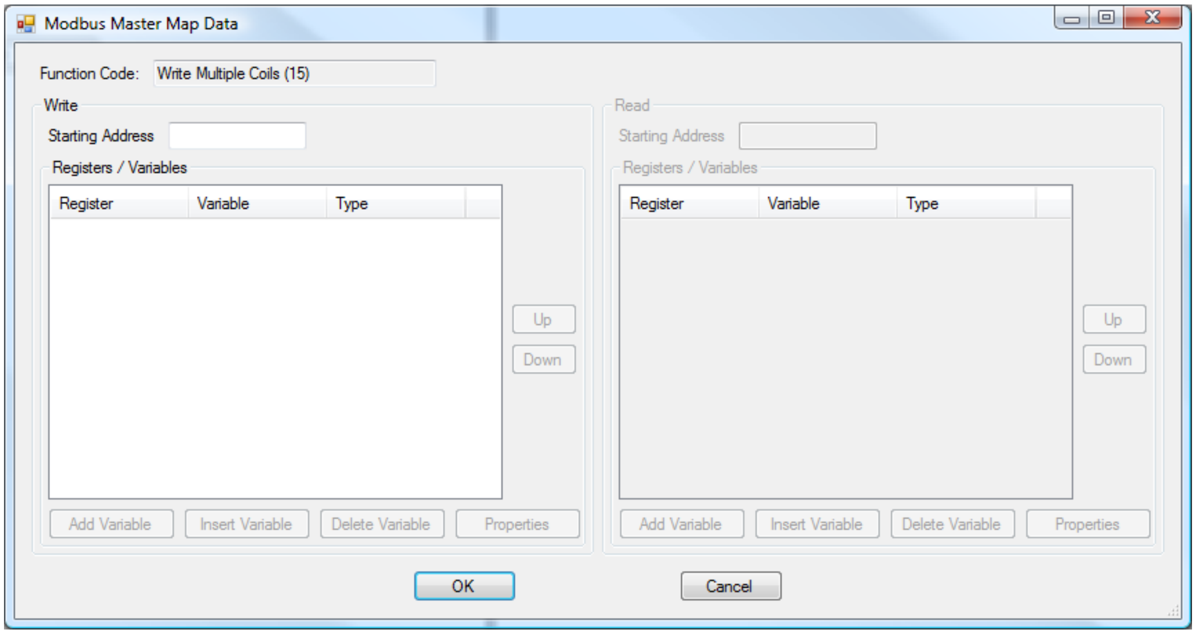
### Supported Function Codes:

| | | |
|---|---|---|
| Read / Write Multiple Registers (23) | Write Single Register (6) | Read Holding Registers (3) |
| Write Multiple Registers (16) | Write Single Coil (5) | Read Discrete Inputs (2) |
| Write Multiple Coils (15) | Read Input Registers (4) | Read Coils (1) |

### Map Data:

For the MODBUS_MASTER2 function block to read or write data (to/from slaves), variables and register assignments must be configured using the Map Data button. Using the Modbus Master Map Data window, variables may be assigned to transmit from and receive to. See window example shown.

Continued Next Page------»


Modbus Master Map Data

Function Code: Write Multiple Coils (15)

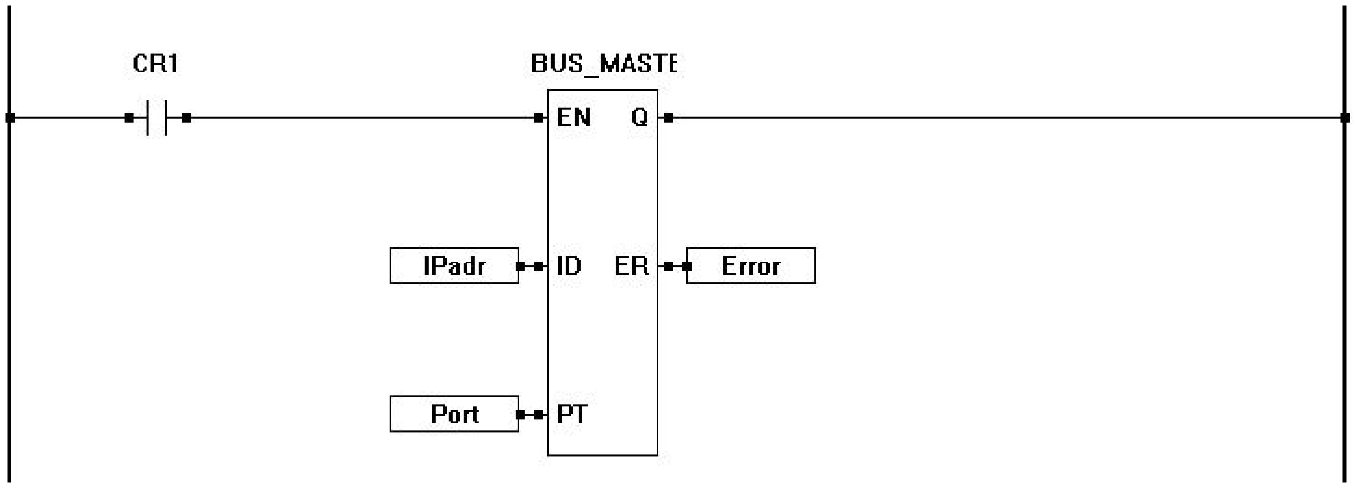
Write

Starting Address

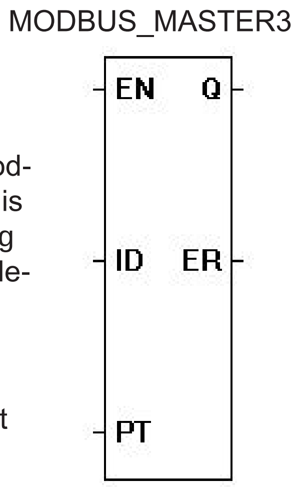
Registers / Variables

| Register | Variable | Type |
|---|---|---|

## Example Circuit:



## MODBUS_MASTER3

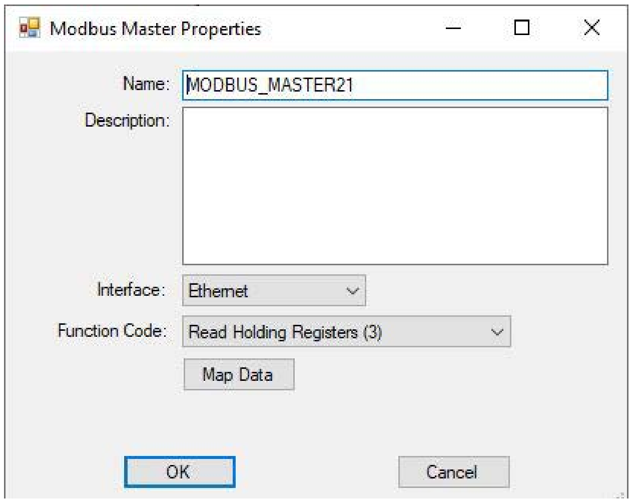


### Description:
The MODBUS_MASTER3 function block is used to communicate to slave devices on an Ethernet Modbus Network. The MODBUS_MASTER3 function block allows the modbus TCP Port and the unit ID to be set when the MODBUS_MASTER3 function block is placed. Additional information is provided that is required for communications including the interface (Ethernet) and Function Code (Type of communication). For additional details and information, see **Chapter 13 - Modbus Networking.**

### Input / Output Connections:
The MODBUS_MASTER3 function block placement requires connections of four input pins (EN, IP, PT, ID) and two output pins (Q, ER).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| IP | Input | X | | | | | IP Address of Slave |
| PT | Input | X | | | | | TCP Port for Communications |
| ID | Input | X | | | | | Unit ID (slave address) |
| ER | Output | X | | | | | Communication Status / Error Number |
| Q | Output | | | X | | | |

### MASTER Properties:



Name: MODBUS_MASTER21
Description:
Interface: Ethernet
Function Code: Read Holding Registers (3)

The ID is the unit ID (slave address) is typically used when communicating via bridges, gateways, and routers that use a single IP address to support multiple independent Modbus units. The slave address (ID) may be passed through the gateway / router device (all dependent upon the devices connected).

If the PT is set to 0, then the default TCP port (502 is used.

The Modbus Master Properties Box shown left is automatically displayed when the MODBUS_MASTER3 function is placed in the ladder diagram.

Select the Ethernet from the Interface Drop-down Box.

Select the Function Code from the Drop-down Box to identify the type of communication to the slave.

Click the Map Data button to map registers to variables during the communication process.

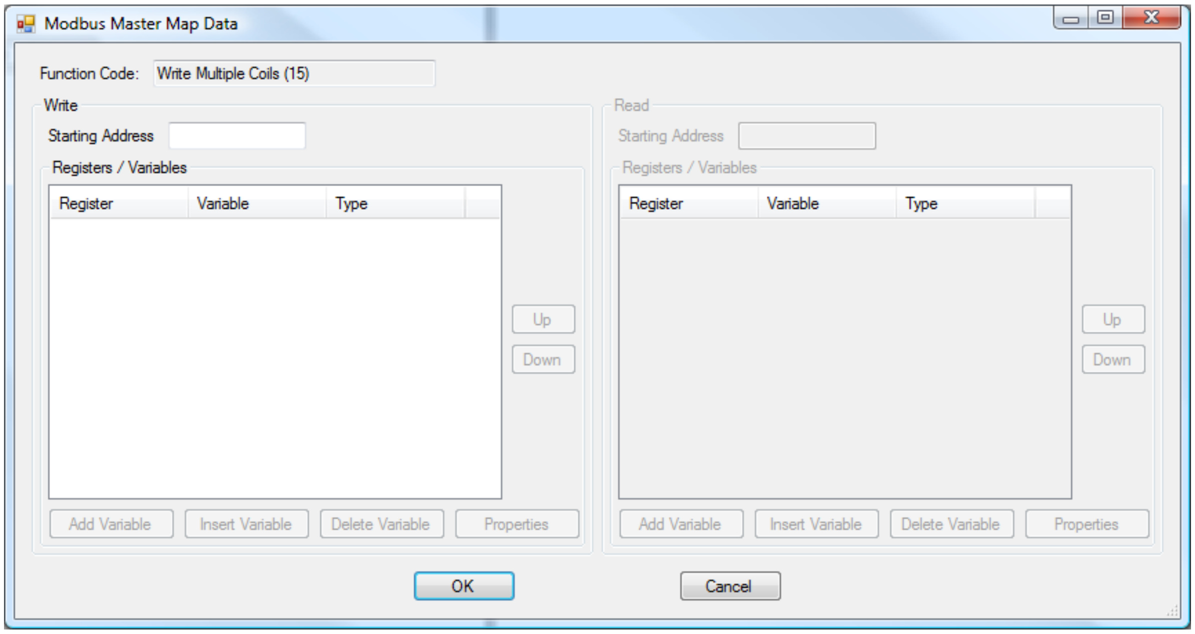
### Supported Function Codes:

| | | |
|---|---|---|
| Read / Write Multiple Registers (23) | Write Single Register (6) | Read Holding Registers (3) |
| Write Multiple Registers (16) | Write Single Coil (5) | Read Discrete Inputs (2) |
| Write Multiple Coils (15) | Read Input Registers (4) | Read Coils (1) |

Continued Next Page------»

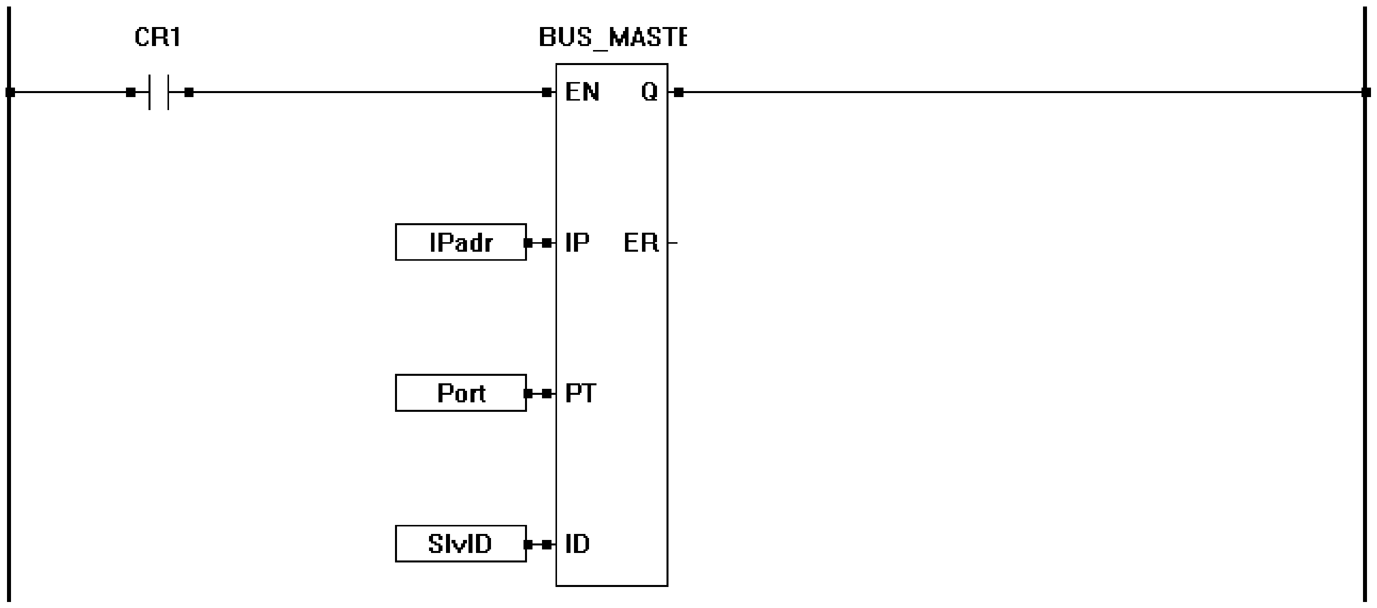
## Map Data:

For the MODBUS_MASTER3 function block to read or write data (to/from slaves), variables and register assignments must be configured using the Map Data button. Using the Modbus Master Map Data window, variables may be assigned to transmit from and receive to. See window example shown.



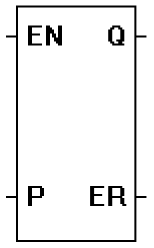
Function Code: Write Multiple Coils (15)

## Example Circuit:



## MODBUS_SET_PROPERTY

MODBUS_SET_PROPERTY



### Description:

The MODBUS_SET_PROPERTY function block is used to set or change the Modbus Slave paramaters. The parameters to change is based on the type of communications implemented for the modbus slave device.

For Modbus Slave using UARTs: The function block sets the ID for the target from the ladder diagram program. This feature is ideal for reading inputs (switches) and setting the slave ID accordingly. For additional details and information, see **Chapter 13 - Modbus Networking.**

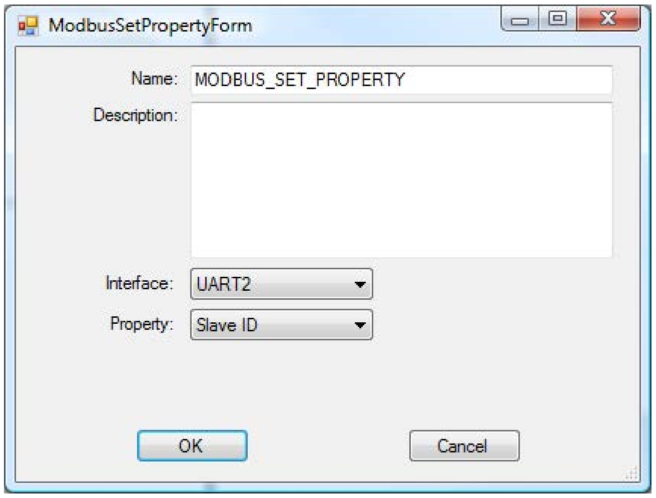
For Modbus Slave using Ethernet/Wi-Fi: The function block sets the TCP Port for Ethenet/Wi-Fi communications from the ladder diagram program. This function is optional for changing the TCP Port for Ethernet Communications using Modbus Slave.

When the EN is true, the slave ID or TCP Port number at input P is set for modbus functionality. The Q output is true when EN is true. The ERR output will identify if a error occurred (if other than zero).

### Input / Output Connections:

The MODBUS_SET_PROPERTY function block placement requires connections of two input pins (EN, P) and two output pins (Q, ERR).

When placing the MODBUS_SET_PROPERTY function, an automatic dialog will appear that is used to set paramters such as the type of interface and the Property to be set. At this time, only the Slave ID is supported for the property.


ModbusSetPropertyForm

Name: MODBUS_SET_PROPERTY

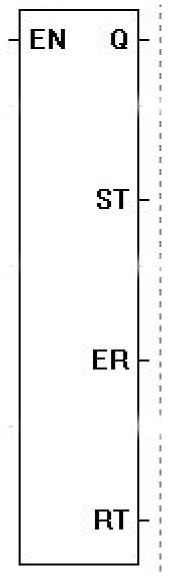
Description:

Interface: UART2

Property: Slave ID

## MQTT_CONNECT

MQTT_CONNECT



### Description:

The MQTT_CONNECT function block opens the connection to the VersaCloud M2M+IoT cloud server (IoT Central, IoT Hub or Exosite Murano) selected in the Project Settings. The output (O) provides the result of the connection. The enable (EN) must be true for the MQTT_CONNECT function to be enabled (and attempt connection).

### Input / Output Connections:

The MQTT_CONNECT function block placement requires connections of one input pin (EN) and four output pins (Q, ST, ER, RT).

The EN (Enable) is active high input and when true, it will attempt to connect to the Versa-Cloud M2M+IoT cloud server (and stay connected based on configuration items set in the dialog when inserting the MQTT_CONNECT function block. The function block connects to the VersaCloud M2M+IoT cloud server that is selected/installed in the MQTT Project Settings.

The **ST** output provides the status (MQTT state) of operation. The ST output values are as follows:

| | |
|---|---|
| 1 | Connecting |
| 2 | Provisioning |
| 3 | Provision Sent |
| 100 | Connect Sent |
| 101 | Connected |
| 200 | Connect Error |
| 300 | TLS Handshaking Failed |
| 301 | Authentication Read Error |
| 302 | Authentication Needs User Password |
| 303 | Authentication Invalid User Password |
| 304 | Authentication Not Authorized (Bad Username or Password) |
| 305 | Connection Refused (Invalid Protocol Version) |
| 306 | Connection Refused (Client ID Rejected) |
| 307 | Connection Refused (Server Unavailable) |

The **ER** output provides MQTT network errors detected (if any). The ER output values are as follows:

| | |
|---|---|
| 0 | No Error |
| -1 | Out of Memory Error |
| -2 | Buffer Error |
| -3 | Timeout Error |
| -4 | Reverved (Should not see) |
| -5 | Operation Currently in Progress |
| -6 | Illegal Value |
| -7 | Reverved (Should not see) |
| -8 | Address in Use |
| -9 | Already Connecting |
| -10 | Connection Already Established |
| -11 | Not Connected |

Continued Next Page------»

| | |
|---|---|
| -12 | Network Interface Error (Cellular, Ethernet, etc) |
| -13 | Connection Aborted |
| -14 | Connection Reset |
| -15 | Connection Closed |
| -16 | Illegal Argument |

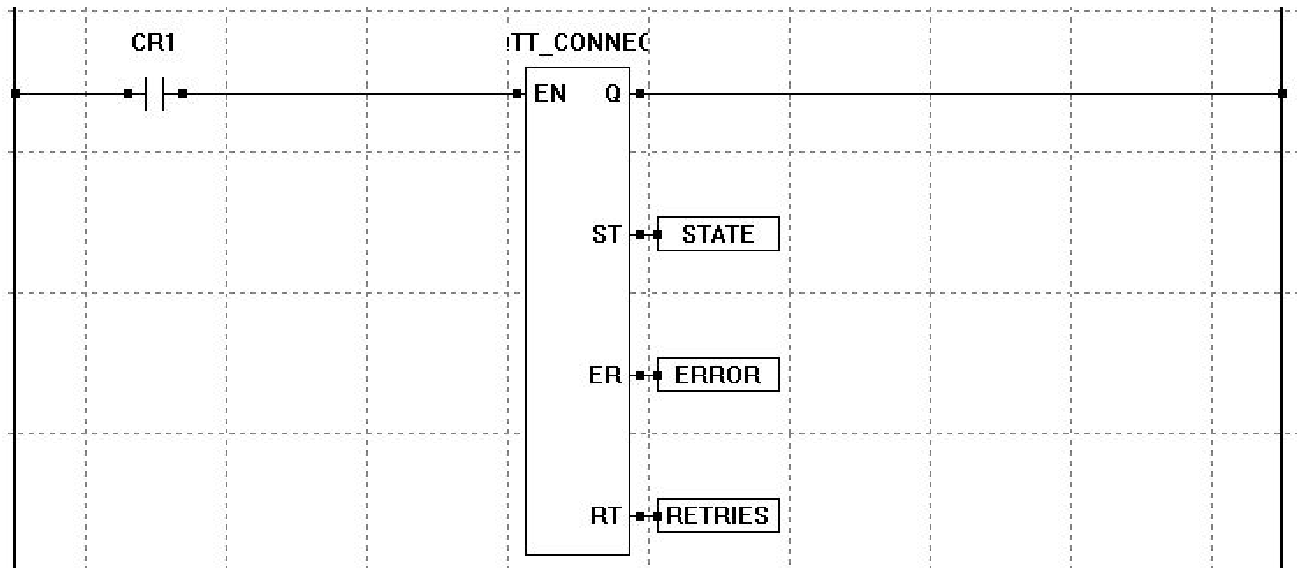
Additional errors below -8192 are specific to the SSL layter. For example, -9984 is a Certificate failed verification (CRL, CA or signature check failed. This can be caused by the certificate not matching, if there is not enough free memory (RAM) in the controller or if the certificate date is not vaild.

The **RT** output isnumber of retries that have occurred. This number is cumulative (until target reset or power cycle). The larger the number, the more times the block has tried to connect to the selected cloud solution (some may be successful, others may not).

The **Q** Output is true when the EN (Enable) input is true.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| ST | Output | X | | | | | MQTT State (see above) |
| ER | Output | X | | | | | MQTT Network erros (see above) |
| RT | Output | X | | | | | Number of connect retries |
| Q | Output | | | X | | | True when EN is true |

## Example Circuit:



The MQTT_CONNECT function block is only available after the MQTT feature has been added to the Project Settings and configured for operation.

The MQTT_CONNECT function block requires an active connection (Ethernet, Wi-Fi or Cellular) for connecting to the VersaCloud M2M+IoT cloud server solution (selected in the MQTT Project Settings).

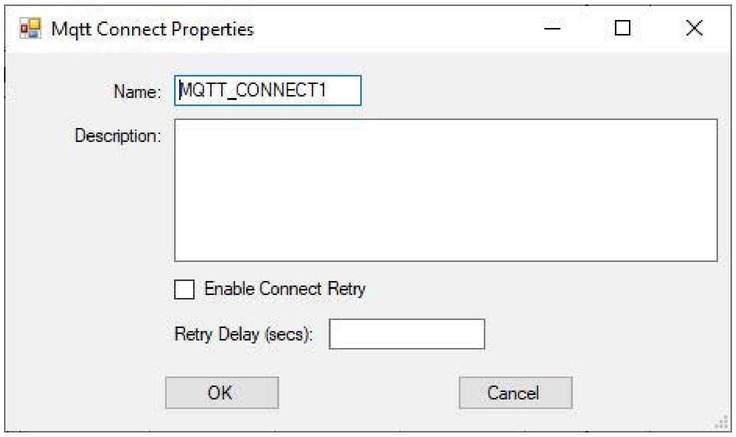
Continued Next Page------»

## Dialog Box:

When placing the MQTT_CONNECT function block, the Mqtt Connect properties window will open automatically. This window has a couple of parameters to be configured, based on needs and preferences.

The **Enable Connect Retry** checkbox is provided for the function block to automatically try reconnecting should the connection be lost.

The **Retry Delay (secs)** field is used to enter the number of seconds to delay between reconnection retries.



Name: MQTT_CONNECT1

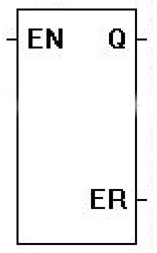
☐ Enable Connect Retry

Retry Delay (secs):

**Related Functions:** MQTT_PUBLISH

## MQTT_PUBLISH

MQTT_PUBLISH



### Description:

The MQTT_PUBLISH function block sends data the VersaCloud M2M+IoT cloud server (IoT Central, IoT Hub or Exosite Murano) selected in the Project Settings. The output (O) provides the result transmission. The enable (EN) is rising edge sensitive for the MQTT_ PUBLISH function to be enabled (and attempt to send data).

### Input / Output Connections:

The MQTT_PUBLISH function block placement requires connections of one input pin (EN) and two output pins (Q, ER).

The EN (Enable) is rising edge sensitive and when a rising edge (false to true) is detected, attempt to transmit data (variable added to the function block, selected when the function block is added) to the VersaCloud M2M+IoT cloud server. The function block sends data to the VersaCloud M2M+IoT cloud server that is selected/installed in the MQTT Project Settings.

Multiple MQTT_PUBLISH blocks may be added tot he ladder diagram, but control over them is required to prevent multiple from attempting to send at the same time (use resource locking).

The data (variables) to send to the VersaCloud M2M+IoT cloud server are selected in the Mqtt Publish Properties window when the MQTT_PUBLISH function block is inserted into the ladder diagram. See the Dialog box section for details on the Mqtt Publish Properties window.

The **ER** output provides MQTT network errors detected (if any). The ER output values are as follows:

| | |
|---|---|
| 0 | No Error |
| -1 | Out of Memory Error |
| -2 | Buffer Error |
| -3 | Timeout Error |
| -4 | Reverved (Should not see) |
| -5 | Operation Currently in Progress |
| -6 | Illegal Value |
| -7 | Reverved (Should not see) |
| -8 | Address in Use |
| -9 | Already Connecting |
| -10 | Connection Already Established |
| -11 | Not Connected |
| -12 | Network Interface Error (Cellular, Ethernet, etc) |
| -13 | Connection Aborted |
| -14 | Connection Reset |
| -15 | Connection Closed |
| -16 | Illegal Argument |

Additional errors below -8192 are specific to the SSL layter. For example, -9984 is a Certificate failed verification (CRL, CA or signature check failed. This can be caused by the certificate not matching, if there is not enough free memory (RAM) in the controller or if the certificate date is not vaild.

The **Q** output goes true for one ladder scan (the scan the send happened on), provided there was no errors.

Continued Next Page------»

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| ER | Output | X | | | | | MQTT Network erros (see above) |
| Q | Output | | | X | | | True one scan when no errors. |

## Dialog Box:

When placing the MQTT_PUBLISH function block, the Mqtt Publish Properties window will open automatically. This window is used to select the variables that will be sent to the VersaCloud M2M+IoT cloud server.

The variables selected / added to the MQTT_PUBLISH block should already exist, or they may be added in the variable add dialog when selecting the variable (See variables in the **Chapter 5 - Ladder Diagram Projects**).

Variable types used with the MQTT_PUBLISH block include BOOLEAN, REAL and INTEGER.

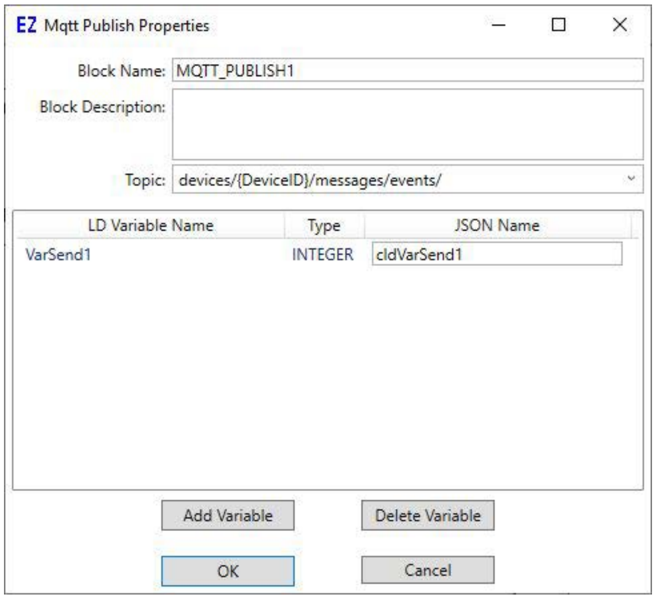
Click the **ADD VARIABLE** button to add a variable to the MQTT_PUBLISH block control. Select (or add a new variable) to use and click **OK**. The variable will now be populated in the Mqtt Publish Properies dialog.

The JSON Name will now by default be populated with the name of the variable. This JSON Name can be edited / changed, but this JSON Name entered is the name used in the cloud portal side to map the data.

The JSON Name entered in this field must match the name of the variable on the cloud portal for data to be sent to and received on the cloud portal side. If these names do not match, data will not be received on the cloud portal.

In the **Topic** drop down menu, select: ***devices/{DeviceID}/messages/events/***

Click **OK** when all the variables have been added to send to the VersaCloud M2M+IoT cloud solution (portal).



Block Name: MQTT_PUBLISH1
Block Description:
Topic: devices/{DeviceID}/messages/events/

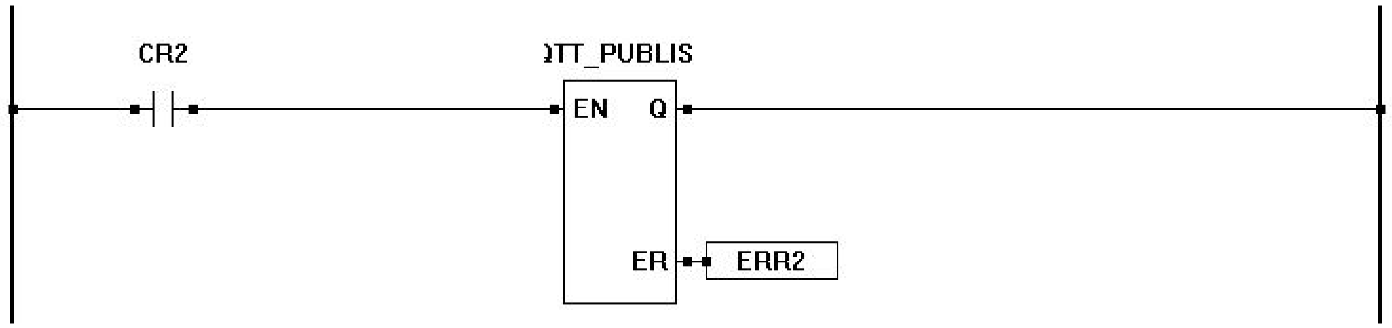
| LD Variable Name | Type | JSON Name |
|---|---|---|
| VarSend1 | INTEGER | cldVarSend1 |

Continued Next Page------»

Variables can be deleted using the **DELETE** button in the Mqtt Publish Properties dialog.

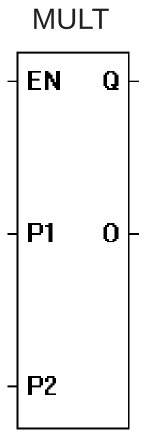
Multiple MQTT_PUBLISH blocks may be used in a single program.

## Example Circuit:



**Related Functions:** MQTT_CONNECT

## MULT



### Description:

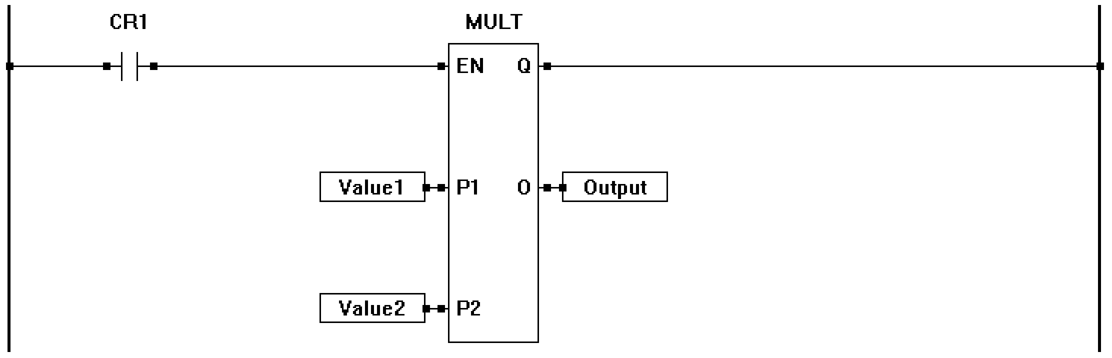
The MULT function multiplies all of the Px inputs together. The number of inputs is specified when the object is placed. The output (O) provides the result of the multiplication.  The enable (EN) must be true for the MULT function to be enabled.

### Input / Output Connections:

The MULT function block placement requires connections of at least 3 input pins (EN, P1, P2) and two output pins (Q, O). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

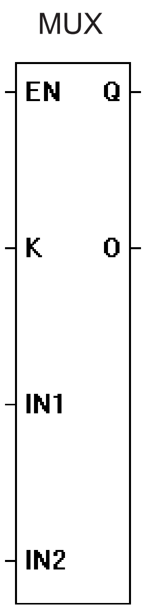
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| O | Output | X | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** ADD, SUB, DIV

## MUX



### Description:

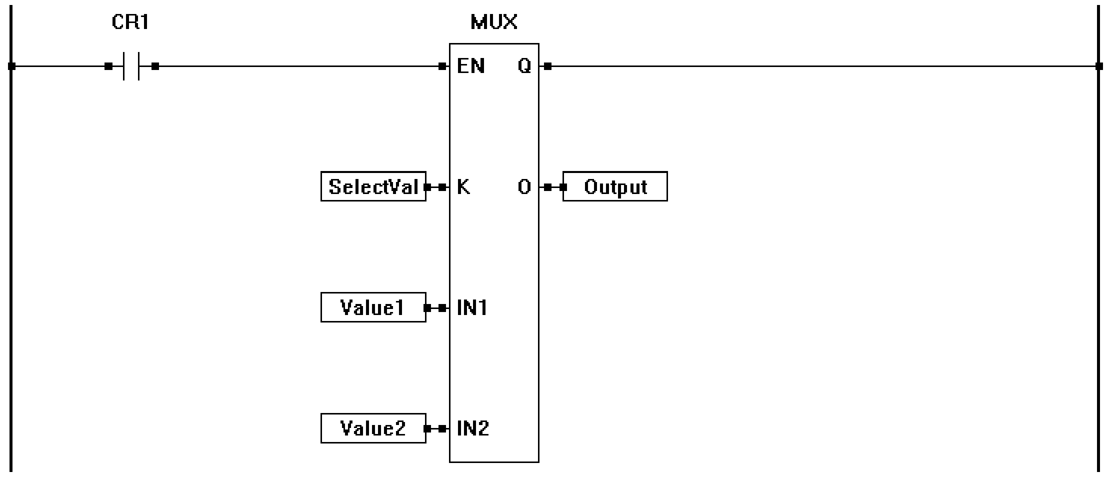
The MUX function multiplexes the INx inputs into one output (O). The number of inputs is specified when the object is placed. The output (O) provides the value of the selected input. The value on input K (starts at zero for IN1) determines the number of the input that will be present on the output. The enable (EN) must be true for the MUX function to be enabled.

### Input / Output Connections:

The MUX function block placement requires connections of at least four input pins (EN, K, IN1, IN2) and two output pints (Q, O).

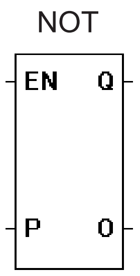
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| K | Input | X | | | | | |
| IN*x* | Input | X | X | | | | |
| Q | Output | | | X | | | |
| O | Output | X | X | | | | = Value of INx selected by K |

### Example Circuit:



### Related Functions: SEL

## NOT



### Description:

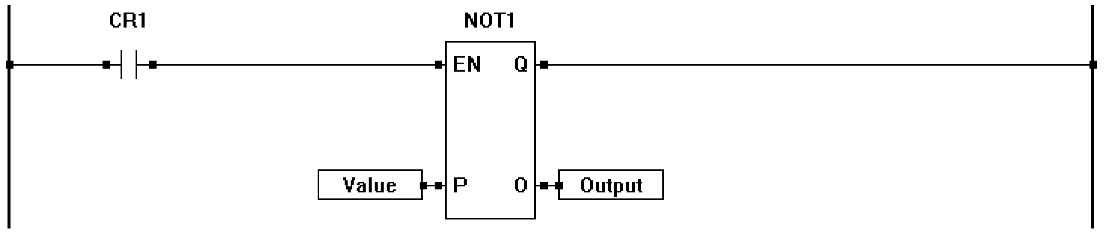
The NOT function provides a one's complement (bit to bit negation) of the P input. The output (O) provides the one's complement. The enable (EN) must be true for the NOT function to be enabled.

### Input / Output Connections:

The NOT function block placement requires connections of two input pins (EN, P) and two output pins (Q, O).

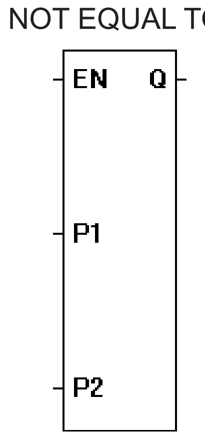
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | | | | | |
| O | Output | X | | | | | One's Complement of P |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** OR, AND, XOR

## NOT EQUAL TO (<>)



### Description:

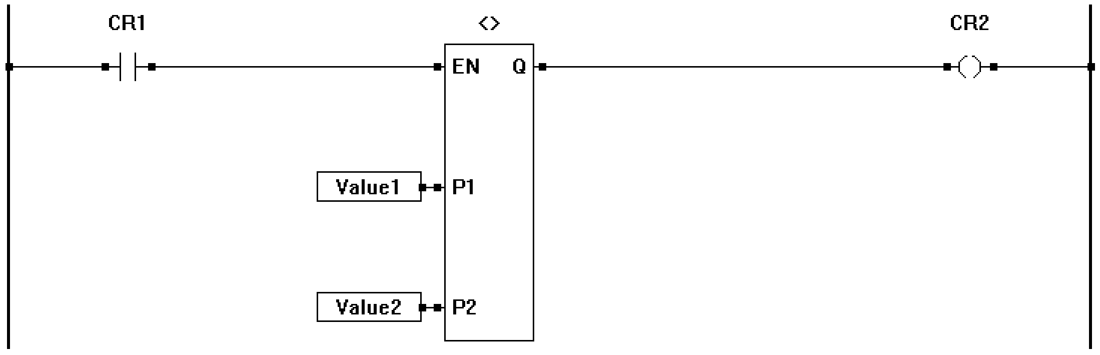
The NOT EQUAL TO provides an if greater than or less than comparison for the Px inputs. The number of inputs is specified when the object is placed. The output (Q) is true if P1 is not equal to P2 and P2 is not equal to P3 and so on. The enable (EN) must be true for the NOT EQUAL TO function to be enabled.

### Input / Output Connections:

The NOT EQUAL TO function block placement requires connections of at least 3 input pins (EN, P1, P2) and one output pin (Q). The EN is always considered an input in the total number of inputs, therefore always add one to the number of P*x* inputs that need to be used.

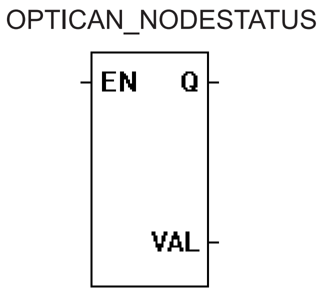
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P*x* | Input | X | X | | | | Number of inputs is dynamic |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** =, <, >, >=, <=

## OPTICAN_NODESTATUS



### Description:

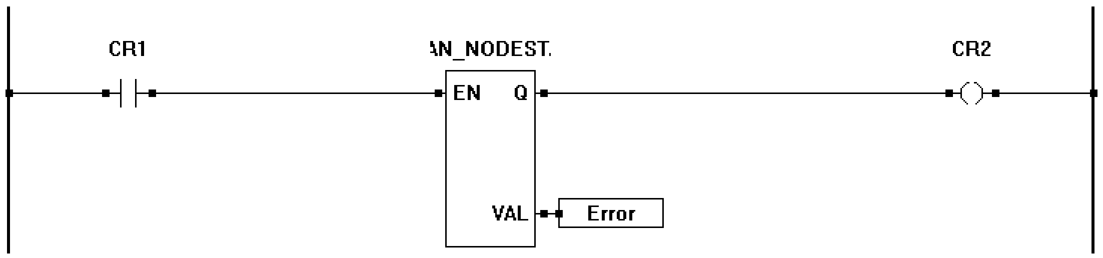
The OPTICAN_NODESTATUS function *listens* for OK of the status register (191) for the specified node over the OptiCAN network. When placing the function, the NODE ID is specified as well as the Timeout. The function block will *listen* for the node status register broadcast of the Node ID and update VAL and Q accordingly. The Timeout value is the duration that the function block will *listen* and not receive a status prior generating an Error on the VAL output pin and the Q output. The Q output is true when the VAL output is valid. See **Chapter 14 - OptiCAN Networking** for more information regarding using the function block and general OptiCAN networking.

### Input / Output Connections:

The OPTICAN_NODESTATUS function block placement requires connections of one input pin (EN) and two output pins (Q, VAL).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| O | Output | X | | | | | See Error Codes |
| Q | Output | | | X | | | |

### Example Circuit:



### Error Codes:

The Node Status register (191) is represented by a 32 bit number. The lower 16 bits represents the current **status code** while the upper 16 bits represents the **error code**.

There are three status codes supported on the OptiCAN network. The status codes are: 1 = Reset, 2 = Active, and 4 = Reset.

The Q output will be true if the VAL output is valid. If invalid (no response from node), then the Q output will be false and the VAL output will equal zero.

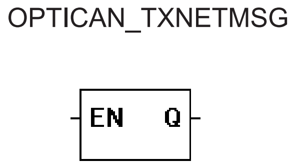
Error codes are divided into two groups. Device specific errors are numbered 0-32767 while common error codes are numbered 32768-65535.

Common Error Codes are as follows:

| | |
|---|---|
| 65535 = CAN Controller Receive Error | 65531 = CAN Controller Bus Off State |
| 65534 = CAN Controller Receive Warning | 65530 = CAN Controller Data Overrun |
| 65533 = CAN Controller Transmit Error | 65519 = OptiCAN Heartbeat Timeout |
| 65532 = CAN Controller Transmit Warning | 65518 = CAN Controller Error |

**Related Functions:** OPTICAN_TXNETMSG

## OPTICAN_TXNETMSG



### Description:

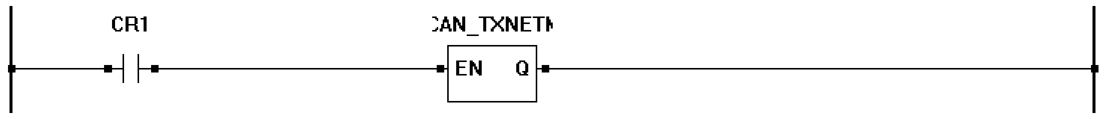
The OPTICAN_TXNETMSG function broadcasts the network control commands Start Network, Stop Network and Reset Network on the OptiCAN network. This function block globally broadcasts, therefore affecting all connected nodes. A Start Network command must be broadcast after power up to start the OptiCAN network nodes communications. When placing the function, a dialog box provides the selection of the type of command to send and an optional description box. See **Chapter 14 - OptiCAN Networking** for more information regarding using the function block and general OptiCAN networking.

### Input / Output Connections:

The OPTICAN_TXNETMSG function block placement requires connections of one input pin (EN) and one output pin (Q).

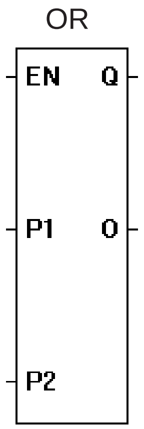
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** OPTICAN_NODESTATUS

## OR



### Description:

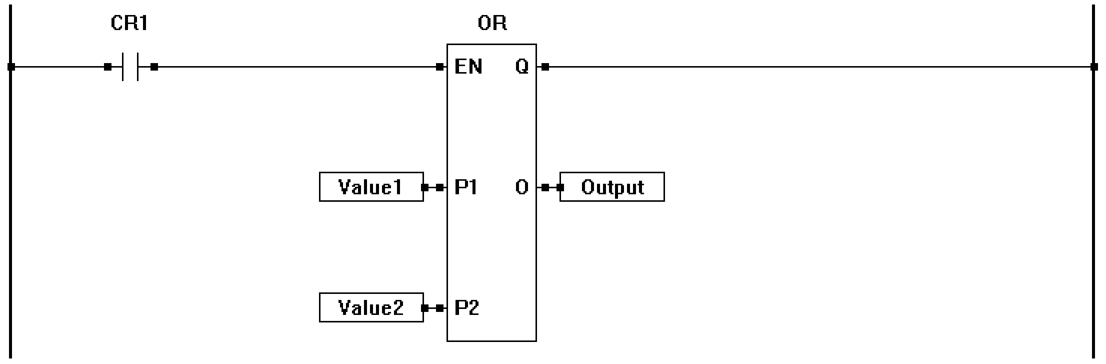
The OR function provides a bitwise OR function of the P1 and P2 inputs. The enable (EN) must be true for the OR function to be enabled. The Q output is true when the OR function is enabled.

### Input / Output Connections:

The OR function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

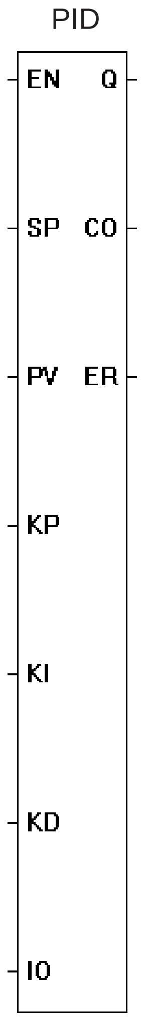
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| O | Output | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** XOR, AND, NOT

## PID

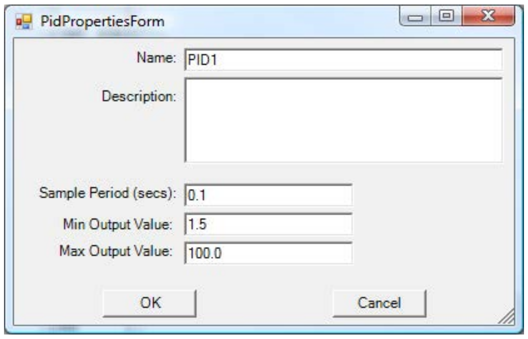


### Description:

The PID function provides an easy to use PID control algorithm. Specific PID information is required when the function is placed as well as the PID inputs. The Q is true when the function is enabled. The CO (Control Output) is the output calculated by the PID. The ER is the error calculation of the PID (SP-PV). The PID function is defined by the difference Equation:

$$u(n) = u(n-1) + Kp[e(n) - e(n-1)] + Ki [T * e(n)] + (Kd / T)[e(n) - 2 * e(n-1) + e(n-2)]$$

| Where: | | | |
|---|---|---|---|
| | u(n) = PID Output | Kp = Proportional Gain | Ki = Integral Gain |
| | Kd = Derivative Gain | e(n) = Error (Setpoint - Process Variable) | T = Sample Period |



Name: PID1
Description:
Sample Period (secs): 0.1
Min Output Value: 1.5
Max Output Value: 100.0

| | |
|---|---|
| Name: | Name of the PID function. |
| Description: | Enter a description. |
| Sample Period (Secs): | The sample period in seconds (Min = .01S, Max = 86,400S), sample period resolution = 50µS. |
| Minimum Output Value: | Minimum PID Output value allowed. |
| Maximum Output Value: | Maximum PID Output value allowed. |

If SP, PV or process error is determined not to be infinite values, the error flag is set and the CO is set to the IO value. When these values are valid (infinite) again, the PID function will return to normal.

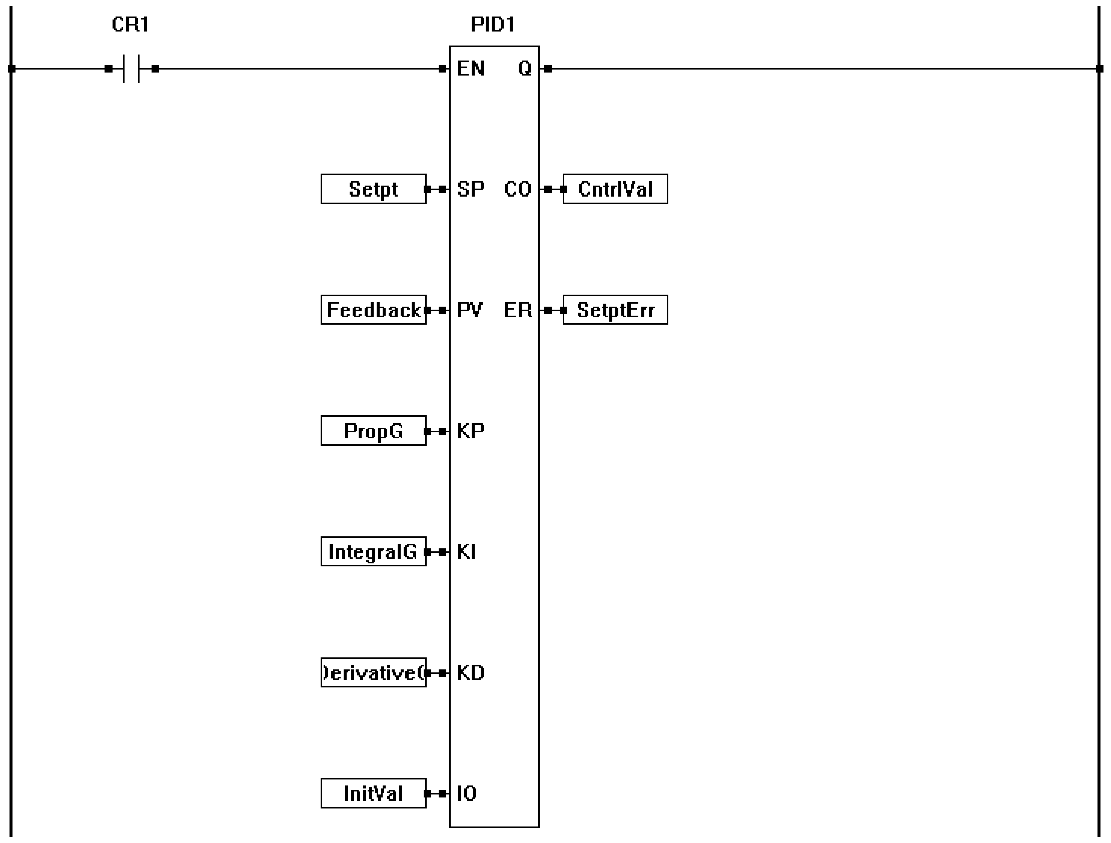
### Input / Output Connections:

The PID function block placement requires connections of 7 input pins (EN, SP, PV, KP, KI, KD, IO) and 3 output pins (Q, CO, ER).

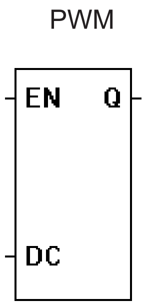
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| SP | Input | | X | | | | Control Set Point |
| PV | Input | | X | | | | Process Feedback Variable |
| KP | Input | | X | | | | Proportional Gain |
| KI | Input | | X | | | | Integral Gain |
| KD | Input | | X | | | | Derivative Gain |
| IO | Input | | X | | | | Initial Value, PID is init to this value |
| CO | Output | | X | | | | Control Output - Calculated Signal |
| ER | Output | | X | | | | Error = Amount of error from set point |
| Q | Output | | | X | | | |

Continued Next Page------»

## Example Circuit:



## PWM



### Description:

The PWM function controls the function of a hardware PWM output channel (this channel is specified when the function is placed). When the EN is true, the hardware PWM channel outputs a square wave with the specified duty cycle (DC) at the frequency pre-programmed (this frequency is determined by the PWM channel and is configured when the PWM channel is installed in the target settings menu unless the PWM_FREQ function overrides this frequency with it's own). The Q output is true when the function is enabled.

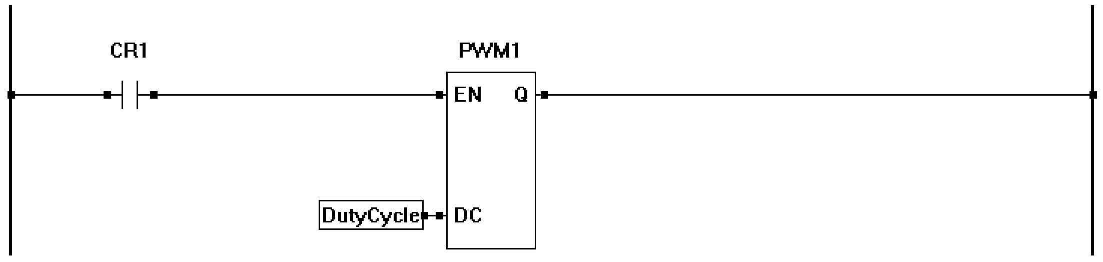
When the PWM function is placed, you must specify the actual hardware PWM channel that the function will control and the Polarity (Starting Low will cause the PWM channel to start with a TTL low, Starting High will cause the PWM channel to start with a TTL high). Refer to **Chapter 8 - Pulse Width Modulation** for details on PWM functionality.

### Input / Output Connections:

The PWM function block placement requires connections of two input pins (EN, DC) and one output pin (Q).

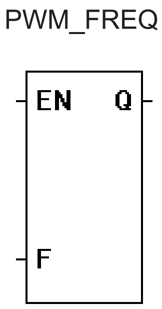
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| DC | Input | X | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** PWM_FREQ

## PWM_FREQ



### Description:

The PWM_FREQ function controls the frequency of a hardware PWM output channel (this channel is specified when the function is placed). When the EN sees a low to high transition, the hardware PWM channel's frequency is changed from it's current value (either from when the PWM channel was installed using the Target..Settings menu or a PWM_FREQ function).

The PWM_FREQ only changes the hardware PWM channel's frequency with a low to high transition on EN. This frequency will be maintained regardless of the EN state. The only time this frequency will change again is when the actual frequency input variable (input F) changes and the EN detects another low to high transition. Q is true during the ladder diagram scan when the frequency is newly applied. All other times, the Q output is low.

When the PWM function is placed, you must specify the actual PWM channel group (CLK A or CLK B) that the function will change the frequency to.

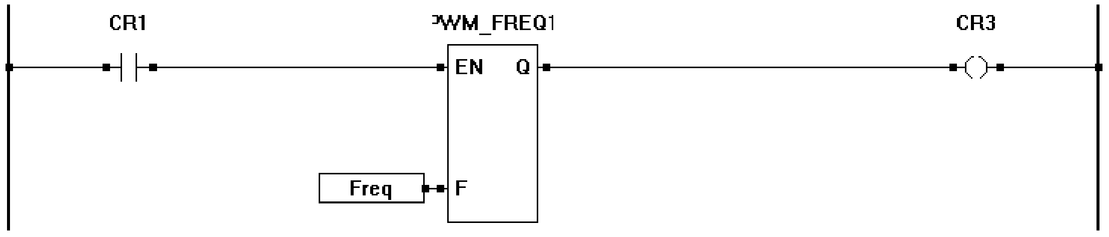
If an invalid frequency is applied to input to F, then the Q Output will remain low as well as the actual PWM output.

### Input / Output Connections:

The PWM function block placement requires connections of two input pins (EN, F) and one output pin (Q).

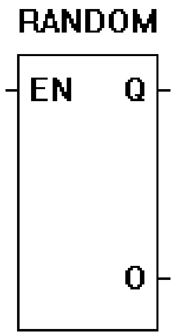
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Rising Edge | |
| F | Input | X | X | | | | Input Frequency |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** PWM

## RANDOM



### Description:

The RANDOM functions provides a random number based on the SEED (function) value. The enable (EN) must be true for the RANDOM function to be enabled. The Q output is true when the RANDOM function is enabled. The output (O) is the random number.

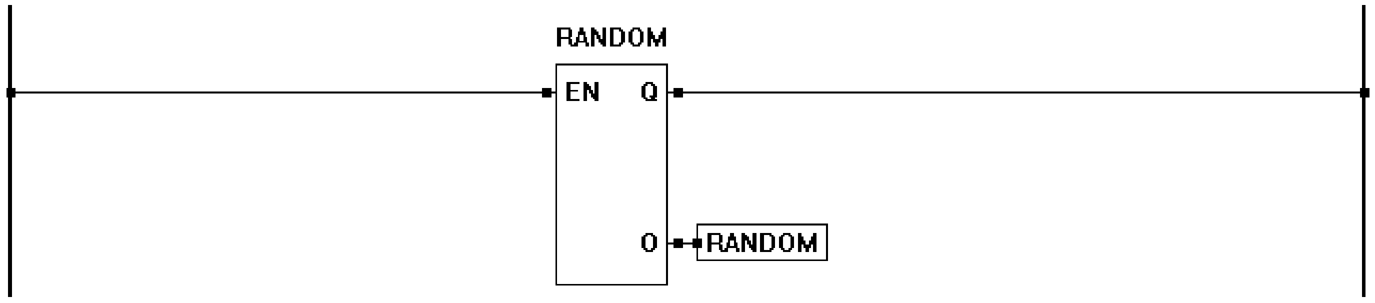
The RANDOM function is designed to be used with the SEED function. Without the SEED, the output will not be random.

### Input / Output Connections:

The RANDOM function block placement requires connections of one input pin (EN) and two output pins (Q, O).

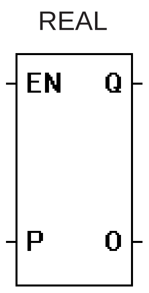
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| O | Output | X | | | | | Random Integer Number |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** SEED

## REAL



### Description:

The REAL function converts the input (P) into an real output (O). The enable (EN) must be true for the REAL function to be enabled. The Q output is true when the REAL function is enabled.

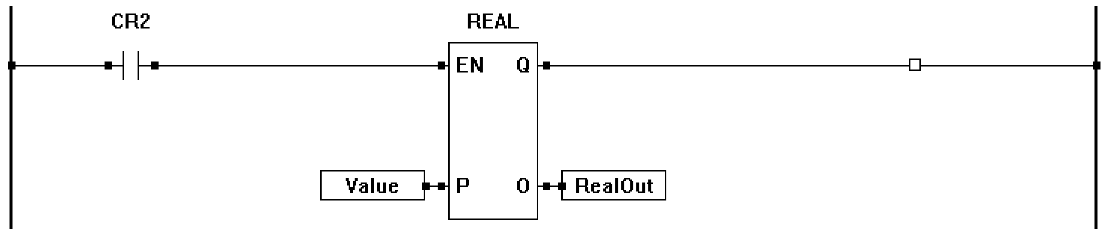
In addition to converting a Boolean, Timer or Integer to an real, the REAL function block can be used to copy one real to another.

### Input / Output Connections:

The REAL function block placement requires connections of two input pins (EN, P) and two output pins (Q, O).

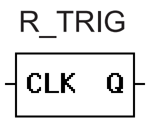
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | X | X | X | | |
| O | | | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** TIMER, BOOLEAN, INTEGER

## R_TRIG



### Description:

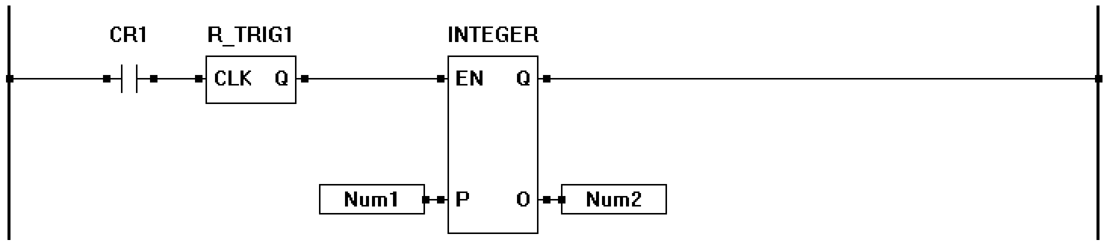
The R_TRIG is a function that may be used to trigger another function on the rising edge of a transition. When the CLK detects a false to true transition, the output (Q) is energized for one scan of the program only.

### Input / Output Connections:

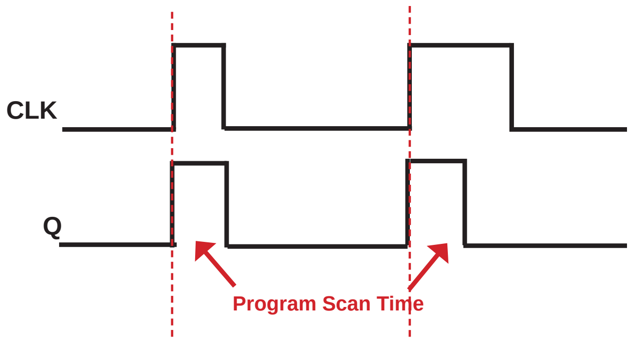
The R_TRIG function block placement requires connections of one input pin (CLK) and one output pin (Q).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| CLK | Input | | | X | | Falling Edge | |
| Q | Output | | | X | | | True for only one scan |

### Example Circuit:

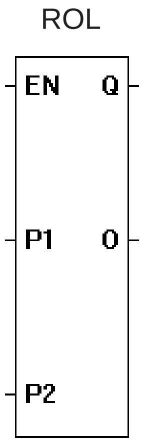


### Timing Diagram:



**Related Functions:** F_TRIG

## ROL



### Description:

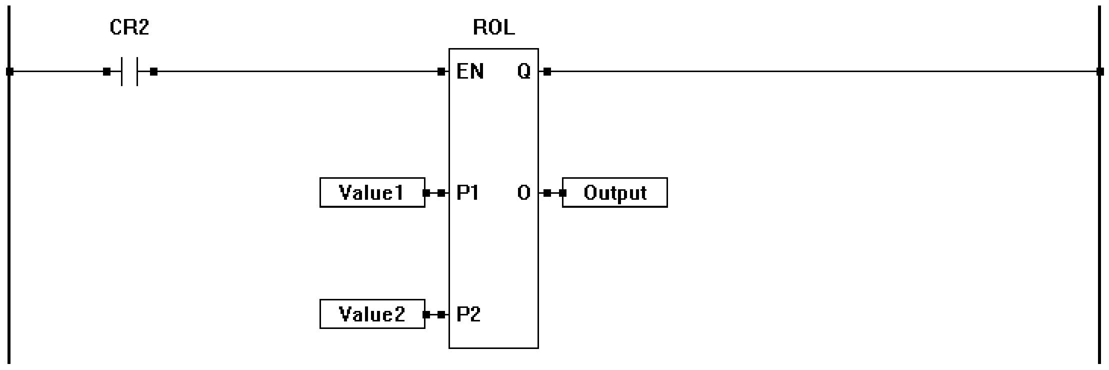
The ROL function provides a left-bit rotation of the P1 input. P2 specifies the number of one-bit rotations. The P1 number is a integer representation of a binary number. The P2 number is an integer representation of the number of binary rotations (shifts) to occur to P1. The actual bit only rotates when the maximum number is reached (example: 32 bit rotation to the input number 1).The enable (EN) must be true for the ROL function to be enabled. The Q output is true when the ROL function is enabled. The O Output is the rotated number (represented in integer form).

### Input / Output Connections:

The ROL function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

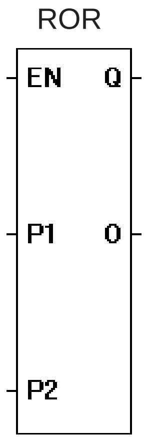
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| O | Output | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** ROR

## ROR



### Description:

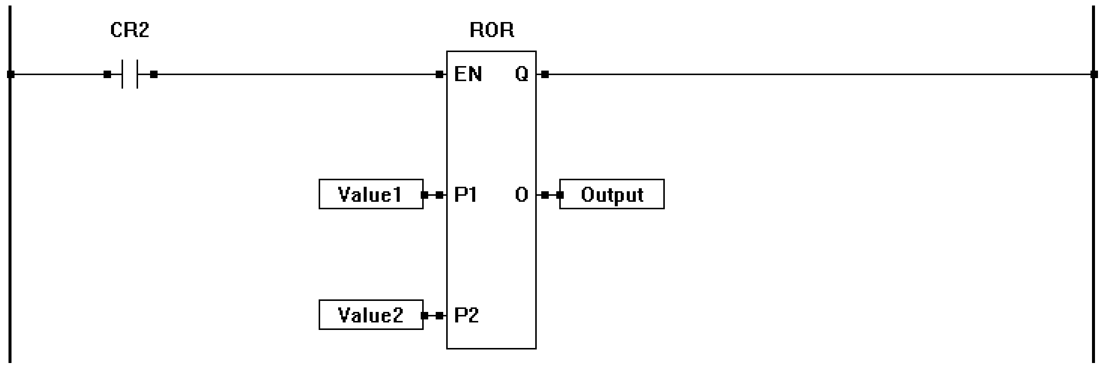
The ROR function provides a right-bit rotation of the P1 input. P2 specifies the number of one-bit rotations. The P1 number is a integer representation of a binary number. The P2 number is an integer representation of the number of binary rotations (shifts) to occur to P1. The actual bit only rotates when the minimum number is reached (example: 32 bit rotation to the input number 32). The enable (EN) must be true for the ROR function to be enabled. The Q output is true when the ROR function is enabled. The O Output is the rotated number (represented in integer form).

### Input / Output Connections:

The ROR function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

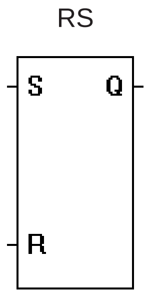
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| O | Output | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** ROL

## RS



### Description:

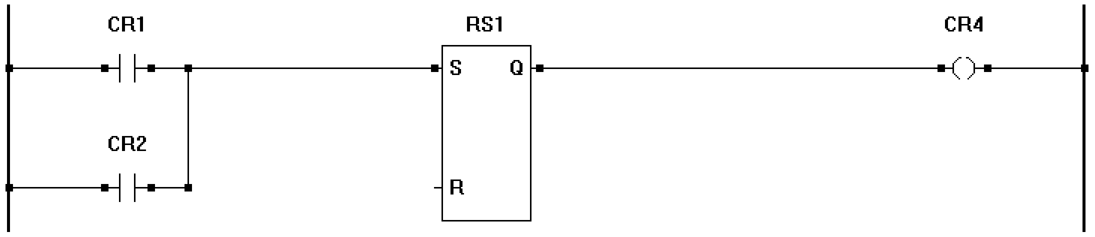
The RS function acts as a reset dominant bistable. If the set input (S) is true, the output (Q) is true. A true on the reset (R) input sets the output (Q) to false (regardless of the set (S) input state).

### Input / Output Connections:

The RS function block placement requires connections of two input pins (S,R) and one output pin (Q).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| R | Input | | | X | | | |
| S | Input | | | X | | | |
| Q | Output | | | X | | | |

### Example Circuit:

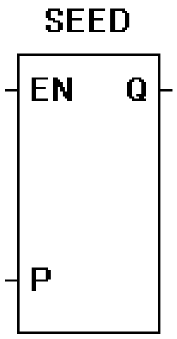


### Truth Table:

| SET | RESET | Q | Q RESULT |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

### Related Functions: SR

## SEED



### Description:

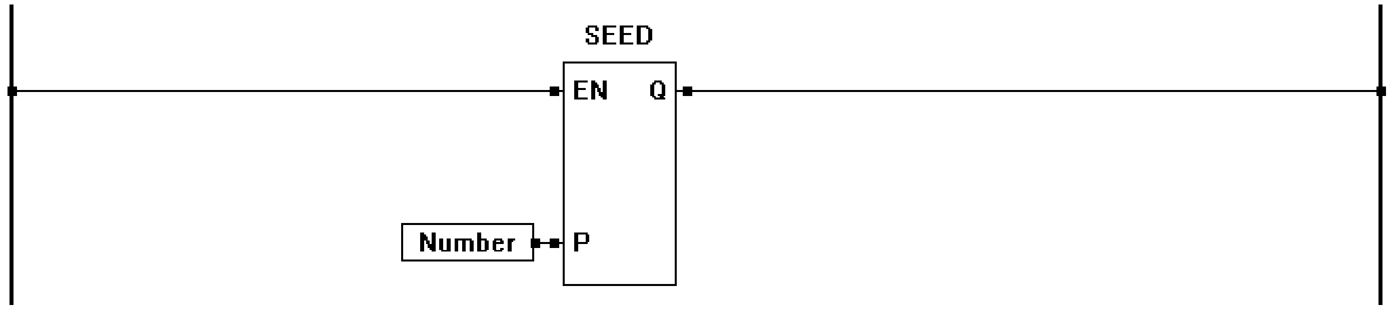
The SEED function provides the number which the RANDOM function uses as the basis for generating a random number. The enable (EN) must be true for the SEED function to be enabled. The Q output is true when the SEED function is enabled.

### Input / Output Connections:

The SEED function block placement requires connections of two input pins (EN, P) and one output pins (Q).

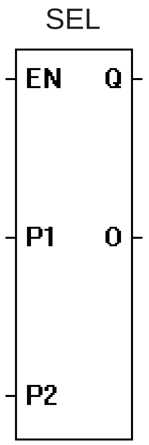
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** RANDOM

## SEL



### Description:

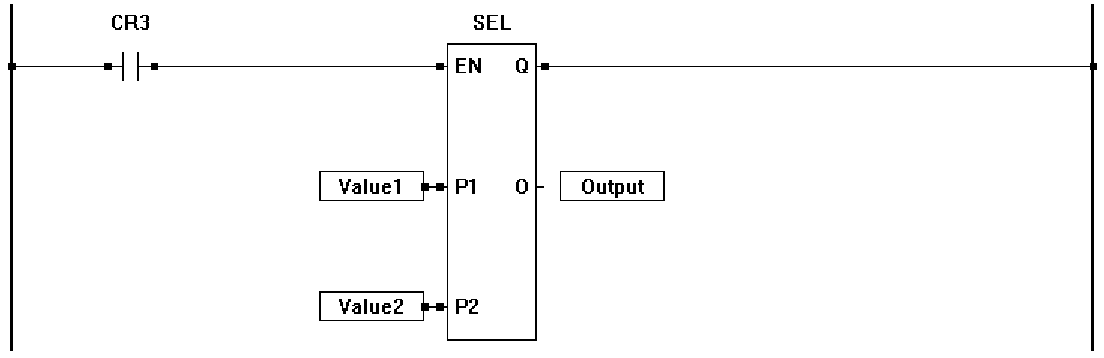
The SEL function provides selection of the P1 or P2 inputs. If enable (EN) is false, the output (O) will be equal to the input P1. If the enable (EN) is true, the output (O) will be equal to the input P2. The Q output is true when the SEL function is enabled.

### Input / Output Connections:

The SEL function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

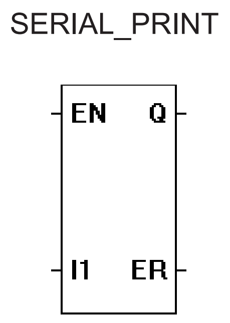
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | X | | | | |
| P2 | Input | X | X | | | | |
| O | Output | X | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** MUX

## SERIAL_PRINT



### Description:

The SERIAL_PRINT function is the transmit block for sending serial information using a multi-purpose serial port.

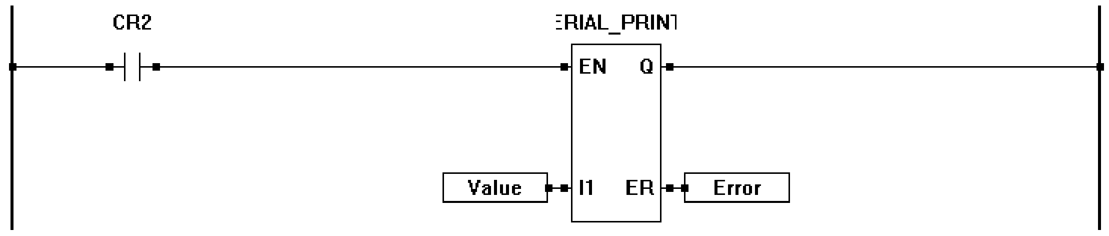
When then EN input senses a rising edge, the block begins the serial transmission of its text that was provided when the SERIAL_PRINT function was placed. The Q output is set true when the transmission is completed. The ER output is set true if there is still data in the buffer when the function block is enabled to transmit again. See **Chapter 11 - Serial Printing**.

### Input / Output Connections:

The SERIAL_PRINT function block placement requires connections of at least one input pin (EN) and two output pins (Q, ER). Additional inputs are based on variables in serial text.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Rising Edge | |
| ERR | Output | X | | | | | Set to non-zero if error |
| I*x* | Input | X | X | X | | | Dynamic Inputs |
| Q | Output | | | X | | | True when transmit is completed |

### Example Circuit:



### Text / Message Formatting:

The SERIAL_PRINT function text formatted per ANSI C “printf”. The function block examples shown are for VT100 terminals. Variables as well as text may be printed. These variables must be formatted correctly. As variables are added to the *text,* the function block will automatically add the appropriate input for the variables.

**Text**

Text is entered exactly as the message is intended. Variables, special formats and escape codes are also added to this field. See the Variables, Other Special Variables and Formats and the Escape Sequence sections for details regarding adding special codes for specific needs.

Continued Next Page------»

**Variables**

Variables are placed in the text using flags and print specification fields. The following is the configuration for adding variables to the text.

%*flag* width .precision    Example Text: OIL PSI %-3d

% - identifies the beginning of a variable or other type of text entry

*flag* - This flag is optional. Use the following flags to change the way data is transmitted.

| **Flag** | **Description** |
|---|---|
| - | Left align the variable within the specified *width*. Default is align right. |
| 0 | If width is prefixed with 0, leading zeros are added until the minimum width is reached. If 0 and - are used together, the 0 is ignored. If 0 is specified in an integer format, the 0 is ignored. |

| | |
|---|---|
| *width* - | This flag is optional. Width is the number of characters that will be printed (total). |
| *.precision* - | This flag is optional. The precision is the number of digits after the decimal point when using REAL variables. |

**Variable Formats**

Variables are formatted based on the variable type. The following are supported variable types and their format.

| | | | |
|---|---|---|---|
| %d | Signed Integer | %X | Upper Case Hexadecimal |
| %u | Unsigned Integer | %f | Real or Float Variable |
| %x | Lower Case Hexadecimal | %b | binary |
| %o | Octal | | |

**Other Special Characters & Formats**

| **To Print** | **Use** | **To Print** | **Use** |
|---|---|---|---|
| % | %% | OFF / ON | %O |
| Boolean 0 or 1 | %d | FALSE / TRUE | %T |

| Examples: | Format | Result | Format | Result |
|---|---|---|---|---|
| | OIL: %d | OIL: 25 | OIL: %04d | OIL: 0025 |
| | LS1: %T | LS1: TRUE | LS1: %O | LS1: OFF |
| | TEMP: %6.2f | TEMP: 234.12 | TEMP: %3.f | TEMP: 234 |

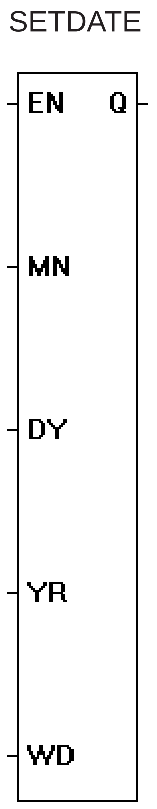
*Escape Sequences located on next page.*

Continued Next Page------»

**Special Printing Codes (Escape Sequences)**

| **Escape Sequence** | **Represents** |
|---|---|
| \a | Bell (Alert) |
| \b | Backspace |
| \f | Form Feed |
| \n | New Line |
| \r | Carriage Return |
| \t | Horizontal Tab |
| \’ | Single Quotation Mark |
| \” | Double Quotation Mark |
| \? | Literal Question Mark |
| \\ | BackSlash |
| \ooo | ASCII character in Octal notation |
| \xhh | ASCII character in Hexadecimal notation, stops on last non-hex character. |
| \uhhhh | Prints hexidecimal, always uses 4 characters |

## SETDATE



### Description:

The SETDATE function sets the current date on the hardware real time clock. The date is set by using variables to apply values to each of the inputs. The enable (EN) must be true for the SETDATE function to be enabled. The Q output is true when the function is enabled. The MN input sets the month (1-12), the DY input sets the day of the month (1-31) , the YR input sets the current year (4 digits) and the WD sets the day of the week (1-7, 1=Sunday). The MN, DY, YR and WD inputs must be connected to Integer variables.

The SETDATE function block will set the date to whatever values are provided on the variables. This actually allows you to set the real time clock using this function to UTC time by formatting the date / time correctly. The SETDTLOCAL, SETTZOFF and GETDTLOCAL function blocks are more appropriate when using UTC time.

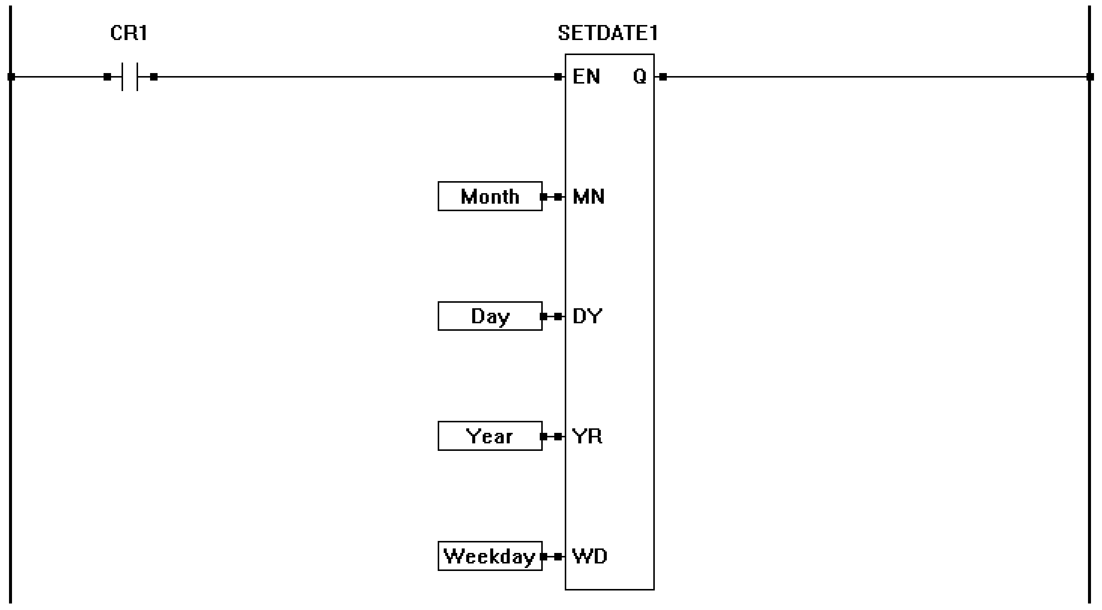
The real real time clock supports two or four digit year when setting, but 4 year is r ecommended. Four year date is required for Unix and UTC time.

### Input / Output Connections:

The SETDATE function block placement requires connections of 5 input pins (EN, MN, DY, YR, WD) and one output pin (Q).

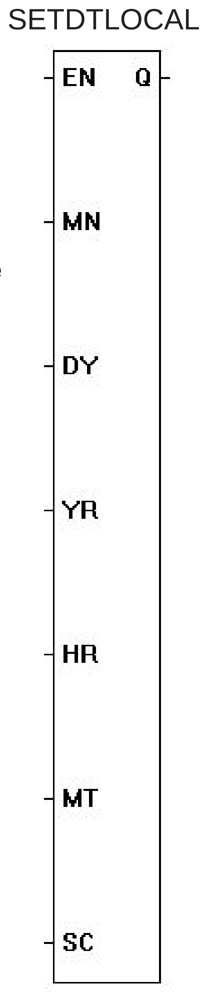
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| Q | Output | | | X | | |
| MN | Input | X | | | | |
| DY | Input | X | | | | |
| YR | Input | X | | | | |
| WD | Input | X | | | | |

### Example Circuit:



**Related Functions:** SETTIME, GETTIME, GETDATE, SETDTLOCAL, GETDTLOCAL, SETTZOFF

## SETDTLOCAL



### Description:

The SETDTLOCAL function block sets the current date and time to the hardware real time clock (stored as UTC time) from local time. The stored UTC time is converted using the time zone offset set by the SETTZOFF function block..

The values of the date and time are set by the integer variables on the input pins. The enable (EN) must be true for the SETDTLOCAL function to be enabled. The Q output is true when the function is enabled. The MN sets the month (1-12), the DY sets the day of the month (1-31), the YR sets the current year (last two digits) and the WD sets the day of the week (1-7, 1=Sunday). The MN, DY, YR and WD inputs must be connected to Integer variables. The HR input sets the hours (0-23), the MT input sets the minutes (0-59) and the SC input sets the seconds (0-59) The HR, MT and SC inputs must be connected to Integer variables.

The SETTZOFF function block must be ran each time the hardware target is started (power cycled) to set the time zone offset with it's value. This offset is not stored or kept in the controller in the event of a power loss or power cycle. Failure to run the SETTZOFF function block will result in incorrect time/date when using the SETDTLOCAL and GETDTLOCAL function blocks.

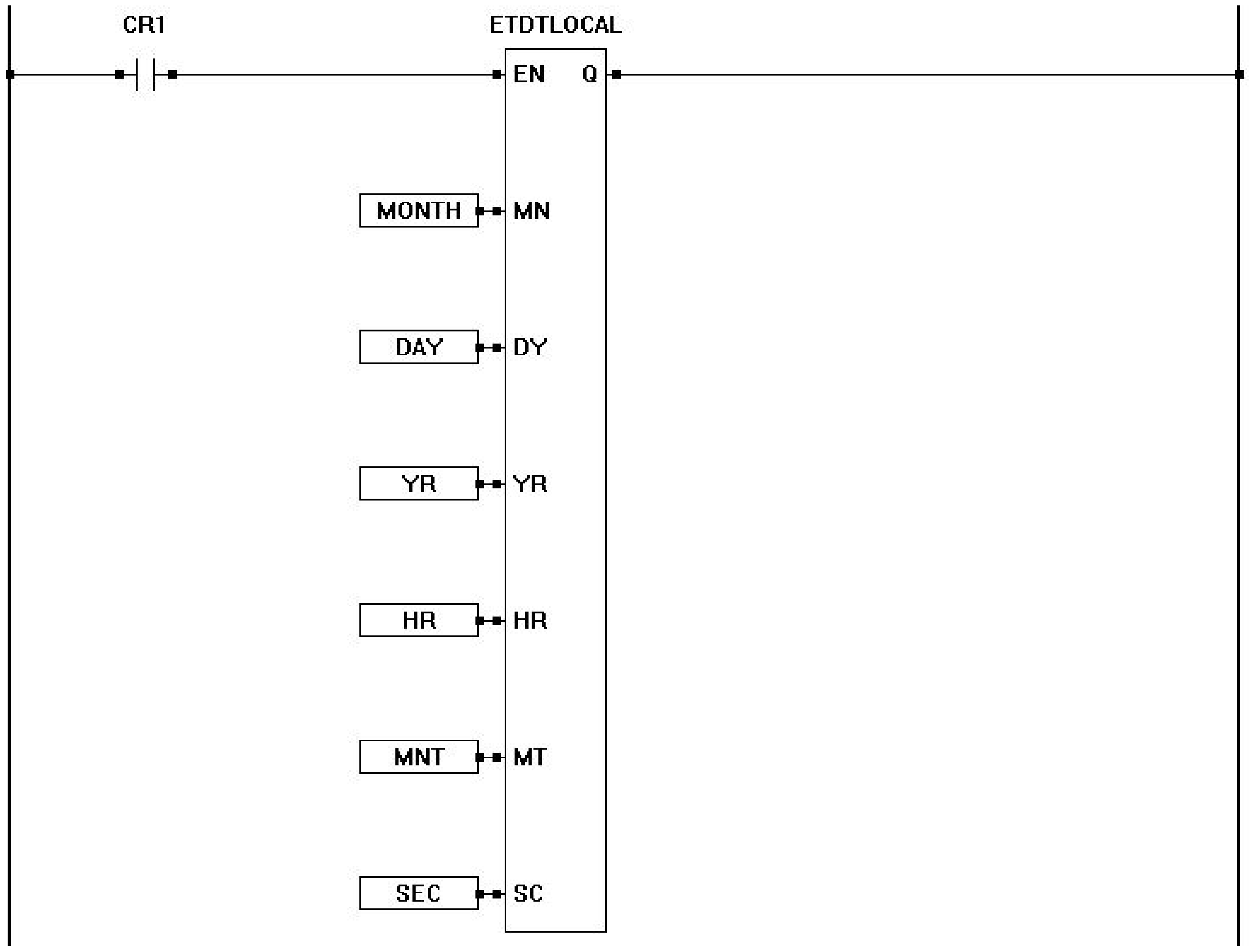
### Input / Output Connections:

The SETDTLOCAL function block placement requires connections of 9 input pins (EN, MN, DY, YR, WD, HR, MT, SC) and 1 output pin (Q).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| MN | Input | X | | | | |
| DY | Input | X | | | | |
| YR | Input | X | | | | |
| WD | Input | X | | | | |
| HR | Input | X | | | | |
| MT | Input | X | | | | |
| SC | Input | X | | | | |
| Q | Output | | | X | | |

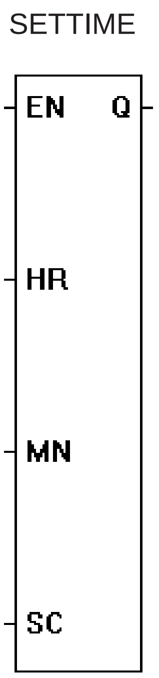
Continued Next Page------»

**Example Circuit:**



**Related Functions:** GETTIME, SETTIME, SETDATE, GETDTLOCAL, SETTZOFF

## SETTIME



### Description:

The SETTIME function sets the current time on the hardware real time clock. The time is set by using variables to apply values to each of the inputs. The enable (EN) must be true for the SETTIME function to be enabled.

The Q output is true when the function is enabled. The HR input sets the hour of the day (0-23) , the MN input sets the minutes (0-59) and the SC sets the seconds (0-59). The HR, MN and SEC inputs must be connected to Integer variables.

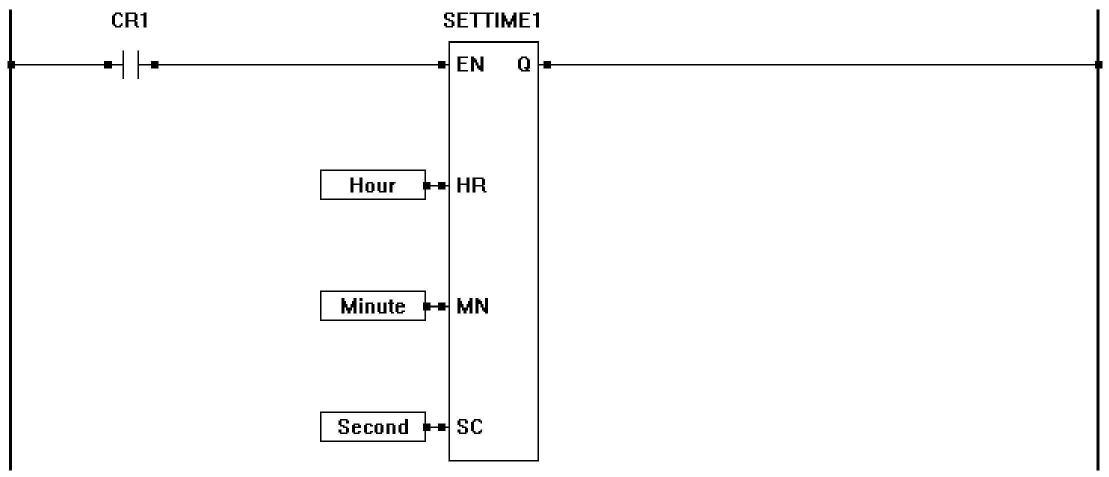
The SETTIME function block will set the time to whatever values are provided on the variables. This actually allows you to set the real time clock using this function to UTC time by formatting the date / time correctly. The SETDTLOCAL, SETTZOFF and GETDTLOCAL function blocks are more appropriate when using UTC time.

### Input / Output Connections:

The SETTIME function block placement requires connections of one input pin (EN) and four output pins (Q, HR, MN, SEC).

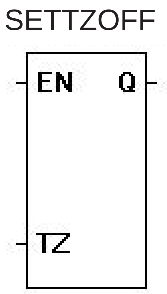
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| Q | Output | | | X | | |
| HR | Input | X | | | | |
| MN | Input | X | | | | |
| SC | Input | X | | | | |

### Example Circuit:



**Related Functions:** SETDATE, GETTIME, GETDATE, SETDTLOCAL, GETDTLOCAL, SETTZOFF

## SETTZOFF



### Description:
The SETTZOFF function block sets the current time zone offset for the real time clock when the real time clock is set to UTC time. The time zone offset  is set in minutes and the ladder diagram uses this offset when using SETDTLOCAL and GETDTLOCAL to set and read the local date time on the real time clock.The time zone offset is set by using a variable to apply the value to the input.  The enable (EN) must be true for the SETTZOFF function to be enabled.

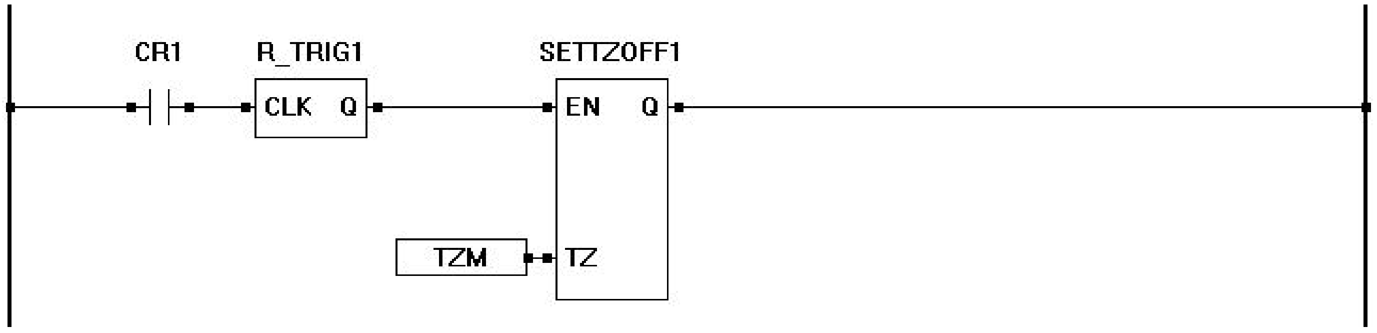
The Q output is true when the function is enabled.  The TZ input sets the time zone offset in minutes (minutes of the time zone you are in from UTC time).

The SETTZOFF function block must be ran each time the hardware target is started (power cycled) to set the time zone offset with it's value.  This offset is not stored or kept in the controller in the event of a power loss or power cycle. Failure to run this function block will result in incorrect time/date when using the SETDTLOCAL and GETDTLOCAL function blocks.

### Input / Output Connections:
The SETTIME function block placement requires connections of two input pins (EN, TZ) and one output pin (Q).

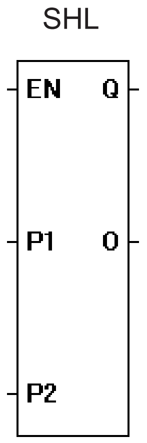
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State |
|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True |
| Q | Output | | | X | | |
| TZ | Input | X | | | | |

### Example Circuit:



**Related Functions:** SETDATE, GETTIME, GETDATE, SETDTLOCAL, GETDTLOCAL

## SHL



### Description:

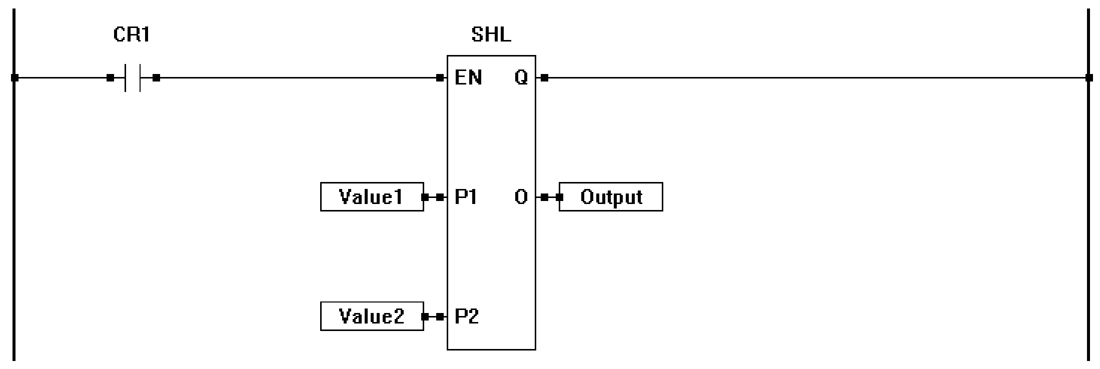
The SHL function provides a left bit shift of the P1 input. The P2 input specifies the number of one-bit left shifts. If the enable (EN) is false, the function is disabled. If the enable (EN) is true, the output (O) will be equal result of the left shifted input in integer form (1..2..4..8..16..32). A shift left when the output is 32 will cause the output to be zero (bit is shifted off). Zeros are always shifted on to the right side when a left shift occurs.

### Input / Output Connections:

The SHL function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

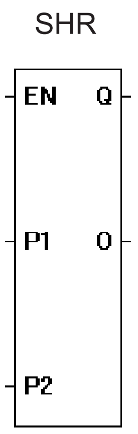
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| O | Output | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** SHR

## SHR



### Description:

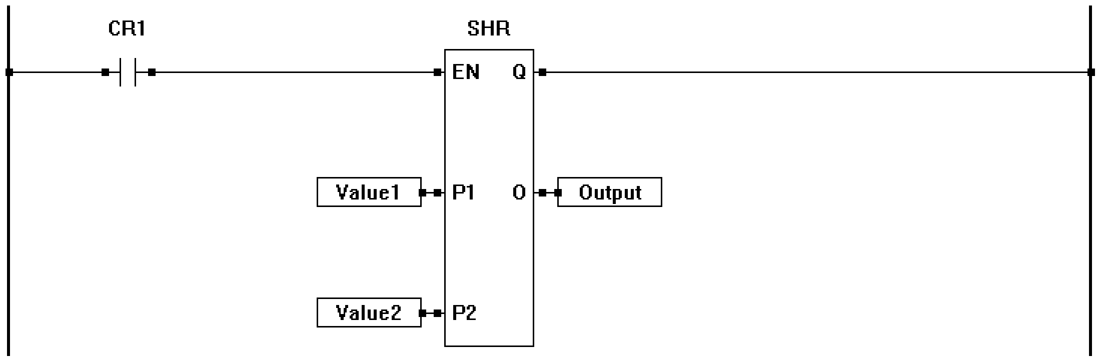
The SHR function provides a right bit shift of the P1 input. The P2 input specifies the number of one-bit right shifts. If the enable (EN) is false, the function is disabled. If the enable (EN) is true, the output (O) will be equal result of the right shifted input in integer form (32..16..8..4..2..1). A shift right when the output is 1 will cause the output to be zero (bit is shifted off). Zeros are always shifted on to the left side when a right shift occurs.

### Input / Output Connections:

The SHR function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

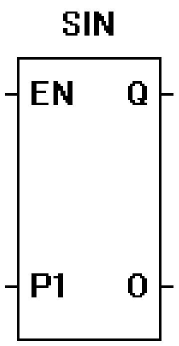
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| O | Output | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** SHL

## SIN



### Description:

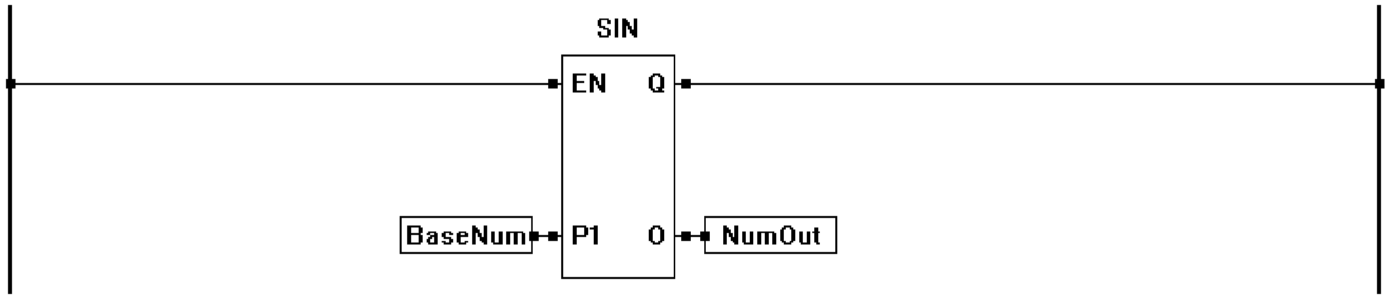
The SIN function provides the sine (O) from the input value (P1). The enable (EN) must be true for the SIN function to be enabled. The Q output is true when the SIN function is enabled.

### Input / Output Connections:

The SIN function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

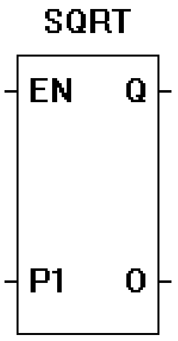
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Sine of P1 Base Number |

### Example Circuit:



**Related Functions:** ASIN, ATAN, COS, TAN, ACOS

## SQRT

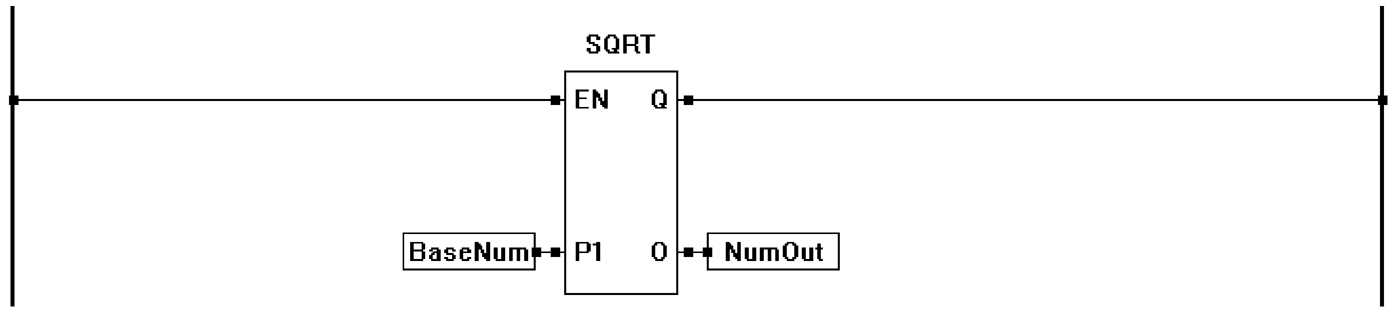


### Description:
The SQRT function provides the square root (O) from the input value (P1). The enable (EN) must be true for the SQRT function to be enabled. The Q output is true when the SQRT function is enabled.

### Input / Output Connections:
The SQRT function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

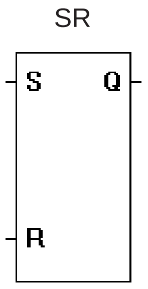
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Square Root of P1 Base Number |

### Example Circuit:



**Related Functions:** EXP, EXPT, LOG, LN

## SR

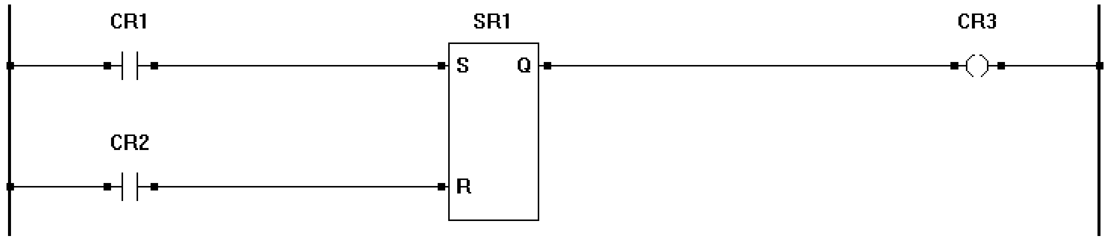


### Description:
The SR function acts as a set dominant bistable. If the set input (S) is true, the output (Q) is true. A true on the reset (R) input sets the output (Q) to false only if the set (S) input is also false.

### Input / Output Connections:
The SR function block placement requires connections of two input pins (S,R) and one output pin (Q).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| R | Input | | | X | | | |
| S | Input | | | X | | | |
| Q | Output | | | X | | | |

### Example Circuit:

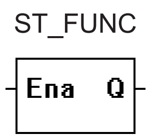


### Truth Table:

| SET | RESET | Q | Q RESULT |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

### Related Functions: RS

## ST_FUNC



### Description:

The ST_FUNC function is a function block for using structured-text functions created in EZ LADDER Toolkit. The ST_FUNC has only local variables and will not allow variables (items) to pass from one ladder scan to another (ie: keeping track of a count). As this uses less memory, it is ideal for using when scan-passing is not required as it uses less overhead (memory). The Enable is used to enable the function and the Q output is active when the function enable is true.

For understanding on Structured Text and how the ST_FUNC operates with structured text functions, refer to **Chapter 26 - Structured Text**.

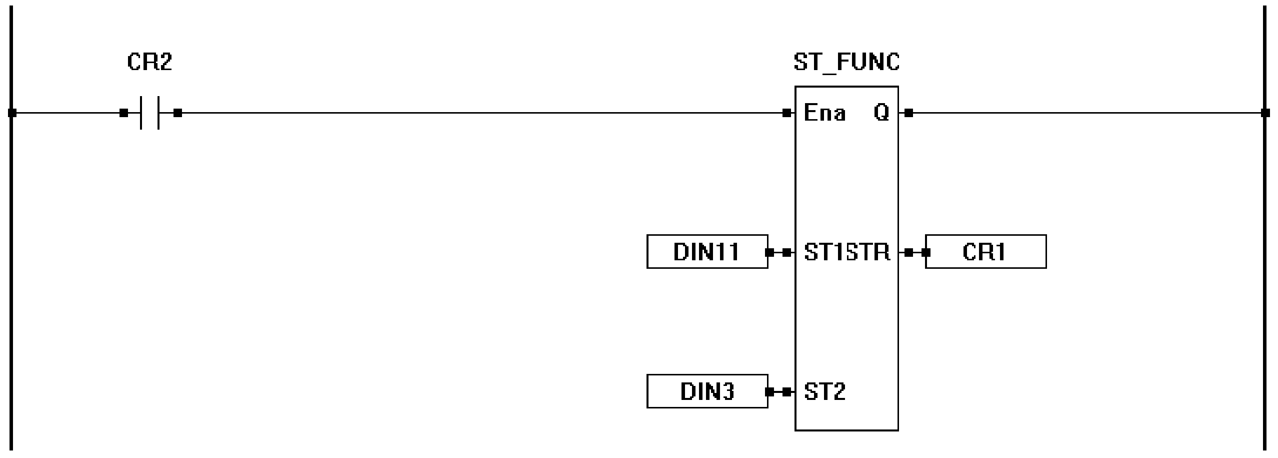
### Input / Output Connections:

**Before an ST_FUNC function may be used in a ladder diagram (called from a ladder diagram program, certain criteria must be met. Refer to Chapter 26-Structured Text for this criteria.**

Input and Output connections are based on the structured text functions you create in EZ LADDER (ST Function Editor). The Enable and Q outputs are by default created.

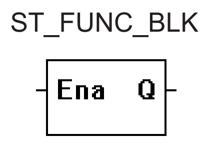
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| Enable | Input | | | X | | Active True | |
| Q | Output | | | X | | | True when EN is true |
| XX | XX | X | X | X | X | | Input/Output Name & Type based solely on User Structured Text Function. |

### Example Circuit:



**Related Functions:** ST_FUNC_BLK

## ST_FUNC_BLK



### Description:

The ST_FUNC_BLK function is a function block for using structured-text functions created in EZ LADDER Toolkit. The ST_FUNC may use local or global variables and will allow variables (items) to pass from one ladder scan to another (ie: keeping track of a count). This function uses more memory than the ST_FUNC function. The Enable is used to enable the function and the Q output is active when the function enable is true.

For understanding on Structured Text and how the ST_FUNC_BLK operates with structured text functions, refer to **Chapter 26 - Structured Text**.

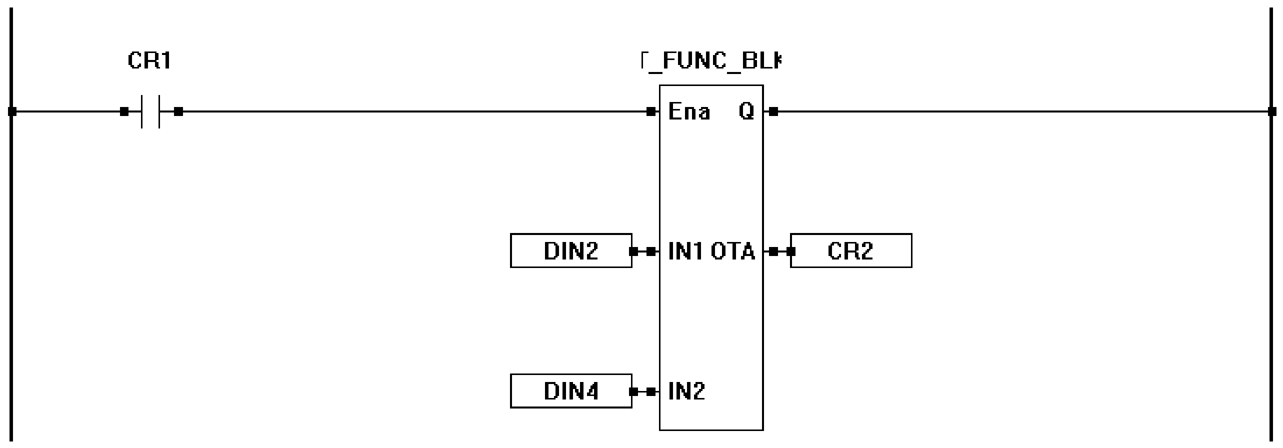
### Input / Output Connections:

**Before an ST_FUNC function may be used in a ladder diagram (called from a ladder diagram program, certain criteria must be met. Refer to Chapter 26-Structured Text for this criteria.**

Input and Output connections are based on the structured text functions you create in EZ LADDER (ST Function Editor). The Enable and Q outputs are by default created.

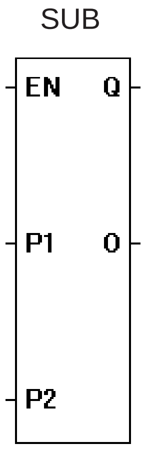
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| Enable | Input | | | X | | Active True | |
| Q | Output | | | X | | | True when EN is true |
| XX | XX | X | X | X | X | | Input/Output Name & Type based solely on User Structured Text Function. |

### Example Circuit:



**Related Functions:** ST_FUNC

## SUB



### Description:

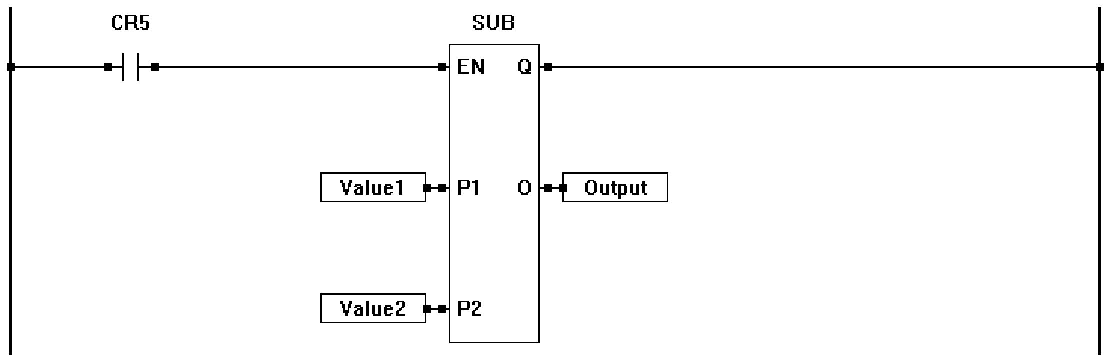
The SUB functions subtracts the P2 input from the P1 input. The output (O) is the result of the subtraction. The enable (EN) must be true for the SUB function to be enabled. The Q output is true when the SUB function is enabled.

### Input / Output Connections:

The SUB function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

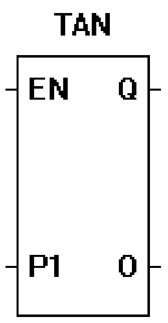
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | X | | | | |
| P2 | Input | X | X | | | | |
| O | Output | X | X | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** ADD, MULT, DIV, ABS

## TAN



### Description:

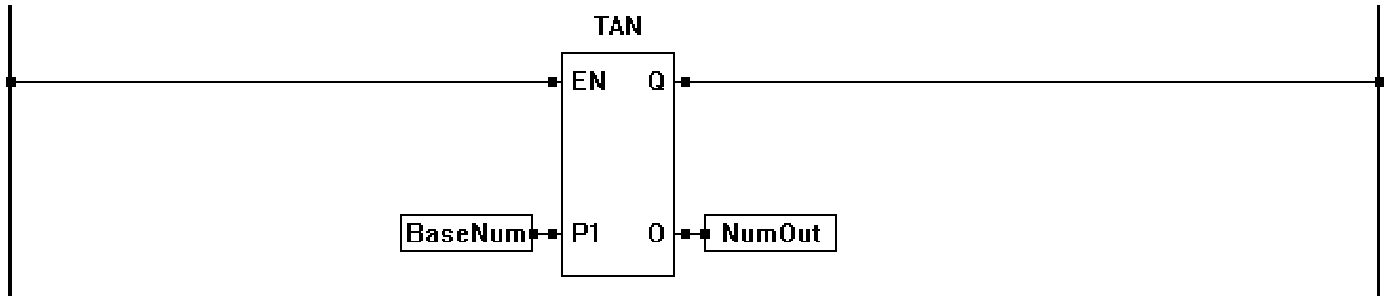
The TAN function provides the Tangent (O) from the input value (P1). The enable (EN) must be true for the TAN function to be enabled. The Q output is true when the TAN function is enabled.

### Input / Output Connections:

The TAN function block placement requires connections of two input pins (EN, P1) and two output pins (Q, O).

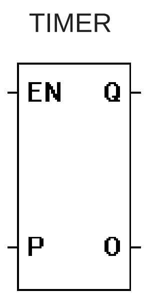
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | | X | | | | Base Number |
| Q | Output | | | X | | | |
| O | Output | | X | | | | Tangent of P1 Base Number |

### Example Circuit:



**Related Functions:** ASIN, ATAN, COS, SIN, ACOS

## TIMER



### Description:

The TIMER function converts the input (P) into an Timer output (O). The enable (EN) must be true for the TIMER function to be enabled. The Q output is true when the TIMER function is enabled. The O output is a representation of the P input value in milliseconds (5=5ms, 1000=1 Second)

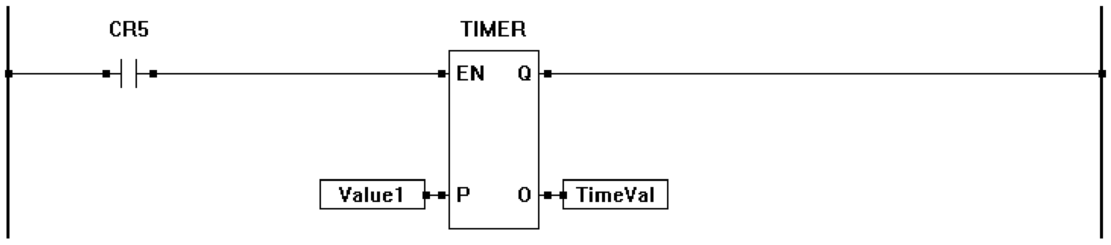
In addition to converting an Integer or Real to a Timer, the Timer function block can be used to copy one timer to another.

### Input / Output Connections:

The TIMER function block placement requires connections of two input pins (EN, P) and two output pins (Q, O).

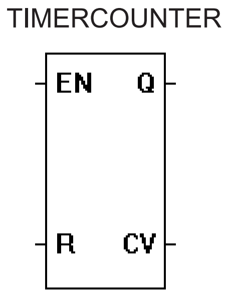
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | X | | X | | |
| O | | | | | X | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** INTEGER, REAL, BOOLEAN

## TIMERCOUNTER



### Description:

The TIMERCOUNTER function block is used to read counter or timer values of the real world inputs connected to the Timer / Counter Capture inputs (pins). The enable (EN) must be true for the TIMERCOUNTER function to be enabled. The Q output is true when the TIMERCOUNTER function is enabled. The CV output is the actual count or time value (based on how the input is configured). The reset (R) input is used to reset the counter or timer.

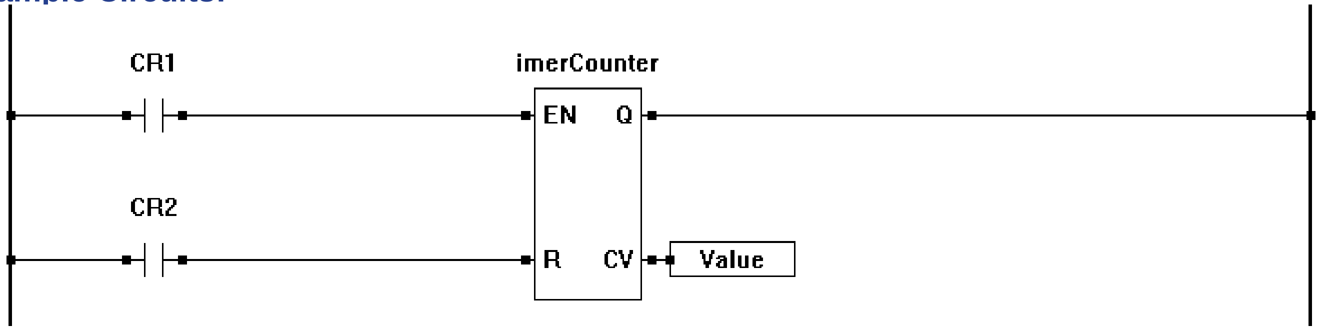
Counter / Timer Capture inputs may be configured as Timers, Free Running Timers or Counters. The capture input(s) must be configured in the EZ LADDER Toolkits Target Settings prior to placing in the ladder diagram.

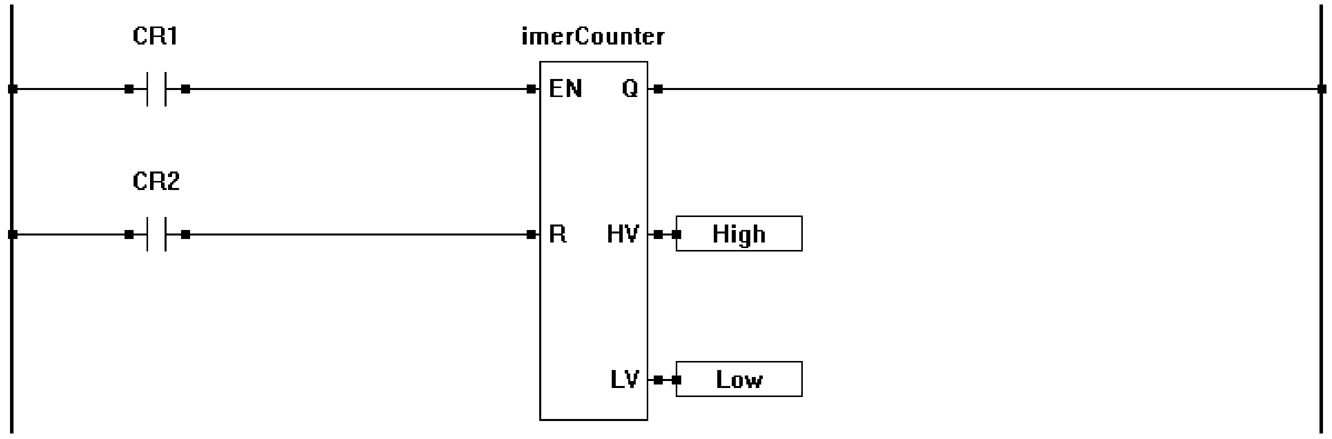
### Input / Output Connections:

The TIMER function block placement requires connections of two input pins (EN, R) and two output pins (Q, CV, HV, LV (based on type of counter timer configured).

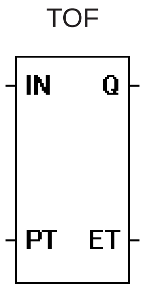
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| R | Input | | | X | | Active True | Resets Counte/Timer |
| CV | Output | X | | | | | Actual Count / Timer Value |
| Q | Output | | | X | | | |
| HV | Output | X | | | | | Time in ticks* Pulse is high (Measuing Duty Cycle only) |
| LV | Output | X | | | | | Time in ticks* Pulse is low (Measuing Duty Cycle only) |
| *Each Tick would be 1/24MHz or 1/24,000,000 | | | | | | | |

### Example Circuits:





## TOF



### Description:

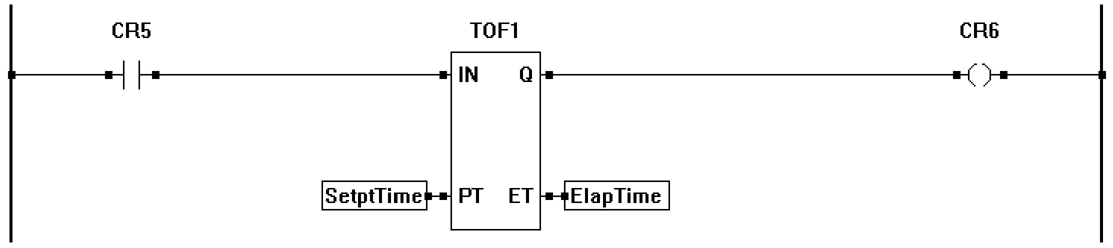
The TOF (off delay timer / time delay on drop-out) is a programmable timer with a variable turn-off time. When the input (IN) input is true, the output (Q) is true. When the input (IN) sees a transition from true to false, the timer begins timing. When the elapsed time (ET) is equal to the preset time (PT), the output (Q) de-energizes (goes false). When the input (IN) sees a false to true to false transition, the timer is reset and begins timing again.

### Input / Output Connections:

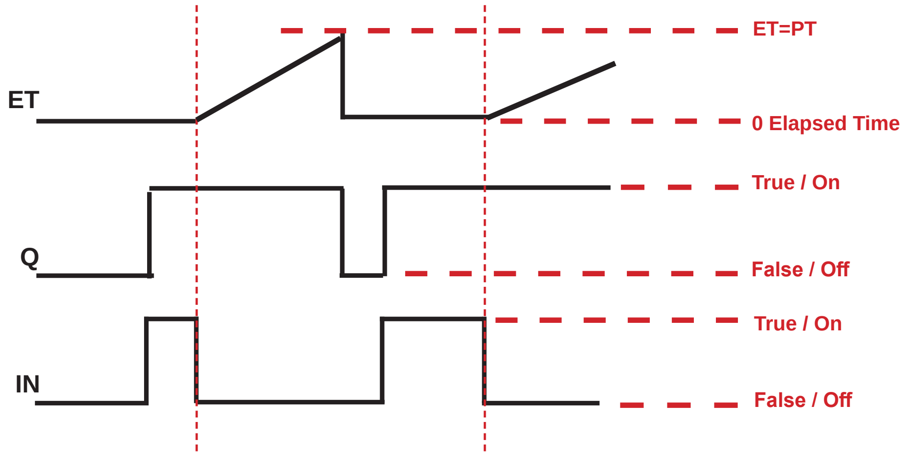
The TOF function block placement requires connections of two input pins (IN, PT) and two output pins (Q, ET).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| IN | Input | | | X | | Falling Edge | |
| PT | Input | | | | X | | |
| ET | | | | | X | | |
| Q | Output | | | X | | | |

### Example Circuit:

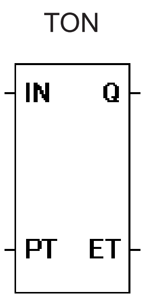


### Timing Diagram:



**Related Functions:** TON, TP

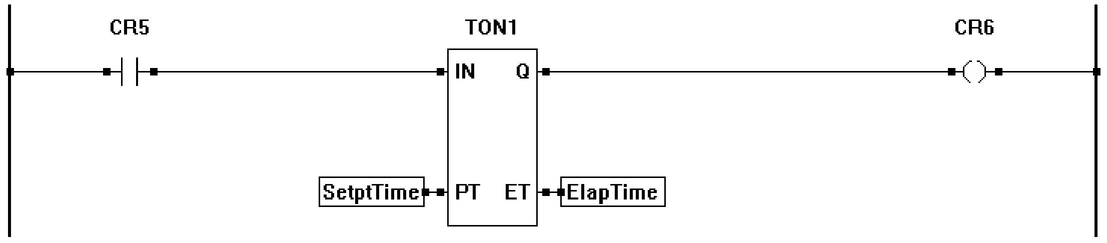
## TON

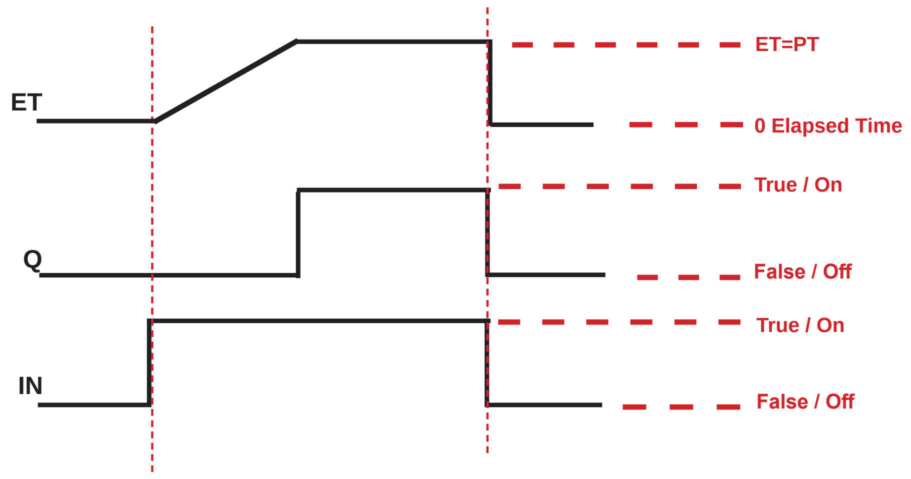


### Description:
The TON (on delay timer / time delay on pick-up) is a programmable timer with a variable turn-on time. When the input (IN) input is true, the timer begins timing. When the elapsed time (ET) is equal to the preset time (PT), the output (Q) energizes (goes true). When the input (IN) sees a true to false transition, the timer is reset and the output (Q) is de-energized (goes false).

### Input / Output Connections:
The TON function block placement requires connections of two input pins (IN, PT) and two output pins (Q, ET).

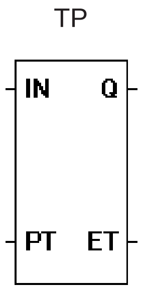
| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| IN | Input | | | X | | Active True | |
| PT | Input | | | | X | | |
| ET | | | | | X | | |
| Q | Output | | | X | | | |

### Example Circuit:


### Timing Diagram:


**Related Functions:** TOF, TP

## TP



### Description:

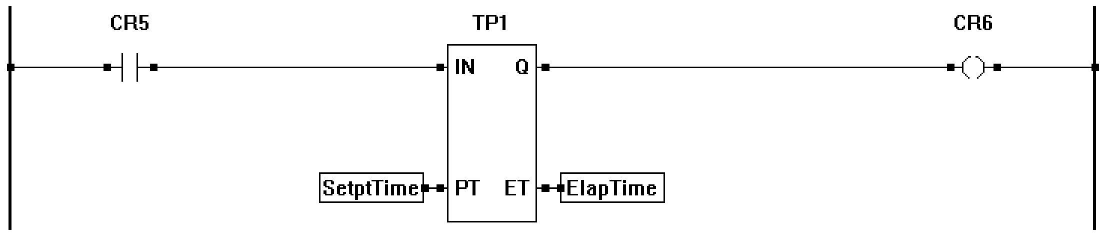
The TP (pulse timer) is a programmable one-shot timer with a variable turn-on time. When the input (IN) input is true, the timer begins timing and the output (Q) is energized. When the elapsed time (ET) is equal to the preset time (PT), the output (Q) de-energizes (goes false). When the input (IN) goes from true to false, the timer is only reset if the elapsed time (ET) is equal to the preset time (PT). If they are not equal, the reset will not occur until they are equal (and IN must still be false).

### Input / Output Connections:

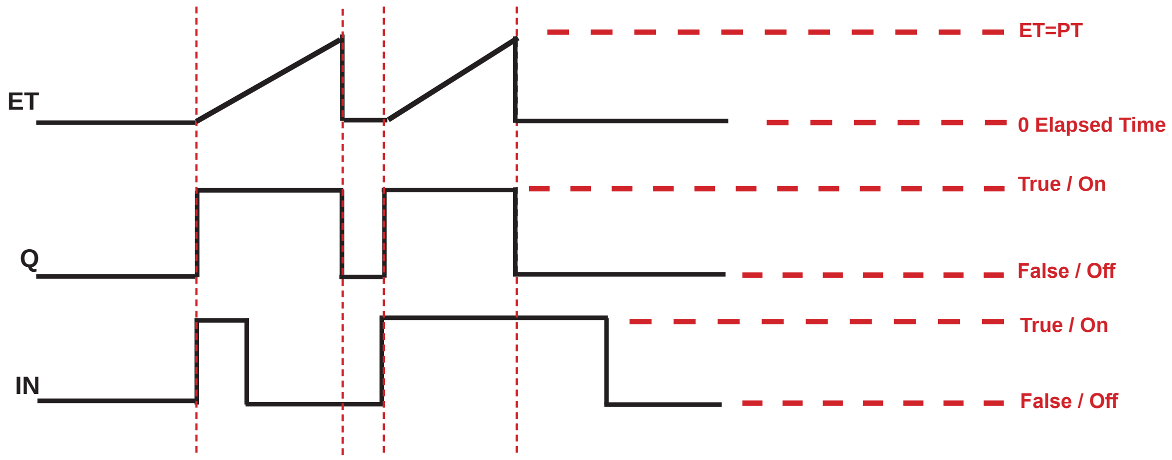
The TP function block placement requires connections of two input pins (IN, PT) and two output pins (Q, ET).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| IN | Input | | | X | | Active True | |
| PT | Input | | | | X | | |
| ET | | | | | X | | |
| Q | Output | | | X | | | |

### Example Circuit:

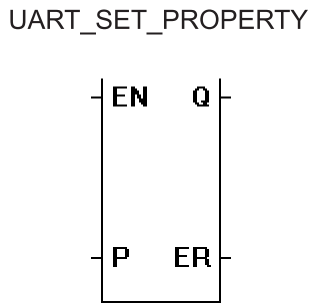


### Timing Diagram:



**Related Functions:** TON, TOF

## UART_SET_PROPERTY



### Description:

The UART_SET_PROPERTY function block allows for adjustment of certain UART parameters from inside the ladder program. The enable (EN) must be true for the UART_SET_PROPERTY function to be enabled. The Q output is true when the UART_SET_PROPERTY function is enabled. The ER output provides an error code should an error occur. The P is the input parameter to adjust.

A dialog box will automatically open when placing the UART_SET_PROPERTY function block. This dialog allows the selection of the UART and Property to adjust.

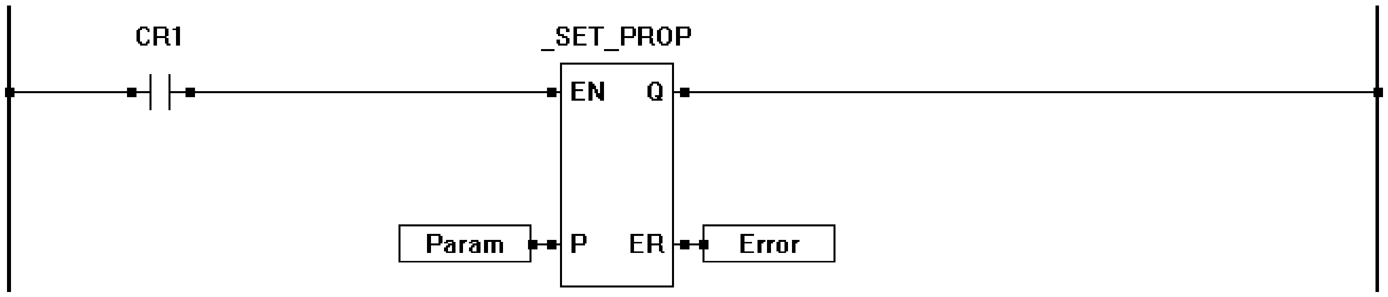
The Baud Rate is the only adjustable parameter as of this release version of EZ LADDER Toolkit.

### Input / Output Connections:

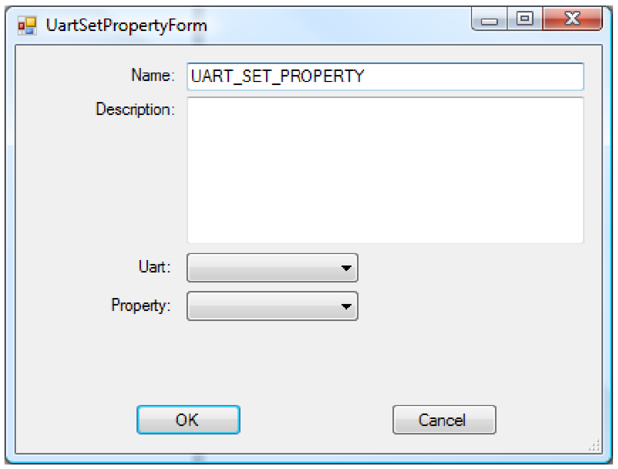
The UART_SET_PROPERTY function block placement requires connections of two input pins (EN, P) and two output pins (Q, ER).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P | Input | X | | | | | Parameter |
| ER | | X | | | | | Error Out Code, (0) = No Error |
| Q | Output | | | X | | | |

### Example Circuit:



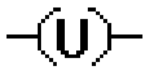
### Dialog Box:



Name: UART_SET_PROPERTY
Description:
Uart:
Property:

## UNLATCH (COIL)

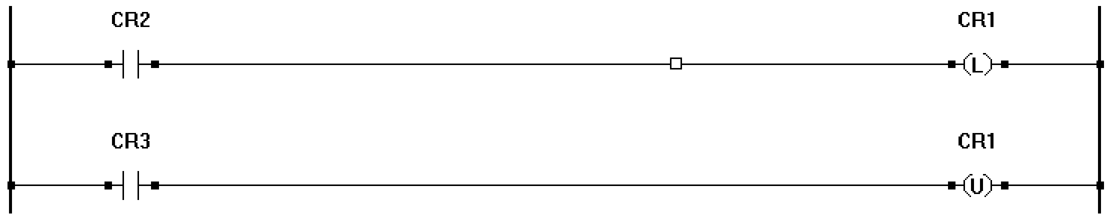
UNLATCH COIL



### Description:

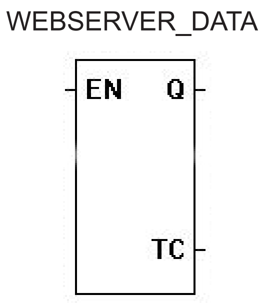
The UNLATCH coil is for use with the LATCH coil operates similar to the DIRECT COIL. The UNLATCH coil will clear it's latched counterpart (LATCH coil with same name). This will cause the LATCH coil to de-energize. LATCH and UNLATCH coils work as pairs. Any boolean variable can be used as a LATCH / UNLATCH coil.

### Example Circuit:



**Related Functions:** LATCH, DIRECT COIL, INVERTED COIL

## WEBSERVER_DATA



### Description:

The WEBSERVER_DATA function block is used in the ladder diagram as the interface to communicate via the embedded webserver to web enabled devices accessing the stored web files on the sd card. This function block allows for variables to be selected that are to be sent / received via the webserver interface.

A dialog box will automatically open when placing the WEBSERVER_DATA function block. This dialog allows the selection variables that may be accessed (read or written to) in the ladder diagram from the webserver page (HTML/JSON) and smart devices conneted to the web page via Ethernet or Wi-Fi.

The WEBSERVER_DATA function block is only available after the WEBSERVER feature has been added to the Project Settings and configured for operation. Multiple WEBSERVER function blocks may be used in the ladder diagram.

The WEBSERVER_DATA function block requires an active connection (Ethernet or Wi-Fi) for connecting to devices and develope web files to interact with the devices and the WEBSERVER_DATA function block. These files reside on the SD Card. Refer to Chapter 24 - Webserver for more information on the webserver features.

### Input / Output Connections:

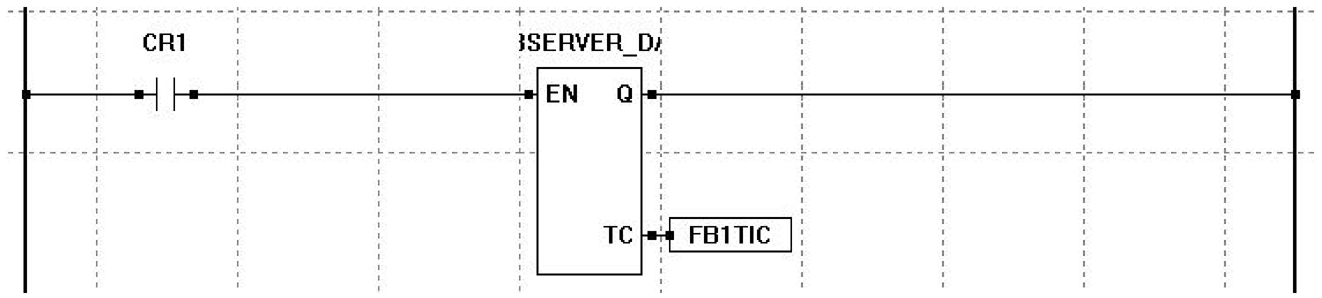
The WEBSERVER_DATA function block placement requires connections of one input pin (EN) and two output pins (Q, TC). Additional parameters are required in the Webserver Data Group Properties dialog when placing the WEBSERVER_DATA function block. See the Dialog Box section for the dialog box information.

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| Q | Output | | | X | | | True when EN is true |
| TC | Output | X | | | | | Current TIC (internal timer) count, updated when Function block communicates with webserver page (s) |

When the EN (Enable) is true, the WEBSERVER_DATA function block is active and available for communication to /from the webserver on the SD Card. If the EN (Enable) is false, any data in the block cannot be update or accessed to/from the webserver.

The TIC output changes each time the communication occurs between the WEBSERVER_DATA function block and the webserver. This value can be monitored in the ladder diagram to identify when communications have occured.

### Example Circuit:

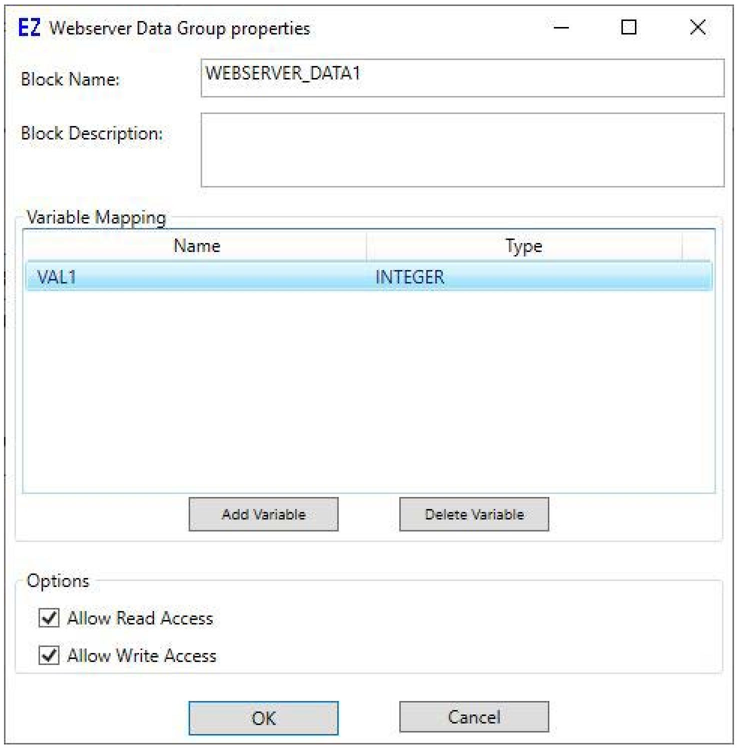


Continued Next Page------»

## Dialog Box:

When placing the WEBSERVER_DATA function block, the Webserver Data Group properties window will open automatically. This window is used to select variables (add and remove) that are used to communicate to the webserver and configure the amount of access (read access, write access or both). The checkboxes provide the amount of access. The EDIT VARIABLE and DELETE VARIABLE buttons are used to add and remove variables from the dialog.

Variables must already exist in the ladder diagram project before they can be selected using the WEBSERVER_DATA function block (added to).


Webserver Data Group properties

Block Name: WEBSERVER_DATA1

Block Description:

Variable Mapping

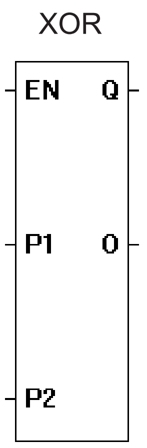
| Name | Type |
|---|---|
| VAL1 | INTEGER |

Options

☑ Allow Read Access

☑ Allow Write Access

## XOR



### Description:

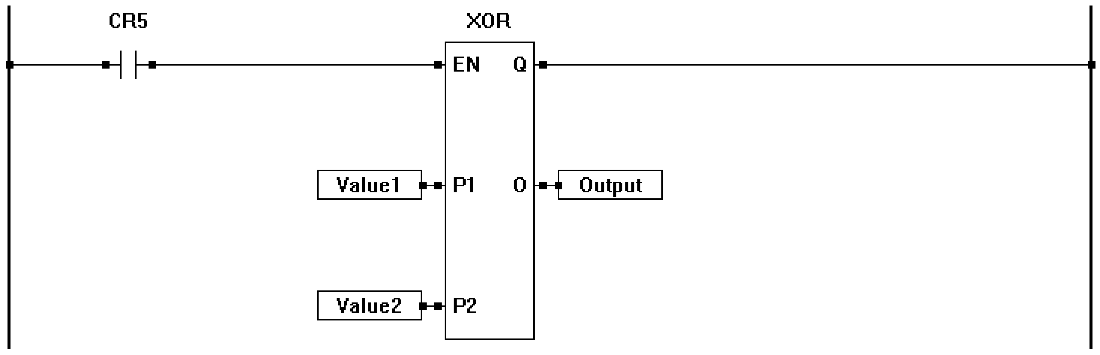
The XOR functions provides a bitwise exclusive OR function of the P1 and P2 inputs. The enable (EN) must be true for the XOR function to be enabled. The Q output is true when the XOR function is enabled.

### Input / Output Connections:

The XOR function block placement requires connections of 3 input pins (EN, P1, P2) and two output pins (Q, O).

| I/O Pin | Type | Integer | Real | Boolean | Timer | Active State | Other Details |
|---|---|---|---|---|---|---|---|
| EN | Input | | | X | | Active True | |
| P1 | Input | X | | | | | |
| P2 | Input | X | | | | | |
| O | Output | X | | | | | |
| Q | Output | | | X | | | |

### Example Circuit:



**Related Functions:** OR, AND, NOT

# Appendix B
## Target Specific ST Function Reference

This chapter provides information on using Target Specifi Structured Text Functions in EZ LADDER Toolkit.

## Chapter Contents

## Target Specific Functions

Target specific structured text functions are functions used in structured text that apply to specific hardware features such as Ethernet, Wi-Fi or cellular. These features are only available on certain controllers (hardware targets, and these functions for accessing these features are only available on supported hardware.

Target specific functions are not standard functions in structured text and as such are specific to EZ LADDER toolkit and only available on supported hardware

Most of the Target Specific functions are found in the structured text editor in the Target Specific Functions tab (2nd tab), though some may be found in the Std Functions Tab (1st tab). Organization and location is based on how broad the feature is supported across targets.

Regardless of the location, all Target specific functions, begin with EZ_ (exampe: EZ_GPIO_Write).

## EZ_CAN_Reset

### Summary:

The EZ_CAN_Reset function is reset the on-board CAN controller and re-initialize with the CAN settings in the Project Settings. Refer to **Chapter 14 - CAN Networking** for more details on CAN communications and Project Settings.

This function can be used at anytime to reset / re-initialize the on-board CAN controller. Often it is used Native CAN Communications (CAN communications at the raw frame level).

### Format:

EZ_CAN_Reset(*FileDescriptor)*;

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_CAN*x*** for the CAN port controller to reset. x is the CAN port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable. |

### Description:

The EZ_CAN_Reset function causes the on-board CAN (Controller Area Network) controller to reset and re-initialize with the CAN setting found in the Project Settings.

### Example:

```
FUNCTION_BLOCK ResetCAN
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF (Enable = TRUE) AND (LastEn = FALSE) THEN

                EZ_CAN_Reset(FD_CAN0);
                LastEn := TRUE;

        END_If;

        If (Enable = FALSE) then
                        LastEn := FALSE;
        END_IF;

        Q := Enable;
```

## EZ_CAN_Rx

### Summary:

The EZ_CAN_Rx function is to receive **Native CAN Communications** via a CAN port. Refer to **Chapter 14 - CAN Networking** for more details on CAN communications and Native CAN Communications.

This function as others with Native CAN Communications allow for sending / receiving CAN data at the raw frame level. Using these functions requires knowledge of CAN communications and CAN frame construction.

### Format:

*varDINT*:=EZ_CAN_Rx(*FileDescriptor, rtrflag, extFrame, ID, data, length)*;

### Arguments:

| | |
|---|---|
| *varDINT* | DINT. Returns status of function.<br>**Negative** Number for error<br>**0** when no message has been received<br>**1** message has been received |
| *FileDescriptor* | **FD_CAN*x*** for the CAN port controller to use. x is the CAN port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable. |
| *rtrflag* | BOOL. Remote Transmission request flag. Identifies when a remote tranmission request has been received. Most often this is zero. |
| *extFrame* | BOOL. Extended frame flag. False (0) for 11-bit CAN id, True (1) for 29-bit CAN id. |
| *ID* | UDINT. CAN PGN number to receive data from. |
| *data* | ARRAY OF USINT. Used as a buffer to store the received data into. |
| *length* | UDINT. Number of bytes received. |

### Description:

The EZ_CAN_Rx function is used to receive CAN data (Native CAN communications) at the raw frame level. Using the selected CAN port (*FileDescriptor*), the function will listen for message(s) from the specific PGN (*ID*) and store the data received in the *data* variable provided. The other communications flags are required to identify the type of CAN communications. *Length* and *rtrflag* are returned as well as the status of the function (*varDINT*) to identifty when messages are received.

### Example:

```
FUNCTION_BLOCK RX_CAN
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        VAR
                rtr : BOOL;                     (* Remote Transmission Request flag *)
                extFrame : BOOL;                (* This is a flag if the frame coming in is a 11 bit or 29 bit identifier*)
                id : udint;                (*This is the PGN number *)
                data : array[0..7] of USINT;  (* the length and type of array to store the CAN bytes *)
                rdCnt : udint;             (* Number of bytes received*)
                Err1 : bool;                    (* return state of the RX command *)
                Infomation : dint;              (* Varible to hold data frm parsed received message*)
                Measure_temp : real;       (* Varible to hold data frm parsed received message*)
```

Continued Next Page------»

```
        END_VAR


        if(Enable) then


                Err1 :=EZ_CAN_Rx(FD_CAN0, rtr, extFrame, id, data, rdCnt);  (*"FD_CAN" is the file directory*)

                IF (id = 16#2C4 )then     (* IF the recived data array has this as the id/PGN then parse the bytes to
these Variables*)

                        Infomation := TO_LSB_UDINT(data ,16#0); (* The first 4 Bytes of the data array is stored in a
Udint variable .*)

                        Measure_temp := INT_TO_REAL (TO_LSB_INT(data ,16#4)) * 10.0; (*the next two bytes start-
ing at 4  stored in a LSB INT and then a INT to real conversion *)

                end_if;


        Q := Enable;


        end_if;

END_FUNCTION_BLOCK
```

## EZ_CAN_SetBitRate

**Summary:**

The EZ_CAN_SetBitRate function is to configure CAN port bit (baud) rate. When used, this function would affect all communications on the CAN port. It would typically be used with **Native CAN Communications** via a CAN port. Refer to **Chapter 14 - CAN Networking** for more details on CAN communications and Native CAN Communications.

Using this functions requires knowledge of CAN communications and CAN frame construction.

This function should only be used with a high level of knowledge of CAN communications and message structure (or under direct consultation from Divelbiss). Use of this function without proper understanding of CAN communications, may result in undesired operation of the CAN network.

**Format:**

*varDINT*:=EZ_CAN_SetBitRate(*FileDescriptor, prescale, tseg1, tseg2, swj, numsamples)*;

**Arguments:**

| | |
|---|---|
| *varDINT* | DINT. Returns status of function.<br>**Negative** Number for error<br>**0** for success |
| *FileDescriptor* | **FD_CAN*x*** for the CAN port controller to use. x is the CAN port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable. |
| *prescale* | DINT*.* Baud Rate Presecale |
| *tseg1* | DINT*.* Time Segment 1. |
| *tseg2* | DINT. Time Segment 2. |
| *sjw* | DINT. Synchronization Jump Width. |
| *numsamples* | DINT. Number of Samples |

**Description:**

The EZ_CAN_SetBitRate function is used to the CAN port settings for communications (bit rate, etc). This should typically be used with Native CAN communications as J1939, NMEA 2000 and OptiCAN are bit rates are configured in the Project Settings (SEE WARNING ABOVE).

For P-Series PLC on a Chip targets, the CAN clock rate is 96 MHz. This frequency must be known when using the function.

To set 250K bit rate, the function would look like: ***var:=EZ_CAN_SetBitRate(fd, 24, 13, 2, 1, 1);***
To set 250K bit rate, the function would look like: ***var:=EZ_CAN_SetBitRate(fd, 12, 13, 2, 1, 1);***

**Example:**

```
FUNCTION_BLOCK FB_SetBitrate
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR
```

Continued Next Page------»

```
        Q := Enable;

        if(NOT lastEn AND Enable) then (* on the rising edge or enable*)
                EZ_CAN_SetBitRate(FD_CAN0, 24, 13, 2, 1, 1);
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_CAN_Status

### Summary:

The EZ_CAN_Status function is to query and get the current status of an on-board CAN controller. It would typically be used with **Native CAN Communications** to determine status, but the function does operate with J1939, NMEA 2000 and OptiCAN. Refer to **Chapter 14 - CAN Networking** for more details on CAN communications and Native CAN Communications.

This function returns information from the on-board CAN controller status. Understanding the status of a CAN controller requires knowledge of CAN communications and conditions that may be seen on network bus.

### Format:

*varUDINT:*=EZ_CAN_Status(*FileDescriptor, txErrors, rxErrors)*;

### Arguments:

*varUDINT* UDINT. Returns status of function.

**0** for No Error

If the status is not zero, then mutiple errors can be set at the same time as they are bit flags. The status would need to be BITWISE ANDed with each value below to identify if that specific flag (error bit) is set.

| | |
|---|---|
| Rx Warning | : 1 (BITWISE AND *varUDINT* with 1) |
| Rx Error | : 1 (BITWISE AND *varUDINT* with 2) |
| Tx Warning | : 1 (BITWISE AND *varUDINT* with 4) |
| Tx Error | : 1 (BITWISE AND *varUDINT* with 8) |
| Bus Error | : 1 (BITWISE AND *varUDINT* with 16) |
| Bus Off | : 1 (BITWISE AND *varUDINT* with 32) |
| Data Overrun | : 1 (BITWISE AND *varUDINT* with 64) |

*FileDescriptor* **FD_CAN*x*** for the CAN port controller to use. x is the CAN port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable.

*txErrors* DINT*.* Transmit Error Count

*rxErrors* DINT*.* Receive Error Count.

### Description:

The EZ_CAN_Status function is used read the status of the on-board CAN controller. This function returns errors if any are seen and has bit flags that can be masked to identify the type of error. Receive and Transmist error counters are also returned.

This function is useful to monitor the status of the CAN network (especially for Native CAN communications) and then take appropriate action based on the type of errors such as restarting the controller, resetting the CAN controller, etc.

### Example:

```
FUNCTION_BLOCK FB_Status
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                stat : udint;
                txErr : dint;
                rxErr : dint;
```

Continued Next Page------»

```
                lastEn : bool;
        END_VAR;

        Q := Enable;

        stat := EZ_CAN_Status(FD_CAN0, txErr, rxErr);

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_CAN_Tx

### Summary:

The EZ_CAN_Tx function is to transmit **Native CAN Communications** via a CAN port. Refer to **Chapter 14 - CAN Networking** for more details on CAN communications and Native CAN Communications.

This function as others with Native CAN Communications allow for sending / receiving CAN data at the raw frame level. Using these functions requires knowledge of CAN communications and CAN frame construction.

### Format:

*varDINT:*=EZ_CAN_Tx(*FileDescriptor, rtrflag, extFrame, ID, data, length)*;

### Arguments:

| | |
|---|---|
| *varDINT* | DINT. Returns status of function.<br>**Negative** Number for error<br>**Greater than 0** Message has been added to the CAN port's transmit buffer with the number returned being the number of bytes sent. |
| *FileDescriptor* | **FD_CAN*x*** for the CAN port controller to use. x is the CAN port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable. |
| *rtrflag* | BOOL*.* Remote Transmission request Identifies when a remote tranmission request has been received. Most often this is zero. Set to '1' or true for remote transmission request. |
| *extFrame* | BOOL*.* Extended frame flag. False (0) for 11-bit CAN id, True (1) for 29-bit CAN id. |
| *ID* | UDINT*.* CAN PGN number to transmit data as. |
| *data* | ARRAY OF USINT. Variable of data to send. Up to 8 bytes can be transmitted in a single frame. |
| *length* | UDINT. Number of bytes to transmit. This number must be greater than or equal to 0 and less than or equal to 8. |

### Description:

The EZ_CAN_Tx function is used to transmit CAN data (Native CAN communications) at the raw frame level. Using the selected CAN port (*FileDescriptor*), the function will load messages into the CAN transmit buffer from the *data* variable provided (to send on the CAN network) using the provided *ID* (PGN). The other communications flags are required to identify the type of CAN communications and size of the message : *extFrame, rtrflag* and *length*.

### Example:

```
FUNCTION_BLOCK FB_TX_CAN
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                cnt : dint;  (*  Number of bytes sent to buffer *)
                rtr : BOOL := false; (*  RTR = Remote Frame Flag   *)
                extFrame : BOOL := true; (*FLAG to set, if you are sending an 11 bit or a 29 bit identifier message*)
                id : udint;   (*PGN*)
                data : array[0..7] of USINT;
                wrCnt : udint;  (*Number of bytes to write/TX *)
                str : string[30];  (*Not used here, but could sent as a string *)
```

Continued Next Page------»

```
                lastEn : bool;
        END_VAR

        Q := Enable;

        if(NOT lastEn AND Enable) then
                wrCnt := 8;
                data[0] := 8; (*First byte of data in the array is 8*)
                data[1] := 7; (*First byte of data in the array is 7*)
                data[2] := 6;  (* ect...*)
                data[3] := 5;
                data[4] := 4;
                data[5] := 3;
                data[6] := 2;
                data[7] := 1;

                (*cnt := EZ_CAN_Rx(FD_CAN0, rtr, extFrame, id, data, rdCnt);*) (* This is the format example *)
                cnt := EZ_CAN_Tx(FD_CAN1, rtr, extFrame, 16#5678, data, wrCnt);


        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_ApplyPower

### Summary:

The EZ_Cell_ApplyPower function is to control the power of the on-board cellular data modem (CDM). This function directly controls the on/off power circuit to the cellular data modem.

### Format:

*INTvar* := EZ_Cell_ApplyPower(*PowerBool*);  or EZ_Cell_ApplyPower(*PowerBool*);

### Arguments:

| | |
|---|---|
| *INTvar* | Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram. |
| | 0 No Error |
| | -1 Null Pointer (Contact Divelbiss for more information) |
| | -2 Initialization Failed (Contact Divelbiss for more information) |
| | -3 Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| | -4 No Connection. A connection could not be made to the cellular network. |
| | -5 Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| | -6 Undefined (Contact Divelbiss for more information) |
| | -7 Signal Strength too low. Retry after better signal strength is detected. |
| | -8 Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| | -9 Not Supported. This function is not supported by the target hardware. |
| *PowerBool* | Control state for modem power ( 0 = Off / not powered, 1 = On / powered) (BOOL). |

### Description:

The EZ_Cell_Activate function controls the On / Off power control circuit for the on-board cellular data modem (CDM). The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *PowerBool* controls the state (0 for off / no power , 1 for on / powered). When powering the CDM on, there will be a delay before the CDM features are operational. See **Chapter 27 - Cellular Connectivity**.

### Example:

```
FUNCTION_BLOCK CellModemPower
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        EZ_Cell_ApplyPower(Enable);          (*Power Cellular data modem on if Enable is true*)

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_Connect

### Summary:

The EZ_Cell_Connect function is to control the cellular data modem (CDM) to connect to the cellular network or disconnect from the cellular network.

### Format:

*INTvar* := EZ_Cell_Connect(*StateBool*); or EZ_Cell_Connect(*StateBool*);

### Arguments:

| | |
|---|---|
| *INTvar* | Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram. |
| | 0 No Error |
| | -1 Null Pointer (Contact Divelbiss for more information) |
| | -2 Initialization Failed (Contact Divelbiss for more information) |
| | -3 Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| | -4 No Connection. A connection could not be made to the cellular network. |
| | -5 Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| | -6 Undefined (Contact Divelbiss for more information) |
| | -7 Signal Strength too low. Retry after better signal strength is detected. |
| | -8 Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| | -9 Not Supported. This function is not supported by the target hardware. |
| *StateBool* | Control state for connect / disconnect ( 0 = disconnect, 1 = connect) (BOOL). |

### Description:

The EZ_Cell_Connect function controls the connection of cellular data modem (CDM) to the cellular network. The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *StateBool* controls the state of the connection to the network (0 for disconnect, 1 for connect). Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem. The cellular data modem state (using **EZ_Cell_GetState**) must be **IDLE** and good registration (using **EZ_Cell_GetRegistration**) and acceptable signal strength (using **EZ_Cell_GetSignalStrength**) for successful implementation of this function (the cellular data modem must be powered on and in an **IDLE** state for this command to function or an error will occur. Power is applied using the **EZ_Cell_ApplyPower** function).

### Example:

```
FUNCTION_BLOCK CellModemConnection
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEnable : bool;                          (*Used for rising edge detect*)
        END_VAR

        IF (Enable = true and lastEnable = false) THEN  (*Look for rising edge only*)
                EZ_Cell_Connect(true);                      (*On rising edge, connect to network*)
        ELSE IF (Enable = false) THEN
                EZ_Cell_Connect(false);                     (* Enable = false then disconnect*)
        END_IF;
        END_IF;
```

Continued Next Page------»

```
        lastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetAPN

**Summary:**

The EZ_Cell_GetAPN function is to read the Access Point Name of the device's cellular data modem. This name can be useful when creating a gateway between youir carrier's cellular network and the public internet.

**Format:**

*INTvar* := EZ_Cell_GetAPN(*IDstring*)

**Arguments:**

*INTvar* Function return holding variable (INT). Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram. The CDM cell state must be IDLE to use this function.

| | |
|---|---|
| 0 | No Error |
| Non 0 | Error |

*IDstring* The string variable to store the APN into. The variable string length must be declared with a large enough size for the ID size (length). On success, the *IDstring* will contain the APN.

**Description:**

The EZ_Cell_GetIAPN communicates to the CDM (cellular data modem). The *INTvar* return variable returns a 0 for No Error or a non-zero number for errors. The variable *IDstring* returns the value of the Access Point Name (APN).

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem. The cellular data modem state (using **EZ_Cell_GetState**) must be **IDLE** (the cellular data modem must be powered on and in an **IDLE** state for this command to function or an error will occur. Power is applied using the **EZ_Cell_ApplyPower** function).

**Example:**

```
FUNCTION_BLOCK RdAPN
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
                ID : string[30];
                Rtn: int;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        if(Enable = True) AND (lastEn = False) then

                 Rtn:= EZ_Cell_GetIAPN(ID);

        end_if;

  Q := Enable;
  lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetICCD

### Summary:

The EZ_Cell_GetICCD function is to read the cellular 4G SIM card Identififcation.

### Format:

*INTvar* := EZ_Cell_GetICCD(*IDstring*)

### Arguments:

| | |
|---|---|
| *INTvar* | Function return holding variable (INT). Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram. The CDM cell state must be IDLE to use this function. |

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

| | |
|---|---|
| *IDstring* | The string variable to store the SIM card identification into. The variable string length must be declared with a large enough size for the ID size (length). On success, the *IDstring* will contain the SIM card serial number / ID. |

### Description:

The EZ_Cell_GetICCD communicates to the CDM (cellular data modem). The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *IDstring* returns the value of the CDM SIM card serial number / ID.

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem. The cellular data modem state (using **EZ_Cell_GetState**) must be **IDLE** (the cellular data modem must be powered on and in an **IDLE** state for this command to function or an error will occur. Power is applied using the **EZ_Cell_ApplyPower** function).

### Example:

```
FUNCTION_BLOCK RdSIMcardID
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
                ID : string[30];
                Rtn: int;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        if(Enable = True) AND (lastEn = False) then
```

Continued Next Page------»

```
                        Rtn:= EZ_Cell_GetICCID(ID);

            end_if;

    Q := Enable;
    lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetIMEI

### Summary:

The EZ_Cell_GetIMEI function returns the cellular data modem's identification number. IMEI numbers are hard coded to the cellular data modem and cannot be changed. This function is used for all 4G Cellular Data Modems (see target information for modem type).

### Format:

*INTvar* := EZ_Cell_GetIMEI(*IMEIstring*); or
EZ_Cell_GetIMEI(*IMEIstring*);

### Arguments:

*INTvar* Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram.

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

*IMEIstring* Cellular data modem IMEI number (identification number). Should be at 15 bytes to prevent being truncated. (STRING)

### Description:

The EZ_Cell_GetIMEI function retrieves the IMEI (cellular data modem's identification number) The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *IMEIstring* (STRING) holds the IMEI data.

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem.

### Example:

```
FUNCTION_BLOCK CellIMEI
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                complete : BOOL := FALSE;
                buffer : string[100];
                IMEI : string[15];
                result : int;
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE))THEN     (*Check for rising edge*)
```

Continued Next Page------»

```
                result := EZ_Cell_GetIMEI(IMEI);            (*Get IMEI from 4G Cellular Data Modem*)
                IF (result = 0) THEN
                        (* Print IMEI to Serial Port *)
                        EZ_FormatString(buffer, 'IMEI: ');
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
                                ;
                        end_while;
                        while EZ_UartWriteStr(FD_UART2,IMEI) <= 0 do
                                ;
                        end_while;
                        (* Print Carriage Return and Line Feed *)
                        EZ_FormatString(buffer, '$N');
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
                                ;
                        end_while;
                END_IF;

                complete := TRUE;

        END_IF;

        IF (Enable = FALSE) THEN
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetIPV4Addr

### Summary:

The EZ_Cell_GetIPV4Addr function returns the IP Address of the cellular data modem (CDM) from the cellular network.

### Format:

*INTvar* := EZ_Cell_ApplyPower(*OurIP,Netmask,DNS1,DNS2*); or
EZ_Cell_ApplyPower(*OurIP,Netmask,DNS1,DNS2*);

### Arguments:

| | |
|---|---|
| *INTvar* | Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram. |

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

| | |
|---|---|
| *OurIP* | Cellular data modem IP address (4 Bytes or more total) (ARRAY[ ] of BYTE). |
| *Netmask* | Cellular data modem subnet address (4 Bytes or more total) (ARRAY[ ] of BYTE). |
| *DNS1* | Cellular data modem DNS1 address (4 Bytes or more total) (ARRAY[ ] of BYTE). |
| *DNS2* | Cellular data modem DNS2 address (4 Bytes or more total) (ARRAY[ ] of BYTE). |

### Description:

The EZ_Cell_GetIPV4Addr function returns the IP address information of the cellular data modem on the cellular network. The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *OurIP* (array of BYTE) holds the IP address, the variable *Netmask* (array of BYTE) holds the subnet, the variable *DNS1* (array of BYTE) holds the DNS1 address, the variable *DNS2* (array of BYTE) holds the DNS2 address. This function is only valid when the CDM is connected to the cellular network (connect using **EZ_Cell_Connect** and verify connected using **EZ_Cell_GetState).**

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem.

### Example:

```
FUNCTION_BLOCK CellModemGetIPV4Addr
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                IP1: DINT;
                IP2: DINT;
                IP3: DINT;
                IP4: DINT;
        END_VAR
```

Continued Next Page------»

```
	VAR
		enableLast : bool;
		OURIP : ARRAY[0..3] OF BYTE;
		NETMASK : ARRAY[0..3] OF BYTE;
		DNS1 : ARRAY[0..3] OF BYTE;
		DNS2 : ARRAY[0..3] OF BYTE;
	END_VAR

	IF ((Enable = TRUE) AND (enableLast = FALSE)) THEN		(*Detect rising edge*)

		EZ_Cell_GetIpV4Addr(OURIP,NETMASK,DNS1,DNS2);		(*Get IP Info*)

		IP1 := OURIP[0];					(*Output IP only to ladder diagram*)
		IP2 := OURIP[1];
		IP3 := OURIP[2];
		IP4 := OURIP[3];

	END_IF;

	enableLast := Enable;
	Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetModelName

### Summary:

The EZ_Cell_GetModelName function is to read the cellular modem (CDM) model name.

### Format:

*INTvar* := EZ_Cell_GetModelName(*Modelstring*)

### Arguments:

| | |
|---|---|
| *INTvar* | Function return holding variable (INT). Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram. The CDM cell state must be IDLE to use this function. |

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

| | |
|---|---|
| *Modelstring* | The string variable to store the CDM model name into. The variable string length must be declared with a large enough size for the ID size (length). On success, the *Modelstring* will contain the cellular data modem's name.<br><br>At this manuals release, the following are valid CDM names: LE910-SVG, LE910-SV1, HL7618RD, HE910-D. Only supported CDMs may be used with P-Series targets. |

### Description:

The EZ_Cell_GetModelName communicates to the CDM (cellular data modem). The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *Modelstring* returns the value of the cellular modem name:

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem. The cellular data modem state (using **EZ_Cell_GetState**) must be **IDLE** (the cellular data modem must be powered on and in an **IDLE** state for this command to function or an error will occur. Power is applied using the **EZ_Cell_ApplyPower** function).

### Example:

```
FUNCTION_BLOCK RdCDMname
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
        MdlName : string[30];
                Rtn: int;
        END_VAR
        VAR_OUTPUT
                Q : bool;
```

Continued Next Page------»

```
        END_VAR

        if(Enable = True) AND (lastEn = False) then

                Rtn:= EZ_Cell_GetModelName(MdlName);

        end_if;

  Q := Enable;
  lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetRegistration

### Summary:

The EZ_Cell_GetRegistration function returns the cellular data modem's current registration (how it would connect to a cellular network).

### Format:

*INTvar* := EZ_Cell_GetRegistration(*RegDINT*); or
EZ_Cell_GetRegistration(*RegDINT*);

### Arguments:

*INTvar* — Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram.

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

*RegDINT* — Registration value for how CDM would connect to cellular network.

| | |
|---|---|
| 0 | Not Registered |
| 1 | Registered Home |
| 2 | Not Registered - Searching |
| 3 | Registration Denied |
| 4 | Unknown |
| 5 | Registered Roaming |

### Description:

The EZ_Cell_GetRegistration function returns the registration for how the cellular data modem would connect to the cellular network. The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *RegDINT* (DINT) holds the registration value (see list above).

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem. The cellular data modem state (using **EZ_Cell_GetState**) must be **IDLE** for successful implementation of this function.

### Example:

```
FUNCTION_BLOCK CellModemReg
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q :  bool;
                RG:  DINT;
        END_VAR
        VAR
                result : INT;
                registration : DINT;
```

Continued Next Page------»

```
        END_VAR

        IF ((Enable = TRUE))THEN                        (*Check for Enable*)

                (* Modem Registration *)
                result := EZ_Cell_GetRegistration(registration);   (*Get registration*)
                IF (result = 0) THEN
                        RG := registration;             (*Set output value based on return*)
                ELSE
                        RG := result;
                END_IF;

        END_IF;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetSignalStrength

### Summary:

The EZ_Cell_GetSignalStrength function returns the cellular data modem's current signal strength of the cellular network signal.

### Format:

*INTvar* := EZ_Cell_GetSignalStrength(*SSDINT*); or
EZ_Cell_GetSignalStrength(*SSDINT*);

### Arguments:

*INTvar* Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram.

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

*SSDINT* Signal Level of cellular data modem to the cellular network.

| | |
|---|---|
| 0 | (-113) dBm or less |
| 1 | (-111) dBm |
| 2-30 | (-109) dBm to (-52) dBm / 2 per step |
| 31 | (-51) dBm or greater |
| 99 | Not known or not detectable |

### Description:

The EZ_Cell_GetSignalStrength function returns the signal strength of the cellular network signal to the cellular data modem. The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *SSDINT* (DINT) holds the signal strength value (see list above).

Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem. The cellular data modem state (using **EZ_Cell_GetState**) must be **IDLE** for successful implementation of this function.

### Example:

```
FUNCTION_BLOCK CellSigStr
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q :  bool;
                SS:  DINT;
        END_VAR
        VAR
                sigStr : DINT;
                result : INT;
        END_VAR
```

Continued Next Page------»

```
          IF ((Enable = TRUE))THEN                          (*check for Enable to be true*)

                    (* Signal Strength *)
                    result := EZ_Cell_GetSignalStrength(sigStr);       (*Get Signal Strength*)
                    IF (result = 0) THEN                                (*Set output value if result is no error*)
                              SS := sigStr;
                    ELSE
                              SS := result;
                    END_IF;

          END_IF;
          Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Cell_GetState

**Summary:**

The EZ_Cell_GetState function returns the cellular data modem's current operational state.

**Format:**

*INTvar* := EZ_Cell_GetState(*StateDINT*); or
EZ_Cell_GetState(*StateDINT*);

**Arguments:**

*INTvar* — Function return holding variable (INT). Returns the status of the function's activity (optional). This must be converted to a DINT before it can exported from structured text to the ladder diagram.

| | |
|---|---|
| 0 | No Error |
| -1 | Null Pointer (Contact Divelbiss for more information) |
| -2 | Initialization Failed (Contact Divelbiss for more information) |
| -3 | Cellular Modem state was not IDLE. Must be IDLE to use this function. |
| -4 | No Connection. A connection could not be made to the cellular network. |
| -5 | Invalid Registration (Wait for registration value to = 1 for HOME or 5 for roaming) |
| -6 | Undefined (Contact Divelbiss for more information) |
| -7 | Signal Strength too low. Retry after better signal strength is detected. |
| -8 | Cellular Data Modem is not powered on. Use **EZ_Cell_ApplyPower** function. |
| -9 | Not Supported. This function is not supported by the target hardware. |

*StateDINT* — Signal Level of cellular data modem to the cellular network.

| | |
|---|---|
| 0 | Off - Not Powered |
| 1 | Powering Up |
| 2 | Idle |
| 3 | Starting / Attempting Connection to cellular the network |
| 4 | Connected to the cellular network |
| 5 | Stopping / Disconnecting from the cellular network |
| 6 | Activating. Activating the cellular data modem on the cellular network. |
| 7 | Deactivating. Deactivating the cellular data modem from the cellular network. |
| 8 | Error. **Requires cycling modem power (off / on) to clear this error state.** |

**Description:**

The EZ_Cell_GetState function returns the current state (status) of the cellular data modem. The *INTvar* return variable returns a 0 for No Error or a negative number for errors. See list above. The variable *StateDINT* (DINT) holds the current state value (see list above). Refer to **Chapter 27 - Cellular Connectivity** for a generic flow chart and descriptions on using the cellular data modem.

**Example:**

```
FUNCTION_BLOCK CellModemStatus
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q :  bool;
                ST:  DINT;
        END_VAR
        VAR
                state : DINT;
                result : INT;
```

Continued Next Page------»

```
        END_VAR

        IF ((Enable = TRUE))THEN                        (*Check for Enable to be True*)

                (* Modem State *)
                result := EZ_Cell_GetState(state);      (*Get Cellular data modem state*)
                IF (result = 0) THEN
                        ST := state;                    (*If result is no error then update output*)
                ELSE
                        ST := result;
                END_IF;

        END_IF;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_Activate

**Summary:**

The EZ_DCCoAP_Activate function is used to activate the target (device) on a DCCoAP Cloud server. This may or may not be required only once per device based on the DCCoAP server(unless it has been removed / deactivated from the server).

**Format:**

*Statvar* := EZ_DCCoAP_Activate(*Response*);

**Arguments:**

| | |
|---|---|
| *Statvar* | Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.<br>-1 = Error<br>0 = Not Used<br>1 = Queued to Send<br>2 = Sending<br>3 = Completed |
| *Response* | Response code from DCCoAP cloud Server (DINT). When the function status (*Statvar*) is equal to 3 (complete) or -1 (error), this variable will return an additional response code with more detail regarding the communications. |

**For Statvar = -1 (Error)**

| Error | Error Codes | Description |
|---|---|---|
| None | 0 | No Error |
| LWIP | -1 | Contact Divelbiss Support for this error. |
| No Connection | -3 | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| Already Activated | -4 | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Activation Failed | -5 | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Null Pointer | -6 | Contact Divelbiss Support for this error. |
| Invalid String | -7 | Contact Divelbiss Support for this error. |
| Queue is Full | -8 | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| Not Idle | -9 | Contact Divelbiss Support for this error. |
| Other Error | -10 | Other unspecified error. Contact Divelbiss Support for this error. |
| Not Activated | -11 | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| Time-out | -20 | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

**Statvar =3 (No Error) Response Codes**

When the function has completed without an error, it will return one of the *Response* codes. This code returned is built with 8 total bits.These bits are divided with the 3 upper bits being the response code before the '.' shown below and the 5 lower bits representing the two digits after the '.' shown below. A conversion would be necessary to use this response code. Typical response codes would be 69 or 129 (shown in red). Contact Divelbiss support for addtional codes.

| Response | Converted Code | Description |
|---|---|---|
| 65 | 2.01 | Created |
| 68 | 2.04 | Changed |
| 69 | 2.05 | Content was received. |

Continued Next Page------»

| | | |
|---|---|---|
| 95 | 2.31 | Continue |
| 129 | 4.01 | Unauthorized - The CIK couldn't be used to authenticate. |
| 130 | 4.02 | Bad Option |
| 131 | 4.03 | Forbidden |
| 132 | 4.04 | Not Found |
| 136 | 4.08 | Request Entity Incomplete |
| 140 | 4.12 | Precondition Failed |

**Description:**

The EZ_DCCoAP_Activate function attempts communications with the DCCoAP cloud server to register the device (target) to the DCCoAP cloud server / portal. Devices may or may not be required to be activated on a DCCoAP cloud server / portal before the device and server can communicate data. If required, each device needs only be activated to the portal one time (unless it is removed from the portal. Requirement for activation is based on the DCCoAP cloud server implementation.

As this function communicates to the DCCoAP cloud server, it must be 'polled' until the *Statvar* variable returns a status of the communciations process -1 (for error) or 3 (for completed). With an error or completed *Statvar*, the *Response* variable will return the details of the device to server communciations (see previous list of errors or if completed). Even though communciation may have been completed successfully, the *Response* code may still represent an issue with activation process to the DCCoAP cloud server (see list of responses and errors listed previously based on *Statvar*.

A *Response* code of 68 or 69 would signify the device was activated successfully.

**Example:**

```
FUNCTION_BLOCK DCCoAPCldActivate
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                S : DINT;
                R : DINT;
        END_VAR
        VAR
                complete : BOOL := FALSE;
                buffer : string[100];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE))THEN

                S := EZ_DCCoAP_Activate(R);

                IF ( S < 0) THEN
                        (* Error occured when sending *)
                        (* Determine what to do. Retry or abort. Up to user. *)
                        EZ_FormatString(buffer, 'Did not communicate to server. Error %d. $N', R);
                        complete := TRUE;

                ELSIF ( S = 3) THEN
                        (* Completed Transmission - Check errors*)

                        IF (R = 69)  THEN
                                (* Completed and successful*)
                                EZ_FormatString(buffer, 'Successfully activated device. $N');
```

Continued Next Page------»

```
                        ELSE
                                (* Completed but error code from server*)
                                EZ_FormatString(buffer, 'Communicated to server but error activating device. Error
code %d.%02d $N', SHR(R,5), (R AND 2#11111));
                        END_IF;

                complete := TRUE;
                END_IF;

                IF (complete = TRUE) THEN
                        (* Print result *)
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
                                ;
                        end_while;
                END_IF;

        END_IF;


        IF (Enable = FALSE) THEN
                S := 0;
                R := 0;
                complete := FALSE;
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_EnableInterface

**Summary:**

The EZ_DCCoAP_EnableInterface function is used to enable and disable cellular data modem and Ethernet interfaces for DCCoAP communications.

**Format:**

*Statvar* := EZ_DCCoAP_EnableInterface(*EthBool*, *CellBool*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 0 | None | No Error |
| -1 | LWIP | Contact Divelbiss Support for this error. |
| -3 | No Connection | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| -4 | Already Activated | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -5 | Activation Failed | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -6 | Null Pointer | Contact Divelbiss Support for this error. |
| -7 | Invalid String | Contact Divelbiss Support for this error. |
| -8 | Queue is Full | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| -9 | Not Idle | Contact Divelbiss Support for this error. |
| -10 | Other Error | Other unspecified error. Contact Divelbiss Support for this error. |
| -11 | Not Activated | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| -20 | Time-out | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

*EthBool*

DCCoAP cloud communications control state for the Ethernet interface (BOOL). A True or '1' enables and a False or '0' disables the Ethernet interface for DCCoAP cloud communications.

*CellBool*

DCCoAP cloud communications control state for the cellular data modem interface (BOOL). A True or '1' enables and a False or '0' disables the celluar data modem interface for DCCoAP cloud communications.

**Description:**

The EZ_DCCoAP_EnableInterface controls the Ethernet and cellular data modem interfaces in relation to DCCoAP cloud communications. The *Statvar* variable returns the status of the function per the list above. If *Statvar* return No Error, the erase was successful. For the variables EthBool and CellBool, a True (1) enables and a False (0) disables the Ethernet and Cellular data modem interfaces for DCCoAP cloud communications.(repectively).

**Example:**

```
FUNCTION_BLOCK DCCoAPCldEnableInterface
        VAR_INPUT
                Enable : bool;
                Eth : BOOL;
```

Continued Next Page------»

```
		Cel : BOOL;
	END_VAR
	VAR_OUTPUT
		Q : bool;
	END_VAR
	VAR
		enableLast : bool := TRUE;
	END_VAR

	IF (Enable = TRUE) THEN
		EZ_DCCoAP_EnableInterface(Eth, Cel);
	END_IF;

	enableLast := Enable;
	Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_EraseCIK

### Summary:

The EZ_DCCoAP_EraseCIK function is used to erase the stored the CIK on the device (target). The CIK is an identification string required for communications to DCCoAP cloud servers. The CIK identifies the device to the server and is unique for each device. This CIK is created and stored when the device is activated DCCoAP cloud server (using EZ_DCCoAP_Activate). The CIK must be erased using this function before it can be re-activated using EZ_DCCoAP_Activate function or stored using the EZ_DCCoAP_EraseCIK function.

### Format:

*Statvar* := EZ_DCCoAP_EraseCIK();

### Arguments:

*Statvar* Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 0 | None | No Error |
| -1 | LWIP | Contact Divelbiss Support for this error. |
| -3 | No Connection | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| -4 | Already Activated | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -5 | Activation Failed | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -6 | Null Pointer | Contact Divelbiss Support for this error. |
| -7 | Invalid String | Contact Divelbiss Support for this error. |
| -8 | Queue is Full | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| -9 | Not Idle | Contact Divelbiss Support for this error. |
| -10 | Other Error | Other unspecified error. Contact Divelbiss Support for this error. |
| -11 | Not Activated | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| -20 | Time-out | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

### Description:

The EZ_DCCoAP_EraseCIK erases / clears the device's stored *CIK*. The *Statvar* variable returns the status of the function per the list above. If *Statvar* return No Error, the erase was successful.

### Example:

```
FUNCTION_BLOCK DCCoAPClearCIK
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
        END_VAR
```

Continued Next Page------»

```
	VAR
		enableLast : bool := TRUE;
		buffer : string[50];
		cr : string[5];
	END_VAR

	IF ((Enable = TRUE) AND (enableLast = FALSE))THEN

		RES := EZ_DCCoAP_EraseCIK();

		IF (RES = 0) THEN
			EZ_FormatString(buffer, 'Successfully erased CIK. $N');
		ELSE
			(*Error Result*)
			EZ_FormatString(buffer, 'Error erasing CIK. Error %d $N', RES);
		END_IF;

		(*Print Result*)
		while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
			;
		end_while;

	END_IF;

	enableLast := Enable;
	Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_GetServerTime

**Summary:**

The EZ_DCCoAP_GetServerTime function is used get the current date time from the DCCoAP cloud server (Unix Time).

**Format:**

*Statvar* := EZ_DCCoAP_GetServerTime(*Response, UnixTime*);

**Arguments:**

| | |
|---|---|
| *Statvar* | Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.<br>-1 = Error<br>0 = Not Used<br>1 = Queued to Send<br>2 = Sending<br>3 = Completed |
| *Response* | Response code from DCCoAP Server (DINT). When the function status (*Statvar*) is equal to 3 (complete) or -1 (error), this variable will return an additional response code with more detail regarding the communications. |
| *UnixTime* | Date and Time returned from DCCoAP cloud server in formatted as Unix time (LINT). |

**For Statvar = -1 (Error)**

| Error | Error Codes | Description |
|---|---|---|
| None | 0 | No Error |
| LWIP | -1 | Contact Divelbiss Support for this error. |
| No Connection | -3 | Unable to make a connection to the DCCoAP server. Check connections. |
| Already Activated | -4 | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Activation Failed | -5 | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Null Pointer | -6 | Contact Divelbiss Support for this error. |
| Invalid String | -7 | Contact Divelbiss Support for this error. |
| Queue is Full | -8 | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| Not Idle | -9 | Contact Divelbiss Support for this error. |
| Other Error | -10 | Other unspecified error. Contact Divelbiss Support for this error. |
| Not Activated | -11 | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| Time-out | -20 | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

**Statvar =3 (No Error) Response Codes**

When the function has completed without an error, it will return one of the *Response* codes. This code returned is built with 8 total bits.These bits are divided with the 3 upper bits being the response code before the '.' shown below and the 5 lower bits representing the two digits after the '.' shown below. A conversion would be necessary to use this response code. Typical response codes would be 69 or 129 (shown in red). Contact Divelbiss support for addtional codes.

| Response | Converted Code | Description |
|---|---|---|
| 65 | 2.01 | Created |
| 68 | 2.04 | Changed |

Continued Next Page------»

| | | |
|---|---|---|
| 69 | 2.05 | Content was received. |
| 95 | 2.31 | Continue |
| 129 | 4.01 | Unauthorized - The CIK couldn't be used to authenticate. |
| 130 | 4.02 | Bad Option |
| 131 | 4.03 | Forbidden |
| 132 | 4.04 | Not Found |
| 136 | 4.08 | Request Entity Incomplete |
| 140 | 4.12 | Precondition Failed |

**Description:**

The EZ_DCCoAP_GetServerTime function attempts communications with the DCCoAP cloud server to to retrieve the current data and time in Unix time format.

As this function communicates to the DCCoAP cloud server, it must be 'polled' until the *Statvar* variable returns a status of the communciations process -1 (for error) or 3 (for completed). With an error or completed *Statvar*, the *Response* variable will return the details of the device to server communciations (see previous list of errors or if completed). *Unix-Time* holds the Unix based time and date returned from the server.

**Example:**

```
FUNCTION_BLOCK DCCoAPcldGetTime
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                S : DINT;
                R : DINT;
                STM : DINT;
        END_VAR
        VAR
                month : DINT;
                day : DINT;
                year : DINT;
                wday : DINT;
                hour : DINT;
                minute : DINT;
                sec : DINT;
                complete : bool := FALSE;
                serverTime : LINT;
                buffer : STRING[50];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                S := EZ_DCCoAP_GetServerTime(R, serverTime);

                IF (S = 3) THEN
                STM := LINT_TO_DINT(serverTime);
                EZ_TimeDateUnixToCalendar(serverTime, month,day,year,wday,hour,minute,sec);
                EZ_FormatString(buffer, 'Server Time: %d/%d/%d %d:%d:%d $N', month, day, year, hour, minute, sec);

                        (*Print Result*)
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
                                ;
```

Continued Next Page------»

```
                    end_while;

                    complete := TRUE;

          END_IF;


          IF (S < 0) THEN  (*Error*)

                              (*Handle error as required by application*)

                    complete := TRUE;

          END_IF;

     END_IF;

     IF (Enable = FALSE) THEN
          S := 0;
          R := 0;
          STM := 0;
          complete := FALSE;
     END_IF;

     Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_GetState

**Summary:**

The EZ_DCCoAP_GetState function is used to get the current status of the device's internal DCCoAP (DCCoAP cloud) state machine.

**Format:**

*Statvar* := EZ_DCCoAP_GetState();

**Arguments:**

*Statvar* Function return holding variable (INT). Returns the status of the function's activity (DCCoAP cloud state machine). This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | |
|---|---|
| 0 | Network Down |
| 1 | Getting Host  Address from DNS server |
| 2 | Establishing / Seting up connection |
| 3 | Idle |
| 4 | Sending |
| 5 | Waiting |
| 6 | Error |

**Description:**

The EZ_DCCoAP_GetState returns the *Statvar* variable (current status) of the devices internal DCCoAP (DCCoAP cloud) state machine. The current status is helpful for controlling DCCoAP cloud server communications using structured text.

**Example:**

```
FUNCTION_BLOCK DCCoAPCldGetState
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                S : DINT;
        END_VAR

        IF (Enable = TRUE) THEN
                S := EZ_DCCoAP_GetState();
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_ReadCIK

**Summary:**

The EZ_DCCoAP_ReadCIK function is used read the device's (target's) stored CIK. The CIK is an identification string that may be required for communications to the DCCoAP cloud server. This CIK is created and stored when the device is activated to the COAP cloud server (using EZ_DCCoAP_Activate). The CIK identifies the device to the server and is unique for each device.

**Format:**

*Statvar* := EZ_DCCoAP_ReadCIK(*CIKreturn*);

**Arguments:**

*Statvar* Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 0 | None | No Error |
| -1 | LWIP | Contact Divelbiss Support for this error. |
| -3 | No Connection | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| -4 | Already Activated | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -5 | Activation Failed | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -6 | Null Pointer | Contact Divelbiss Support for this error. |
| -7 | Invalid String | Contact Divelbiss Support for this error. |
| -8 | Queue is Full | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| -9 | Not Idle | Contact Divelbiss Support for this error. |
| -10 | Other Error | Other unspecified error. Contact Divelbiss Support for this error. |
| -11 | Not Activated | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| -20 | Time-out | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

*CIKreturn* Variable to hold CIK stored in device (STRING). The CIK is 40 characters long, so this variable (or any variables used to handle this data) should be at least 40 bytes in size.

**Description:**

The EZ_DCCoAP_ReadCIK reads the CIK stored on the device. The *Statvar* variable returns the status of the function per the list above. If *Statvar* return No Error will result in the actual CIK being returned in the *CIKreturn* variable.

**Example:**

```
FUNCTION_BLOCK DCCoAPCldReadCIK
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES: DINT;
        END_VAR
```

Continued Next Page------»

```
        VAR
                enableLast : bool := TRUE;
                cik : string[40];
                buffer : string[70];
                cr : string[5];
        END_VAR

        IF ((Enable = TRUE) AND (enableLast = FALSE))THEN

                RES := EZ_DCCoAP_ReadCIK(cik);

                IF (RES = 0 and cik[0] <> 0) THEN
                        EZ_FormatString(buffer, 'Device CIK: ');  (*Print title/header to UART*)
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
                                ;
                        end_while;
                        (*Print CIK*)
                        while EZ_UartWriteStr(FD_UART2, cik) <= 0 do
                                ;
                        end_while;
                        (*Print Carriage Return and Line Feed*)
                        EZ_FormatString(cr, '$N');
                        while EZ_UartWriteStr(FD_UART2, cr) <= 0 do
                                ;
                        end_while;
                ELSE
                        (*Error Result*)
                        IF (RES = -11) THEN
                                (*Device not activated error*)
                                EZ_FormatString(buffer, 'Error reading CIK. Device not activated $N');
                        ELSE
                                (*Other error*)
                                EZ_FormatString(buffer, 'Error reading CIK: %d $N', RES);

                        END_IF;

                        (*Print Result*)
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
                                ;
                        end_while;
                END_IF;

        END_IF;

        enableLast := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_SendData

**Summary:**

The EZ_DCCoAP_SendData function is used send data (not time / date stamped) to the DCCoAP cloud server.

**Format:**

*Statvar* := EZ_DCCoAP_SendData(*Response, Var1, Var2...*);

**Arguments:**

| | |
|---|---|
| *Statvar* | Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.<br>-1 = Error<br>0 = Not Used<br>1 = Queued to Send<br>2 = Sending<br>3 = Completed |
| *Response* | Response code from DCCoAP cloud Server (DINT). When the function status (*Statvar*) is equal to 3 (complete) or -1 (error), this variable will return an additional response code with more detail regarding the communications. |
| *Var1, Var2...* | List of variables to send from device to DCCoAP cloud server. Variables can be any standard type of variable (DINT, REAL, STRING, etc). **The variable names in this function must mach the variable names as setup on the DCCoAP cloud server to send / receive.** |

**For Statvar = -1 (Error)**

| Error | Error Codes | Description |
|---|---|---|
| None | 0 | No Error |
| LWIP | -1 | Contact Divelbiss Support for this error. |
| No Connection | -3 | Unable to make a connection to the DCCoAP server. Check connections. |
| Already Activated | -4 | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Activation Failed | -5 | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Null Pointer | -6 | Contact Divelbiss Support for this error. |
| Invalid String | -7 | Contact Divelbiss Support for this error. |
| Queue is Full | -8 | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| Not Idle | -9 | Contact Divelbiss Support for this error. |
| Other Error | -10 | Other unspecified error. Contact Divelbiss Support for this error. |
| Not Activated | -11 | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| Time-out | -20 | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

**Statvar =3 (No Error) Response Codes**

When the function has completed without an error, it will return one of the *Response* codes. This code returned is built with 8 total bits.These bits are divided with the 3 upper bits being the response code before the '.' shown below and the 5 lower bits representing the two digits after the '.' shown below. A conversion would be necessary to use this response code. Typical response codes would be 68, 69 or 129 (shown in red). Contact Divelbiss support for addtional codes.

| Response | Converted Code | Description |
|---|---|---|
| 65 | 2.01 | Created |

Continued Next Page------»

| | | |
|---|---|---|
| 68 | 2.04 | Changed |
| 69 | 2.05 | Content was received. |
| 95 | 2.31 | Continue |
| 129 | 4.01 | Unauthorized - The CIK couldn't be used to authenticate. |
| 130 | 4.02 | Bad Option |
| 131 | 4.03 | Forbidden |
| 132 | 4.04 | Not Found |
| 136 | 4.08 | Request Entity Incomplete |
| 140 | 4.12 | Precondition Failed |

**Description:**

The EZ_DCCoAP_SendData function attempts communications with the DCCoAP cloud server and sends the variables listed (*Var1*, *Var2*...) to the server.

As this function communicates to the DCCoAP cloud server, it must be 'polled' until the *Statvar* variable returns a status of the communciations process -1 (for error) or 3 (for completed). With an error or completed *Statvar*, the *Response* variable will return the details of the device to server communciations (see previous list of errors or if completed).

**Example:**

```
FUNCTION_BLOCK DCCoAPCldWriteData
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                S : DINT;
                R : DINT;
        END_VAR
        VAR
                STTEST : string[40];
                IN1 : INT;
                response : DINT;
                FLTEST : REAL;
                complete : BOOL := FALSE;
        END_VAR
        IN1 := 51;
        STTEST := 'Test Record Upload';
        FLTEST := 102.1;

        IF ((Enable = TRUE) AND (complete = FALSE))THEN

                S := EZ_DCCoAP_SendData(R, IN1, STTEST, FLTEST);

                IF ( S < 0) THEN
                        (* Error occured when sending *)
                        (* Determine what to do. Retry or abort. Up to user. *)
                        complete := TRUE;

                ELSIF ( S = 3) THEN
                        (* Completed Transmission - Check errors*)

                        IF ((R = 68) OR (R=69)) THEN
                                (* Completed and successful*)
```

Continued Next Page------»

```
                        complete := TRUE;
                    END_IF;

                complete := TRUE;
                END_IF;

    END_IF;


    IF (Enable = FALSE) THEN
        S := 0;
        R := 0;
        complete := FALSE;
    END_IF;

    Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_SendDataRecord

**Summary:**

The EZ_DCCoAP_SendDataRecord function is used send data with date / time stamp to the DCCoAP cloud server.

**Format:**

*Statvar* := EZ_DCCoAP_SendDataRecord(*Response, UnixTime, Var1, Var2...*);

**Arguments:**

| | |
|---|---|
| *Statvar* | Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.<br>-1 = Error<br>0 = Not Used<br>1 = Queued to Send<br>2 = Sending<br>3 = Completed |
| *Response* | Response code from DCCoAP cloud Server (DINT). When the function status (*Statvar*) is equal to 3 (complete) or -1 (error), this variable will return an additional response code with more detail regarding the communications. |
| *UnixTime* | Variable to hold time / date stamp to send to DCCoAP cloud server in Unix time format (LINT). |
| *Var1, Var2...* | List of variables to send from device to DCCoAP cloud server. Variables can be any standard type of variable (DINT, REAL, STRING, etc). **The variable names in this function must mach the variable names as setup on the DCCoAP cloud server.** |

**For Statvar = -1 (Error)**

| Error | Error Codes | Description |
|---|---|---|
| None | 0 | No Error |
| LWIP | -1 | Contact Divelbiss Support for this error. |
| No Connection | -3 | Unable to make a connection to the DCCoAP server. Check connections. |
| Already Activated | -4 | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Activation Failed | -5 | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| Null Pointer | -6 | Contact Divelbiss Support for this error. |
| Invalid String | -7 | Contact Divelbiss Support for this error. |
| Queue is Full | -8 | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| Not Idle | -9 | Contact Divelbiss Support for this error. |
| Other Error | -10 | Other unspecified error. Contact Divelbiss Support for this error. |
| Not Activated | -11 | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| Time-out | -20 | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

**Statvar =3 (No Error) Response Codes**

When the function has completed without an error, it will return one of the *Response* codes. This code returned is built with 8 total bits.These bits are divided with the 3 upper bits being the response code before the '.' shown below and the 5 lower bits representing the two digits after the '.' shown below. A conversion would be necessary to use this response code. Typical response codes would be 69 or 129 (shown in red). Contact Divelbiss support for addtional codes.

Continued Next Page------»

| Response | Converted Code | Description |
|---|---|---|
| 65 | 2.01 | Created |
| 68 | 2.04 | Changed |
| 69 | 2.05 | Content was received. |
| 95 | 2.31 | Continue |
| 129 | 4.01 | Unauthorized - The CIK couldn't be used to authenticate. |
| 130 | 4.02 | Bad Option |
| 131 | 4.03 | Forbidden |
| 132 | 4.04 | Not Found |
| 136 | 4.08 | Request Entity Incomplete |
| 140 | 4.12 | Precondition Failed |

**Description:**

The EZ_DCCoAP_SendDataRecord function attempts communications with the DCCoAP cloud server and sends the variables listed (*Var1*, *Var2*...) to the DCCoAP cloud server along with the time / date stamp variable *UnixTime* (formatted to Unix time).

As this function communicates to the DCCoAP cloud server, it must be 'polled' until the *Statvar* variable returns a status of the communciations process -1 (for error) or 3 (for completed). With an error or completed *Statvar*, the *Response* variable will return the details of the device to server communciations (see previous list of errors or if completed).

**Example:**

```
FUNCTION_BLOCK DCCoAPCldWriteDataRecord
        VAR_INPUT
                Enable : bool;
                TM : DINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                S : DINT;
                R : DINT;
        END_VAR
        VAR
                STTEST : string[40];
                IN1 : INT;
                response : DINT;
                FLTEST : REAL;
                complete : BOOL := FALSE;
                TimeLint : LINT;
        END_VAR
        IN1 := 51;
        STTEST := 'Test Record Upload';
        FLTEST := 102.1;

        IF ((Enable = TRUE) AND (complete = FALSE))THEN

                TimeLint := DINT_TO_LINT(TM);

                S := EZ_DCCoAP_SendDataRecord(R, TimeLint, IN1, STTEST, FLTEST);

                IF ( S < 0) THEN
                        (* Error occured when sending *)
                        (* Determine what to do. Retry or abort. Up to user. *)
                        complete := TRUE;
```

Continued Next Page------»

```
                ELSIF ( S = 3) THEN
                        (* Completed Transmission - Check errors*)

                        IF ( R = 69 ) THEN
                                (* Completed and successful*)

                                complete := TRUE;
                        END_IF;

                complete := TRUE;
                END_IF;

        END_IF;


        IF (Enable = FALSE) THEN
                S := 0;
                R := 0;
                complete := FALSE;
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_DCCoAP_WriteCIK

### Summary:

The EZ_DCCoAP_WriteCIK function is used manually write and store the CIK to the device (target). The CIK is an identification string that may be required for communications to the DCCoAP cloud server. The CIK identifies the device to the server and is unique for each device. This CIK is created and stored when the device is activated to the DCCoAP cloud server (using EZ_DCCoAP_Activate). This allows for a supplement method for writing a CIK to the device (CIK must have been created previously). The CIK must be erased (using EZ_DCCoAP_EraseCIK) or have not been written previously before this function may be used to write the CIK.

### Format:

*Statvar* := EZ_DCCoAP_WriteCIK(*CIK*);

### Arguments:

*Statvar* Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 0 | None | No Error |
| -1 | LWIP | Contact Divelbiss Support for this error. |
| -3 | No Connection | Unable to make a connection to the DCCoAP cloud server. Check connections. |
| -4 | Already Activated | Device has already been activated on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -5 | Activation Failed | The activation for this device failed on the DCCoAP cloud server. Contact Divelbiss support for more information. |
| -6 | Null Pointer | Contact Divelbiss Support for this error. |
| -7 | Invalid String | Contact Divelbiss Support for this error. |
| -8 | Queue is Full | The send / receive queue is full. Evaluate the amount and frequency of data being transmitted to the DCCoAP cloud server. |
| -9 | Not Idle | Contact Divelbiss Support for this error. |
| -10 | Other Error | Other unspecified error. Contact Divelbiss Support for this error. |
| -11 | Not Activated | The device is trying to communicate to the DCCoAP cloud server but has not been activated on the server. |
| -20 | Time-out | A communications time-out occurred. Check your network and settings (Wi-Fi, Ethernet). |

*CIK* Variable to hold CIK to be stored in the device (STRING). The CIK is 40 characters long, so this variable (or any variables used to handle this data) should be at least 40 bytes in size.

### Description:

The EZ_DCCoAP_WriteCIK writes / stores the *CIK* variable value to the device. The *Statvar* variable returns the status of the function per the list above. If *Statvar* return No Error, the write was successful.

### Example:

```
FUNCTION_BLOCK DCCoAPCldWriteCIK
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
```

Continued Next Page------»

```
		RES : DINT;
	END_VAR
	VAR
		enableLast : bool := TRUE;
		cik : string[40];
		buffer : string[50];
		cr : string[5];
	END_VAR
	cik := '0123456789012345678901234567890123456789';

	IF ((Enable = TRUE) AND (enableLast = FALSE))THEN

		RES := EZ_DCCoAP_WriteCIK(cik);

		IF (RES = 0) THEN
			EZ_FormatString(buffer, 'Successfully wrote CIK: ');
			while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
				;
			end_while;
			(*Print CIK*)
			while EZ_UartWriteStr(FD_UART2, cik) <= 0 do
				;
			end_while;
			(*Print Carriage Return and Line Feed*)
			EZ_FormatString(cr, '$N');
			while EZ_UartWriteStr(FD_UART2, cr) <= 0 do
				;
			end_while;
		ELSE
			(*Error Result*)
			EZ_FormatString(buffer, 'Error writing CIK: %d $N', RES);

			(*Print Result*)
			while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do
				;
			end_while;
		END_IF;

	END_IF;

	enableLast := Enable;
	Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_EEPromRead

### Summary:

The EZ_EEPromRead function is used to read data from the on-board PLC on a Chip or external chip FRAM EEPROM memory.

### Format:

*DINTvar* := EZ_EEPromRead(*FileDescriptor*, *StartAddr*, *ReturnData*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_PLCHIP_PXX_EEPROM** for internal PLC on a Chip EEPROM.<br>**FD_FM24XXX** for external FRAM. |
| *StartAddr* | EEPROM memory start address to read from (DINT) |
| *ReturnData* | Holding variable for data read from EEPROM. Supports ALL variable types. |
| *DINTvar* | Function return holding variable (DINT). # of bytes read based on the variable type used in *ReturnData*. |

### Description:

The EZ_EEPromRead function accesses the EEPROM memory identified in *FileDescriptor*, locates the beginning address identified by *StartAddr* and reads the number of bytes for the variable type of *ReturnData*. The number of bytes read is returned in *UDINTvar* and the actual read EEPROM data is stored in the *ReturnData* variable.

### Example:

```
FUNCTION_BLOCK ExampleReadEEPROM
        VAR_INPUT
                Enable : bool;
                AdrStart: DINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                NByte : DINT;
                ReturnData : DINT;
        END_VAR

        IF Enable = 1 THEN
                NByte := EZ_EEPromRead(FD_PLCHIP_PXX_EEPROM,AdrStart,ReturnData);
        ELSE
                NByte := 0;
                ReturnData := 0;
        End_If;
                Q := Enable;
END_FUNCTION_BLOCK
```

## EZ_EEPromReadArray

**Summary:**

The EZ_EEPromReadArray function is used to read a data array from the on-board PLC on a Chip or external chip FRAM EEPROM memory.

**Format:**

*DINTvar* := EZ_EEPromReadArray(*FileDescriptor*, *StartAddr*, *Buffer*, *Offset*, *Len*);

**Arguments:**

| | |
|---|---|
| *FileDescriptor* | **FD_PLCHIP_PXX_EEPROM** for internal PLC on a Chip EEPROM.<br>**FD_FM24XXX** for external FRAM. |
| *StartAddr* | EEPROM memory start address to read from (DINT) |
| *Buffer* | Destination buffer to hold the read EEPROM contents (ARRAY[ ] of USINT). |
| *Offset* | *Offset* location in the *Buffer* where to start writing read EEPROM data to (DINT). |
| *Len* | Length / Number of bytes to read from the EEPROM (DINT) |
| *DINTvar* | Function return holding variable (DINT). # of bytes read. |

**Description:**

The EZ_EEPromReadArray function accesses the EEPROM memory identified in *FileDescriptor*, locates the beginning address identified by *StartAddr* and reads the number of bytes idenfified by *Len* and stores the data as a byte array into the array identified in *Buffer* beginning at the *offset* location. The number of bytes read is returned in *UDINTvar*.

**Example:**

```
FUNCTION_BLOCK ExampleReadEEPROMArray
        VAR_INPUT
                Enable : bool;
                AdrStart: DINT;
        END_VAR
        VAR_TEMP
                Vals : ARRAY[0..11] of USINT;
                B : DINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                NByte : DINT;
                OVal1 : DINT;
                OVal2 : DINT;
                OVal3 : DINT;
        END_VAR

        IF Enable = 1 THEN
                NByte := EZ_EEPromReadArray(FD_PLCHIP_PXX_EEPROM,AdrStart,Vals,0,12);  (*Call function*)

                Oval1 := TO_LSB_DINT(Vals,0);    (*Update output vars with actual read eeprom data from array*)
                Oval2 := TO_LSB_DINT(Vals,4);
                Oval3 := TO_LSB_DINT(Vals,8);
        ELSE
                NByte := 0;
```

Continued Next Page------»

```
                    Oval1 := 0;                                         (*Update output vars with 0*)
                    Oval2 := 0;
                    Oval3 := 0;
          End_If;
                    Q := Enable;
END_FUNCTION_BLOCK
```

## EZ_EEPromWrite

**Summary:**

The EZ_EEPromWrite function is used to write data to the on-board PLC on a Chip or external chip FRAM EEPROM memory.

**Format:**

*DINTvar* := EZ_EEPromWrite(*FileDescriptor*, *StartAddr*, *Datavar*);

**Arguments:**

| | |
|---|---|
| *FileDescriptor* | **FD_PLCHIP_PXX_EEPROM** for internal PLC on a Chip EEPROM.<br>**FD_FM24XXX** for external FRAM. |
| *StartAddr* | EEPROM memory start address to write to. (DINT) |
| *Datavar* | Variable holding data to be written to EEPROM. Supports ALL variable types. |
| *DINTvar* | Function return holding variable (DINT). # of bytes written based on the variable type used in *ReturnData*. |

**Description:**

The EZ_EEPromWrite function accesses the EEPROM memory identified in *FileDescriptor*, locates the beginning address identified by *StartAddr* and writes the number of bytes for the variable type of *Datavar*. The number of bytes written is returned in *UDINTvar*.

**Example:**

```
FUNCTION_BLOCK ExampleEEPROMWrite
        VAR_INPUT
                Enable : bool;
                AdrStart: DINT;
                Datavar : DINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                NByte : DINT;
        END_VAR

        IF Enable = 1 THEN
        (*Is function enabled?*)
                NByte := EZ_EEPromWrite(FD_PLCHIP_PXX_EEPROM,AdrStart,Datavar); (*Call function*)
        ELSE
                NByte := 0;
                (*Set to 0 if function disabled*)
        End_If;
                Q := Enable;
END_FUNCTION_BLOCK
```

## EZ_EEPromWriteArray

### Summary:

The EZ_EEPromWriteArray function is used to write a data array to the on-board PLC on a Chip or external chip FRAM EEPROM memory.

### Format:

*DINTvar* := EZ_EEPromWriteArray(*FileDescriptor*, *StartAddr*, *Buffer*, *Offset*, *Len*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_PLCHIP_PXX_EEPROM** for internal PLC on a Chip EEPROM.<br>**FD_FM24XXX** for external FRAM. |
| *StartAddr* | EEPROM memory start address to write array to (DINT) |
| *Buffer* | Source buffer for the data array to write to EEPROM (ARRAY[ ] of USINT). |
| *Offset* | *Offset* location in the *Buffer* where to start reading write EEPROM data from (DINT). |
| *Len* | Length / Number of bytes to write to the EEPROM (DINT) |
| *DINTvar* | Function return holding variable (DINT). # of bytes written. |

### Description:

The EZ_EEPromWriteArray reads data from the *Buffer* (array of USINT), beginning with the *Offset* location and reading the number of bytes identified by *Len*. It then accesses the EEPROM memory identified by *FileDescriptor* and writes the number of bytes of the array beginning at the *StartAddr* location of the EEPROM memory. The *UDINTvar* returns the number of bytes written.

### Example:

```
FUNCTION_BLOCK ExampleWriteEEPROMArray
        VAR_INPUT
                Enable : bool;
                AdrStart: DINT;
                EVal1 : DINT;
                EVal2 : DINT;
                EVal3 : DINT;
        END_VAR
        VAR_TEMP
                Vals : ARRAY[0..11] of USINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                NByte : DINT;
        END_VAR

        IF Enable = 1 THEN
                LSB_DINT_TO_ARRAY(Vals,0,Eval1);          (*Load Array with input values to store*)
                LSB_DINT_TO_ARRAY(Vals,4,Eval2);
                LSB_DINT_TO_ARRAY(Vals,8,Eval3);

                NByte := EZ_EEPromWriteArray(FD_PLCHIP_PXX_EEPROM,AdrStart,Vals,0,12); (*Call function*)

        ELSE
```

Continued Next Page------»

```
                NByte := 0;                  (*Update output vars with 0*)
        End_If;
                Q := Enable;
END_FUNCTION_BLOCK
```

## EZ_Eth_DHCPRelease

### Summary:

The EZ_Eth_DHCPRelease function is used to release the Ethernet communication port's DHCP IP address. This will result in the Ethernet port having no IP address on the network.

### Format:

*BOOLvar* := EZ_Eth_DHCPRelease();

### Arguments:

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns True (1) upon success. |

### Description:

The EZ_Eth_DHCPRelease releases the Ethernet communication port's DHCP IP address resulting in the Ethernet port having no IP address on the network. Upon successful release, the *BOOLvar* variable returns a True (1).

### Example:

```
FUNCTION_BLOCK ReleaseDHCP
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                LastEnable : bool;
        End_VAR

        IF (Enable = TRUE)AND (LastEnable = False) THEN            (*Rising Edge detector*)

                Q := EZ_Eth_DHCPRelease();

        ELSE
                Q := False;

        END_IF;

        LastEnable := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_DHCPRenew

**Summary:**

The EZ_Eth_DHCPRenew function is used to renew (refresh or get  a new) the Ethernet communication port's IP address (DHCP) on an ethernet nework that supports DHCP IP.

**Format:**

*BOOLvar* := EZ_Eth_DHCPRenew();

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns True (1) upon success. |

**Description:**

The EZ_Eth_DHCPRenew renews (refreshes or gets a new) the Ethernet communication port's DHCP IP address on a DHCP supported network. Upon successful release, the *BOOLvar* variable returns a True (1).

**Example:**

```
FUNCTION_BLOCK RenewDHCP
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                LastEnable : bool;
        End_VAR

        IF (Enable = TRUE)AND (LastEnable = False) THEN            (*Rising Edge detector*)

                Q := EZ_Eth_DHCPRenew();

        ELSE
                Q := False;

        END_IF;

        LastEnable := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_GetHostname

**Summary:**

The EZ_Eth_GetHostname function is used to read and return the Ethernet communication port's Hostname.

**Format:**

*BOOLvar* := EZ_Eth_GetHostname(*stringbuf*);

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the function. Returns True (1) upon success |
| *stringbuf* | Variable (STRING). This variable holds the returned Hostname. |

**Description:**

The EZ_Eth_GetHostname reads the target's (device's) Ethernet communication hostname and stores it in the *string-buf* variable. *BOOLvar* is True (1) when upon success.

**Example:**

```
FUNCTION_BLOCK RdEthName
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
                Rtn : bool;
                hname: string[30];
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = true) AND (LastEn = False) then
                Rtn := EZ_Eth_GetHostname(hname);
                LastEn := true;
        End_If;


        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_GetIPV4Addr

**Summary:**

The EZ_Eth_GetIPV4Addr function is used to retrieve the device's (target's) current Ethernet communications port network IP address information.

**Format:**

*BOOLvar* := EZ_Eth_GetIPV4Addr(*StatIP*,*OurIP,Subnet,Gateway,DNS1,DNS2*); or
EZ_Eth_GetIPV4Addr(*StatIP,OurIP,Subnet,Gateway,DNS1,DNS2*);

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the function's activity (optional). Returns True (1) upon success. |
| *StatIP* | Type of IP used: Static or Dynamic (BOOL). True when using Static IP, False for Dynamic. |
| *OurIP* | Ethernet Port IP address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *Subnet* | Ethernet Port subnet address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *Gateway* | Ethernet Port gateway address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *DNS1* | Ethernet Port DNS1 address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *DNS2* | Ethernet Port DNS2 address (4 Bytes or more total) (ARRAY[ ] of USINT). |

**Description:**

The EZ_Eth_GetIPV4Addr function returns the IP address information of the Ethernet port on an ethernet network. The *BOOLvar* return variable returns a True (1) when complete.The variable *StatIP* (BOOL) hold the type of IP address (0 for dynamic, 1 for static), *OurIP* (array of USINT) holds the IP address, the variable *Subnet* (array of USINT) holds the subnet, the variable *Gateway* holds the gateway (array of USINT), the variable *DNS1* (array of USINT) holds the DNS1 address, the variable *DNS2* (array of USINT) holds the DNS2 address.

**Example:**

```
FUNCTION_BLOCK GetIPInfo
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                IP1: DINT;
                IP2: DINT;
                IP3: DINT;
                IP4: DINT;
                StIP: Bool;
        END_VAR
        VAR
                enableLast : bool;
                OURIP : ARRAY[0..3] OF USINT;
                NETMASK : ARRAY[0..3] OF USINT;
                GWAY : ARRAY[0..3] OF USINT;
                DNS1 : ARRAY[0..3] OF USINT;
                DNS2 : ARRAY[0..3] OF USINT;
                SIP : bool;
        END_VAR

        IF ((Enable = TRUE) AND (enableLast = FALSE))THEN  (*Detect rising edge*)
```

Continued Next Page------»

```
                EZ_Eth_GetIpV4Addr(SIP,OURIP,NETMASK,GWAY,DNS1,DNS2);       (*Get IP Info*)

                IP1 := USINT_TO_DINT(OURIP[0]);                        (*Output IP only to ladder diagram*)
                IP2 := USINT_TO_DINT(OURIP[1]);
                IP3 := USINT_TO_DINT(OURIP[2]);
                IP4 := USINT_TO_DINT(OURIP[3]);

                StIP := SIP;  (*Output type of IP to ladder diagram*)

        END_IF;

        enableLast := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_GetLinkActive

**Summary:**

The EZ_Eth_GetLinkActive function is used to check if the device's (target's) current Ethernet physical Link is active (port is connected to a switch or hub).

**Format:**

*BOOLvar* := EZ_Eth_GetLinkActive();

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the Ethernet physical Link. False (0) for not active or True (1) for active. |

**Description:**

The EZ_Eth_GetLinkActive function is used to check if the Ethernet physical LINK is active. The *BOOLvar* variable returns the status of the link (False (0) for not active or True (1) for active.

**Example:**

```
FUNCTION_BLOCK EthLinkActive
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                LK : bool;
        END_VAR
        VAR
                enableLast : bool;
                LINK : bool;
        END_VAR

        IF ((Enable = TRUE) AND (enableLast = FALSE))THEN  (*Detect rising edge*)

        LINK := EZ_Eth_GetLinkActive();

                IF (LINK > 0) THEN
                        LK := LINK;                        (*Set output to LINK (1) if true*)
                ELSE
                        LK := 0;
                END_IF;

        END_IF;

        enableLast := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_SetEnableAutoIpV4

**Summary:**

The EZ_Eth_SetEnableAutoIpV4 function is used to enable the device (target) Ethernet communication port's Auto IP setting and configure if this change is to be persistent when power is cycled or the device is re-booted.

**Format:**

*BOOLvar* := EZ_Eth_SetEnableAutoIpV4(*Persist*);

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the function. Returns True (1) upon success |
| *Persist* | Variable to set if the device (target) should keep the Auto IPV4 enabled setting after re-boot (BOOL) or use the settings in the target's bootloader. When True (1), this sets a flag for the bootloader to make this change for all subsequent re-boots. |

**Description:**

The EZ_Eth_SetEnableAutoIpV4 sets the target's (device's) Ethernet communication port to for Auto IPV4. The Persist variable configures if the Auto IPV4 setting is to be kept after a re-boot of the target. When True (1), this sets a flag for the bootloader to make this change for all subsequent re-boots and when Fasle (0), this change will be discarded after re-boot. *BOOLvar* is True (1) when upon success. Auto IPV4 determines if an auto IP address will be assigned if no IP address is received using DHCP (timed-out).

When this function it used to enable and set as persistent (permanent), the DHCP Enabled setting in the bootloader is disabled (no DHCP). If the persistent is not set (not permanent), then DHCP remains enabled and on any reboot or power cycle, the AutoIPV4 will not retain the enabled setting.

**Example:**

```
UNCTION_BLOCK EthAuto
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
                Rtn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = true) AND (LastEn = False) then
                Rtn := EZ_Eth_SetEnableAutoIpV4(1);
                LastEn := true;
        End_If;


        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_SetEnableDHCPIPV4

**Summary:**

The EZ_Eth_SetEnableDHCPIPV4 function is used to configure the device (target) Ethernet communication port's IP address type to DHCP and configure if the this change is to be persistent when power is cycled or the device is re-booted.

**Format:**

*BOOLvar* := EZ_Eth_SetEnableDHCPIPV4(*Persist*);

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the function. Returns True (1) upon success |
| *Persist* | Variable to set if the device (target) should keep DHCP as the IP address type after re-boot (BOOL) or use the settings in the target's bootloader. When True (1), this sets a flag for the bootloader to make this change for all subsequent re-boots. |

**Description:**

The EZ_Eth_SetEnableDHCPIPV4 sets the target's (device's) Ethernet communication port to use DHCP for the IP configuration (gets IP from DHCP server). The Persist variable configures if the DHCP setting is to be kept after a re-boot of the target. When True (1), this sets a flag for the bootloader to make this change for all subsequent re-boots and when Fasle (0), this change will be discarded after re-boot. *BOOLvar* is True (1) when upon success.

When this function is used to enable and set as persistent (permanent), the Auto IPV4 Config is also enabled. If the persistent is not set (not permanent), then the Auto IPV4 Config remains disabled and on any re boot or power cycle, the DHCP will not retain the enabled setting.

**Example:**

```
FUNCTION_BLOCK Set2DHCP
        VAR_INPUT
                Enable : bool;
                Perm : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                enableLast : bool;
                STS : bool;
        END_VAR

        IF ((Enable = TRUE) AND (enableLast = FALSE))THEN  (*Detect rising edge*)

        STS := EZ_Eth_SetEnableDHCPIpV4(Perm);     (*Set to DHCP and if keep setting on reboot based on Perm*)

        END_IF;

        enableLast := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_SetHostname

### Summary:

The EZ_Eth_SetHostname function is used to set the Ethernet communication port's Hostname. When this function is used to set the Hostname, the hostname will be kept until changed with this function or in the Bootloader screen.

### Format:

*BOOLvar* := EZ_Eth_SetHostname(*stringbuf*);

### Arguments:

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the function. Returns True (1) upon success |
| *stringbuf* | Variable (STRING) holding the name to use for the Ethernet Hostname. |

### Description:

The EZ_Eth_SetHostname sets the target's (device's) Ethernet communication hostname to the values specified in the *stringbuf* variable. The Hostname is stored until changed by this function or manually in the target's Bootloader screen. *BOOLvar* is True (1) when upon success.

### Example:

```
FUNCTION_BLOCK EthName
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
                Rtn : bool;
                hname: string := 'Pump_1';
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = true) AND (LastEn = False) then
                Rtn := EZ_Eth_SetHostname(hname);
                LastEn := true;
        End_If;


        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_Eth_SetStaticIPV4Addr

### Summary:

The EZ_SetStaticIPV4Addr function is used to configure the target's (device's) Ethernet communication port to Static IP type (IP address set locally, not DHCP), sets the IP address information to the target (device) and configures if the this change is to be persistent when power is cycled or the device is re-booted.

### Format:

*BOOLvar* := EZ_SetStaticIPV4Addr(*OurIP,Subnet,Gateway,DNS1,DNS2,Persist*); or
EZ_SetStaticIPV4Addr(*OurIP,Subnet,Gateway,DNS1,DNS2,Persist*);

### Arguments:

| | |
|---|---|
| *BOOLvar* | Function return holding variable (BOOL). Returns the status of the function's activity (optional). Returns True (1) upon success. |
| *OurIP* | Ethernet Port IP address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *Subnet* | Ethernet Port subnet address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *Gateway* | Ethernet Port gateway address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *DNS1* | Ethernet Port DNS1 address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *DNS2* | Ethernet Port DNS2 address (4 Bytes or more total) (ARRAY[ ] of USINT). |
| *Persist* | Variable to set if the device (target) should keep Static as the IP address type after re-boot (BOOL) or use the settings in the target's bootloader. When True (1), this sets a flag for the bootloader to make this change for all subsequent re-boots. |

### Description:

The EZ_SetStaticIPV4Addr function configures the type of IP address to static, configures the IP address information and configures if static is to be kept after re-boot of the target's (device's) Ethernet port. The *BOOLvar* return variable returns a True (1) when complete.The variable *OurIP* (array of USINT) holds the IP address, the variable *Subnet* (array of USINT) holds the subnet, the variable *Gateway* holds the gateway (array of USINT), the variable *DNS1* (array of USINT) holds the DNS1 address, the variable *DNS2* (array of USINT) holds the DNS2 address and the variable persist holds the setting if static IP is to be used after re-boot (when True (1), this sets a flag for the bootloader to make this change for all subsequent re-boots).

### Example:

```
FUNCTION_BLOCK Set2Static
        VAR_INPUT
                Enable : bool;
                Perm : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                enableLast : bool;
                STS : bool;
                OURIP : ARRAY[0..3] OF USINT := [192, 168,1,1];       (*Configure IP address info as*)
                SUBNET : ARRAY[0..3] OF USINT := [0,0,0,0];           (*part of variable definition*)
                GWAY : ARRAY[0..3] OF USINT := [0,0,0,0];
                DNS1 : ARRAY[0..3] OF USINT := [123,45,67,9];
                DNS2 : ARRAY[0..3] OF USINT := [101,121,32,5];
        END_VAR
```

Continued Next Page------»

```
        IF ((Enable = TRUE) AND (enableLast = FALSE))THEN  (*Detect rising edge*)

        (*Set to Static and if keep setting on reboot*)
                STS := EZ_Eth_SetStaticIpV4Addr(OURIP,SUBNET,GWAY,DNS1,DNS2,Perm);

        END_IF;

        enableLast := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FormatString

**Summary:**

The EZ_FormatString function is used to format a string to specific requirements similar to the standard 'C' PrintF. This is useful for formatting a string to send serial data or displaying data.

**Format:**

*DINTvar* := EZ_FormatString(*StrBuffer*, '*FormatString1 VarformatFlag,FormatString2 VarformatFlag..', Var1,Var2)*;

**Arguments:**

| | |
|---|---|
| *StrBuffer* | Destination string variable where the formatted resulting string is stored. |
| *DINTvar* | Function return holding variable (DINT). # of bytes written to the *StrBuffer* variable. |
| *FormatStringx* | String that formats the data of the variable to be added to the *StrBuffer*. All the *FormatString* entries are surrouned by single quotes ('). |
| *Varx* | The variables to format and add to the *StrBuffer*. The number of variables must match the number of *FormatString* entries and must be in the same order for proper functionality. Each Variable is separated by a comma (,). |
| *VarFormatFlag* | Variable format and Flags are used to identify variable types and formats as well as special control characters needed for specific formatting. Follows 'C' PrintF standard. |

%flag width .precision Example Text: OIL PSI %-3d

% - identifies the beginning of a variable or other type of text entry

flag - This flag is optional. Use the following flags to change the way data is transmitted.

| Flag | Description |
|---|---|
| - | Left align the variable within the specified width. Default is align right. |
| 0 | If width is prefixed with 0, leading zeros are added until the minimum width is reached.<br>If 0 and - are used together, the 0 is ignored. If 0 is specified in an integer format, the0 is ignored. |
| width | This flag is optional. Width is the number of characters that will be printed (total). |
| .precision | This flag is optional. The precision is the number of digits after the decimal point when using REAL variables. |

**Some common variable formats and flags are:**

| | |
|---|---|
| ***%d*** - Signed Integer | ***%X*** - Upper Case Hexadecimal |
| ***%u*** - Unsigned Integer | ***%f*** - Real or Float Variable |
| ***%x*** - Lower Case Hexadecimal | **%b** - binary |
| ***%o*** - Octal | %s - String |

To Print a '%' use ***%%***

To Print a Boolean as 0 or 1 use ***%d***

To Print a Boolean as OFF or ON use ***%O***

To Print a Boolean as FALSE or TRUE, use ***%T***

Toi add a carriage return, use ***$N***.

Continued Next Page------»

**Description:**

The EZ_FormatString takes one to multiple variables as an input and uses the *FormatString* and *Flags* to format the data that is stored in the *StrBuffer*. The StrBuffer must be large enough to hold the final formatted string data.

**Example:**

```
FUNCTION_BLOCK DisplayFormatData
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                Line1 : string[16];
                Line2 : string[16];
                Value1: Real;
                Value2: DINT;
                Value3: DINT;
                Value4: Real;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        Value1 := 12.567;               (*Set value for each variable*)
        Value2 := 24;
        Value3 := 967;
        Value4 := 8.1415;

        IF Enable = 1 THEN;

        (*Format Line1 as Value1 as V1: Real with 2 digits each side of decimal pt & Value 2 as V2: Integer*)
        EZ_FormatString(Line1,'V1:%2.2f  V2:%d',Value1,Value2);

        (*Format Line2 as Value3 as V3: Integer & Value 4 as V4: Real with 1 digit each side of decimal pt*)
        EZ_FormatString(Line2,'V3:%d  V4:%1.1f',Value3,Value4);

        EZ_LcdWrite(0,0,Line1);                 (*Write Line1 to LCD display*)
        EZ_LcdWrite(1,0,Line2);                 (*Write Line2 to LCD display*)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Close

**Summary:**

The EZ_FS_Close function is used to close an open file on the file system (SD Card). Any cached data is flushed and written to the file before it is closed.

**Format:**

*DINTvar* := EZ_FS_Close(*FileHandle*);

**Arguments:**

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

**Description:**

The EZ_FS_Close function closes the open file identified by *FileHandle*. Any data that is cached is automatically flushed from the cache and written to the file before it is closed. *DINTvar* returns the status of the file system action (CLOSE in this case).

**It is recommended for data reliability that open file(s) be closed upon completion of the action that caused them to be opened (read, write, append, etc). In the event of a power loss, un-written data (cached, file not closed) will be lost.**

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK FileSystem
    VAR_INPUT
        Enable : bool;
    END_VAR
    VAR_OUTPUT
        Q : bool;
        R: DINT;
    END_VAR
    VAR
    FileHandle : DINT;
    LastEnable : bool;
    DATA : STRING[40];
    BWR : UDINT;
    fresult : DINT;
    END_VAR

    DATA := 'Test value write'; (*set data to store as test*)

    (*Rising Edge only*)
    IF ((Enable = 1) AND (LastEnable = 0)) THEN
        (*Open / Create file test1.log and append to it*)
        fresult := EZ_FS_Open('m:\\test1.log', 16#112, FileHandle);

        IF (fresult = 0) THEN           (*File opened or created successfully*)
        (*Write test string to file*)
        fresult := EZ_FS_WriteStr(FileHandle,DATA,BWR);
        END_IF;

        (*Close file after use - Always*)
        fresult := EZ_FS_Close(FileHandle);

        R := fresult;
    END_IF;

    LastEnable := Enable;
    Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Flush

### Summary:

The EZ_FS_Flush function is used to flush any data that is cached and write it to the SD Card. This requires a an SD Card file to be opened and data existing (from a previous write data function) in the cache. Typically, little data is cached and the EZ_FS_CLOSE function automatically performs this flushing effect before closing a file on the SD Card.

### Format:

*DINTvar* := EZ_FS_Flush(*FileHandle*);

### Arguments:

*FileHandle* The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed).

*DINTvar* Function return holding variable (DINT). Returns the status of the function block. The codes are:

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

### Description:

The EZ_FS_Flush function flushes and stores any cached data in the open file identified by *FileHandle*. *DINTvar* returns the status of the file system action (FLUSH in this case). This function is typically not required if recommendations are followed (close all open files after action) as this flushing is performed automatically during a **EZ_FS_Close** operation. When files are opened, then accessed as needed (read/write) and then closed, this function is not required.

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK FileSystemFlush
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : STRING[100];
        BWR : UDINT;
        fresult : DINT;
        END_VAR

        DATA := '012345678abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'; (*set data to store
as test*)

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log and append to it*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#112, FileHandle);

                IF (fresult = 0) THEN           (*File opened or created successfully*)
                (*Write test string to file*)
                fresult :=      EZ_FS_WriteStr(FileHandle,DATA,BWR);
                END_IF;

                (*Flush the cache to file*)             (* This example is for referenc of the EZ_FS_Flush*)
                                                        (* Function only. The method of use is not recommended*)
                fresult := EZ_FS_Flush(FileHandle);     (* The EZ_FS_Close function should be used to close the*)
                                                        (* Open file after a write as the RECOMMENDED METHOD*)

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Open

**Summary:**

The EZ_FS_Open function is used to open file system (SD Card) files. This function includes additional command flags to configure additional features as part of the file open action (such as create, append, etc).

**Format:**

*DINTvar* := EZ_FS_Open(*FilePath, ModeFlag, FileHandle*);

**Arguments:**

*FileHandle* The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed).

*FilePath* The path used to find the file to be opened (STRING). The SD Card path begins with 'M:\\' The path must contain the entire path including the file name and extension.

*ModeFlag* Determines additional modes of operation associated with the EZ_FS_Open function (DINT) (this call only). Mode flags may be combined to simplify the EZ_FS_Open by combining the hex values shown.

**The Mode Flags are:**

| Value (hex) | Description |
|---|---|
| 16#00 | File Action - Open Existing File (Fails if file does not exist) |
| 16#01 | File Action - Read. Used to read data only. (Combine with Write for read/write access). |
| 16#02 | File Action - Write. Used to write data only. (Combine with Read for read/write access). |
| 16#04 | File Action - Create New. Used to read data only. (Fails if file already exists). |
| 16#08 | File Action - Create Always. Creates a new file. If the file already exists it will be truncated and over-written. |
| 16#10 | File Action - Open Always. Opens the file is it already exists. The file is created if it does not exist. (Combine with Append for locating the end of the file to begin writing to). If Append is not used then the EZ_FS_Seek function will be required to locate the end of the file data to begin writing. |
| 16#100 | File Action - Append. (Combined with Write for finding the end of the file and appending or Combine with Open Always to locate the end of the file to append upon file open). |

**Combination Examples:**

*Open existing file for read:*
fresult := EZ_FS_Open('m:\\test1.log', 16#1, fileHandle);
*Open existing file for write:*
fresult := EZ_FS_Open('m:\\test1.log', 16#2, fileHandle);
*Open existing file for read/write, create if not exist:*
fresult := EZ_FS_Open('m:\\test1.log', 16#13, fileHandle);
*Open existing file write, append, create if not exist:*
fresult := EZ_FS_Open('m:\\test1.log', 16#112, fileHandle);

*DINTvar* Function return holding variable (DINT). Returns the status of the function block. The codes are:

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |

Continued Next Page------»

| | |
|---|---|
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

**Description:**

The EZ_FS_Open function is used to open (optionally create or append) a file on the file system (SD Card). The file identifcation is automatically assigned and returned as *FileHandle* (DINT). This value must be kept and used for all subsequent file actions until the file is closed (using the EZ_FS_Close function). The *FilePath* (STRING) can be used direclty in the function call or as a string variable. This must be the full path, filename (including extension) for the file to be opened. The *ModeFlag* (DINT) can be used directly in the function call or as a DINT variable. This *ModeFlag* identifies the additional characteristics of the file open action (see list previous page). *DINTvar* returns the function status (success or error (see list above).

**It is recommended for data reliability that open file(s) be closed upon completion of the action that caused them to be opened (read, write, append, etc). In the event of a power loss, un-written data (cached, file not closed) will be lost.**

**Example:**

```
FUNCTION_BLOCK FileSystem
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : STRING[40];
        BWR : UDINT;
        fresult : DINT;
```

Continued Next Page------»

```
    END_VAR

    DATA := 'Test value write'; (*set data to store as test*)

    (*Rising Edge only*)
    IF ((Enable = 1) AND (LastEnable = 0)) THEN
            (*Open / Create file test1.log and append to it*)
            fresult := EZ_FS_Open('m:\\test1.log', 16#112, FileHandle);

            IF (fresult = 0) THEN              (*File opened or created successfully*)
            (*Write test string to file*)
            fresult :=          EZ_FS_WriteStr(FileHandle,DATA,BWR);
            END_IF;

            (*Close file after use - Always*)
            fresult :=          EZ_FS_Close(FileHandle);

            R := fresult;
    END_IF;

    LastEnable := Enable;
    Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Read

**Summary:**

The EZ_FS_Read function is used read data from the file system (SD Card). To use the EZ_FS_Read function, a file must be open (see EZ_FS_Open) and the location in the file to read from (see the EZ_FS_Seek function).

**Format:**

*DINTvar* := EZ_FS_Read(*FileHandle, DataBuffer, Offset, LenBytes, NBytes*);

**Arguments:**

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *DataBuffer* | Data buffer to store data read from the file (Array of USINT). |
| *Offset* | Offset (location) into *DataBuffer* to begin writing data to that is read from the file (UDINT). |
| *LenBytes* | Number of bytes to read from from the file (UDINT). |
| *NBytes* | Number of Bytes actually read from the file (UDINT). Returned from the function. |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

**Description:**

The EZ_FS_Read function is used to read data from the a file of the file system (SD Card). Before this function may be used to read data, a file must be open and the file's pointer set at the location to start reading data from (See EZ_FS_ Open and EZ_FS_Seek functions).

Continued Next Page------»

The data is read from the file identified by *FileHandle* beginning the file's internal pointer location. The number of bytes set by *LenBytes* is attempted to be read from the file and stored in the *DataBuffer* beginning at the *Offset* location. The *NBytes* returns the actual number of bytes read from the file. The *DINTvar* returns the status of the function's actions (successful or error, see list on the previous page).

**It is recommended for data reliability that open file(s) be closed upon completion of the action that caused them to be opened (read, write, append, etc). In the event of a power loss, un-written data (cached, file not closed) will be lost.**

**Example:**

```
FUNCTION_BLOCK FSReadbytes
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
                BT : DINT;
                DT1 : DINT;
                DT2 : DINT;
                DT3 : DINT;
                DT4 : DINT;
                DT5 : DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : ARRAY[0..4] of USINT;
        Loc : UDINT := 11;
        fresult : DINT;
        BWR : UDINT;
        END_VAR

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log and append to it*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#13, FileHandle);

                IF (fresult = 0) THEN           (*File opened successfully*)
                (*Seek location in file to read from*)
                fresult := EZ_FS_Seek(FileHandle,Loc);
                END_IF;

                IF (fresult = 0) THEN           (*Location pointer set*)
                (*read data from file*)
                fresult := EZ_FS_Read(FileHandle,DATA,0,5,BWR);
                BT := UDINT_TO_DINT(BWR);
                DT1 := USINT_TO_DINT(DATA[0]);
                DT2 := USINT_TO_DINT(DATA[1]);
                DT3 := USINT_TO_DINT(DATA[2]);
                DT4 := USINT_TO_DINT(DATA[3]);
                DT5 := USINT_TO_DINT(DATA[4]);

                ELSE
                DT1 := 0;
```

Continued Next Page------»

```
                    DT2 := 0;
                    DT3 := 0;
                    DT4 := 0;
                    DT5 := 0;
                    BT := 0;

                    END_IF;

                    (*Close file after use - Always*)
                    fresult := EZ_FS_Close(FileHandle);

                    R := fresult;


          END_IF;

          LastEnable := Enable;
          Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_ReadStr

### Summary:

The EZ_FS_ReadStr function is used to read string data from the file system (SD Card). To use the EZ_FS_ReadStr function, a file must be open (see EZ_FS_Open) and the location in the file to read from (see the EZ_FS_Seek function).

### Format:

*DINTvar* := EZ_FS_ReadStr(*FileHandle, DataBuffer, LenBytes, NBytes*);

### Arguments:

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *DataBuffer* | Data buffer to store string data read from the file (STRING). Size of string must be sufficient to hold the number of bytes being read from the file. |
| *LenBytes* | Number of bytes to read from from the file (UDINT). |
| *NBytes* | Number of Bytes actually read from the file (UDINT). Returned from the function. |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

### Description:

The EZ_FS_ReadStr function is used to read string data from the a file of the file system (SD Card). Before this function may be used to read data, a file must be open and the file's pointer set at the location to start reading data from (See EZ_FS_Open and EZ_FS_Seek functions).

Continued Next Page------»

The data is read from the file identified by *FileHandle* beginning with the file's internal pointer location. The number of bytes set by *LenBytes* is attempted to be read from the file and stored in the *DataBuffer*. The *NBytes* returns the actual number of bytes read from the file. The *DINTvar* returns the status of the function's actions (successful or error, see list on the previous page).

**It is recommended for data reliability that open file(s) be closed upon completion of the action that caused them to be opened (read, write, append, etc). In the event of a power loss, un-written data (cached, file not closed) will be lost.**

**Example:**

```
FUNCTION_BLOCK FSReadString
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
                BT : DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : STRING[5];
        ReadBytes : UDINT := 5;
        Loc : UDINT := 11;
        fresult : DINT;
        BWR : UDINT;
        END_VAR

        DATA := 'Test value write'; (*set data to store as test*)

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log and append to it*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#13, FileHandle);

                IF (fresult = 0) THEN           (*File opened successfully*)
                (*Seek location in file to read from*)
                fresult :=      EZ_FS_Seek(FileHandle,Loc);
                END_IF;

                IF (fresult = 0) THEN           (*Location pointer set*)
                (*read string from file*)
                fresult :=      EZ_FS_ReadStr(FileHandle,DATA,ReadBytes,BWR);
                BT := UDINT_TO_DINT(BWR);
                END_IF;

                (*Close file after use - Always*)
                fresult :=      EZ_FS_Close(FileHandle);

                IF (fresult = 0) THEN           (*All file functions successful*)
                (* Send data to Serial Port (UART) as string*)
                        while EZ_UartWriteStr(FD_UART3, DATA) <= 0 do (*write to serial port*)
                                ;
                        end_while;
```

Continued Next Page------»

```
                END_IF;

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Seek

### Summary:

The EZ_FS_Seek function is used to locate a position (pointer position) within an open file. This location would typically be used for writing or reading data (See reading and writing functions EZ_FS_xxx). For writing data, this is typically the end of the file. A file must be open for this function to operate.

### Format:

*DINTvar* := EZ_FS_Seek(*FileHandle, Offset*);

### Arguments:

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *Offset* | Number of bytes to offset the location pointer by (location inside the file) (UDINT). |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

### Description:

The EZ_FS_Seek function seeks and places the location pointer in the file identified by *FileHandle*, offset by the *Offset* number of bytes. The *DINTvar* returns the function status (successful or error - See list above.) Typically this function is used to locate the end of a file for writing (requires the use of the EZ_FS_Size function) but it can be used to position to read / write data at any position in the file.

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK FSSeekWrite
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : STRING[50];
        fresult : DINT;
        FS : DINT;
        BWR : UDINT;
        END_VAR

        DATA := 'Written 2 End Current File Opened'; (*set data to store as test*)

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log /create if not exist*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#13, FileHandle);

                IF (fresult = 0) THEN           (*File opened or created successfully*)
                (*Get size of file*)
                FS := EZ_FS_Size(FileHandle);

                (*Seek end of file*)
                fresult := EZ_FS_Seek(FileHandle,DINT_TO_UDINT(FS));
                END_IF;

                IF (fresult = 0) THEN           (*File end found*)
                (*Write test string to file*)
                fresult := EZ_FS_WriteStr(FileHandle,DATA,BWR);

                END_IF;

                (*Close file after use - Always*)
                fresult := EZ_FS_Close(FileHandle);

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Size

### Summary:

The EZ_FS_Size function is used read and return a file system (SD Card) file size in bytes.

### Format:

*DINTvar* := EZ_FS_Size(*FileHandle*);

### Arguments:

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *DINTvar* | Function return holding variable (DINT). Returns the number of bytes in the file identified by *FileHandle*. |

### Description:

The EZ_FS_Size function reads and returns the file size of the file identified by *FileHandle*. The *DINTvar* return value is the actual size of the file in bytes. A file must be opened for this function to operate (see the EZ_FS_Open function).

### Example:

```
FUNCTION_BLOCK FSSeekWrite
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : STRING[50];
        fresult : DINT;
        FS : DINT;
        BWR : UDINT;
        END_VAR

        DATA := 'Written 2 End Current File Opened'; (*set data to store as test*)

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log /create if not exist*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#13, FileHandle);

                IF (fresult = 0) THEN           (*File opened or created successfully*)
                (*Get size of file*)
                FS := EZ_FS_Size(FileHandle);

                (*Seek end of file*)
                fresult := EZ_FS_Seek(FileHandle,DINT_TO_UDINT(FS));
                END_IF;
```

Continued Next Page------»

```
                IF (fresult = 0) THEN                    (*File end found*)
                (*Write test string to file*)
                fresult := EZ_FS_WriteStr(FileHandle,DATA,BWR);

                END_IF;

                (*Close file after use - Always*)
                fresult := EZ_FS_Close(FileHandle);

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Truncate

### Summary:

The EZ_FS_Truncate function is used to truncate an open file (contents and size) based on the file pointer location (within the file). A file must be open (see the EZ_FS_Open function) and the file's internal location pointer be set (see the EZ_FS_Seek function) to the desired location.

### Format:

*DINTvar* := EZ_FS_Truncate(*FileHandle*);

### Arguments:

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

### Description:

The EZ_FS_Truncate function actually truncates the file (contents and file size) at the file's current location pointer (internal to the file). The function EZ_FS_Seek is used to set the file's current location point. This function works only on the open file identified in the *FileHandle*. The Dint returns the function status (successful or error, see list above).

For example, if the file's locaton point were set at the beginning of a file and the EZ_FS_Truncate was called, the file would be erased and file size set to zero. If the file's locaton point were set at the middle of a file and the EZ_FS_Trun-

Continued Next Page------»

cate was called, the last half of the file file would be erased and file size set to half the original size. If the file's locaton point were set at the end of a file and the EZ_FS_Truncate was called, the file would not be affected at all.

**Example:**

```
FUNCTION_BLOCK FSTrunc
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        fresult : DINT;
        Offset : UDINT := 20;
        END_VAR

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#13, FileHandle);

                IF (fresult = 0) THEN           (*File opened or created successfully*)
                (*Locate file point 20 bytes into file*)
                fresult := EZ_FS_Seek(FileHandle,Offset);
                END_IF;


                IF (fresult = 0) THEN           (*Pointer Located*)
                (*Truncate file*)
                fresult := EZ_FS_Truncate(FileHandle);
                END_IF;

                (*Close file after use - Always*)
                fresult := EZ_FS_Close(FileHandle);

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_Write

**Summary:**

The EZ_FS_Write function is used write data to the file system (SD Card). To use the EZ_FS_Write function, a file must be open (see EZ_FS_Open) and the location in the file to write to (usually the end) (see EZ_FS_Seek or Append Mode of EZ _FS_Open).

**Format:**

*DINTvar* := EZ_FS_Write(*FileHandle, DataBuffer, Offset, Len, NBytes*);

**Arguments:**

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *DataBuffer* | Data to be written to the file (Array of USINT). |
| *Offset* | Offset / Location into the *DataBuffer* to start writing from (UDINT). This is the location in the *DataBuffer* that will be the beginning point where data that will be written to the file will be read from. |
| *Len* | Length of data in bytes from *DataBuffer* to write to the file (UDINT) beginning with *Offset*. |
| *NBytes* | Number of Bytes actually written to the file (UDINT). Returned from the function |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| **Value** | **Description** |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

Continued Next Page------»

**Description:**

The EZ_FS_Write function writes data from *DataBuffer* variable to the file identified by the *FileHandle*. A file must already be open and location to write to (usually the end of the file) (See the functions EZ_FS_Open and EZ_FS_Seek). The *Offset* variable identifies the beginning point within the *DataBuffer* to begin getting data to write to the file. The *Len* variable identifies the number of bytes to gather from *DataBuffer* and write to the file. The number of bytes actually written to the file is returned by the *NBytes* variable. The *DINTvar* returns the status of the function (success or error). See the list on the previous page.

**Example:**

```
FUNCTION_BLOCK FSWrite
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
                BYT: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        BWR : UDINT;
        Offset : UDINT := 10;
        Length : UDINT := 6;
        fresult : DINT;
        DATA : ARRAY[0..15] of USINT;
        END_VAR

                (* Set values of data in byte array to send*)
        DATA[0] := 48; DATA[1] := 49; DATA[2] := 50; DATA[3] := 51; DATA[4] := 52; DATA[5] := 53; DATA[6] := 54;
        DATA[7] := 55; DATA[8] := 56; DATA[9] := 57; DATA[10] := 65; DATA[11] := 66; DATA[12] := 67; DATA[13] := 68;
        DATA[14] := 69; DATA[15] := 70;

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log and append to it*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#112, FileHandle);

                IF (fresult = 0) THEN           (*File opened or created successfully*)
                (*Write data array to file*)
                fresult :=      EZ_FS_Write(FileHandle,DATA,Offset,Length,BWR);
                BYT := UDINT_TO_DINT(BWR);              (*Convert to DINT to view in ladder*)
                END_IF;

                (*Close file after use - Always*)
                fresult :=      EZ_FS_Close(FileHandle);

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_FS_WriteStr

**Summary:**

The EZ_FS_WriteStr function is used to write string data to the file system (SD Card). The string data can be formatted using other structured text functions as required. To use the EZ_FS_WriteStr function, a file must be open (see EZ_FS_Open) and the location in the file to write to (usually the end) (see EZ_FS_Seek or Append Mode of EZ _FS_Open).

**Format:**

*DINTvar* := EZ_FS_WriteStr(*FileHandle, Data, NBytes*);

**Arguments:**

| | |
|---|---|
| *FileHandle* | The file system File identification of the file (DINT).This value is generated by the file system when an EZ_FS_Open structured text function is called automatically. This FileHandle must be used by all EZ_FS_ (file system) structured text commands when accessing this file (it must be kept until the file is closed). |
| *Data* | String data to be written to the file. Can be entered directly or as variable (STRING). |
| *NBytes* | Number of Bytes actually written to the file (UDINT). Returned from the function. |
| *DINTvar* | Function return holding variable (DINT). Returns the status of the function block. The codes are: |

| Value | Description |
|---|---|
| 0 | File operation succeeded with no errors. |
| 1 | A hard error occurred. The SD Card may have a problem. All file system functions will not work except for EZ_FS_CLOSE. |
| 2 | Assertion Failed. A problem may exist with the FAT file structure. |
| 3 | The physical drive cannot be accessed. Verify the SD Card is installed. |
| 4 | File Not Found - Verify the file exists or create the file on the SD Card |
| 5 | The file path was not found - Verify the path is correct to the file. |
| 6 | The file path name (string) is invalid. Correct the path name (string). |
| 7 | Access Denied. Access is prohibited as due to incorrect file system flag (see EZ_FS_Open) or the directory is full. |
| 8 | Access Denied. A duplicate name is trying to be used and duplicate names are not allowed. |
| 9 | The file or file directory object is invalid. The file may have been closed or not opened. |
| 10 | Write Protected. The SD Card is write protected. |
| 11 | Drive Number is invalid. An invalid drive number was used in the path name (string). |
| 13 | There is no valid FAT volume. The SD Card needs formatted to FAT 16 or 32. The SD card FAT file system may be corrupted or the SD Card may have failed. |
| 15 | The file system operation did not complete in the allotted time |
| 16 | The file / file system is currently locked by another function or process and not and is not available. Retry after the lock is released. |
| 17 | Long File Name buffer could not be created due to memory allocation or size. |
| 18 | Too Many files are open. The maximum number of allowed open files has been reached. |
| 19 | Invalid parameter. A parameter used is invalid. |

**Description:**

The EZ_FS_WriteStr function writes string data as directly entered or by *Data* variable to the file identified by the *FileHandle*. A file must already be open and location to write to (usually the end of the file) (See the functions EZ_FS_Open and EZ_FS_Seek). The number of bytes actually written to the file is returned by the *NBytes* variable. The *DINTvar* returns the status of the function (success or error). See the list above.

Continued Next Page------»

**It is recommended for data reliability that open file(s) be closed upon completion of the action that caused them to be opened (read, write, append, etc). In the event of a power loss, un-written data (cached, file not closed) will be lost.**

**Example:**

```
FUNCTION_BLOCK FileSystem
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                R: DINT;
        END_VAR
        VAR
        FileHandle : DINT;
        LastEnable : bool;
        DATA : STRING[40];
        BWR : UDINT;
        fresult : DINT;
        END_VAR

        DATA := 'Test value write'; (*set data to store as test*)

        (*Rising Edge only*)
        IF ((Enable = 1) AND (LastEnable = 0)) THEN
                (*Open / Create file test1.log and append to it*)
                fresult := EZ_FS_Open('m:\\test1.log', 16#112, FileHandle);

                IF (fresult = 0) THEN           (*File opened or created successfully*)
                (*Write test string to file*)
                fresult :=      EZ_FS_WriteStr(FileHandle,DATA,BWR);
                END_IF;

                (*Close file after use - Always*)
                fresult :=      EZ_FS_Close(FileHandle);

                R := fresult;
        END_IF;

        LastEnable := Enable;
        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GetBootloaderVersion

### Summary:

The EZ_GetBootloaderVersion function reads and returns the current hardware target's on-board Bootloader software version number into an array of 4 bytes of unsigned integers (USINT).

### Format:

EZ_GetBootloaderVersion(*Array[] of Byte buffer, Offset*);

### Arguments:

| | |
|---|---|
| *Array[] Byte buffer* | Array of unsigned integer (USINT) bytes (4 minimum) to store the returned version numbers into. |
| *Offset* | DINT. Location (number of bytes) into the array of unsigned integers (Array[] Byte buffer) to begin writing the version information. It will write starting at this byte offset and write all four bytes. |

### Description:

The EZ_GetBootloaderVersion reads the actual PLC on a Chip (hardware target) stored Bootloader version number. The typical bootloader version is formatted as: xxx.xxx.xxx.xxx. The number of digits can vary. For example, the version may be 1.1.4.0. This can be useful, depending upon the application for identifying the version number for compatibility and sending the version number using communications or to a local display.

### Example:

This example reads the bootloader version and converts each individual USINT byte into a DINT and stores it in the VB1 - VB4 variables. These are outputs in the ladder program. The USINT bytes or DINT values can be used using other functions to display to the LCD (if available) or send via communications (serial, etc).

```
FUNCTION_BLOCK FB_BootLDRinfo
   VAR_INPUT
      Enable : bool;
   END_VAR
   VAR_OUTPUT
      Q : bool;
      VB1: DINT;
      VB2: DINT;
      VB3: DINT;
      VB4: DINT;
   END_VAR
   VAR
      lastEn : bool;
      BootloaderVer : array[0..3] of USINT;
   END_VAR

   if(Enable = True) AND (lastEn = False) then
      EZ_GetBootloaderVersion(BootloaderVer, 0);        (* Get bootloader USINT bytes *)

      VB1:= USINT_TO_DINT(BootloaderVer[0]);            (* Convert to DIN for output of FB *)
      VB2:= USINT_TO_DINT(BootloaderVer[1]);            (* Convert to DIN for output of FB *)
      VB3:= USINT_TO_DINT(BootloaderVer[2]);            (* Convert to DIN for output of FB *)
```

Continued Next Page------»

```
        VB4:= USINT_TO_DINT(BootloaderVer[3]);              (* Convert to DIN for output of FB *)

    end_if;

    Q := Enable;
    lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_GetKernelVersion

### Summary:

The EZ_GetKernelVersion function reads and returns the current hardware target's on-board loaded kernel software version number into an array of 4 bytes of unsigned integers (USINT).

### Format:

EZ_GetKernelVersion(*Array[] of Byte buffer, Offset*);

### Arguments:

| | |
|---|---|
| *Array[] Byte buffer* | Array of unsigned integer (USINT) bytes (4 minimum) to store the returned version numbers into. |
| *Offset* | DINT. Location (number of bytes) into the array of unsigned integers (Array[] Byte buffer) to begin writing the version information. It will write starting at this byte offset and write all four bytes. |

### Description:

The EZ_GetKernelVersion reads the actual PLC on a Chip (hardware target) stored kernel version number. The typical kernel version is formatted as: xxx.xxx.xxx.xxx. The number of digits can vary. For example, the version may be 1.1.4.0. This can be useful, depending upon the application for identifying the version number for compatibility and sending the version number using communications or to a local display.

### Example:

This example reads the kernel version and converts each individual USINT byte into a DINT and stores it in the VB1 - VB4 variables. These are outputs in the ladder program.  The USINT bytes or DINT values can be used using other functions to display to the LCD (if available) or send via communications (serial, etc).

```
FUNCTION_BLOCK FB_Krnlinfo
  VAR_INPUT
    Enable : bool;
  END_VAR
  VAR_OUTPUT
    Q : bool;
    VB1: DINT;
    VB2: DINT;
    VB3: DINT;
    VB4: DINT;
  END_VAR
  VAR
    lastEn : bool;
   KrnlVer : array[0..3] of USINT;
  END_VAR

  if(Enable = True) AND (lastEn = False) then
    EZ_GetKernelVersion(KrnlVer, 0);           (* Get Kernel Version USINT bytes *)

    VB1:= USINT_TO_DINT(KrnlVer[0]);                (* Convert to DIN for output of FB *)
    VB2:= USINT_TO_DINT(KrnlVer[1]);                (* Convert to DIN for output of FB *)
    VB3:= USINT_TO_DINT(KrnlVer[2]);                (* Convert to DIN for output of FB *)
```

Continued Next Page------»

```
        VB4:= USINT_TO_DINT(KrnlVer[3]);                    (* Convert to DIN for output of FB *)

    end_if;

    Q := Enable;
    lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_GetLadderBuild

**Summary:**

The EZ_GetLadderBuild function reads and returns the target's loaded ladder diagram software Build number into an array of 4 bytes of unsigned integers (USINT). The build number increments automatically each time a program is compiled (unless changed in the Project settings, version tab).

**Format:**

*varUDINT* := EZ_GetLadderBuild();

**Arguments:**

| | |
|---|---|
| *varUDINT* | Unsigned Double Integer (UDINT). This variable is used to store the current ladder build into when read by the function. |

**Description:**

The EZ_GetLadderBuild reads the actual ladder diagram build number. The build number is a single number. The build number automatically increments each time the ladder diagram is compiled in EZ LADDER. It can also be changed in the EZ LADDER Project Settings Menu, the version tab. This build maybe useful, depending upon the application for reporting purposes and sending the build number using communications or to a local display.

**Example:**

This example reads the build and converts it into a DINT and stores it in the BLD variable. The BLD variable is an output in the ladder program. The UDINT or DINT values can be used using other functions to display to the LCD (if available) or send via communications (serial, etc).

```
FUNCTION_BLOCK FB_LDRBuildinfo
   VAR_INPUT
      Enable : bool;
   END_VAR
   VAR_OUTPUT
      Q : bool;
      BLD: DINT;
   END_VAR
   VAR
      lastEn : bool;
      LdrBuild : UDINT;
   END_VAR

   if(Enable = True) AND (lastEn = False) then
     LDrBuild:= EZ_GetLadderBuild();            (* Get Ladder Build *)
     BLD:= UDINT_TO_DINT(LDrBuild);
   end_if;

   Q := Enable;
   lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_GetLadderVersion

### Summary:

The EZ_GetLadderVersion function reads and returns the current hardware target's on-board loaded ladder diagram software version number into an array of 4 bytes of unsigned integers (USINT).

### Format:

EZ_GetLadderVersion(*Array[] of Byte buffer, Offset*);

### Arguments:

| | |
|---|---|
| *Array[] Byte buffer* | Array of unsigned integer (USINT) bytes (4 minimum) to store the returned version numbers into. |
| *Offset* | DINT. Location (number of bytes) into the array of unsigned integers (Array[] Byte buffer) to begin writing the version information. It will write starting at this byte offset and write all four bytes. |

### Description:

The EZ_GetLadderVersion reads the actual running ladder diagram's version number. This version is set by the programmer in the Project Settings menu's version tabl. (The typical ladder diagram version is formatted as: xxx.xxx.xxx.xxx. The number of digits can vary. For example, the version may be 1.1.4.0. This can be useful, depending upon the application for identifying the version number for compatibility and sending the version number using communications or to a local display.

### Example:

This example reads the ladder diagram version and converts each individual USINT byte into a DINT and stores it in the VB1 - VB4 variables. These are outputs in the ladder program. The USINT bytes or DINT values can be used using other functions to display to the LCD (if available) or send via communications (serial, etc).

```
FUNCTION_BLOCK FB_LDRinfo
    VAR_INPUT
        Enable : bool;
    END_VAR
    VAR_OUTPUT
        Q : bool;
        VB1: DINT;
        VB2: DINT;
        VB3: DINT;
        VB4: DINT;
    END_VAR
    VAR
        lastEn : bool;
       LadderVer : array[0..3] of USINT;
    END_VAR

    if(Enable = True) AND (lastEn = False) then
        EZ_GetLadderVersion(LadderVer, 0);          (* Get Ladder Diagram version USINT bytes *)

        VB1:= USINT_TO_DINT(LadderVer[0]);                 (* Convert to DIN for output of FB *)
        VB2:= USINT_TO_DINT(LadderVer[1]);                 (* Convert to DIN for output of FB *)
```

Continued Next Page------»

```
        VB3:= USINT_TO_DINT(LadderVer[2]);                    (* Convert to DIN for output of FB *)
        VB4:= USINT_TO_DINT(LadderVer[3]);                    (* Convert to DIN for output of FB *)

    end_if;

    Q := Enable;
    lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_GetSerialNumber

**Summary:**

The EZ_GetSerialNumber function reads and returns the current hardware target serial number that is stored in the PLC on a Chip.

**Format:**

*DINTvar* := EZ_GetSerialNumber();

**Arguments:**

| | |
|---|---|
| *DINTvar* | Function return holding variable (DINT). Returns the actual stored hardware target's Serial Number (stored in the PLC on a Chip). |

**Description:**

The EZ_GetSerialNumber reads the actual PLC on a Chip (hardware target) stored Serial Number and returns the serial number as *DINTvar*.

**Example:**

```
FUNCTION_BLOCK ReadSN
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_TEMP
        FncReturn : DINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                SNum: DINT;
        END_VAR

        IF Enable = True THEN;                          (*Check for function enabled?*)
                FncReturn := EZ_GetSerialNumber();      (*Read Serial Number from unit*)
                SNum := FncReturn;                      (*Copy to output var to view*)
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GetTickCount

**Summary:**

The EZ_GetTickCount function is used get the current PLC on a Chip processor Tick count.

**Format:**

*UDINTvar* := EZ_GetTickCount();

**Arguments:**

| | |
|---|---|
| *UDINTvar* | Returns the number of elapsed milliseconds since the system started. |

**Description:**

The EZ_GetTickCount queries the PLC on a Chip and returns the number of milliseconds (ticks) since the system was started (powered up).

**Example:**

```
FUNCTION_BLOCK GetTickCount
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                Cval: DINT;
        END_VAR
        VAR_TEMP
                Tmx : UDINT;
                Tval: DINT;
        END_VAR

        IF Enable = 1 THEN;
                Tval := UDINT_TO_DINT(EZ_GetTickCount()); (*Get tickcount and convert to DINT to make viewable*)
                Cval := Tval;                             (*Copy to output variable to view*)
        ELSE
                Tval := 0;              (*Update output vars with 0*)
        End_If;
                Q := Enable;
END_FUNCTION_BLOCK
```

## EZ_GPIO_Init

### Summary:

The EZ_GPIO_Init function is used to initialize and configure GPIO pins (PLC on a Chip Digital I/O) from structured text as an input pin or an output pin.

### Format:

*BOOLvar* := EZ_GPIO_Init(*GPIOnum*, *PinSel*, *Mode*);

### Arguments:

| | |
|---|---|
| *GPIOnum* | GPIO pin number to configure (UDINT). This is the actual GPIO number (not a variable name associated with the GPIO number. |
| *PinSel* | Pin configuration. Enter 16#100 for digital input or 16#200 for digital output (UDINT). |
| *Mode* | Configurable Pull-up or Pull-down mode. 1 for pull-down, 2 for pull-up (PLC on Chip internal resistance)(UDINT). |
| *BOOLvar* | Function return holding variable (BOOL). Returns true (1) when configuration is successful. |

### Description:

The EZ_GPIO_Init configures PLC on a Chip device GPIO (general purpose I/O) pins from structured text as either input or output and can set a pull-up or pull-down mode of operation. As this function configures GPIO directly from structured text, the GPIO is not required to be added or configured in the ladder diagram. Once GPIO is configured, structured text function blocks can access the GPIO directly without any ladder diagram integration.

This function may also be called without the use of the Boolvar variable as shown in the example.

This function should only be used to configure GPIO not configured in the ladder diagram.

### Example:

```
FUNCTION_BLOCK ConfigIO
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_TEMP
        Tmp : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 Then;

        Tmp := EZ_GPIO_Init(118,16#100,2);     (*Configure I/O Input using variable to call*)

        EZ_GPIO_Init(119,16#100,2);    (*Configure I/O Input without variable to call*)
        EZ_GPIO_Init(124,16#200,2);    (*Configure I/O output without variable to call*)
        EZ_GPIO_Init(125,16#200,2);    (*Configure I/O output without variable to call*)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GPIO_Read

### Summary:

The EZ_GPIO_Read function is used to read the current status of a GPIO (digital input) within a structured text function or function block.

### Format:

*BOOLvar* := EZ_GPIO_Read(*GPIOnum*);

### Arguments:

| | |
|---|---|
| *GPIOnum* | GPIO number to read the current status (UDINT). This is the actual GPIO number (not a vari able name associated with the GPIO number. |
| *BOOLvar* | Function return holding variable (BOOL). Returns the actual status of the digital input as true (1) or false (0). |

### Description:

The EZ_GPIO_Read directly accesses the digital input from the structured text (outside of the ladder diagram digital I/O) and reads the current status of the input then returns the status as *Boolvar*. A zero represents false and a 1 represents true.

### Example:

```
FUNCTION_BLOCK ReadGPIO
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_TEMP
        Input1 : bool;
        Input2 : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 THEN;

        Input1:=EZ_GPIO_READ(7);            (* Read GPIO Input *)
        Input2:=EZ_GPIO_READ(9);            (* Read GPIO Input *)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GPIO_Write

### Summary:

The EZ_GPIO_Write function is used to set the current state of a GPIO (digital output) within a structured text function or function block.

### Format:

EZ_GPIO_Write(*GPIOnum,State*);

### Arguments:

| | |
|---|---|
| *GPIOnum* | GPIO number to set the current state of (UDINT). This is the actual GPIO number (not a vari able name associated with the GPIO number. |
| *State* | The desired state to set the GPIO (digital output) to (true or 1, false or 0) (BOOL) |

### Description:

The EZ_GPIO_Write directly accesses the digital output from the structured text (outside of the ladder diagram digital I/O) and sets the current *State* of the output. A zero represents false and a 1 represents true. There is no return value from this function.

### Example:

```
FUNCTION_BLOCK WriteGPIO
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 THEN;

        EZ_GPIO_Write(122,1);                (* Write GPIO ON *)
        EZ_GPIO_Write(123,0);                (* Write GPIO OFF *)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GPS_GetDateTimeUTC

### Summary:

The EZ_GPS_GetDateTimeUTC function is used to access the GPS message RMC and return the current month, day, year, hour, minute and second that was received from the GPS satellite. A sytem tick time is also received to identify when the last data was received.

### Format:

BOOLvar := EZ_GPS_GetDateTimeUTC(*Tick,Month,Day,Year,Hour,Minute,Second*);

### Arguments:

| | |
|---|---|
| *BOOLvar* | Return status holding variable (BOOL). Set to 1 or True when data is valid (data received within last 2 seconds). Set to 0 or False when data is invalid (data not received within last 2 seconds). |
| *Tick* | Tick time the last message was received.(UDINT) |
| *Month* | Message returned date value for Month (DINT). |
| *Day* | Message returned date value for Day (DINT). |
| *Year* | Message returned date value for Year (DINT). |
| *Hour* | Message returned time value for Hour (DINT). |
| *Minute* | Message returned time value for Minute (DINT). |
| *Second* | Message returned time value for Second (DINT). |

### Description:

The EZ_GPS_GetDateTimeUTC reads and decodes the message RMC from the satellite via the GPS and then returns the *Tick* (system tick time) and variables for the date and time. The *BoolVar* return variable is set 1 or True for as long as data is not older than 2 seconds (2 seconds since last message was received). This return value and the *Tick* time may be used to identify if the GPS data is valid. The date / time when valid may be used to sycn a real time clock or for data-logging (with our without real time clock).

### Example:

```
FUNCTION_BLOCK GPSDateTime
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                valid : bool := False;
                MN : DINT;
                DY : DINT;
                YR : DINT;
                HR : DINT;
                MNT : DINT;
                SEC : DINT;
        END_VAR
        VAR
                tickRecv : UDINT;
                lastTickRecv : UDINT;
                month : DINT;
                day : DINT;
                year : DINT;
                hours : DINT;
                minutes : DINT;
```

Continued Next Page------»

```
		seconds : DINT;
	END_VAR

	IF (Enable = TRUE) THEN;
		valid := EZ_GPS_GetDateTimeUTC(tickRecv,month,day,year,hours,minutes,seconds);
		MN := month;
		DY := day;
		YR := year;
		HR := hours;
		MNT := minutes;
		SEC := seconds;

	END_IF;

	Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GPS_GetMovement

**Summary:**

The EZ_GPS_GetMovement function is used to access the GPS message RMC and return the speed of movement and course of movement for the GPS receiver. A sytem tick time is also received to identify when the last data was received.

**Format:**

BOOLvar := EZ_GPS_GetMovement(*Tick,Speed, Course*);

**Arguments:**

| | |
|---|---|
| *BOOLvar* | Return status holding variable (BOOL). Set to 1 or True when data is valid (data received within last 2 seconds). Set to 0 or False when data is invalid (data not received within last 2 seconds). |
| *Tick* | Tick time the last message was received.(UDINT) |
| *Speed* | Speed over the ground in Knots (REAL). |
| *Course* | Course - Track angle in degrees (REAL). |

**Description:**

The EZ_GPS_GetMovement reads and decodes the message RMC from the satellite via the GPS and then returns the *Tick* (system tick time) and variables for the *Speed* (speed over the ground in knots) and *Course* (Track angle in degrees). The *BoolVar* return variable is set 1 or True for as long as data is not older than 2 seconds (2 seconds since last message was received). This return value and the *Tick* time may be used to identify if the GPS data is valid.

**Example:**

```
FUNCTION_BLOCK GPSMove
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                valid : bool := FALSE;
                SPD : REAL;
                CRSE: REAL;
        END_VAR
        VAR
                tickRecv : UDINT;
                speed : REAL;
                course : REAL;
        END_VAR

        IF (Enable = TRUE) THEN
                valid := EZ_GPS_GetMovement(tickRecv,speed,course);
                SPD := speed;
                CRSE:= course;

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GPS_GetPosition

### Summary:

The EZ_GPS_GetPosition function is used to access the GPS message GGA and return location information including # of satellites, latitude, longitude, and altitude. A sytem tick time is also received to identify when the last data was received.

### Format:

BOOLvar := EZ_GPS_GetPosition(*Tick,Sats,Latitude,Longitude,Altitude*);

### Arguments:

| | |
|---|---|
| *BOOLvar* | Return status holding variable (BOOL). Set to 1 or True when data is valid (data received within last 2 seconds). Set to 0 or False when data is invalid (data not received within last 2 seconds). |
| *Tick* | Tick time the last message was received.(UDINT) |
| *Sats* | # of Satelites used (BYTE). |
| *Latitude* | Latitude (LREAL). Format: +/-dddmm.sss for degrees + minutes + seconds<br>+ for East, - for West |
| *Longitude* | Longitude (LREAL). Format: +/-dddmm.sss for degrees + minutes + seconds<br>+ for North, - for South |
| *Altitude* | Altitude in meters.(REAL). |

### Description:

The EZ_GPS_GetPosition reads and decodes the message GGA from the satellite via the GPS and then returns the *Tick* (system tick time) and variables for the *Sats* (number of satelites used), *Latitude*, *Longitude*, and *Altitude* (meters). The *BoolVar* return variable is set 1 or True for as long as data is not older than 2 seconds (2 seconds since last message was received). This return value and the *Tick* time may be used to identify if the GPS data is valid.

This data received using this function cannot be passed directly to the ladder diagram without data type conversions as the data types are not supported directly in the ladder diagram. The data may be sent via serial (UART), LCD display and VersaCloud M2M+IoT or COAP cloud solutions as a string.

### Example:

```
FUNCTION_BLOCK GPSpos
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                valid : bool := FALSE;
        END_VAR
        VAR
                tickRecv : UDINT;
                lastTickRecv : UDINT;
                satsUsed : BYTE;
                latitude : LREAL;
                longitude : LREAL;
                altitude : REAL;
                buffer : string[200];
```

Continued Next Page------»

```
        buffer2 : string[100];
    END_VAR

    IF (Enable = TRUE) THEN
        valid := EZ_GPS_GetPosition(tickRecv,satsUsed,latitude,longitude,altitude);

        if (tickRecv <> lastTickRecv) then

        (*Since data format not supported in ladder - using LCD and serial port to view data*)

            EZ_FormatString(buffer, 'valid: %d, tickRecv: %d, satsUsed: %d, latitude: %f, longitude: %f,
altitude: %f $N', valid, tickRecv, satsUsed, latitude, longitude, altitude);
            EZ_FormatString(buffer2, 'lat:%f', latitude);        (*Write latitude to LCD display*)
            EZ_LcdWrite(0,0,buffer2);
            EZ_FormatString(buffer2, 'long:%f', longitude);    (*Write longitude to LCD display*)
            EZ_LcdWrite(1,0,buffer2);
            while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do        (*Write all values to Serial Port*)
                ;
            end_while;
            lastTickRecv := tickRecv;
        end_if;
    END_IF;

    Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_GPS_GetPrecision

### Summary:

The EZ_GPS_GetPrecision function is used to access the GPS message GSA and return the precision of measurement including 3D Fix quality, Dilution of Precisioin (PDOP), Horizontal Dilution of Precision (HDOP) and Vertical Dilution of Precision (VDOP). A sytem tick time is also received to identify when the last data was received.

### Format:

BOOLvar := EZ_GPS_GetPrecision(*Tick,Fix,PDOP,HDOP,VDOP*);

### Arguments:

| | |
|---|---|
| *BOOLvar* | Return status holding variable (BOOL). Set to 1 or True when data is valid (data received within last 2 seconds). Set to 0 or False when data is invalid (data not received within last 2 seconds). |
| *Tick* | Tick time the last message was received.(UDINT) |
| *Fix* | 3D Fix Quality (BYTE). 1 = No fix, 2 = 2D fix, 3 = 3D fix. |
| *PDOP* | Dilution of Precision (REAL). |
| *HDOP* | Horizontal Dilution of Precision (REAL). |
| *VDOP* | Vertical Dilution of Precision (REAL). |

### Description:

The EZ_GPS_GetPrecision reads and decodes the message GSA from the satellite via the GPS and then returns the *Tick* (system tick time) and variables for the *Fix* (3D fix quality), *PDOP* (Dilution of Precision), *HDOP* (Horizontal Dilution of Precision) and *VDOP* (Vertical Dilution of Precision). The *BoolVar* return variable is set 1 or True for as long as data is not older than 2 seconds (2 seconds since last message was received). This return value and the *Tick* time may be used to identify if the GPS data is valid.

### Example:

```
FUNCTION_BLOCK GPS_Prec
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                valid : bool := FALSE;
        END_VAR
        VAR
                tickRecv : UDINT;
                lastTickRecv : UDINT;
                mode2 : BYTE;
                pdop : REAL;
                hdop : REAL;
                vdop : REAL;
                buffer : string[200];
        END_VAR

        IF (Enable = TRUE) THEN
                valid := EZ_GPS_GetPrecision(tickRecv,mode2,pdop,hdop,vdop);

                if (tickRecv <> lastTickRecv) then

                (* Send data to Serial Port (UART) as string*)
```

Continued Next Page------»

```
                              EZ_FormatString(buffer, 'valid: %d, tickRecv: %d, mode2: %d, pdop: %f, hdop: %f, vdop: %f
$N', valid, tickRecv, mode2, pdop, hdop, hdop);

                              while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do (*write to serial port*)
                                      ;
                              end_while;
                              lastTickRecv := tickRecv;
                      end_if;
          END_IF;

          Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_I2CReadData

### Summary:

The EZ_I2CReadData function is to receive data from an I2C device on a P-Series PLC on Chip I2C bus. This function makes it possible to add I2C devices to a PLC on Chip design and interface with them. Refer to **Chapter 16 - I2C Devices** for more details on supported I2C devices and using I2C.

This function as others with I2C Custom Device Communications requires knowledge of I2C bus architecture and an understanding of implementing I2C communications to devices with different configuration parameters.

This function should only be used by those with experience in I2C communications or with direct consutation of Divelbiss. Using the function incorrectly could render the I2C bus inoperative or result in loss of stable communications.

### Format:

*varDINT*:=EZ_I2CReadData(*FileDescriptor, flags, clock, data, length)*;

### Arguments:

| | |
|---|---|
| *varDINT* | DINT. Returns status of function.<br>**Negative** Number for error or positive number is the number of bytes transferred from the I2C device. |
| *FileDescriptor* | **FD_I2C*x*** for the I2C port to use. x is the I2C port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable. |
| *flags* | UDINT. Bit flags for writing the start and stop bits.<br>These values can be ANDed together and used at the same time<br>1 : Write start bit at start of transfer.<br>2: Write stop bit at end of transfer |
| *clock* | UDNIT. Clock rate in KHz. Maximum 1 MHz allowed. |
| *data* | ARRAY OF USINT. Used as a buffer to store the received data into. |
| *length* | UDINT. Number of bytes to transfer from the device. |

### Description:

The EZ_I2CReadData function is used to receive I2C data (Custom Device communications) from I2C devices that are not natively supported in P-Series PLC on a Chip targets (not listed in the Project Settings Devices). Using the selected CAN port (*FileDescriptor*), the function will read data from the I2C device based on the *flags* and store the received data in the *data* variable. The *varDINT* returns the status of the function (and the number ot bytes transferred if successful). The *length* sets the number of bytes to transfer.

A high level of understanding of I2C communications is required to use this function. For example: generally, the EZ_I2CWriteData function must be used prior to this function (EZ_I2CReadData) to set the address of the device.

### Example:

**This example is provided only as an example of function block using the function. It is written to interface to a specific device. Any I2C devices will use the same syntax, but will be different (based on their needs for commications).**

```
FUNCTION_BLOCK FB_Read
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
```

Continued Next Page------»

```
		Q : bool;
	END_VAR
	VAR
		lastEn : bool;
		buf : array[0..31] of byte;
		err1 : dint;
		err2 : dint;
		err3 : dint;
	END_VAR

	Q := Enable;

	if(enable AND NOT lastEn) then
		buf[0] := 16#A0;
		buf[1] := 16#20;
		err1 := EZ_I2CWriteData(FD_I2C0, 16#01, 400, buf, 2);
		buf[0] := 16#A1;
		err2 := EZ_I2CWriteData(FD_I2C0, 16#01, 400, buf, 1);
		err3 := EZ_I2CReadData(FD_I2C0, 16#02, 400, buf, 32);

	end_if;

	lastEn := enable;

END_FUNCTION_BLOCK
```

## EZ_I2CWriteData

**Summary:**

The EZ_I2CWriteData function is to write data from an I2C device on a P-Series PLC on Chip I2C bus. This function makes it possible to add I2C devices to a PLC on Chip design and interface with them. Refer to **Chapter 16 - I2C Devices** for more details on supported I2C devices and using I2C.

This function as others with I2C Custom Device Communications requires knowledge of I2C bus architecture and an understanding of implementing I2C communications to devices with different configuration parameters.

This function should only be used by those with experience in I2C communications or with direct consutation of Divelbiss. Using the function incorrectly could render the I2C bus inoperative or result in loss of stable communications.

**Format:**

*varDINT*:=EZ_I2CWriteData(*FileDescriptor, flags, clock, data, length)*;

**Arguments:**

| | |
|---|---|
| *varDINT* | DINT. Returns status of function.<br>**Negative** Number for error or positive number is the number of bytes transferred from the I2C device. |
| *FileDescriptor* | **FD_I2C*x*** for the I2C port to use. x is the I2C port number. This file descriptor is found with other file descriptors (FD) in the structured text editor (Variables tab at the bottom). Hard-coded or use DINT variable. |
| *flags* | UDINT. Bit flags for writing the start and stop bits.<br>These values can be ANDed together and used at the same time<br>1 : Write start bit at start of transfer.<br>2: Write stop bit at end of transfer |
| *clock* | UDNIT*.* Clock rate in KHz. Maximum 1 MHz allowed. |
| *data* | ARRAY OF USINT. Variable of data to send to I2C device. |
| *length* | UDINT. Number of bytes to transfer to the device. |

**Description:**

The EZ_I2CWriteData function is used to transmit I2C data (Custom Device communications) to I2C devices that are not natively supported in P-Series PLC on a Chip targets (not listed in the Project Settings Devices). Using the selected CAN port (*FileDescriptor*)*,* the function will send data in the variable *data* to the I2C device based on the *flags*. The *varDINT* returns the status of the function (and the number ot bytes transferred if successful). The *length* sets the number of bytes to transfer.

A high level of understanding of I2C communications is required to use this function. For example: generally, the EZ_I2CWriteData function must be used prior to this function (EZ_I2CReadData) to set the address of the device. An understanding of what is required to interface to I2C devices is required.

**Example:**

**This example is provided only as an example of function block using the function. It is written to interface to a specific device. Any I2C devices will use the same syntax, but will be different (based on their needs for commications).**

```
FUNCTION_BLOCK FB_Read
        VAR_INPUT
                Enable : bool;
        END_VAR
```

Continued Next Page------»

```
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
                buf : array[0..31] of byte;
                err1 : dint;
                err2 : dint;
                err3 : dint;
        END_VAR

        Q := Enable;

        if(enable AND NOT lastEn) then
                buf[0] := 16#A0;
                buf[1] := 16#20;
                err1 := EZ_I2CWriteData(FD_I2C0, 16#01, 400, buf, 2);
                buf[0] := 16#A1;
                err2 := EZ_I2CWriteData(FD_I2C0, 16#01, 400, buf, 1);
                err3 := EZ_I2CReadData(FD_I2C0, 16#02, 400, buf, 32);

        end_if;

        lastEn := enable;

END_FUNCTION_BLOCK
```

## EZ_KeypadGetKey

### Summary:

The EZ_KeypadGetKey function is to get a keypad press from the keypad buffer. All keys are debounced (50mS). Key autorepeating begins after 500mS and then repeated every 150mS if the button is still pressed.

This function requires the hardware (target) to support and have installed a keypad using the standard keypad inteface for PLC on a Chip.

### Format:

*varDINT*:=EZ_KeypadGetKey();

### Arguments:

*varDINT* DINT. Returns the value (ASCII) of the key pressed. Returns -1 if no key is pressed.

Button values are based on Divelbiss basic keypad layouts.

| Button | ASCII | Button | ASCII | Button | ASCII | Button | ASCII |
|---|---|---|---|---|---|---|---|
| 0 | 48 | 5 | 53 | F1/A | 65 | DOWN/F | 70 |
| 1 | 49 | 6 | 54 | F2/B | 66 | ENTER | 13 |
| 2 | 50 | 7 | 55 | F3/C | 67 | CLEAR | 8 |
| 3 | 51 | 8 | 56 | F4/D | 68 | - | 45 |
| 4 | 52 | 9 | 57 | UP/E | 69 | . | 46 |

### Description:

The EZ_KeypadGetKey function is read a current keypress from a keypad and return the ASCII value or -1 if no key is pressed. The ASCII values are based on the arguments above and the Divelbiss basic keypad matrix.

All keys are debounce (50mSec) and provide a repeat feature. Autorepeating begins if a button has been pressed for 500mSec and then the values is repeated every additional 150mSec the button is still pressed.

If more than one key is pressed, none will be returned. Multiple keypresses are not supported at this time.

## EZ_LcdClear

### Summary:

The EZ_LcdClear function is used to clear / erase any displayed value on a connected LCD display. The display must be configured in the ladder diagram project settings prior to using this function. This function supports the character (CHR) and graphics (GFX) displays.

### Format:

EZ_LcdClear();

### Arguments:

*No arguments are required.*

### Description:

The EZ_LcdClear directly accesses the LCD display and erases (clears) all displayed values, text and/or graphics.

### Example:

```
FUNCTION_BLOCK ClearLCD
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 THEN;

        EZ_LcdClear();

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdDrawImage

### Summary:

The EZ_LcdDrawImage function is used to draw / render pre-made images on the graphics (GFX) LCD display. The images are stored in the same directory as the ladder diagram project file (.dld).

The images must be stored in the same directory as the ladder diagram project file (.dld) to be embedded for display. When located in the correct directory, they will be listed in the Variables tab in the structured text editor.

### Format:

*DINT* := EZ_LcdDrawImage(*ImageName*, *DINTX*, *DINTY*);

### Arguments:

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for success or non-zero for error. |
| *ImageName* | STRING*.* Name of the image file to render (embed). |
| *DINTX* | DINT. Horizontal (X) starting location to render the image. |
| *DINTY* | DINT. Vertical (Y) starting location to render the image. |

### Description:

The EZ_LcdDrawImage is used to write / render images to the graphics (GFX) LCD display. The *ImageName* is the actual image name of the stored image in the ladder project directory (can be found in the *Variables* ST editor tab). The *DINTX* and *DINTY* are the coordinates of the display to render the image. The *DINT* returns a zero when successful.

The image may be cropped or not display correctly if the image size and location would cause the image to have insufficient room for displaying.

### Example:

```
FUNCTION_BLOCK FB_Logo
        VAR_INPUT
                Enable : bool;
                x : dint;
                y : dint;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                EZ_LcdDrawImage('logo', x, y);
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdDrawLine

### Summary:

The EZ_LcdDrawLine function is used to draw / render a line on the graphics (GFX) LCD display.

### Format:

*DINT* := EZ_LcdDrawLine(*DINTx1*, *DINTy1*, *DINTx2, DINTy2, DINTcolor*);

### Arguments:

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for succes, -1 if x or y coordinates are invalid |
| *DINTx1* | DINT. Starting X (horizontal) position of line |
| *DINTy1* | DINT. Starting Y (vertical) position of line |
| *DINTx2* | DINT. Ending X (horizontal) position of line |
| *DINTy2* | DINT. Ending Y (vertical) position of line |
| *DINTcolor* | DINT. Line (pixel) Color, 0 for Off. or non-zero for color. Monochrome displays either off or on. |

### Description:

The EZ_LcdDrawLine is used to write / render a line on the graphics (GFX) LCD display. X and Y locations for starting and ending the line must be provided. The color (0 for off or non-zero for color (or on for monochrome) must be provided. The *DINT* returns a zero when successful or -1 if any of the X or Y positions are invalid.

### Example:

```
FUNCTION_BLOCK FB_DrawLine
        VAR_INPUT
                Enable : bool;
                x1 : dint;
                y1 : dint;
                x2 : dint;
                y2 : dint;
                color : dint;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                EZ_LcdDrawLine(x1, y1, x2, y2, color);
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdDrawRectangle

### Summary:

The EZ_LcdDrawRectangle function is used to draw / render an unfilled rectangle on the graphics (GFX) LCD display. The rectange is always drawn with a 1-pixel border. The border does not affect the inside or fill of the rectangle.

### Format:

*DINT* := EZ_LcdDrawRectangle(*DINTx1*, *DINTy1*, *DINTx2, DINTy2, DINTcolor*);

### Arguments:

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for succes, -1 if x or y coordinates are invalid |
| *DINTx1* | DINT. Starting X (horizontal) position of the rectangle |
| *DINTy1* | DINT. Starting Y (vertical) position of rectangle |
| *DINTx2* | DINT. Ending X (horizontal) position of rectangle |
| *DINTy2* | DINT. Ending Y (vertical) position of rectangle |
| *DINTcolor* | DINT. Border Color, 0 for Off. or non-zero for color. Monochrome displays either off or on. |

### Description:

The EZ_LcdDrawRectangle is used to write / render an unfilled rectangle on the graphics (GFX) LCD display. X and Y locations for starting and ending the rectangle must be provided. The color (0 for off or non-zero for color (or on for monochrome) for the border must be provided. The *DINT* returns a zero when successful or -1 if any of the X or Y positions are invalid.

### Example:

```
FUNCTION_BLOCK FB_Disp1
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                (* draw rectangle*)
                EZ_LcdDrawRectangle(0, 0, 127, 63, 1);
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdDrawRectangleFilled

### Summary:

The EZ_LcdDrawRectangleFilled function is used to draw / render a filled rectangle on the graphics (GFX) LCD display. The rectange is always drawn with a 1-pixel border. The border does not affect the inside or fill of the rectangle.

### Format:

*DINT* := EZ_LcdDrawRectangleFilled(*DINTx1*, *DINTy1*, *DINTx2, DINTy2, DINTborder, DINTfill*);

### Arguments:

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for succes, -1 if x or y coordinates are invalid |
| *DINTx1* | DINT. Starting X (horizontal) position of the rectangle |
| *DINTy1* | DINT. Starting Y (vertical) position of rectangle |
| *DINTx2* | DINT. Ending X (horizontal) position of rectangle |
| *DINTy2* | DINT. Ending Y (vertical) position of rectangle |
| *DINTborder* | DINT. Border Color, 0 for Off. or non-zero for color. Monochrome displays either off or on. |
| *DINTfill* | DINT. Fill Color, 0 for Off. or non-zero for color. Monochrome displays either off or on. |

### Description:

The EZ_LcdDrawRectangleFilled is used to write / render a filled rectangle on the graphics (GFX) LCD display. X and Y locations for starting and ending the rectangle must be provided. The *DINTborder* (0 for off or non-zero for color (or on for monochrome) for the border must be provided. The *DINTfill* (0 for off or non-zero for color (or on for monochrome) for the fill must be provided. The *DINT* returns a zero when successful or -1 if any of the X or Y positions are invalid.

Using the border color in conjunction with the fill color, the user has the ability to combine fills and border colors to draw and erase. For example, if a filled rectangle is drawn with a border color set, the area will be completely filled; if both colors (filled and border) are set to zero, it will clear the screen in that area.

### Example:

```
FUNCTION_BLOCK FB_Disp
        VAR_INPUT
                Enable : bool;
                br : dint;
                fl : dint;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                EZ_LcdDrawRectangleFilled(0, 0, 63, 63, br, fl);
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdInit

### Summary:

The EZ_LcdInit function is used initialize the LCD Display. This may be used to reinitialize the display from the ladder diagram program (using structured text). The display must be configured in the ladder diagram project settings prior to using this function.

### Format:

EZ_LcdInit();

### Arguments:

*No arguments are required.*

### Description:

The EZ_LcdInit directly accesses the LCD display initializes / restores the LCD Display communications parameters and port.

### Example:

```
FUNCTION_BLOCK INITLCD
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 THEN;

        EZ_LcdInit();

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdSetFont

### Summary:

The EZ_LcdSetFont function is set the font used on the (GFX) LCD display. Currently, only the default font (lcd_6x8) is supported.

The default font (lcd_6x8) is the only supported font at this time. It is automatically set as the font to be used.

### Format:

*DINT* := EZ_LcdSetFont(*StringFont*);

### Arguments:

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for success. |
| *StringFont* | STRING. String name of the font to use (only supported fonts). Supported fonts will appear in the *Variables* tab in the ST editor. |

### Description:

The EZ_LcdSetFont is used to set the font used on the graphics (GFX) LCD display. Only supported fonts may be used (at this time only the lcd_6x8 font is supported). *DINT* returns zero when sucessful.

### Example:

```
FUNCTION_BLOCK FB_Setfont
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                (* Init display *)

                EZ_LcdSetFont('lcd_6x8');
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdSetFontSize

**Summary:**

The EZ_LcdSetFontSize function is used to set the font size (scaling) used on the (GFX) LCD display. This function can be used multiple times to display multiple sizes of text.

**Format:**

*DINT* := EZ_LcdSetFontSize(*DINTscale*);

**Arguments:**

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for succes, -1 if scale is invalid |
| *DINTscale* | DINT. Font size scale to use. Supported scale is 1-4. |

**Description:**

The EZ_LcdSetFontSize is used to set the font scaling used on the graphics (GFX) LCD display. *DINT* returns zero when sucessful or -1 if the scale is invalid. The scaling may be set from 1 to 4 (size multiplier).

**Example:**

```
FUNCTION_BLOCK FB_Disp1
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                (* Init display *)

                EZ_LcdSetFontSize(2);
                EZ_LcdWriteString(2, 2, 'Test1');

                EZ_LcdSetFontSize(1);
                EZ_LcdWriteString(2, 18, 'Test2 ');
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdSetPixel

**Summary:**

The EZ_LcdSetSetPixel function is used to draw / control a pixel on the (GFX) LCD display. The color of the pixel can be set (off, color or on for monochrome).

**Format:**

*DINT* := EZ_LcdSetSetPixel(*DINTx*, *DINTy*, *DINTcolor*);

**Arguments:**

| | |
|---|---|
| *DINT* | DINT. Returns a 0 for succes, -1 if x or y coordinates are invalid |
| *DINTx* | DINT. Starting X (horizontal) position of the pixel |
| *DINTy* | DINT. Starting Y (vertical) position of the pixel |
| *DINTcolor* | DINT. Pixel Color, 0 for Off. or non-zero for color. Monochrome displays either off or on. |

**Description:**

The EZ_LcdSetSetPixel is used to to display / control a pixel on the GFX LCD display. The X and Y positions ( *DINTx* , *DINTy)* of the pixel must be provided. The color (*DINTcolor)* sets the color of the pixel as either 0 for off, or a number for a color (monochrome displays are 0 for off or a number for on). The *DINT* returns a zero when successful or a -1 if the X or Y locations is out of range.

**Example:**

```
FUNCTION_BLOCK FB_DrawLine
        VAR_INPUT
                Enable : bool;
                x1 : dint;
                y1 : dint;
                color : dint;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then
                EZ_LcdSetSetPixel(x1, y1, color);
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdWrite

**Summary:**

The EZ_LcdWrite function is used to write / display data on the LCD Display. The display must be configured in the ladder diagram project settings prior to using this function.

**Format:**

EZ_LcdWrite(*Row,Column,Data*);

**Arguments:**

| | |
|---|---|
| *Row* | The desired row of the LCD display to display the data to. Rows are numbered 0 to the number of rows on the LCD Display. ( A 4 row display will be numbered 0 to 3). (DINT) |
| *Column* | The column in the row to begin writing / displaying the data to. Columns are numbered 0 to the number of columns on the LCD Display. ( A 16 column display will be numbered 0 to 15) (DINT) |
| Data | the string of data to write (display) on the LCD display. The data may be preformatted using the EZ_FormatString function. |

**Description:**

The EZ_LcdWrite directly accesses the LCD display from the structured text (outside of the ladder diagram), sets the current location as the *Row* and *Column* then writes (displays) the information in the *Data* string.

**Example:**

```
FUNCTION_BLOCK WRITELCD
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                data : string[16];
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        data := ' Display Line 1 ';     (*Set value to display*)

        IF Enable = 1 THEN;

        EZ_LcdWrite(0,0,data);

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_LcdWriteString

### Summary:

The EZ_LcdWriteString function is used to write / display text data on the GFX LCD Display. The display must be configured in the ladder diagram project settings prior to using this function.

### Format:

*DINT* := EZ_LcdWriteString(*DINTx*, *DINTy*,*StringData, UDINTFlag*);

### Arguments:

| | |
|---|---|
| *DINT* | DINT. Returns status of the function (result).<br>0 for success<br>-3 A character to display was not defined in the font.<br>-4 The base font wideth is too large (maximum is 8).<br>-100 No Found Setup (Check display configurations) |
| *DINTx* | DINT. The desired X (horizontal) location of the GFX LCD display to display the data to. |
| *DINTy* | DINT. The desired Y (vertical) location of the GFX LCD display to display the data to. |
| *StringData* | STRING. The string of data to write (display) on the GFX LCD display. |
| *UDINTFlag* | UDINT. Optional The flag determines special display characteristics of the text that is being displayed. When not used, normal text is displayed.<br>1: Reverse Text<br>4: Underlined Text |

### Description:

The EZ_LcdWriteString directly accesses the LCD display from the structured text (outside of the ladder diagram), sets the current location as the *DINTx* and *DINTy* (horizontal and vertical coordinates) then writes (displays) the information in the *StringData*. The optional flags allow for reversed (blocked) text and underlined text. The return value is 0 for success for negatie for errors (see above).

### Example:

```
FUNCTION_BLOCK FB_Disp1
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
        VAR
                lastEn : bool;
        END_VAR

        Q := Enable;

        if NOT lastEn AND Enable then

                EZ_LcdSetFontSize(1);

                EZ_LcdWriteString(2, 26, 'Reverse', 16#1);      (* draw with reverse *)
                EZ_LcdWriteString(2, 34, 'Underline', 16#4);    (* draw with underline *)
        end_if;

        lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_ModbusMaster_UartEnableIsr

**Summary:**

The EZ_ModbusMaster_UartEnableIsr is used to enable or diable Modbus Master communications (when the target is the Modbus Master).

**Format:**

BoolVar := EZ_ModbusMaster_UartEnableIsr(*FileDescriptor,State*);

**Arguments:**

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *State* | 0 or 1, False or True for UART Modbus Enable (BOOL) |
| *BOOLvar* | Function return holding variable (BOOL). Returns the number of bytes read. |

**Description:**

The EZ_ModbusMaster_UartEnableIsr controls the Modbus Master UART enable (on/off). When the State is 0, the Modbus Master function for the FileDescriptor UART is enabled (Modbus Read/Write functions work). When the State is 0, the Modbus Master function for the FileDescriptor UART is disabled (Modbus Read/Write functions don't work).

**The Modbus_Master function block should not be used when disabled. Using the Modbus_Master function block after it has been disabled will cause the program to lock up.**

**Example:**

```
FUNCTION_BLOCK ModbusCtrl
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                FncReturn1 : BOOL;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 THEN;             (*Look for Enable*)
        FncReturn1 := EZ_ModbusMaster_UartEnableIsr(FD_UART3,1); (*Enable*)

        ELSE
        FncReturn1 := EZ_ModbusMaster_UartEnableIsr(FD_UART3,0); (*Disable*)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_MQTT_Connect

**Summary:**

The EZ_MQTT_Connect function attempts to make a connection to a VersaCloud M2M+IoT cloud solution server. This is similar to the ladder diagram function block, this function provides control via structured text.

**Format:**

BoolVar := EZ_MQTT_Connect(*Enable,RetryTimeout,Status,Error,NumRetries*);

**Arguments:**

*Enable* Connect / Disconnect for MQTT control. (BOOL). **0** for Disconnect, **1** for Connect.

*RetryTimeout* Time between connection retries in seconds. When set to zero (0), the function will not attempt reconnection in the event of a loss of communication (unplanned disconnect). Any other seconds entered is the delay time between reconnection attempts (**DINT**).

*Status* Status (output) of the connection (or attempt). (**DINT**). The status returns one of the following based on the MQTT state:

| | |
|---|---|
| 1 | Connecting |
| 2 | Provisioning |
| 3 | Provision Sent |
| 100 | Connect Sent |
| 101 | Connected |
| 200 | Connect Error |
| 300 | TLS Handshaking Failed |
| 301 | Authentication Read Error |
| 302 | Authentication Needs Info (causes may include unable to read client ID, username or password from EEPROM) |
| 303 | Authentication Invalid User Password |
| 304 | Authentication Not Authorized (Bad Username or Password) |
| 305 | Connection Refused (Invalid Protocol Version) |
| 306 | Connection Refused (Client ID Rejected) |
| 307 | Connection Refused (Server Unavailable) |

*Error* Error (output), if an error detected. (DINT). The erros returns one of the following network errors:

| | |
|---|---|
| 0 | No Error |
| -1 | Out of Memory Error |
| -2 | Buffer Error |
| -3 | Timeout Error |
| -4 | Reverved (Should not see) |
| -5 | Operation Currently in Progress |
| -6 | Illegal Value |
| -7 | Reverved (Should not see) |
| -8 | Address in Use |
| -9 | Already Connecting |
| -10 | Connection Already Established |
| -11 | Not Connected |
| -12 | Network Interface Error (Cellular, Ethernet, etc) |
| -13 | Connection Aborted |
| -14 | Connection Reset |
| -15 | Connection Closed |

Continued Next Page------»

| | |
|---|---|
| -16 | Illegal Argument |

Additional errors below -8192 are specific to the SSL layter. For example, -9984 is a Certificate failed verification (CRL, CA or signature check failed. This can be caused by the certificate not matching, if there is not enough free memory (RAM) in the controller or if the certificate date is not vaild.

| | |
|---|---|
| *NumRetries* | Number of retries that have occurred (output). (DINT). This number is cumulative (until target reset or power cycle). The larger the number, the more times the function has tried to connect to the selected cloud solution (some may be successful, others may not). |

**Description:**

The EZ_MQTT_Connect connects (or attemps to connect) and stay connected to the VersaCloud M2M+IoT solution selected in the Project Settings based on the flags and options used in the function. *Status* and *Error* outputs are provided for the state of the connection and troubleshooting.

EZ_MQTT_Connect requires the SNTP feature to be installed in the Project Settings for the ladder program to compile successfully and be functional. MQTT uses SNTP to sync the real time clock using UTC time and this time is used to validate communication and SSL to VersaCloud M2M+IoT servers.

When using SNTP, the real time clock cannot be set using the ladder diagram or structured text without 'breaking' MQTT functionality.

**Example:**

```
FUNCTION_BLOCK CnctMQTT
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                rtrytm : DINT;
                Rtn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                S : DINT;
                E : DINT;
                R : DINT;

        END_VAR

        if(Enable = True) then

                Rtn:= EZ_MQTT_Connect(1,60,S,E,R);

  Else;

        Rtn:= EZ_MQTT_Connect(0,60,S,E,R);

  end_if;

  Q := Enable;


END_FUNCTION_BLOCK
```

## EZ_MQTT_Publish

**Summary:**

The EZ_MQTT_Publish function sends data to a VersaCloud M2M+IoT cloud solution server. This is similar to the ladder diagram function block, this function provides sending via structured text.

**Format:**

*DINTsts* := EZ_MQTT_Publish(*StringTopic,StringPayload,DintPayloadlength[,DintDup,DintQOS,DintRetained]*);

**Examples:** err := EZ_MQTT_Publish('devices/{DeviceID}/messages/events/', payload); **or**
err := EZ_MQTT_Publish('devices/{DeviceID}/messages/events/',payload, 0, 0, 0);

**Arguments:**

| | |
|---|---|
| *StringTopic* | Specific topic string required to connect to VersaCloud M2M+IoT. (**STRING**)<br>**For Azure IOT,** set to: ***devices/{DeviceID}/messages/events/*** . This automatically gets replaced underneath when running with the actual device ID stored in the project settings (stored in EEPROM).<br>**For Exosite Murano**, set to ***$resource/data_in .*** This automatically gets and updates the ID stored in the project settings (stored in EEPROM). |
| *StringPayload* | Actual data to transmit to VersaCloud M2M+IoT solution. (**STRING**). This must ve a valid **json** string. |
| *DintPayloadlength* | Number of bytes to send. (**DINT**) |
| *DINTsts* | Status (output) of the transmission (or attempt). (**DINT**). Error (output), if an error detected. (DINT). The status returns one of the following network errors: |

| | |
|---|---|
| 0 | No Error |
| -1 | Out of Memory Error |
| -2 | Buffer Error |
| -3 | Timeout Error |
| -4 | Reverved (Should not see) |
| -5 | Operation Currently in Progress |
| -6 | Illegal Value |
| -7 | Reverved (Should not see) |
| -8 | Address in Use |
| -9 | Already Connecting |
| -10 | Connection Already Established |
| -11 | Not Connected |
| -12 | Network Interface Error (Cellular, Ethernet, etc) |
| -13 | Connection Aborted |
| -14 | Connection Reset |
| -15 | Connection Closed |
| -16 | Illegal Argument |

| | |
|---|---|
| *String DintDup* | Optional. Use only at the recommendation and provided information from Divelbiss. (**DINT**) |
| *DintQOS* | Optional. Use only at the recommendation and provided information from Divelbiss. (**DINT**) |
| *DintRetained* | Optional. Use only at the recommendation and provided information from Divelbiss. (**DINT**) |

Continued Next Page------»

**Description:**

The EZ_MQTT_Publish sends (or attemps to send) the payload json data to the VersaCloud M2M+IoT solution selected in the Project Settings based on the flags and options used in the function. The *DINTsts* return value is provided for the state of the network (and sending).

EZ_MQTT_Publish requires the SNTP feature to be installed in the Project Settings for the ladder program to compile successfully and be functional. MQTT uses SNTP to sync the real time clock using UTC time and this time is used to validate communication and SSL to VersaCloud M2M+IoT servers.

When using SNTP, the real time clock cannot be set using the ladder diagram or structured text without 'breaking' MQTT functionality.

**Example:**

```
FUNCTION_BLOCK FB_MqttPublish1
   VAR_INPUT
      Enable : bool;
      spd : dint;
      load : dint;
   END_VAR
   VAR_OUTPUT
      Q : bool;
   END_VAR
   VAR
      lastEn: bool;
      payload: string[200];
      err : dint;
   END_VAR

   Q := Enable;

   if Enable AND NOT lastEn then
      EZ_FormatString(payload, '{"engine_load": %d, "engine_speed": %d}', load, spd);
      err := EZ_MQTT_Publish('devices/{DeviceID}/messages/events/', payload);
   end_if;

   lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_MQTT_RECEIVE

### Summary:

The EZ_MQTT_RECEIVE function is used to receive data from a VersaCloud M2M+IoT cloud solution server using MQTT. This is similar to the ladder diagram function block, this function provides receiving via structured text.

### Format:

*DINTsts :=(StringTopic,StringPayload,DintPayloadlength[,DintDup,DintQOS,DintRetained]*);

**Examples:** *DINTsts* := EZ_MQTT_RECEIVE(*StringTopic*, *StringPayload*, *DintPayloadlength*); **or**
*DINTsts* := EZ_MQTT_RECEIVE(*StringTopic*, *StringPayload*, *DintPayloadlength*, *DintDup*, *DintQOS*, *DintRetained*);

### Arguments:

| | |
|---|---|
| *StringTopic* | Actual received MQTT topic (**STRING**) |
| *StringPayload* | Actual Payload received fromVersaCloud M2M+IoT solution. (**STRING**). |
| *DintPayloadlength* | Number of bytes received in payload. (**DINT**) |
| *DINTsts* | Status (output) of the receive (or attempt). (**DINT**).<br>0 No message received<br>1 Message received |
| *DintDup* | Duplication Flag. Indicates message received is a duplicate and was resent because the inteded recipient or broker did not acknowledge the original message. (**DINT**). Usually only relevant for QOS grater than 0. Refer to MQTT client library or broker implementation detail. |
| *DintQOS* | Quality of Service. (**DINT**). Refer to MQTT client library or broker implementation detail. |
| *DintRetained* | Retained Flag (**DINT**). Refer to MQTT client library or broker implementation detail. |

### Description:

The EZ_MQTT_RECEIVE received (or attemps to receive) the payload json data from the VersaCloud M2M+IoT solution (MQTT) selected in the Project Settings based on the flags and options used in the function. The *DINTsts* return value is provided for identifying when data is received.

EZ_MQTT_Publish requires the SNTP feature to be installed in the Project Settings for the ladder program to compile successfully and be functional. MQTT uses SNTP to sync the real time clock using UTC time and this time is used to validate communication and SSL to VersaCloud M2M+IoT servers.

When using SNTP, the real time clock cannot be set using the ladder diagram or structured text without 'breaking' MQTT functionality.

### Example:

```
FUNCTION_BLOCK FB_MqttReceive
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;


        END_VAR
        VAR
                topic: string[100];
                payload: string[200];
                payloadLen: dint;
                txTopic: string[100];
                txPayload: string[200];
                err : dint;
                result : dint;
                str_command1 : string := '$$home_36274/toggle1';
                str_command2 : string := '$$home_36274/toggle2';
        END_VAR

        var_external

        Overide : dint;
        setpoint : dint;
        end_var

        Q := Enable;

        if Enable then

                (* topic := ''; *)
                err := EZ_MQTT_Receive(topic, payload, payloadLen);

                if err = 1 then
                        (* $iothub/methods/POST/Command1/?$rid=2 *)


                        if F_strncmp(topic, str_command1, 29) = 0 then
                                txTopic := CONCAT('$$iothub/methods/res/204/', RIGHT(topic, len(topic) - 30));   (*EZ_
FormatString(txPayload, '{}');*)
                                err := EZ_MQTT_Publish(txTopic, payload);
                                EZ_ScanString (payload, '%d', setpoint);

                        elsif
                                F_strncmp(topic, str_command2, 29) = 0  then
                           txTopic := CONCAT('$$iothub/methods/res/204/', RIGHT(topic, len(topic) - 30));
                           err := EZ_MQTT_Publish(txTopic, payload);
                                EZ_ScanString (payload, '%d', Overide);



                        end_if;
                end_if;
        end_if;

END_FUNCTION_BLOCK
```

## EZ_ResetTarget

### Summary:

The EZ_ResetTarget is used to force restart force restart the ladder diagram and PLC on a Chip target using a Watchdog reset with an optional flag to control the Watchdog PLC on a Chip reset.

### Format:

EZ_RestartTarget(*resetType*);

### Arguments:

| | |
|---|---|
| *resetType* | Optional Boolean flag or variable. When resetType is "0" or not specified, and the function is called, the function will force the PLC on a Chip watchdog to trigger a reset. This causes the PLC on a Chip to reset and restart the ladder diagram.<br>When resetType is specified and not zero, the function sends a restart command for the ladder diagram to restart (no watchdog or PLC on a Chip reset). |

### Description:

The EZ_ResetTarget provides a method from the ladder diagram / structured text for being able to restart / reboot the ladder diagram (with options for forcing the PLC on a Chip to restart also). This can be useful in the event that a reboot is required for updating programs and kernels; or in the event a reboot may resolve certain undesired operation.

Regardless of *resetType*, when the ladder restarts, if the system is configured to update ladder diagram and/or kernel from SD card and updated files are on the SD card, the update(s) will occur during the restart.

### Example:

```
FUNCTION_BLOCK FrcRstLadder
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
        LastEn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        if(Enable = True) AND (lastEn = False) then
                EZ_ResetTarget(0);                    (* Force Reset with Watchdog *)

        end_if;

  Q := Enable;
  lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_RTC_GETDTLOCAL

### Summary:

The EZ_RTC_GETDTLOCAL function is used to read the current date and time from the real time clock (when the real time clock was set using UTC time). This function reads the UTC time, converts the data to local unix time (number of seconds since 1-1-1970, offset by the time zone) as a long integer. The value can be converted to actual DINT variables for month, day, etc. using the **EZ_TimeDateUnixToCalendar** function. For this function to work correctly, the EZ_RTC_SETTZOFF function must have been ran prior (must be run on each program start) to set the time zone offset of your current time zone from UTC time (in minutes). **This function allows for using the time zone offset to return the local time (unix time).**

### Format:

*DINT* := EZ_RTC_GETDTLOCAL(*FileDescriptor, UnixTime*);

### Arguments:

| | |
|---|---|
| *DINT* | **DINT**. Function return value. >=0 for success or negative number for error. |
| *FileDescriptor* | **FD_PLCHIP-PXX_RTC** for internal PLC on a Chip Real Time Clock. |
| *UnixTime* | **LINT**. Variable to hold Unix time (offset by time zone offset). |

### Description:

The EZ_RTC_GETDTLOCAL provides a method from the ladder diagram / structured text to read the hardware's real time clock and get the local date and time (Unix time) from stored UTC time. The returned time is always returned as Unix time (number of seconds since 1-1-1970, offset by time zone) and stored in *UnixTime*. The *FileDescriptor* is set for the PLC on a Chip real time clock. The *DINT* is the status of the function (>=0 for success or negative for error).

The return can be converted to actual individual double integer (DINT) variable for use in the ladder diagram by using the **EZ_TimeDateUnixToCalendar** function.

The EZ_RTC_SETTZOFF function must have been used (on each power up / boot / program start) to set the time zone offset (number of minutes difference between your current time zone and UTC time). This value is used in the conversion process for the EZ_RTC_GETDTLOCAL function. If the EZ_RTC SETTZOFF is not used / set, the returned date/time will not be correct.

The EZ_RTC_SETTZOFF structured text function and the SETTTZOFF ladder function block function both set the time zone offset. If either is ran, then the time zone offset is set until a power cycle occurs (must be ran each time the program is started).

If the EZ_RTC_SETTZOFF function is not called or is set to zero (0), then the EZ_RTC_GETDTLOCAL function will return the same value as the EZ_RTC_GETDTUTC function (UTC time with no offset).

### Example:

```
FUNCTION_BLOCK RDRTC
        VAR_INPUT
                Enable : bool;
                RO: bool; (*run once input flag*)
                TZO: DINT; (*time zone offset*)
        END_VAR
        VAR
                LastRO: bool;
                Sts: DINT;
                LocalDT: LINT;
        END_VAR
```

Continued Next Page------»

```
	VAR_OUTPUT
		Q : bool;
		M : DINT;
		D : DINT;
		Y : DINT;
		W : DINT;
		H : DINT;
		MN: DINT;
		SC: DINT;
	END_VAR


	If (RO = true) and (LastRO = False) then		(*Set time zone offset once*)
			EZ_RTC_SETTZOFF(TZO);
			LastRO := true;
		End_if;

	If (Enable = True) then

		Sts := EZ_RTC_GETDTLOCAL(FD_PLCHIP_PXX_RTC,LocalDT); (* get unix time*)
		EZ_TimeDateUnixToCalendar(LocalDT,M,D,Y,W,H,MN,SC);

	End_If;

	If (Enable = False) then
		M := 0;
		D := 0;
		Y := 0;
		W := 0;
		H := 0;
		MN := 0;
		SC := 0;

	End_If;


	Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_RTC_GETDTUTC

### Summary:

The EZ_RTC_GETDTUTC function is used to read the current date and time from the real time clock (when the real time clock was set using UTC time). This function reads the UTC time, converts the data to unix time (number of seconds since 1-1-1970) as a long integer. **This function does NOT allow for using the time zone offset to return the local time (unix time).**

### Format:

*DINT* := EZ_RTC_GETDTUTC(*FileDescriptor, UnixTime*);

### Arguments:

| | |
|---|---|
| *DINT* | **DINT**. Function return value. >=0 for success or negative number for error. |
| *FileDescriptor* | **FD_PLCHIP-PXX_RTC** for internal PLC on a Chip Real Time Clock. |
| *UnixTime* | **LINT**. Variable to hold Unix time |

### Description:

The EZ_RTC_GETDTUTC provides a method from the ladder diagram / structured text to read the hardware's real time clock and get the Unix time from stored UTC time. The returned time is always returned as Unix time (number of seconds since 1-1-1970) and stored in *UnixTime*. The *FileDescriptor* is set for the PLC on a Chip real time clock. The *DINT* is the status of the function (>=0 for success or negative for error).

The return can be converted to actual individual double integer (DINT) variable for use in the ladder diagram by using the **EZ_TimeDateUnixToCalendar** function.

### Example:

```
FUNCTION_BLOCK RDRTC
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                Sts: DINT;
                UXDT: LINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = True) then

                Sts := EZ_RTC_GETDTUTC(FD_PLCHIP_PXX_RTC,UXDT); (* get unix time*)

        End_If;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_RTC_SETDTLOCAL

### Summary:

The EZ_RTC_SETDTLOCAL function is used to set the current date and time in the real time clock to UTC time (from unix time, offset by time zone offset). This function sets the UTC time by converting the data from unix time (number of seconds since 1-1-1970, offset by the time zone) as a long integer into UTC time and storing this time in the real time clock. The local date and time variables for month, day, etc. can be converted into unix time using the **EZ_TimeDate-CalendarToUnix** function. For this function to work correctly, the EZ_RTC_SETTZOFF function must have been ran prior (must be run on each program start) to set the time zone offset of your current time zone from UTC time (in minutes). **This function allows for using the time zone offset to set UTC time based on the current time zone offset (local time/date).**

### Format:

*DINT* := EZ_RTC_SETDTLOCAL(*FileDescriptor, UnixTime*);

### Arguments:

| | |
|---|---|
| *DINT* | **DINT**. Function return value. >=0 for success or negative number for error. |
| *FileDescriptor* | **FD_PLCHIP-PXX_RTC** for internal PLC on a Chip Real Time Clock. |
| *UnixTime* | **LINT**. Variable to hold Unix time (offset by time zone offset). |

### Description:

The EZ_RTC_SETDTLOCAL provides a method from the ladder diagram / structured text to set the hardware's real time clock to UTC time using Unix time. The time is always stored as UTC, converted from Unix time (*UnixTime* (number of seconds since 1-1-1970)), then offset by time zone. The *FileDescriptor* is set for the PLC on a Chip real time clock. The *DINT* is the status of the function (>=0 for success or negative for error).

The EZ_RTC_SETTZOFF function must have been used (on each power up / boot / program start) to set the time zone offset (number of minutes difference between your current time zone and UTC time). This value is used in the conversion process for the EZ_RTC_SETDTLOCAL function. If the EZ_RTC SETTZOFF is not used / set, the returned date/time will not be correct.

The EZ_RTC_SETTZOFF structured text function and the SETTTZOFF ladder function block function both set the time zone offset. If either is ran, then the time zone offset is set until a power cycle occurs (must be ran each time the program is started).

If the EZ_RTC_SETTZOFF function is not called or is set to zero (0), then the EZ_RTC_SETDTLOCAL function will store the same value as the EZ_RTC_SETDTUTC function (UTC time with no offset).

The EZ_RTC_SETDTLOCAL requires a 4 digit year as part of the unix time. A two digit year will cause incorrect time/date settings.

### Example:

```
FUNCTION_BLOCK SETLOCRTC
        VAR_INPUT
                Enable : bool;
                RO : bool;
                TZ : DINT;
                M : DINT;
                D : DINT;
                Y : DINT;
                H : DINT;
                MN : DINT;
```

Continued Next Page------»

```
                SC : DINT;
        END_VAR
        VAR
                Sts: DINT;
                UXDT: LINT;
                LastEN : bool;
                LastRO : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (RO = true) and (LastRO = false) then
                EZ_RTC_SETTZOFF(TZ);
                LastRO := true;
        End_If;

        If (Enable = True) and (LastEn = false) then

                EZ_TimeDateCalendarToUnix(UXDT,M,D,Y,H,MN,SC);
                Sts := EZ_RTC_SETDTLOCAL(FD_PLCHIP_PXX_RTC,UXDT); (* get unix time*)

        End_If;

        If (RO = false) then
                LastRO := false;
        End_If;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_RTC_SETDTUTC

### Summary:

The EZ_RTC_SETDTUTC function is used to set the current date and time on the real time clock (using UTC time). This function uses the provided unix time (number of seconds since 1-1-1970) and converts the data to UTC. **This function does NOT allow for using the time zone offset when storing date / time as UTC ( from unix time).**

### Format:

*DINT* := EZ_RTC_SETDTUTC(*FileDescriptor, UnixTime*);

### Arguments:

| | |
|---|---|
| *DINT* | **DINT**. Function return value. >=0 for success or negative number for error. |
| *FileDescriptor* | **FD_PLCHIP-PXX_RTC** for internal PLC on a Chip Real Time Clock. |
| *UnixTime* | **LINT**. Variable to hold Unix time. |

### Description:

The EZ_RTC_SETDTUTC provides a method from the ladder diagram / structured text to set the hardware's real time clock and to UTC time (using unix time). The passed in time is always returned as Unix time (number of seconds since 1-1-1970) and stored in *UnixTime*. The *FileDescriptor* is set for the PLC on a Chip real time clock. The *DINT* is the status of the function (>=0 for success or negative for error).

The *UnixTime* can be converted from individual double integers (DINTs) variables from the ladder diagram by using the **EZ_TimeDateCalendarToUnix** function.

The EZ_RTC_SETDTLOCAL requires a 4 digit year as part of the unix time. A two digit year will cause incorrect time/date settings.

### Example:

```
FUNCTION_BLOCK SETUTCRTC
        VAR_INPUT
                Enable : bool;
                M : DINT;
                D : DINT;
                Y : DINT;
                H : DINT;
                MN : DINT;
                SC : DINT;
        END_VAR
        VAR
                Sts: DINT;
                UXDT: LINT;
                LastEN : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = True)and (LastEN = false) then

                EZ_TimeDateCalendarToUnix(UXDT,M,D,Y,H,MN,SC);
                Sts := EZ_RTC_SETDTUTC(FD_PLCHIP_PXX_RTC,UXDT); (* get unix time*)
                LastEN := true;
```

Continued Next Page------»

```
        End_If;

        If (Enable = false) then
                LastEN := false;
        End_If;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_RTC_SETTZOFF

**Summary:**

The EZ_RTC_SETTZOFF function is used to set the current time zone offset (in minutes frrom UTC time) when using the real time clock (with UTC time). This function set the number of minutes difference between UTC time and the time zone the target is in.

The EZ_RTC_SETTZOFF function must have been used (on each power up / boot / program start) to set the time zone offset (number of minutes difference between your current time zone and UTC time). This value is used in the conversion process for the EZ_RTC_GETDTLOCAL and EZ_RTC_SETDTLOCAL functions. If the EZ_RTC SETTZOFF is not used / set, the returned date/time will not be correct.

The EZ_RTC_SETTZOFF structured text function and the SETTTZOFF ladder function block function both set the time zone offset. If either is ran, then the time zone offset is set until a power cycle occurs (must be ran each time the program is started).

If the EZ_RTC_SETTZOFF function is not called or is set to zero (0), then the EZ_RTC_SETDTLOCAL function will store the same value as the EZ_RTC_SETDTUTC function (UTC time with no offset).

**Format:**

EZ_RTC_SETTZOFF(*Offset*);

**Arguments:**

| | |
|---|---|
| *Offset* | **DINT***.* Variable to hold Offset in minutes. |

**Description:**

The EZ_RTC_SETTZOFF sets from structured text the offset in minutes from UTC to the time zone the target is located in when using the hardware's real time clock with UTC time. This offset allows for functions to use local time/date to set the real time clock and the real time clock to be read and viewed in local time/date. The passed in *Offset* is the number of minutes difference (+ or -) between UTC time and the time zone the hardware target is in.

**Example:**

```
FUNCTION_BLOCK SETLOCRTC
        VAR_INPUT
                Enable : bool;
                RO : bool;
                TZ : DINT;
                M : DINT;
                D : DINT;
                Y : DINT;
                H : DINT;
                MN : DINT;
                SC : DINT;
        END_VAR
        VAR
                Sts: DINT;
                UXDT: LINT;
                LastEN : bool;
                LastRO : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
```

Continued Next Page------»

```
        If (RO = true) and (LastRO = false) then
                EZ_RTC_SETTZOFF(TZ);
                LastRO := true;
        End_If;

        If (Enable = True) and (LastEn = false) then

                EZ_TimeDateCalendarToUnix(UXDT,M,D,Y,H,MN,SC);
                Sts := EZ_RTC_SETDTLOCAL(FD_PLCHIP_PXX_RTC,UXDT); (* get unix time*)

        End_If;

        If (RO = false) then
                LastRO := false;
        End_If;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_ScanString

**Summary:**

The EZ_ScanString is used to parse data from a string (such as a string buffer received during communcation) and save the data to variables. This function serves to nearly the reverse of the EZ_FormatString function.

**Format:**

*DINTvar* := EZ_ScanString(*StrBuffer*, '*VarFormatFlag1 VarformatFlag2 VarformatFlag3..', Var1,Var2, Var3)*;

**Arguments:**

| | |
|---|---|
| *StrBuffer* | Source string variable to parse the variable data from. |
| *DINTvar* | Function return holding variable (DINT). Number of variables read from the *StrBuffer* variable. |
| *Varx* | The variables were the formatted data (using *VarFormatFlag*) will be stored.The number of variables must match the number of *FormatString* entries and must be in the same order for proper functionality. Each Variable is separated by a comma (,). |
| *VarFormatFlag* | Variable format and Flags are used to identify variable types and formats as well as special control characters needed for specific formatting. Follows 'C' scanf standard. |

%flag width .precision Example Text: OIL PSI %-3d

% - identifies the beginning of a variable or other type of text entry

flag - Use the following control how data is received.

| Flag | Description |
|---|---|
| width | This flag is optional. Width is the number of characters that will be printed (total). |

**Some common variable formats and flags are:**

| | |
|---|---|
| ***%d*** - Signed Integer | ***%x*** - Upper Case Hexadecimal |
| ***%u*** - Unsigned Integer | ***%o*** - Octal |

The parsing will begin on whitespace characters (space), commas (,) or tabs. It can also detect specific characters in the string.

**For example:** *nlngth*:= EZ_ScanString(*stbuff*, '*CH1%3d CH2%2d CH3%3d', Var1,Var2, Var3)*; will parse 3 characters after CH1, 2 characters after CH2 and 3 characters after CH3 in the *stbuff* and store them in *Var1*, *Var2* and *Var3* respectively.

**Description:**

The EZ_FormatString parses (pulls) one to multiple sets of characters from a string, converts and saves the values as variables based on the *VarFormatFlag* items used. The variable types and and order must be correct and match the *VarFormatFlag* for the function to properly parse and save the data.

**Example:**

```
FUNCTION_BLOCK ParseStringtoVars
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
        LastEn : bool;
        data : string[20];
        lgth : dint;
```

Continued Next Page------»

```
        END_VAR
        VAR_OUTPUT
                Q : bool;
                val1 : dint;
                val2 : dint;
        END_VAR

        data := 'CH1345CH2789';

        if(Enable = True) AND (lastEn = False) then

                lgth := EZ_ScanString(data,'CH1%3dCH2%3d', val1, val2);

        end_if;

  Q := Enable;
  lastEn := Enable;

END_FUNCTION_BLOCK
```

## EZ_SNTP_START

**Summary:**

The EZ_SNTP_START function is used to begin polling the pre-programmed SNTP (Simple Network Time Protocol) servers to update / sync the real time clock to UTC time. The SNTP servers are set in the Project Settings.

The SNTP feature (and SNTP servers) must be installed and configured in the Project Settings before this function is available.

When using SNTP, the real time clock cannot be set using the ladder diagram or structured text without 'breaking' MQTT functionality.

SNTP functionality requires a communication interface such as Ethernet or Wi-Fi. These devices / features must be supported on the hardware target and installed in the Project Settings prior to installing the SNTP feature in the Project Settings.

SNTP functionality requires the real time clock. The real time clock must be supported on the hardware target and installed in the Project Settings prior to installing the SNTP feature in the Project Settings.

**Format:**

EZ_SNTP_START();

**Arguments:**

There are no Arguments for the EZ_SNTP_START functions

**Description:**

The EZ_SNTP_START function causes the target to start polling the pre-programmed SNTP servers configured in the Project Settings (SNTP feature). SNTP will poll the server(s) at the rate specified in the Project Settings and update / sync the real time clock to the SNTP (UTC time).

Refer to Chapter 12 - Real Time Clock for more details on SNTP Project Settings and dialogs.

**Example:**

```
FUNCTION_BLOCK StrtSNTP
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = true) and (LastEN = False) then
                EZ_SNTP_START();
                LastEN := true;
        End_If;

        IF (Enable = False) then
                LastEN := False;
        End_If;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_SNTP_STOP

### Summary:

The EZ_SNTP_STOP function is used to end polling the pre-programmed SNTP (Simple Network Time Protocol) servers and stop updating / syncing the real time clock to UTC time. The SNTP servers are set in the Project Settings.

The SNTP feature (and SNTP servers) must be installed and configured in the Project Settings before this function is available.

When using SNTP, the real time clock cannot be set using the ladder diagram or structured text without 'breaking' MQTT functionality.

SNTP functionality requires a communication interface such as Ethernet or Wi-Fi. These devices / features must be supported on the hardware target and installed in the Project Settings prior to installing the SNTP feature in the Project Settings.

SNTP functionality requires the real time clock. The real time clock must be supported on the hardware target and installed in the Project Settings prior to installing the SNTP feature in the Project Settings.

### Format:

EZ_SNTP_STOP();

### Arguments:

There are no Arguments for the EZ_SNTP_STOP functions

### Description:

The EZ_SNTP_STOP function causes the target to end (stop) polling the pre-programmed SNTP servers configured in the Project Settings (SNTP feature). SNTP will not update / sync the real time clock to the SNTP (UTC time).

Refer to Chapter 12 - Real Time Clock for more details on SNTP Project Settings and dialogs.

### Example:

```
FUNCTION_BLOCK StpSNTP
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                LastEn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        If (Enable = true) and (LastEN = False) then
                EZ_SNTP_START();
                LastEN := true;
        End_If;

        IF (LastEN = true) and (Enable = False) then
                EZ_SNTP_STOP();
        End_If;
```

Continued Next Page------»

```
        IF (Enable = False) then
                LastEN := False;
        End_If;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_SpiWriteData

### Summary:

The EZ_SpiWriteData is used to read / write data to an SPI device on an SPI port. For P-Series targets, the minimum clock rate is 2KHz.

### Format:

DINTvar := EZ_SpiWriteData(*FileDescriptor, Flags, Clockrate, GPIO, TxBuffer, RxBuffer, Len*);

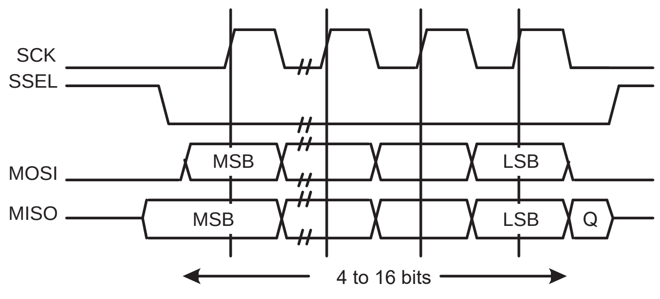
### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_SPI0** for internal PLC on a Chip SPI Port 0.<br>**FD_SPI1** for internal PLC on a Chip SPI Port 1. |
| *Flags* | 16#10000 for active low SPI clock, else uses high SPI Clock<br>16#20000 to capture data on first clock transition. CPHA = 0, else CPHA gets set to 1.<br>Low 8 bits are data fram size (4-16). Typical size is 8 bits. |
| *Clockrate* | Clock rate in Khz, maximum 10 Mhz. (UDINT) |
| *GPIO* | GPIO number for chip select of SPI device (UDINT) |
| *TxBuffer* | Array for data transmit. (Array of USINT) |
| *RxBuffer* | Array for data receive. (Array of USINT) |
| *Len* | Number of frame transfer, sized based on data frame size (UDINT) |
| *DINTvar* | Function return holding variable (BOOL). Returns the number of bytes transferred or negative number if an error occurs. |

### Description:

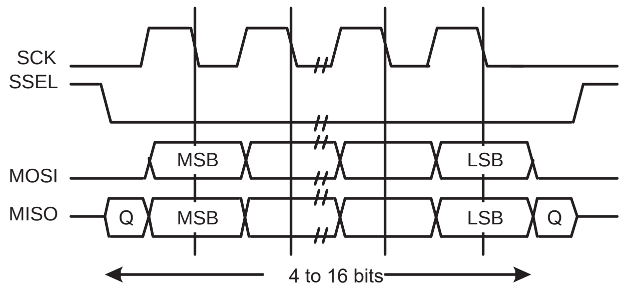
The EZ_SpiWriteData is used to control SPI devices (read and write) connected to an SPI port. Refer to the following timing diagrams for the four modes of operation. The control (read / write) is based entirely on the *Flags* and *Clockrate* as described above. The *FileDescriptor* must match the SPI port being used. The *TxBuffer* and *RxBuffer* are used to hold transmit and recieve data. The *DINTvar* returns teh number of bytes transferred or is negative if an error is detected.

### Diagrams:

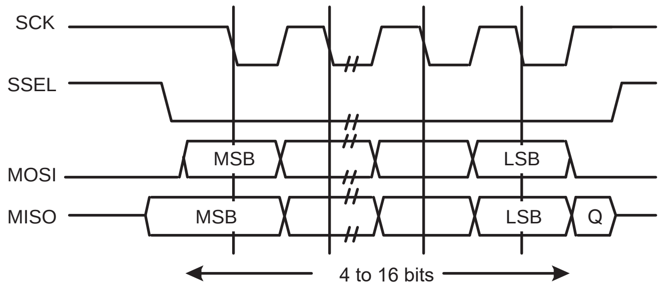


Single transfer with CPOL=0 and CPHA=0

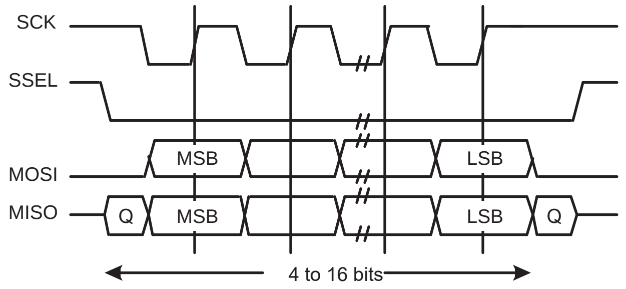
Continued Next Page------»



Single transfer with CPOL=0 and CPHA=1



Single transfer with CPOL=1 and CPHA=0



Single transfer with CPOL=1 and CPHA=1

**Example:**

```
FUNCTION_BLOCK DAC8552_108
        VAR_INPUT
                Enable : bool;
                dac1 : dint;
                dac2 : dint;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR
```

Continued Next Page------»

```
        VAR_TEMP
                txBuf, rxBuf : array [0..2] of byte;
        END_VAR

        Q := False;

        if Enable then

                (* write dac1 value *)
                txBuf[0] := 16#10;

                MSB_INT_TO_ARRAY(txBuf, 1, DINT_TO_INT(dac1));

                EZ_SpiWriteData(
                        FD_SPI1,        (* Use SPI0 *)
                        16#00008,       (* SPI clk idle high, CPHA=1, 8 bit transfers *)
                        1000,           (* 1MHZ Max Clock rate *)
                        108,            (* Use GPIO108 for Chip Select *)
                        txBuf,          (* data to transmit to device *)
                        rxBuf,          (* data to receive data into *)
                        3);             (* transfer 3 bytes *)

                (* write dac2 value *)
                txBuf[0] := 16#24;

                MSB_INT_TO_ARRAY(txBuf, 1, DINT_TO_INT(dac2));

                EZ_SpiWriteData(
                        FD_SPI1,        (* Use SPI0 *)
                        16#00008,       (* SPI clk idle high, CPHA=1, 8 bit transfers *)
                        1000,           (* 1MHZ Max Clock rate *)
                        108,            (* Use GPIO108 for Chip Select *)
                        txBuf,          (* data to transmit to device *)
                        rxBuf,          (* data to receive data into *)
                        3);             (* transfer 3 bytes *)

                Q := true;
        end_if;

END_FUNCTION_BLOCK
```

## EZ_TimeDateCalendarToUnix

**Summary:**

The EZ_TimeDateCalendarToUnix converts standard date and time to a Unix time (number of elapsed seconds since Jan 1, 1970).

**Format:**

EZ_TimeDateCalendarToUnix(LINTvar,*Month,Day,Year,Hour,Min,Sec*);

**Arguments:**

| | |
|---|---|
| *LINTvar* | Variable to hold result of time / date conversion to Unix Time. (LINT). |
| *Month* | Variable for Month input from real time clock or other source (DINT). |
| *Day* | Variable for Day input from real time clock or other source (DINT). |
| *Year* | Variable for Year input from real time clock or other source (DINT). |
| *Hour* | Variable for Hour input from real time clock or other source (DINT). |
| *Min* | Variable for Minute input from real time clock or other source (DINT). |
| *Sec* | Variable for Seconds input from real time clock or other source (DINT). |

**Description:**

The EZ_TimeDateCalendarToUnix is used to convert standard date and time to Unix time (number of elapsed seconds since Jan 1, 1970). The function uses inputs *Month*, *Day*, *Year*, *Hour*, *Min*, *Sec* usually from the real time clock (or other source such as future time date event input) and converts to Unix time and stores the result in *LINTvar*.

When using the EZ_TimeDateCalendarToUnix function to convert to unix time, the date year should be entered (passed) as a four-digit year. A four digit year is required for correct conversion.

**Example:**

```
FUNCTION_BLOCK DTCONVERT
        VAR_INPUT
                Enable : bool;
                MTH : DINT;
                DY : DINT;
                YR : DINT;
                HR : DINT;
                MINS : DINT;
                SEC : DINT;
        END_VAR
        VAR
                UXTIME : LINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                OMTH : DINT;
                ODY : DINT;
                OYR : DINT;
                OHR : DINT;
                OMINS : DINT;
                OSEC : DINT;
                OWKD : DINT;
        END_VAR
```

Continued Next Page------»

```
        IF Enable = True THEN;
        EZ_TimeDateCalendarToUnix(UXTIME,MTH,DY,YR,HR,MINS,SEC);       (*Convert to Unix*)
        EZ_TimeDateUnixToCalendar(UXTIME,OMTH,ODY,OYR,OWKD,OHR,OMINS,OSEC);    (*Convert to Date
                                                                               Time*)
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_TimeDateUnixToCalendar

**Summary:**

The EZ_TimeDateUnixToCalendar converts Unix time (number of elapsed seconds since Jan 1, 1970) to standard date and time.

**Format:**

EZ_TimeDateUnixToCalendar(LINTvar,*Month,Day,Year,Weekday,Hour,Min,Sec*);

**Arguments:**

| | |
|---|---|
| *LINTvar* | Variable to beginning Unix time (to convert to calendar based). (LINT). |
| *Month* | Function output Variable for Month (DINT). |
| *Day* | Function output Variable for Day (DINT). |
| *Year* | Function output Variable for Year 4 Digit year. (DINT). |
| Weekday | Function output Variable for Day of Week (DINT). |
| *Hour* | Function output Variable for Hour (DINT). |
| *Min* | Function output Variable for Minute (DINT). |
| *Sec* | Function output Variable for Seconds (DINT). |

**Description:**

The EZ_TimeDateUnixToCalendar is used to convert Unix time (number of elapsed seconds since Jan 1, 1970) to calendar based time and date. The function uses outputs *Month*, *Day*, *Year*, *Weekday*, *Hour*, *Min*, *Sec* from the function. The LINTvar is used to hold the beginning Unix time value.

**Example:**

```
FUNCTION_BLOCK DTCONVERT
        VAR_INPUT
                Enable : bool;
                MTH : DINT;
                DY : DINT;
                YR : DINT;
                HR : DINT;
                MINS : DINT;
                SEC : DINT;
        END_VAR
        VAR
                UXTIME : LINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                OMTH : DINT;
                ODY : DINT;
                OYR : DINT;
                OHR : DINT;
                OMINS : DINT;
                OSEC : DINT;
                OWKD : DINT;
        END_VAR
```

Continued Next Page------»

```
        IF Enable = True THEN;
        EZ_TimeDateCalendarToUnix(UXTIME,MTH,DY,YR,HR,MINS,SEC);                (*Convert to Unix*)
        EZ_TimeDateUnixToCalendar(UXTIME,OMTH,ODY,OYR,OWKD,OHR,OMINS,OSEC);     (*Convert to Date
                                                                                Time*)
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_UartEnableIsr

### Summary:

The EZ_UartEnableIsr function is used enable or disable UARTs.

### Format:

*BOOLvar* := EZ_UartEnableIsr(*FileDescriptor*, *State*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *State* | State to set the UART (0 for disabled, 1 for enabled) |
| *BOOLvar* | Function return holding variable (BOOL). |

### Description:

The EZ_UartEnableIsr function is used to control on-board PLC on a Chip UART (Serial Ports) state of functionality as Enabled or Disabled. Disabled UARTs will not receive or transmit any data and the UARTs interrupt is disabled. Structure Text buffers (ST Buffers) should be enabled and appropriately sized in the EZ LADDER Projects Settings menu when using structured text UART functions.

### Example:

```
FUNCTION_BLOCK EnableUART2
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                State : bool;
                FncReturn : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
        END_VAR

        IF Enable = 1 THEN;

        FncReturn := EZ_UartEnableIsr(FD_UART2,1);   (*Enable UART2*)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_UartGetBytesToRead

**Summary:**

The EZ_UartGetBytestoRead function access the UART and return the number of bytes available to read from the UART's receive buffer.

**Format:**

*DINTvar* := EZ_UartGetBytestoRead(*FileDescriptor*);

**Arguments:**

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *DINTvar* | Function return holding variable (DINT). # of Bytes available to read in the UART's receive buffer. |

**Description:**

The EZ_UartGetBytestoRead function accesses the UART's receive buffer and returns the number of bytes that are in the receive buffer as *DINTvar* (# of bytes available to read). Structure Text buffers (ST Buffers) should be enabled and appropriately sized in the EZ LADDER Projects Settings menu when using structured text UART functions.

**Example:**

```
FUNCTION_BLOCK GetbytesUART3
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                FncReturn : DINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                ByteAvail: DINT;
        END_VAR

        IF Enable = 1 THEN;

        FncReturn := EZ_UartGetBytestoRead(FD_UART3);     (*Get Bytes from UART3*)
        ByteAvail := FncReturn;                           (*Store value in function output var*)

        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_UartIsTxFinished

### Summary:

The EZ_UartIsTxFinished function access the UART and determines if the UART transmit has completed or has not completed.

### Format:

*BOOLvar* := EZ_UartIsTxFinished(*FileDescriptor*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *BOOLvar* | Function return holding variable (BOOL). False / 0 when transmit has not completed; True or 1 when transmit has completed. |

### Description:

The EZ_UartIsTxFinished function accesses the UART and determines if the current transmit cycle has completed or not and returns the *BOOLvar* as True if complete or False if not complete. If using this function with a UART configured for RS85, this function will turn off the RS485 transmitter after all the characters have been transmitted.

### Example:

```
FUNCTION_BLOCK UART3WrtChk
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                FncReturn1 : bOOl;
                LastEnable : bool;
                FncReturn2 : DINT;
                Data : string[80];
        END_VAR
        VAR_OUTPUT
                Q : bool;
                Done : bool;
        END_VAR
        (* Set Data String to send*)
        Data := 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789^&#!ABCDEFGHIJKLMNOPQRSTUVWX
YZ0123456789^&#!';

        IF Enable = 1 AND LastEnable = 0 THEN;          (*look for rising edge to transmit ie run once*)
        FncReturn2 := EZ_UartWriteStr(FD_UART3,Data);   (* Send data to tx buffer*)
        END_IF;

        IF Enable = 1 THEN;                             (*Look for Enable to keep checking if done*)
        FncReturn1 := EZ_UartIsTxFinished(FD_UART3);    (*Check if finished*)
        Done := FncReturn1;                             (*Set FncReturn to output variable to view in ladder*)
        END_IF;

        Q := Enable;
        LastEnable := Enable;                           (*set flag for rising edge run once*)

END_FUNCTION_BLOCK
```

## EZ_UartRead

**Summary:**

The EZ_UartRead function access the UART and reads data from the UARTs receive buffer into a byte array of USINT.

**Format:**

*DINTvar* := EZ_UartRead(*FileDescriptor,RxBuffer, Offset, Len*);

**Arguments:**

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *RxBuffer* | Byte array buffer variable to store received data (ARRAY [ ] of USINT) |
| *Offset* | Zero based offset value into *RxBuffer* to begin storing received data (DINT). |
| *Len* | Number of bytes to read and store into *TxBuffer* (DINT). |
| *DINTvar* | Function return holding variable (DINT). Returns the number of bytes read. |

**Description:**

The EZ_UartRead function accesses the UART and reads the *Len* number of bytes from the UART's receive buffer then stores the read bytes into the byte array variable *RXBuffer* beginning in the byte array variable *RXBuffer* at the *Offset* location. If there are less available bytes in the UART's receive buffer than specified in the *Len*, then just the number available is actually read and stored in *RXbuffer*. The *DINTvar* return variable stores the actual number of bytes read and stored in *RXbuffer*. The EZ_UartGetBytestoRead may be used to identify the number of bytes available and to trigger an actual read using this function. Structure Text buffers (ST Buffers) should be enabled and appropriately sized in the EZ LADDER Projects Settings menu when using structured text UART functions.

**Example:**

```
FUNCTION_BLOCK UART3RdBytes
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                FncReturn1 : DINT;
                FncReturn2 : DINT;
                DATA : ARRAY[0..4] of USINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                NByte : DINT;
                RByte1 : DINT;
                RByte2 : DINT;
                RByte3 : DINT;
                RByte4 : DINT;
        END_VAR

        IF Enable = 1 THEN;                             (*Look for Enable to keep checking if done*)
        FncReturn1 := EZ_UartGetBytestoRead(FD_UART3);  (*Check if for # bytes*)
        NByte := FncReturn1;
                IF FncReturn1 >= 4 THEN;                        (*Check for 4 bytes*)
```

Continued Next Page------»

```
                        FncReturn2 := EZ_UartRead(FD_UART3,DATA,0,4);      (*Read 4 bytes*)
                        RByte1 := DATA[0];                          (*Store in output to view*)
                        RByte2 := DATA[1];
                        RByte3 := DATA[2];
                        RByte4 := DATA[3];
                END_IF;
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_UartSetBaudRate

### Summary:

The EZ_SetBaudRate function access the UART sets / changes the Baud Rate of the UART.

### Format:

*DINTvar* := EZ_SetBaudRate(*FileDescriptor,BaudRate*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *BaudRate* | Baud Rate for UART (ie: 19200, 9600) (UDINT). Supports baud rates 4800, 9600, 19200, 57600 and 115200. |
| *DINTvar* | Function return holding variable (DINT). Returns the number of bytes read. |

### Description:

The EZ_SetBaudRate function accesses the UART and sets the UART's baud rate for communications. The BaudRate is used to configure the baud rate in typical bits per second (ie: 9600, 19200, 57600, etc). The DINTvar returns a 0 when the baud rate has been successfully set. Structure Text buffers (ST Buffers) should be enabled and appropriately sized in the EZ LADDER Projects Settings menu when using structured text UART functions.

### Example:

```
FUNCTION_BLOCK UARTCtrlBaud
        VAR_INPUT
                Enable : bool;
                BR19200 : bool;
        END_VAR
        VAR
                FncReturn1 : DINT;
                FncReturn2 : DINT;
                FncReturn3 : DINT;
                DATA : ARRAY[0..1] of USINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RByte1 : DINT;
                RByte2 : DINT;
        END_VAR

        IF Enable = 1 THEN;                     (*Look for Enable to keep checking if done*)
                IF BR19200 = 1 THEN;            (*Change baud rate input set*)
                        FncReturn3 := EZ_UartSetBaudRate(FD_UART3,19200);         (*Set to 19200*)
                ELSE
                        FncReturn3 := EZ_UartSetBaudRate(FD_UART3,9600);          (*Set to 9600*)
                END_IF;

        FncReturn1 := EZ_UartGetBytestoRead(FD_UART3);                    (*Check if for # bytes*)
                IF FncReturn1 >= 2 THEN;                                  (*Check for 2 bytes*)
```

Continued Next Page------»

```
                        FncReturn2 := EZ_UartRead(FD_UART3,DATA,0,2);      (*Read 2 bytes*)
                        RByte1 := DATA[0];                                  (*Store in output to view*)
                        RByte2 := DATA[1];
                END_IF;
        END_IF;

        Q := Enable;

END_FUNCTION_BLOCK
```

## EZ_UartWrite

### Summary:

The EZ_UartWrite function access the UART and loads data into the UARTs transmit buffer from a byte array source.

### Format:

*DINTvar* := EZ_UartWrite(*FileDescriptor,TxBuffer, Offset, Len*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *TxBuffer* | Byte array buffer variable to transmit data from (ARRAY [ ] of USINT). |
| *Offset* | Zero based offset value into *TxBuffer* to begin reading transmit data (DINT). |
| *Len* | Number of bytes of Byte array buffer *TxBuffer* to write beginning at *Offset* (DINT). |
| *DINTvar* | Function return holding variable (DINT). Returns the number of bytes written or 0 if the UART is busy. |

### Description:

The EZ_UartWrite function accesses the UART and loads the data in the byte array *TxBuffer* into the UART's transmit buffer beginning with the *Offset* byte and continuing for the *Len* number of bytes.This will copy the data into the buffer and begin transmitting. This function will return before all the data has been transmitted, so the EZ_UartIsTxFinished function should be used to identify the completion of the transmit. The *DINTvar* return variable returns the number of bytes written to the transmit buffer or a 0 if the UART is busy. Structure Text buffers (ST Buffers) should be enabled and appropriately sized in the EZ LADDER Projects Settings menu when using structured text UART functions.

### Example:

```
FUNCTION_BLOCK UART3WrtByteChk
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                FncReturn1 : bOOl;
                LastEnable : bool;
                FncReturn2 : DINT;
                DATA : ARRAY[0..15] of USINT;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                Done : bool;
        END_VAR

        (* Set values of data in byte array to send*)
        DATA[0] := 48; DATA[1] := 49; DATA[2] := 50; DATA[3] := 51; DATA[4] := 52; DATA[5] := 53; DATA[6] := 54;
        DATA[7] := 55; DATA[8] := 56; DATA[9] := 57; DATA[10] := 65; DATA[11] := 66; DATA[12] := 67; DATA[13] := 68;
        DATA[14] := 69; DATA[15] := 70;

        IF Enable = 1 AND LastEnable = 0 THEN;              (*look for rising edge to transmit ie run once*)
        FncReturn2 := EZ_UartWrite(FD_UART3,DATA,0,16);     (* Send data to tx buffer*)
        END_IF;
```

Continued Next Page------»

```
        IF Enable = 1 THEN;                                *Look for Enable to keep checking if done*)
        FncReturn1 := EZ_UartIsTxFinished(FD_UART3);       (*Check if finished*)
        Done := FncReturn1;                                (*Set FncReturn to output variable to view in ladder*)
        END_IF;

        Q := Enable;
        LastEnable := Enable;                              (*set flag for rising edge run once*)

END_FUNCTION_BLOCK
```

## EZ_UartWriteStr

### Summary:

The EZ_UartWriteStr function accesses the UART and loads data into the UARTs transmit buffer from a string source.

### Format:

*DINTvar* := EZ_UartWriteStr(*FileDescriptor,StrBuffer*);

### Arguments:

| | |
|---|---|
| *FileDescriptor* | **FD_UART1** for internal PLC on a Chip UART1. |
| | **FD_UART2** for internal PLC on a Chip UART2. |
| | **FD_UART3** for internal PLC on a Chip UART3. |
| | **FD_UART4** for internal PLC on a Chip UART4. |
| *StrBuffer* | Holding Varialble with String data. (STRING) |
| *DINTvar* | Function return holding variable (DINT). Returns the number of bytes written or 0 if the UART is busy. |

### Description:

The EZ_UartWriteStr function accesses the UART and loads the data in *StrBuffer* into the UART's transmit buffer.This will copy the entire string into the buffer and begin transmitting. This function will return before all the data has been transmitted, so the EZ_UartIsTxFinished function should be used to identify the completion of the transmit. The *DINT-var* return variable returns the number of bytes written to the transmit buffer or a 0 if the UART is busy. Structure Text buffers (ST Buffers) should be enabled and appropriately sized in the EZ LADDER Projects Settings menu when using structured text UART functions.

### Example:

```
FUNCTION_BLOCK UART3WrtChk
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR
                FncReturn1 : bOOl;
                LastEnable : bool;
                FncReturn2 : DINT;
                Data : string[80];
        END_VAR
        VAR_OUTPUT
                Q : bool;
                Done : bool;
        END_VAR
        (* Set Data String to send*)
        Data := 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789^&#!ABCDEFGHIJKLMNOPQRSTUVWX
YZ0123456789^&#!';

        IF Enable = 1 AND LastEnable = 0 THEN;                (*look for rising edge to transmit ie run once*)
        FncReturn2 := EZ_UartWriteStr(FD_UART3,Data);         (* Send data to tx buffer*)
        END_IF;

        IF Enable = 1 THEN;                                   (*Look for Enable to keep checking if done*)
        FncReturn1 := EZ_UartIsTxFinished(FD_UART3);          (*Check if finished*)
        Done := FncReturn1;                                   (*Set FncReturn to output variable to view in ladder*)
        END_IF;
```

Continued Next Page------»

```
        Q := Enable;
        LastEnable := Enable;                                    (*set flag for rising edge run once*)

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_Access_Points

**Summary:**

The EZ_WiFi_Get_Access_Points function is used scan and return all the AP (access points / wi-fi networks) in range of the Wi-Fi module. This function returns a string of data for each access point detected.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Get_Access_Points(*Error,AccessPts*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*AccessPts*

Variable to hold detected access point data (STRING). An access point's data is returned as a single string and this string will need parsed.

Continued Next Page------»

**Description:**

The EZ_WiFi_Get_Access_Points function commands the on-board Wi-Fi module to scan for all available wi-fi networks (Access points / APs) within range and return information about the access point as *AccessPts* (STRING). For proper operation, this function must be repeatedly called until the *Statvar* = 2 signifying it has completed. When *Statvar* = 1, it signifies that some access point data is available (but the function hasn't completed scanning and finished). As the function is repeatedly 'polled', all networks are located before the *Statvar* = 2.

When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (see previous page Error values).

The *AccessPts* data is received as a single string and must be parsed as needed for information (application specific). This function :

> Returns a list of up to 16 APs and Ad-Hoc networks available in the surrounding area. Each line contains the following comma-separated fields:
> <SSID>,<AP Type>,<BSSID>,<security type>,<channel>,<RSSI>
>
> <AP Type> = ADHOC|AP
> <Security type> = NONE|WEP64|WEP128|WPA|WPA2
> <RSSI> = Value between 0-255 which represents (SNR+NoiseFloor). Higher RSSI values indicate weaker signal strength.
>
> For example:
> Jetta,AP,06:14:6C:69:4A:7C,WPA,1,25
> RTL8186-default,AP,00:E0:4C:81:86:86,NONE,1,77
> dlink_test,AP,00:1C:F0:9A:63:7A,NONE,1,68

**Example:**

```
FUNCTION_BLOCK WifiGetAccessPoints
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                response: DINT;
                complete: bool := FALSE;
                err : INT := 0;
                getStr : STRING[60];
                descriptionString : STRING[50];
                buff : STRING[10];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                        result := EZ_Wifi_Get_Access_Points(err, getStr);
                        RES := INT_TO_DINT(result);    (*set functionblock RES to result*)
                        ER := INT_TO_DINT(err);        (*set functionblock ER to err*)

                        IF (result = 1 or result = 2) THEN
```

Continued Next Page------»

```
                    (*Write result to serial port. Don't print NULL Strings*)
                    IF (getStr[0] <> 0) THEN

                        EZ_FormatString(buff, '$N');
                        EZ_FormatString(descriptionString, '$NReading Access Points: $N');
                        while EZ_UartWriteStr(FD_UART3, descriptionString) <= 0 do
                        ;
                        end_while;

                        (*Print Result*)
                        while EZ_UartWriteStr(FD_UART3, getStr) <= 0 do
                            ;
                        end_while;
                        (*Print Carriage Return and Line Feed*)
                        while EZ_UartWriteStr(FD_UART3, buff) <= 0 do
                            ;
                        end_while;
                    END_IF;
                END_IF;

                IF (result = 2) THEN
                    complete := TRUE;
                END_IF;

                IF (result < 0) THEN
                    EZ_FormatString(getStr, 'Read AP List Error Res: %d Er: %d $N', result, err);
                    (*Print Error Information*)
                    while EZ_UartWriteStr(FD_UART3, getStr) <= 0 do
                        ;
                    end_while;
                    complete := TRUE;
                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
            RES := 0;
            complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_GetChannel

### Summary:

The EZ_WiFi_Get_Channel function reads the channel (used to connect to a wi-fi network) from the on-board Wi-Fi module. Only one channel is stored.

This function should be called / polled until it's status is compete (*Statvar* =2).

### Format:

*Statvar* := EZ_WiFi_Get_Channel(*Error, Channel*);

### Arguments:

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Channel*

The Channel number for Wi-Fi communications read from the Wi-Fi module (INT). Channels will range from 0 to 12.

### Description:

The EZ_WiFi_Get_Channel reads the *Channel* used for communicating to the Wi-Fi network from the Wi-Fi module. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values).

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK WifiGetChannel
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
                WC : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
                channel : INT;
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Get_Channel(err, channel);   (*read channel# from Wi-Fi module*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN
                        WC := INT_TO_DINT(channel);  (*Copy to functionblock output pin to see in ladder*)
                        (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Read Channel Complete. Channel: %d $N', WC);

                        ELSE

                        EZ_FormatString(buffer, 'Write Channel Error - Res: %d Er: %d $N', result, err);

                        END_IF;

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                                ;
                        end_while;
                        complete := TRUE;                    (*Finished*)
                        ER := INT_TO_DINT(err);

                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_Connection_Status_1

**Summary:**

The EZ_WiFi_Get_Connection_Status_1 function is used retrieve the current Wi-Fi connection status information including port status, transfer rate, signal level and link quality.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Get_Connection_Status_1(*Error,PortStat,XferRate,Siglevel, LinkQual*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*PortStat*

Variable to hold the current Wi-Fi port status (INT). Returns as follows:

**PortStat Values**

| | |
|---|---|
| 0 | Wireless LAN adapter not present |
| 1 | Wireless LAN adapter disabled |
| 2 | Searching for initial connection |
| 4 | Connected |
| 5 | Out of Range |

*XferRate*

Variable to hold the current Wi-Fi data transfer rate (INT). Transfer rate values can range from

Continued Next Page------»

| | |
|---|---|
| | 1 to 54 mbps. |
| *Siglevel* | Variable to hold the current Wi-Fi signal level (INT). Signal level is reported as a % with range of 0 to 100 (%). |
| *LinkQual* | Variable to hold the current Wi-Fi Link Quality (INT). Link quality is reported as a % with range of 0 to 100 (%). |

**Description:**

The EZ_WiFi_Get_Connection_Status_1 returns connection status regarding the on-board Wi-Fi module's current connection. For proper operation, this function must be repeatedly called until the *Statvar* = 2 signifying it has completed.

When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (see previous page Error values).

The *PortStat* (INT) variable returns the current port status (0,1,2,4 or 5) per the list on the previous page. The *XferRate* (INT) variable returns the current transfer rate. The *Siglevel* (INT) variable holds the current signal level as a percent (0-100).The *LinkQual glevel* (INT) variable holds the current link quality (connection quality) as a percent (0-100).

**Example:**

```
FUNCTION_BLOCK WifiGetConnectionStatus1
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                buffer : STRING[100];

                err : INT := 0;
                _ps : INT := 0;
                _xr : INT := 0;
                _sl : INT := 0;
                _lq : INT := 0;

        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Get_Connection_Status_1(err, _ps, _xr, _sl, _lq);
                RES := INT_TO_DINT(result);    (*Copy to functionblock output*)
                ER := INT_TO_DINT(err);        (*Copy to functionblock output*)

                IF (result <> 0) THEN

                    IF (result = 2 and err = 0) THEN
                    (*Successful read from Wifi module*)
                    EZ_FormatString(buffer,'Read Connection Status 1 Complete. %d,%d,%d,%d $N',_ps,_xr,_sl,_lq);

                        ELSE
                        (*Error while reading from Wifi module*)
```

Continued Next Page------»

```
                        EZ_FormatString(buffer,'Read Connection Status 1 Error Res: %d Er: %d $N', result, err);

                        END_IF;

                        (*Write result to serial port*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do
                                    ;
                        end_while;

                        complete := TRUE;
                  END_IF;

            END_IF;

            IF (Enable = FALSE) THEN
                  RES := 0;
                  complete := FALSE;
            END_IF;

            Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_Connection_Status_2

**Summary:**

The EZ_WiFi_Get_Connection_Status_2 function is used retrieve a report of the current WLAN connection (see info in Arguments and Description below).

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Get_Connection_Status_2(*Error,Status*);

**Arguments:**

*Statvar* — Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error* — Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Status* — Variable to hold the current Wi-Fi port status (STRING). The status is return as one string that will require parsing to get data based on application requirements.

**Description:**

The EZ_WiFi_Get_Connection_Status_2 returns connection information regarding the on-board Wi-Fi modules current connection to a network (WLAN). For proper operation, this function must be repeatedly called until the *Statvar* = 2 signifying it has completed.

Continued Next Page------»

When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (see previous page Error values).

The *Status* (STRING) variable returns the current connection information (see information below) as a single string. This string will require parsing to retrieve data based on actual applications needs.

Returns a report of the current WLAN connection.
<SSID>,<BSSID>,<security type>,<WPA status>,<channel>,<SNR>

<Security type> = NONE|WEP64|WEP128|WPA|WPA2
<WPA status> = Completed|Not Completed
This indicates whether WPA negotiation completed or not, appears only when WPA/WPA2 security is specified.

Return *Status* example:
Jetta,06:14:6C:69:4A:7C,WPA,Completed,1,68

**Example:**

```
FUNCTION_BLOCK WifiGetConnectionStatus2
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool;
                err : INT := 0;
                returnString : STRING[50];
                descriptionString : STRING[50];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Get_Connection_Status_2(err, returnString);
                RES := INT_TO_DINT(result);    (*Copy to functionblock output*)
                ER := INT_TO_DINT(err);                (*Copy to functionblock output*)

                IF (result <> 0) THEN

                   EZ_FormatString(descriptionString, 'Reading Connection Status 2: ');
                   while EZ_UartWriteStr(FD_UART3, descriptionString) <= 0 do
                                ;
                   end_while;

                   IF (result <> 2 or err <> 0) THEN
                   (*Error while reading from Wifi module*)
                    EZ_FormatString(returnString, 'Read Connection Status 1 Error. Res: %d Er: %d $N', result, err);

                    END_IF;

                    IF (returnString[0] = 0) THEN
```

Continued Next Page------»

```
                        EZ_FormatString(returnString, 'No connection. Null string returned.');
                        END_IF;

                        (*Write result to serial port*)
                        while EZ_UartWriteStr(FD_UART3, returnString) <= 0 do
                                    ;
                        end_while;

                        complete := TRUE;

            END_IF;


      END_IF;

      IF (Enable = FALSE) THEN
            RES := 0;
            complete := FALSE;
      END_IF;

      Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_Mode

### Summary:

The EZ_WiFi_Get_Mode function is used read the current Wi-Fi mode of operation (Client or Host). See **Chapter 30 - Ethernet / Wi-Fi** for more information modes of operation (Client and Host mode).

This function should be called / polled until it's status is compete (*Statvar* =2).

### Format:

*Statvar* := EZ_WiFi_Get_Mode(*Error, Mode*);

### Arguments:

*Statvar* Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error* Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Mode* Function return holding variable (INT). Returns the current Wi-Fi mode of operation (0 for Client or 1 for Host). This must be converted to a DINT before it can exported from structured text to the ladder diagram.See **Chapter 30 - Ethernet / Wi-Fi** for more information modes of operation (Client and Host mode).

### Description:

The EZ_WiFi_Get_Mode reads and returns the operational mode of the on-board Wi-Fi to either Client (0) or Host (1). The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding Wi-Fi modes of operation.

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK WifiRDMode
	VAR_INPUT
		Enable : bool;
	END_VAR
	VAR_OUTPUT
		Q : bool;
		RES : DINT;
		ER : DINT;
		MD : DINT;
	END_VAR
	VAR
		result : INT;
		complete: bool := FALSE;
		err : INT := 0;
		buffer : STRING[50];
		cmode : INT;
	END_VAR

	IF ((Enable = TRUE) AND (complete = FALSE)) THEN

		result := EZ_Wifi_Get_Mode(err,cmode);			(*Read Mode*)
		RES := INT_TO_DINT(result);
		MD := INT_TO_DINT(cmode);

		IF (result <> 0) THEN

			IF (result = 2 and err = 0) THEN			(*Format for writing to uart*)
			EZ_FormatString(buffer, 'Set Read Completed :%d',MD);

			ELSE

			EZ_FormatString(buffer, 'Read Mode Error - Res: %d Er: %d $N', result, err);

			END_IF;

			(*Print result*)
			while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do  (*write to uart*)
				;
			end_while;
			complete := TRUE;		(*Finished*)
			ER := INT_TO_DINT(err);

		END_IF;

	END_IF;

	IF (Enable = FALSE) THEN
		RES := 0;
		complete := FALSE;
	END_IF;

	Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_Passcode

**Summary:**

The EZ_WiFi_Get_Passcode function is used read an SSID's passcode (to connect to a wi-fi network) from the on-board Wi-Fi module SSID (network) connections list. A total of 10 passcodes (one for each SSID) may be stored. These SSIDs storage locations are the same as described in Chapter 30. See **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage. The passcode location # should match the SSID location # of the desired SSID connections. The passcode is not viewable and is always returned as astericks (*). This function may be used to identify if a passcode has been set or the number of charactes in the passcode only.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Get_Passcode(*Error, Index, Passcode*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Index*

Index number to read the *Passcode* from (INT)(referred to as slot in Chapter 19). Valid *Index* is 0-9. The passcode for an SSID must be read from the same *Index* as the SSID itself (ie: the passcode *Index* should equal the SSID *Index* for proper operation).

*Passcode*

Variable to hold the Passcode read from the Wi-Fi module (STRING). This passcode is used to authenticate the to the network for the SSID stored at *Index* location. The Passcode

Continued Next Page------»

string variable declaration should be large enough to handle the Passcode. If the Passcode is larger than the declared variable, it will be truncated.The passcode is not viewable and is always returned as astericks (*).

**Description:**

The EZ_WiFi_Get_Passcode reads the *Passcode* from the Wi-Fi module's memory at the *Index* location (0-9). The passcode stored at the *Index* location should be the passcode to connect to the SSID of the same *Index* location. For example, the SSID stored at index location (slot) 5 would use the passcode stored at index location 5. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

The passcode is not viewable and is always returned as astericks (*). This function may be used to identify is a passcode has been set or the number of charactes in the passcode only.

**Example:**

```
FUNCTION_BLOCK WifiGetPasscode
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
                buffer2 : STRING[40];
                index : INT := 3;
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Get_Passcode(err, index, buffer2);   (*read passcode to index location 3*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Read Passcode Complete %d : Passcode : ', index);

                        (*Print partial result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                                ;
                        end_while;

                        (*Print Passcode to uart*)
                        while EZ_UartWriteStr(FD_UART3, buffer2) <= 0 do  (*write to uart*)
                        ;
                        end_while;
```

Continued Next Page------»

```
                        (*Print CRLF to uart*)
                        while EZ_UartWriteStr(FD_UART3, '$N') <= 0 do  (*write to uart*)
                        ;
                        end_while;


                        ELSE
                        EZ_FormatString(buffer, 'Write Passcode Error %d Res: %d Er: %d $N', index, result, err);

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                              ;
                        end_while;
                        END_IF;


                        complete := TRUE;                   (*Finished*)
                        ER := INT_TO_DINT(err);

                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_Security

**Summary:**

The EZ_WiFi_Get_Security function is read an SSID's security setting (needed to connect to a wi-fi network) from the on-board Wi-Fi module SSID (network) connections list. A total of 10 security settings (one for each SSID) may be stored. These SSIDs storage locations are the same as described in Chapter 30. See **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage. The security setting location # should match the SSID location # and passcode location # for the SSID.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Get_Security(*Error, Index, Type*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Index*

Index number to read the Security *Type* from (INT)(referred to as slot in Chapter 19). Valid *Index* is 0-9. The Security *Type* for an SSID to function properly must be read from the same *Index* as the SSID itself (ie: the SSID *Index* and passcode *Index* should equal the Security *Type Index* for proper operation).

*Type*

Variable to hold the Security *Type* to be read from the Wi-Fi module (INT). This Security *Type*

Continued Next Page------»

is used to authenticate the to the network for the SSID stored at *Index* location. Supported values are:

| Value | Security Type |
|---|---|
| 0 | No Security |
| 3 | WPA |
| 4 | WPA2 |

## Description:

The EZ_WiFi_Get_Security reads the security *Type* value for the SSID from the Wi-Fi module's memory at the *Index* location (0-9). The security *Type* stored at the *Index* location is the security *Type* of the SSID (required to connect to the SSID) of the same *Index* location. For example, the SSID stored at index location (slot) 5 would use the passcode stored at index location 5 and the security *Type* stored at index location 5. The supported *Types* are listed above. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

## Example:

```
FUNCTION_BLOCK WifiGetSecurity
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
                ST : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
                SEC : INT;
                index : INT := 3;
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Get_Security(err, index, SEC);   (*Read security type from index location 3*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN
                        ST := INT_TO_DINT(SEC);
                        (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Read Security Complete %d $N', SEC);

                        ELSE

                        EZ_FormatString(buffer, 'Write Security Error %d Res: %d Er: %d $N', index, result, err);
```

Continued Next Page------»

```
                END_IF;

                (*Print result*)
                while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                    ;
                end_while;
                complete := TRUE;               (*Finished*)
                ER := INT_TO_DINT(err);

            END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
            RES := 0;
            complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Get_SSID

**Summary:**

The EZ_WiFi_Get_SSID function is used read an SSID from the on-board Wi-Fi module SSID (network) connections list. A total of 10 SSIDs may be stored. These SSIDs storage locations are the same as described in Chapter 30. See **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Get_SSID(*Error, Index, SSID*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Index*

Index number to read the *SSID* from (INT)(referred to as slot in Chapter 19). Valid *Index* is 0-9.

*SSID*

Variable to hold SSID that is read from the Wi-Fi module (STRING). The SSID string variable declaration should be large enough to handle the SSID. If the SSID is larger than the declared variable, it will be truncated.

Continued Next Page------»

**Description:**

The EZ_WiFi_Get_SSID reads an *SSID* value from the Wi-Fi module's memory at the *Index* location (0-9).The *Statvar* variable returns the status of the function per the list on the previous page. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

**Example:**

```
FUNCTION_BLOCK WifiGetSSID
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool;
                err : INT;
                buffer : STRING[25];
                buffer2 : STRING[80];
                index : INT;
        END_VAR

        index := 3;                                                       (*Set index / slot to 3*)

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Get_SSID(err, index, buffer);    (*keep calling / checking function until complete*)
                RES := INT_TO_DINT(result);
                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN
                        (*No Error, format print data description for uart*)
                        EZ_FormatString(buffer2, 'Read SSID Complete. SSID: ');

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer2) <= 0 do  (*Print data to uart*)
                                ;
                        end_while;

                        (*No Error, copy SSID for print to uart*)
                        buffer2 := buffer;
                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer2) <= 0 do  (*Print data to uart*)
                                ;
                        end_while;

                        (*Print CRLF to uart*)
                        while EZ_UartWriteStr(FD_UART3, '$N') <= 0 do  (*Print data to uart*)
                                ;
                        end_while;
```

Continued Next Page------»

```
                    ELSE
                    (*Error, format print data for uart*)
                    EZ_FormatString(buffer2, 'Read SSID Error %d Res: %d Er: %d $N', index, result, err);

                    (*Print result*)
                    while EZ_UartWriteStr(FD_UART3, buffer2) <= 0 do  (*Print data to uart*)
                         ;
                    end_while;
                    END_IF;

                    complete := TRUE;                    (*Finished*)
                    ER := INT_TO_DINT(err);

               END_IF;

          END_IF;

          IF (Enable = FALSE) THEN
               RES := 0;
               complete := FALSE;
          END_IF;

          Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Set_Channel

**Summary:**

The EZ_WiFi_Set_Channel function is used write and store the channel (used to connect to a wi-fi network) to the on-board Wi-Fi module module. Only one channel may be stored.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Set_Channel(*Error, Channel*);

**Arguments:**

*Statvar* Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error* Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Channel* The Channel number for Wi-Fi communications to store on the Wi-Fi module (INT). Supports channels 0-12.

**Description:**

The EZ_WiFi_Set_Channel writes / stores the *Channel* used for communicating to the Wi-Fi network to the Wi-Fi module. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values).

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK WifiSetChannel
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Set_Channel(err, 6);   (*Store channel #6 to Wi-Fi module*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Write Channel Complete');

                        ELSE

                        EZ_FormatString(buffer, 'Write Channel Error - Res: %d Er: %d $N', result, err);

                        END_IF;

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                                ;
                        end_while;
                        complete := TRUE;                       (*Finished*)
                        ER := INT_TO_DINT(err);

                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Set_Mode

### Summary:

The EZ_WiFi_Set_Mode function is used to configure the Wi-Fi mode of operation as Client or Host. See **Chapter 30 - Ethernet / Wi-Fi** for more information modes of operation (Client and Host mode).

This function should be called / polled until it's status is compete (*Statvar* =2).

### Format:

*Statvar* := EZ_WiFi_Set_Mode(*Error, Mode*);

### Arguments:

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Mode*

Either a variable or hard-coded. When 0, sets the Wi-Fi mode to Client and when 1, sets the Wi-Fi mode to Host. See **Chapter 30 - Ethernet / Wi-Fi** for more information modes of operation (Client and Host mode).

### Description:

The EZ_WiFi_Set_Mode sets the operational mode of the on-board Wi-Fi to either Client (0) or Host (1).The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding Wi-Fi modes of operation.

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK WifiSetMode
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Set_Mode(err, 0);             (*Set to Client Mode*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN        (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Set Mode Completed');

                        ELSE

                        EZ_FormatString(buffer, 'Set MOde Error - Res: %d Er: %d $N', result, err);

                        END_IF;

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART2, buffer) <= 0 do  (*write to uart*)
                                ;
                        end_while;
                        complete := TRUE;               (*Finished*)
                        ER := INT_TO_DINT(err);

                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;
```

## EZ_WiFi_Set_Passcode

**Summary:**

The EZ_WiFi_Set_Passcode function is used write and store an SSID's passcode (to connect to a wi-fi network) to the on-board Wi-Fi module SSID (network) connections list. A total of 10 passcodes (one for each SSID) may be stored. These SSIDs storage locations are the same as described in Chapter 30. See **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage. The passcode location # should match the SSID location #.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Set_Passcode(*Error, Index, Passcode*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Index*

Index number to store the *Passcode* to (INT)(referred to as slot in Chapter 19). Valid *Index* is 0-9. The passcode for an SSID to function properly must be stored in the same *Index* as the SSID itself (ie: the passcode *Index* should equal the SSID *Index* for proper operation).

*Passcode*

Variable to hold the Passcode to be stored in the Wi-Fi module (STRING). This passcode is used to authenticate the to the network for the SSID stored at *Index* location. The Passcode string variable declaration should be large enough to handle the Passcode. If the Passcode is larger than the declared variable, it will be truncated.

Continued Next Page------»

**Description:**

The EZ_WiFi_Set_Passcode writes / stores a *Passcode* value to the Wi-Fi module's memory at the *Index* location (0-9). The passcode stored at the *Index* location should be the passcode to connect to the SSID of the same *Index* location. For example, the SSID stored at index location (slot) 5 would use the passcode stored at index location 5. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

When writing passcodes to the Wi-Fi module, the write may take up to 20 seconds to complete.

**Example:**

```
FUNCTION_BLOCK WifiSetPasscode
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
                index : INT := 3;
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Set_Passcode(err, index, '12345678');   (*Store passcode to index location 3*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Write Passcode Complete %d $N', index);

                        ELSE

                        EZ_FormatString(buffer, 'Write Passcode Err %d Res: %d Er: %d $N', index, result, err);

                        END_IF;

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                                ;
                        end_while;
                        complete := TRUE;                   (*Finished*)
                        ER := INT_TO_DINT(err);

                END_IF;

        END_IF;
```

Continued Next Page------»

```
        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Set_Security

**Summary:**

The EZ_WiFi_Set_Security function is used write and store an SSID's security setting (to connect to a wi-fi network) to the on-board Wi-Fi module SSID (network) connections list. A total of 10 security settings (one for each SSID) may be stored. These SSIDs storage locations are the same as described in Chapter 30. See **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage. The security setting location # should match the SSID location # and passcode location # for the SSID.

This function should be called / polled until it's status is compete (*Statvar* =2).

**Format:**

*Statvar* := EZ_WiFi_Set_Security(*Error, Index, Type*);

**Arguments:**

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Index*

Index number to store the Security *Type* to (INT)(referred to as slot in Chapter 19). Valid *Index* is 0-9. The Security *Type* for an SSID to function properly must be stored in the same *Index* as the SSID itself (ie: the SSID *Index* and passcode *Index* should equal the Security *Type Index* for proper operation).

*Type*

Variable to hold the Security *Type* to be stored in the Wi-Fi module (INT). This Security *Type* is

Continued Next Page------»

used to authenticate the to the network for the SSID stored at *Index* location. Supported values are:

| Value | Security Type |
|---|---|
| 0 | No Security |
| 3 | WPA |
| 4 | WPA2 |

**Description:**

The EZ_WiFi_Set_Security writes / stores the security *Type* value for the SSID to the Wi-Fi module's memory at the *Index* location (0-9). The security *Type* stored at the *Index* location should be the security *Type* of the SSID (required to connect to the SSID) of the same *Index* location. For example, the SSID stored at index location (slot) 5 would use the passcode stored at index location 5 and the security *Type* stored at index location 5. The supported *Types* are listed above. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

**Example:**

```
FUNCTION_BLOCK WifiSetSecurity
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
                index : INT := 3;
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Set_Security(err, index, 3);   (*Store security type WPA to index location 3*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN (*Format for writing to uart*)
                        EZ_FormatString(buffer, 'Write Security Complete %d $N', index);

                        ELSE

                        EZ_FormatString(buffer, 'Write Security Error %d Res: %d Er: %d $N', index, result, err);

                        END_IF;

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*write to uart*)
                                ;
                        end_while;
```

Continued Next Page------»

```
                        complete := TRUE;                   (*Finished*)
                        ER := INT_TO_DINT(err);

                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Set_SSID

### Summary:

The EZ_WiFi_Set_SSID function is used write and store an SSID to the on-board Wi-Fi module SSID (network) connections list. A total of 10 SSIDs may be stored. These SSIDs storage locations are the same as described in Chapter 30. See **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

This function should be called / polled until it's status is compete (*Statvar* =2).

### Format:

*Statvar* := EZ_WiFi_Set_SSID(*Error, Index, SSID*);

### Arguments:

*Statvar* Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error* Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

*Index* Index number to store the *SSID* to (INT)(referred to as slot in Chapter 19). Valid *Index* is 0-9.

*SSID* Variable to hold SSID to be stored in the Wi-Fi module (STRING). The SSID string variable declaration should be large enough to handle the SSID. If the SSID is larger than the declared variable, it will be truncated.

Continued Next Page------»

**Description:**

The EZ_WiFi_Set_SSID writes / stores a known *SSID* value to the Wi-Fi module's memory at the *Index* location (0-9). The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values). Refer to **Chapter 30 - Ethernet / Wi-Fi** for more information regarding SSID storage.

Due to encryption calculations, when writing SSIDs to the Wi-Fi module, it may take up to 20 seconds to complete.

**Example:**

```
FUNCTION_BLOCK WifiSetSSID
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
                ER : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool;
                err : INT;
                buffer : STRING[50];
                index : INT;
        END_VAR

        buffer := 'SSIDvalue01123456ABCDEF';        (*SSID to write to Wi-Fi module*)
        index := 0;                                                         (*Set index / slot to 0*)

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Set_SSID(err, index, buffer);     (*keep calling / checking function until complete*)
                RES := INT_TO_DINT(result);
                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN
                        (*No Error, format print data for uart*)
                        EZ_FormatString(buffer, 'Write SSID Complete %d $N', index);

                        ELSE

                        (*Error, format print data for uart*)
                        EZ_FormatString(buffer, 'Write SSID Error %d Res: %d Er: %d $N', index, result, err);

                        END_IF;

                        (*Print result*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do  (*Print data to uart*)
                                ;
                        end_while;

                        complete := TRUE;               (*Finished*)
                        ER := INT_TO_DINT(err);
```

Continued Next Page------»

```
                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

## EZ_WiFi_Soft_Reset

### Summary:

The EZ_WiFi_Soft_Reset function is used peform a soft reset to the on-board Wi-Fi module. When changing Wi-Fi Module settings a power cycle or a soft reset using this function is required.

This function should be called / polled until it's status is compete (*Statvar* =2).

### Format:

*Statvar* := EZ_WiFi_Soft_Reset(*Error*);

### Arguments:

*Statvar*

Function return holding variable (INT). Returns the status of the function's activity. This must be converted to a DINT before it can exported from structured text to the ladder diagram.

**Statvar Values**

| | | |
|---|---|---|
| 2 | Complete | The Wi-Fi module completed processing the command. |
| 1 | DataAvailable | At least 1 AP record is available. The command is still processing and is not complete yet. |
| 0 | Processing | The Wi-Fi module is processing the command. |
| -1 | Locked | Communication to the Wi-Fi module is locked. Try again later. |
| -2 | Busy | The Wi-Fi module is processing another functions request.Try again later. |
| -3 | Invalid Parameter | The function / command used an invalid parameter. Check the calling function and it's parameters and make corrections. |
| -4 | Error State | The Wi-Fi module is in an error state (communications error from target to on-board Wi-Fi module). Wait and re-try again. If error persists or is constant, contact Divelbiss support. |
| -5 | Initializing | The Wi-Fi module is in being initialized. Try again later. |
| -6 | Not Supported | The installed W-Fi module is not supported or the Wi-Fi module has an internal problem and is undetectable. Contact Divelbiss support. |

*Error*

Variable to returned errors returned from Wi-Fi module (INT). These errors are only valid and should be checked after statvar = 2 (complete). Common expected errors are listed below. For other error codes, contact Divelbiss support.

**Error Values**

| | |
|---|---|
| 42 | Illegal Value |
| 44 | Number was expected but not received. |
| 48 | String was expected not not received. |
| 119 | WPA passphrase was too short, must be 8-63 characters. |
| 406 | Command failed because Wi-Fi module is currently busy. |

### Description:

The EZ_WiFi_Soft_Reset forces a soft-reset to the on-board Wi-Fi module. Whenever Wi-Fi module settings are changed, a Wi-Fi module power cycle or soft reset is required. The EZ_WiFi_Soft_Reset function is used to perform this soft reset as needed. The *Statvar* variable returns the status of the function per the list above. When *Statvar* = 2 (complete), the *Error* value must be checked for addtional errors (listed above as Error values).

Continued Next Page------»

**Example:**

```
FUNCTION_BLOCK WifiSoftReset
        VAR_INPUT
                Enable : bool;
        END_VAR
        VAR_OUTPUT
                Q : bool;
                RES : DINT;
        END_VAR
        VAR
                result : INT;
                complete: bool := FALSE;
                err : INT := 0;
                buffer : STRING[50];
        END_VAR

        IF ((Enable = TRUE) AND (complete = FALSE)) THEN

                result := EZ_Wifi_Soft_Reset(err); (*Reset Wi-Fi Module*)
                RES := INT_TO_DINT(result);

                IF (result <> 0) THEN

                        IF (result = 2 and err = 0) THEN
                        (*Successful soft reset*)
                        EZ_FormatString(buffer, 'Soft Reset Complete $N');

                        ELSE
                        (*Error while reading from Wifi module*)

                        EZ_FormatString(buffer, 'Soft Reset Error Res: %d Er: %d $N', result, err);

                        END_IF;

                        (*Write result to serial port*)
                        while EZ_UartWriteStr(FD_UART3, buffer) <= 0 do
                                        ;
                        end_while;

                complete := TRUE;
                END_IF;

        END_IF;

        IF (Enable = FALSE) THEN
                RES := 0;
                complete := FALSE;
        END_IF;

        Q := complete;

END_FUNCTION_BLOCK
```

# Appendix C
## Standard ST Function Reference

This chapter provides information on using Standard Structured Text Functions in EZ LADDER Toolkit.

## Chapter Contents

## Standard ST Functions

Standard structured text functions are functions used in structured text that are supported across all hardware targets. These typically include functions that convert variable types or modify a variable in some way.

Standard ST functions are found in the structured text editor in the Standard Functions tab (1st tab).

The variable definitions provided below are used in descriptions for the functions shown in this Appendix. More details on their definition and structured text can be found in **Chapter 26 - Structured Text**.

| Keyword | Data Type | Bit Size | Internal Data Type | Input/Output Data Type |
|---|---|---|---|---|
| BOOL | Boolean | 1 | X | X |
| SINT | Short Integer | 8 | X | |
| INT | Integer | 16 | X | |
| DINT | Double Integer | 32 | X | X |
| LINT | Long Integer | 64 | X | |
| USINT | Unsigned Short Integer | 8 | X | |
| UINT | Unsigned Integer | 16 | X | |
| UDINT | Unsigned Double Integer | 32 | X | |
| ULINT | Unsigned Long Integer | 64 | X | |
| REAL | Real Numbers | 32 | X | X |
| LREAL | Long Real Numbers | 64 | X | |
| BYTE | Bit String of length 8 | 8 | X | |
| WORD | Bit String of length 16 | 16 | X | |
| DWORD | Bit String of length 32 | 32 | X | |
| LWORD | Bit String of length 64 | 64 | X | |
| ARRAY [..] OF | Array of Type | --- | X | |
| STRING [# of bytes] | ASCII String of [x] bytes in length | --- | X | |

| Function Name | Function Details |
|---|---|
| **ABS** | **Format:**<br>*AbsNum* := ABS(*AnyNum*);<br>**Description:**<br>The ABS function returns the absolute as *AbsNum* of the provided (passed in) *AnyNum* value.<br>**Supported Variable Types:**<br>*AbsNum* and *AnyNum* supports variable types LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.<br>**Example:**<br>ABSPSI := ABS(PSI); |
| **ACOS** | **Format:**<br>*ACOSReal* := ACOS(*AnyReal*);<br>**Description:**<br>The ACOS function returns the Arcosine as *ACOSReal* of the provided (passed in) *Any-Real* value.<br>**Supported Variable Types:**<br>*ACOSReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= ACOS(PB); |
| **ASIN** | **Format:**<br>*ASINReal* := ASIN(*AnyReal*);<br>**Description:**<br>The ASIN function returns the Arcsine as *ASINReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*ASINReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= ASIN(PB); |
| **ATAN** | **Format:**<br>*ATANReal* := ATAN(*AnyReal*);<br>**Description:**<br>The ATAN function returns the Arctangent as *ATANReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*ATANReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= ATAN(PB); |
| **BYTE_TO_DINT** | **Format:**<br>*DINT* := BYTE_TO_DINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_DINT function converts the *AnyByte* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*DINT* supports DINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>ABC := BYTE_TO_DINT(XYZ); |

| Function Name | Function Details |
|---|---|
| **BYTE_TO_DWORD** | **Format:**<br>*DWORD* := BYTE_TO_DWORD(*AnyByte*);<br>**Description:**<br>The BYTE_TO_DWORD function converts the *AnyByte* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>MNO := BYTE_TO_DWORD(XYZ); |
| **BYTE_TO_INT** | **Format:**<br>*INT* := BYTE_TO_INT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_INT function converts the *AnyByte* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>DEF := BYTE_TO_INT(XYZ); |
| **BYTE_TO_LINT** | **Format:**<br>*LINT* := BYTE_TO_LINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_LINT function converts the *AnyByte* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>KLM := BYTE_TO_LINT(XYZ); |
| **BYTE_TO_LREAL** | **Format:**<br>*LREAL* := BYTE_TO_LREAL(*AnyByte*);<br>**Description:**<br>The BYTE_TO_LREAL function converts the *AnyByte* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *AnyByte* supports BYTE variables type.<br>**Example:**<br>GHI := BYTE_TO_LREAL(XYZ); |
| **BYTE_TO_LWORD** | **Format:**<br>*LWORD* := BYTE_TO_LWORD(*AnyByte*);<br>**Description:**<br>The BYTE_TO_LWORD function converts the *AnyByte* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>CDE := BYTE_TO_LWORD(XYZ); |

| Function Name | Function Details |
|---|---|
| **BYTE_TO_REAL** | **Format:**<br>*REAL* := BYTE_TO_REAL(*AnyByte*);<br>**Description:**<br>The BYTE_TO_REAL function converts the *AnyByte* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>PBX := BYTE_TO_REAL(XYZ); |
| **BYTE_TO_SINT** | **Format:**<br>*SINT* := BYTE_TO_SINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_SINT function converts the *AnyByte* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>RDS := BYTE_TO_SINT(XYZ); |
| **BYTE_TO_UDINT** | **Format:**<br>*UDINT* := BYTE_TO_UDINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_UDINT function converts the *AnyByte* (Passed in) value into a unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>BXD := BYTE_TO_UDINT(XYZ); |
| **BYTE_TO_UINT** | **Format:**<br>*UINT* := BYTE_TO_UINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_UINT function converts the *AnyByte* (Passed in) value into a unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>RSA := BYTE_TO_UINT(XYZ); |
| **BYTE_TO_ULINT** | **Format:**<br>*ULINT* := BYTE_TO_ULINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_ULINT function converts the *AnyByte* (Passed in) value into a unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>CBA := BYTE_TO_ULINT(XYZ); |

| Function Name | Function Details |
|---|---|
| **BYTE_TO_USINT** | **Format:**<br>*USINT* := BYTE_TO_USINT(*AnyByte*);<br>**Description:**<br>The BYTE_TO_ULSNT function converts the *AnyByte* (Passed in) value into a unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *AnyByte* supports BYTE variabl types.<br>**Example:**<br>DGA := BYTE_TO_USINT(XYZ); |
| **BYTE_TO_WORD** | **Format:**<br>*WORD* := BYTE_TO_WORD(*AnyByte*);<br>**Description:**<br>The BYTE_TO_WORD function converts the *AnyByte* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>LKT := BYTE_TO_WORD(XYZ); |
| **CONCAT** | **Format:**<br>*STRINGout* := CONCAT(*AnyStringIN1, AnyStringIN2*);<br>**Description:**<br>The CONCAT function concatenates (combines)  two strings input strings (*AnyStringIN1, AnyStringIN2*) into a single string and stores it in *STRINGout.*<br>**Supported Variable Types:**<br>*STRINGout, AnyStringIN1, AnyStringIN2* supports STRING variable type (must be de-clared large enough to handle string size).<br>**Example:**<br>STRCMB := CONTCAT(STR1, STR2); |
| **COS** | **Format:**<br>*COSReal* := COS(*AnyReal*);<br>**Description:**<br>The COS function returns the Cosine as *COSReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*COSReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= COS(PB); |
| **DELETE** | **Format:**<br>*STRINGout* := DELETE(*AnyStringIN1, INTLGTH, INTPSTN*);<br>**Description:**<br>The DELETE function deletes *INTLGTH* characters from *AnyStringIN1,* beginning at *INTPSTN* and stores the result in *STRINGout.*<br>**Supported Variable Types:**<br>*STRINGout, AnyStringIN1* supports STRING variable type. *INTLGTH, INTPSTN* support LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>STRRSLT := DELETE(STR1, LENGTH, POSITION); |

| Function Name | Function Details |
|---|---|
| **DINT_TO_BYTE** | **Format:**<br>*AnyByte* := DINT_TO_BYTE(*DINT*);<br>**Description:**<br>The DINT_TO_BYTE function converts the *DINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*DINT* supports DINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := DINT_TO_BYTE(ABC); |
| **DINT_TO_DWORD** | **Format:**<br>*DWORD* := DINT_TO_DWORD(*DINT*);<br>**Description:**<br>The DINT_TO_DWORD function converts the *DINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *DINT* supports DINT variable type.<br>**Example:**<br>MNO := DINT_TO_DWORD(ABC); |
| **DINT_TO_INT** | **Format:**<br>*INT* := DINT_TO_INT(*DINT*);<br>**Description:**<br>The DINT_TO_INT function converts the *DINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>DEF := DINT_TO_INT(ABC); |
| **DINT_TO_LINT** | **Format:**<br>*LINT* := DINT_TO_LINT(*DINT*);<br>**Description:**<br>The DINT_TO_LINT function converts the *DINT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>KLM := DINT_TO_LINT(ABC); |
| **DINT_TO_LREAL** | **Format:**<br>*LREAL* := DINT_TO_LREAL(*DINT*);<br>**Description:**<br>The DINT_TO_LREAL function converts the *DINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *DINT* supports DINT variable type.<br>**Example:**<br>GHI := DINT_TO_LREAL(ABC); |

| Function Name | Function Details |
|---|---|
| **DINT_TO_LWORD** | **Format:**<br>*LWORD* := DINT_TO_LWORD(*DINT*);<br>**Description:**<br>The DINT_TO_LWORD function converts the *DINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *DINT* supports DINT variable type.<br>**Example:**<br>CDE := DINT_TO_LWORD(ABC); |
| **DINT_TO_REAL** | **Format:**<br>*REAL* := DINT_TO_REAL(*DINT*);<br>**Description:**<br>The DINT_TO_REAL function converts the *DINT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *DINT* supports DINT variable type.<br>**Example:**<br>PBX := DINT_TO_REAL(ABC); |
| **DINT_TO_SINT** | **Format:**<br>*SINT* := DINT_TO_SINT(*DINT*);<br>**Description:**<br>The DINT_TO_SINT function converts the *DINT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>RDS := DINT_TO_SINT(ABC); |
| **DINT_TO_UDINT** | **Format:**<br>*UDINT* := DINT_TO_UDINT(*DINT*);<br>**Description:**<br>The DINT_TO_UDINT function converts the *DINT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>BXD := DINT_TO_UDINT(ABC); |
| **DINT_TO_UINT** | **Format:**<br>*UINT* := DINT_TO_UINT(*DINT*);<br>**Description:**<br>The DINT_TO_UINT function converts the *DINT* (Passed in) value into a unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>RSA := DINT_TO_UINT(ABC); |

| Function Name | Function Details |
|---|---|
| **DINT_TO_ULINT** | **Format:**<br>*ULINT* := DINT_TO_ULINT(*DINT*);<br>**Description:**<br>The DINT_TO_ULINT function converts the *DINT* (Passed in) value into a unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>CBA := DINT_TO_ULINT(ABC); |
| **DINT_TO_USINT** | **Format:**<br>*USINT* := DINT_TO_USINT(*DINT*);<br>**Description:**<br>The DINT_TO_USINT function converts the *DINT* (Passed in) value into a unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>DGA := DINT_TO_USINT(ABC); |
| **DINT_TO_WORD** | **Format:**<br>*WORD* := DINT_TO_WORD(*DINT*);<br>**Description:**<br>The DINT_TO_WORD function converts the *DINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *DINT* supports DINT variable type.<br>**Example:**<br>LKT := DINT_TO_WORD(ABC); |
| **DWORD_TO_BYTE** | **Format:**<br>*AnyByte* := DWORD_TO_BYTE(*DWORD*);<br>**Description:**<br>The DWORD_TO_BYTE function converts the *DWORD* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := DWORD_TO_BYTE(MNO); |
| **DWORD_TO_DINT** | **Format:**<br>*DINT* := DWORD_TO_DINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_DINT function converts the *DWORD* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := DWORD_TO_DINT(MNO); |

| Function Name | Function Details |
|---|---|
| **DWORD_TO_INT** | **Format:**<br>*INT* := DWORD_TO_INT(*DWORD*);<br>**Description:**<br>The DWORD_TO_INT function converts the *DWORD* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *INT* supports INT variable type.<br>**Example:**<br>ABC := DWORD_TO_INT(MNO); |
| **DWORD_TO_LINT** | **Format:**<br>*LINT* := DWORD_TO_LINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_LINT function converts the *DWORD* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *LINT* supports LINT variable type.<br>**Example:**<br>KLM := DWORD_TO_LINT(MNO); |
| **DWORD_TO_LREAL** | **Format:**<br>*LREAL* := DWORD_TO_LREAL(*DWORD*);<br>**Description:**<br>The DWORD_TO_LREAL function converts the *DWORD* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>GHI := DWORD_TO_LREAL(MNO); |
| **DWORD_TO_LWORD** | **Format:**<br>*LWORD* := DWORD_TO_LWORD(*DWORD*);<br>**Description:**<br>The DWORD_TO_LWORD function converts the *DWORD* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>CDE := DWORD_TO_LWORD(MNO); |
| **DWORD_TO_REAL** | **Format:**<br>*REAL* := DWORD_TO_REAL(*DWORD*);<br>**Description:**<br>The DWORD_TO_REAL function converts the *DWORD* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>PBX := DWORD_TO_REAL(MNO); |

| Function Name | Function Details |
|---|---|
| **DWORD_TO_SINT** | **Format:**<br>*SINT* := DWORD_TO_SINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_SINT function converts the *DWORD* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>RDS := DWORD_TO_SINT(MNO); |
| **DWORD_TO_UDINT** | **Format:**<br>*DINT* := DWORD_TO_UDINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_UDINT function converts the *DWORD* (Passed in) value into an un-signed double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>BXD := DWORD_TO_UDINT(MNO); |
| **DWORD_TO_UINT** | **Format:**<br>*UINT* := DWORD_TO_UINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_UINT function converts the *DWORD* (Passed in) value into an un-signed integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>RSA := DWORD_TO_UINT(MNO); |
| **DWORD_TO_ULINT** | **Format:**<br>*ULINT* := DWORD_TO_ULINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_ULINT function converts the *DWORD* (Passed in) value into an un-signed long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>CBA := DWORD_TO_ULINT(MNO); |
| **DWORD_TO_USINT** | **Format:**<br>*USINT* := DWORD_TO_USINT(*DWORD*);<br>**Description:**<br>The DWORD_TO_USINT function converts the *DWORD* (Passed in) value into an un-signed short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>DGA := DWORD_TO_USINT(MNO); |

| Function Name | Function Details |
|---|---|
| **DWORD_TO_WORD** | **Format:**<br>*WORD* := DWORD_TO_WORD(*DWORD*);<br>**Description:**<br>The DWORD_TO_WORD function converts the *DWORD* (Passed in) value into a WORD and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>LKT := DWORD_TO_WORD(MNO); |
| **EXP** | **Format:**<br>*EXPReal* := EXP(*AnyReal*);<br>**Description:**<br>The EXP function returns the natureal exponential as *EXPReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*EXPReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= EXP(PB); |
| **EXPT** | **Format:**<br>*EXPTNum* := EXPT(*AnyNum1, AnyNum2*);<br>**Description:**<br>The EXPT function returns the exponentation of the provided (passed in) *AnyNum1 and AnyNum2* values.<br>**Supported Variable Types:**<br>*EXPTNum*, *AnyNum1 and AnyNum2* supports variable types LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.<br>**Example:**<br>LMI:= EXPT(TUV, RXL); |
| **FIND** | **Format:**<br>*AnyInt* := FIND(*AnyStringIN1, AnyStringIN2*);<br>**Description:**<br>The FIND function searches for the *AnyStringIN2* provided inside the *AnyStringIN1* provided. It will return the location where *AnyStringIN2* begins as *AnyInt*. If the *AnyStringIN2* provided is not found, then it returns a zero (0).<br>**Supported Variable Types:**<br>*AnyStringIN1 and AnyStringIN2* supports STRING variable type. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>ABC := FIND(STR1, STR2); |
| **INSERT** | **Format:**<br>*STRINGout* := INSERT(*AnyStringIN1, AnyStringIN2, AnyInt*);<br>**Description:**<br>The INSERT function inserts *AnyStringIN1* into *AnyStringIN2* starting after the position provided as *AnyInt*.<br>**Supported Variable Types:**<br>*STRINGout, AnyStringIN1 and AnyStringIN2* supports STRING variable type. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>STRRSLT := INSERT(STR1, STR2, ABC); |

| Function Name | Function Details |
|---|---|
| **INT_TO_BYTE** | **Format:**<br>*AnyByte* := INT_TO_BYTE(*INT*);<br>**Description:**<br>The INT_TO_BYTE function converts the *INT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := INT_TO_BYTE(DEF); |
| **INT_TO_DINT** | **Format:**<br>*DINT* := INT_TO_DINT(*INT*);<br>**Description:**<br>The INT_TO_DINT function converts the *INT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := INT_TO_DINT(DEF); |
| **INT_TO_DWORD** | **Format:**<br>*DWORD* := INT_TO_DWORD(*INT*);<br>**Description:**<br>The INT_TO_DWORD function converts the *INT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *INT* supports INT variable type.<br>**Example:**<br>MNO := INT_TO_DWORD(DEF); |
| **INT_TO_LINT** | **Format:**<br>*LINT* := INT_TO_LINT(*INT*);<br>**Description:**<br>The INT_TO_LINT function converts the *INT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *INT* supports INT variable type.<br>**Example:**<br>KLM := INT_TO_LINT(DEF); |
| **INT_TO_LREAL** | **Format:**<br>*LREAL* := INT_TO_LREAL(*INT*);<br>**Description:**<br>The INT_TO_LREAL function converts the *INT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *INT* supports INT variable type.<br>**Example:**<br>GHI := INT_TO_LREAL(DEF); |

| Function Name | Function Details |
|---|---|
| **INT_TO_LWORD** | **Format:**<br>*LWORD* := INT_TO_LWORD(*INT*);<br>**Description:**<br>The INT_TO_LWORD function converts the *INT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *INT* supports INT variable type.<br>**Example:**<br>CDE := INT_TO_LWORD(DEF); |
| **INT_TO_REAL** | **Format:**<br>*REAL* := INT_TO_REAL(*INT*);<br>**Description:**<br>The INT_TO_REAL function converts the *INT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *INT* supports INT variable type.<br>**Example:**<br>PBX := INT_TO_REAL(DEF); |
| **INT_TO_SINT** | **Format:**<br>*SINT* := INT_TO_SINT(*INT*);<br>**Description:**<br>The INT_TO_SINT function converts the *INT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *INT* supports INT variable type.<br>**Example:**<br>RDS := DINT_TO_SINT(DEF); |
| **INT_TO_UDINT** | **Format:**<br>*UDINT* := INT_TO_UDINT(*INT*);<br>**Description:**<br>The INT_TO_UDINT function converts the *INT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *INT* supports INT variable type.<br>**Example:**<br>BXD := INT_TO_UDINT(DEF); |
| **INT_TO_UINT** | **Format:**<br>*UINT* := INT_TO_UINT(*INT*);<br>**Description:**<br>The INT_TO_UINT function converts the *INT* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *INT* supports INT variable type.<br>**Example:**<br>RSA := INT_TO_UINT(DEF); |

| Function Name | Function Details |
|---|---|
| **INT_TO_ULINT** | **Format:**<br>*ULINT* := INT_TO_ULINT(*INT*);<br>**Description:**<br>The INT_TO_ULINT function converts the *INT* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *INT* supports INT variable type.<br>**Example:**<br>CBA := INT_TO_ULINT(DEF); |
| **INT_TO_USINT** | **Format:**<br>*USINT* := INT_TO_USINT(*INT*);<br>**Description:**<br>The INT_TO_USINT function converts the *INT* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *INT* supports INT variable type.<br>**Example:**<br>DGA := INT_TO_USINT(DEF); |
| **INT_TO_WORD** | **Format:**<br>*WORD* := INT_TO_WORD(*INT*);<br>**Description:**<br>The INT_TO_WORD function converts the *INT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *INT* supports INT variable type.<br>**Example:**<br>LKT := INT_TO_WORD(DEF); |
| **LEFT** | **Format:**<br>*STRINGout* := LEFT(*AnyStringIN, AnyInt*);<br>**Description:**<br>The LEFT function returns the left most *AnyInt* number of characters of *AnyStringIN* and stores them in *STRINGout.*<br>**Supported Variable Types:**<br>*STRINGout and AnyStringIN* supports STRING variable type. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>STRRSLT := LEFT(STR1, ABC); |
| **LEN** | **Format:**<br>*AnyInt* := LEN(*AnyStringIN*);<br>**Description:**<br>The LEN function returns the number of characters (length) of *AnyStringIN* and stores the result in *AnyInt.*<br>**Supported Variable Types:**<br>*AnyStringIN* supports STRING variable type. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>ABC := LEN(STR1); |

| Function Name | Function Details |
|---|---|
| LIMIT | **Format:**<br>*AnyNumLimit* := LIMIT(*AnyNumMin, AnyNum, AnyNumMax*);<br>**Description:**<br>The LIMIT function limits the *AnyNum* passed value between the *AnyNumMin* (minimum limit) and the *AnyNumMax* (maximum limit). When the *AnyNum* value is between the two values, the output of the function will be equal to *AnyNum* (stored in *AnyNumLimit*). When  less than or equal to the minimum or greater than or equal to the maximum, the output of the function (stored in *AnyNumLimit*) will be equal to either the minimum or maximum, respectively.<br>**Supported Variable Types:**<br>*AnyNumLimit*, *AnyNumMin*, *AnyNum* and *AnyNumMax* supports variable types LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.<br>**Example:**<br>LMTVAL := LIMIT(MN,VIN,MX); |
| LINT_TO_BYTE | **Format:**<br>*AnyByte* := LINT_TO_BYTE(*LINT*);<br>**Description:**<br>The LINT_TO_BYTE function converts the *LINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := LINT_TO_BYTE(KLM); |
| LINT_TO_DINT | **Format:**<br>*DINT* := LINT_TO_DINT(*LINT*);<br>**Description:**<br>The LINT_TO_DINT function converts the *LINT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type. *DINT* supports DINT variable type.<br>**Example:**<br>ABC := LINT_TO_DINT(KLM); |
| LINT_TO_DWORD | **Format:**<br>*DWORD* := LINT_TO_DWORD(*LINT*);<br>**Description:**<br>The LINT_TO_DWORD function converts the *LINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *LINT* supports LINT variable type.<br>**Example:**<br>MNO := LINT_TO_DWORD(KLM); |

| Function Name | Function Details |
|---|---|
| **LINT_TO_INT** | **Format:**<br>*INT* := LINT_TO_INT(*LINT*);<br>**Description:**<br>The LINT_TO_INT function converts the *LINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *INT* supports INT variable type.<br>**Example:**<br>DEF := LINT_TO_INT(KLM); |
| **LINT_TO_LREAL** | **Format:**<br>*LREAL* := LINT_TO_LREAL(*LINT*);<br>**Description:**<br>The LINT_TO_LREAL function converts the *LINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *LINT* supports LINT variable type.<br>**Example:**<br>GHI := LINT_TO_LREAL(KLM); |
| **LINT_TO_LWORD** | **Format:**<br>*LWORD* := LINT_TO_LWORD(*LINT*);<br>**Description:**<br>The LINT_TO_LWORD function converts the *LINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *LINT* supports LINT variable type.<br>**Example:**<br>CDE := LINT_TO_LWORD(KLM); |
| **LINT_TO_LREAL** | **Format:**<br>*LREAL* := LINT_TO_LREAL(*LINT*);<br>**Description:**<br>The LINT_TO_LREAL function converts the *LINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *LINT* supports LINT variable type.<br>**Example:**<br>GHI := LINT_TO_LREAL(KLM); |
| **LINT_TO_SINT** | **Format:**<br>*SINT* := LINT_TO_SINT(*LINT*);<br>**Description:**<br>The LINT_TO_SINT function converts the *LINT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *LINT* supports LINT variable type.<br>**Example:**<br>RDS := LINT_TO_SINT(KLM); |

| Function Name | Function Details |
|---|---|
| **LINT_TO_UDINT** | **Format:**<br>*UDINT* := LINT_TO_UDINT(*LINT*);<br>**Description:**<br>The LINT_TO_UDINT function converts the *LINT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *LINT* supports LINT variable type.<br>**Example:**<br>BXD := LINT_TO_UDINT(KLM); |
| **LINT_TO_UINT** | **Format:**<br>*UINT* := LINT_TO_UINT(*LINT*);<br>**Description:**<br>The LINT_TO_UINT function converts the *LINT* (Passed in) value into an unsigned inte-ger and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *LINT* supports LINT variable type.<br>**Example:**<br>RSA := LINT_TO_UINT(KLM); |
| **LINT_TO_ULINT** | **Format:**<br>*ULINT* := LINT_TO_ULINT(*LINT*);<br>**Description:**<br>The LINT_TO_ULINT function converts the *LINT* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *LINT* supports LINT variable type.<br>**Example:**<br>CBA := LINT_TO_ULINT(KLM); |
| **LINT_TO_USINT** | **Format:**<br>*USINT* := LINT_TO_USINT(*LINT*);<br>**Description:**<br>The LINT_TO_USINT function converts the *LINT* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *LINT* supports LINT variable type.<br>**Example:**<br>DGA := LINT_TO_USINT(KLM); |
| **LINT_TO_WORD** | **Format:**<br>*WORD* := LINT_TO_WORD(*LINT*);<br>**Description:**<br>The LINT_TO_WORD function converts the *LINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *LINT* supports LINT variable type.<br>**Example:**<br>LKT := LINT_TO_WORD(KLM); |

| Function Name | Function Details |
|---|---|
| **LN** | **Format:**<br>*LNReal* := LN(*AnyReal*);<br>**Description:**<br>The LN function returns the natural logarithm of the passed in variable *AnyReal* and stores the result in *LNReal.*<br>**Supported Variable Types:**<br>*LNReal* and *AnyReal* supports variable types LREAL, REAL.<br>**Example:**<br>LNRSLT:= LN(PBX); |
| **LOG** | **Format:**<br>*LGReal* := LOG(*AnyReal*);<br>**Description:**<br>The LOG function returns the logarith base 10 of the passed in variable *AnyReal* and stores the result in *LGReal.*<br>**Supported Variable Types:**<br>*LGReal* and *AnyReal* supports variable types LREAL, REAL.<br>**Example:**<br>LGRSLT:= LOG(PBX); |
| **LREAL_TO_BYTE** | **Format:**<br>*AnyByte* := LREAL_TO_BYTE(*LREAL*);<br>**Description:**<br>The LREAL_TO_BYTE function converts the *LREAL* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := LREAL_TO_BYTE(GHI); |
| **LREAL_TO_DINT** | **Format:**<br>*DINT* := LREAL_TO_DINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_DINT function converts the *LREAL* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := LREAL_TO_DINT(GHI); |
| **LREAL_TO_DWORD** | **Format:**<br>*DWORD* := LREAL_TO_DWORD(*LREAL*);<br>**Description:**<br>The LREAL_TO_DWORD function converts the *LREAL* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>MNO := LREAL_TO_DWORD(GHI); |

| Function Name | Function Details |
|---|---|
| **LREAL_TO_INT** | **Format:**<br>*INT* := LREAL_TO_INT(*LREAL*);<br>**Description:**<br>The LREAL_TO_INT function converts the *LREAL* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *INT* supports INT variable type.<br>**Example:**<br>DEF := LREAL_TO_INT(GHI); |
| **LREAL_TO_LINT** | **Format:**<br>*LINT* := LREAL_TO_LINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_LINT function converts the *LREAL* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *LINT* supports LINT variable type.<br>**Example:**<br>KLM := LREAL_TO_LINT(GHI); |
| **LREAL_TO_LWORD** | **Format:**<br>*LWORD* := LREAL_TO_LWORD(*LREAL*);<br>**Description:**<br>The LREAL_TO_LWORD function converts the *LREAL* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>CDE := LREAL_TO_LWORD(GHI); |
| **LREAL_TO_SINT** | **Format:**<br>*SINT* := LREAL_TO_SINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_SINT function converts the *LREAL* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>RDS := LREAL_TO_SINT(GHI); |
| **LREAL_TO_UDINT** | **Format:**<br>*UDINT* := LREAL_TO_UDINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_UDINT function converts the *LREAL* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>BXD := LREAL_TO_UDINT(GHI); |

| Function Name | Function Details |
|---|---|
| **LREAL_TO_UINT** | **Format:**<br>*UINT* := LREAL_TO_UINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_UINT function converts the *LREAL* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>RSA := LREAL_TO_UINT(GHI); |
| **LREAL_TO_ULINT** | **Format:**<br>*ULINT* := LREAL_TO_ULINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_ULINT function converts the *LREAL* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>CBA := LREAL_TO_ULINT(GHI); |
| **LREAL_TO_USINT** | **Format:**<br>*USINT* := LREAL_TO_USINT(*LREAL*);<br>**Description:**<br>The LREAL_TO_USINT function converts the *LREAL* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>DGA := LREAL_TO_USINT(GHI); |
| **LREAL_TO_WORD** | **Format:**<br>*WORD* := LREAL_TO_WORD(*LREAL*);<br>**Description:**<br>The LREAL_TO_WORD function converts the *LREAL* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *LREAL* supports LREAL variable type.<br>**Example:**<br>LKT := LREAL_TO_WORD(GHI); |
| **LSB_DINT_TO_ARRAY** | **Format:**<br>LSB_DINT_TO_ARRAY(*ArrayAnyType, DINToffset, DINTval*);<br>**Description:**<br>The LSB_DINT_TO_ARRAY function converts the *DINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* and *DINTval* supports DINT variable type.<br>**Example:**<br>LSB_DINT_TO_ARRAY(TXBUFF, OFST, VLIN); |

| Function Name | Function Details |
|---|---|
| **LSB_DWORD_TO_ARRAY** | **Format:**<br>LSB_DWORD_TO_ARRAY(*ArrayAnyType, DINToffset, DWORDval*);<br>**Description:**<br>The LSB_DWORD_TO_ARRAY function converts the *DWORDval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *DWORDval* supports DWORD variable type.<br>**Example:**<br>LSB_DWORD_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_INT_TO_ARRAY** | **Format:**<br>LSB_INT_TO_ARRAY(*ArrayAnyType, DINToffset, INTval*);<br>**Description:**<br>The LSB_INT_TO_ARRAY function converts the *INTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *INTval* supports INT variable type.<br>**Example:**<br>LSB_INT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_LINT_TO_ARRAY** | **Format:**<br>LSB_LINT_TO_ARRAY(*ArrayAnyType, DINToffset, LINTval*);<br>**Description:**<br>The LSB_LINT_TO_ARRAY function converts the *LINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *LINTval* supports LINT variable type.<br>**Example:**<br>LSB_LINT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_LREAL_TO_ARRAY** | **Format:**<br>LSB_LREAL_TO_ARRAY(*ArrayAnyType, DINToffset, LREALval*);<br>**Description:**<br>The LSB_LREAL_TO_ARRAY function converts the *LREALval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *LREALval* supports LREAL variable type.<br>**Example:**<br>LSB_LREAL_TO_ARRAY(TXBUFF, OFST, VLIN); |

| Function Name | Function Details |
|---|---|
| **LSB_LWORD_TO_ARRAY** | **Format:**<br>LSB_LWORD_TO_ARRAY(*ArrayAnyType, DINToffset, LWORDval*);<br>**Description:**<br>The LSB_LWORD_TO_ARRAY function converts the *LWORDval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *LWORDval* supports LWORD variable type.<br>**Example:**<br>LSB_LWORD_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_REAL_TO_ARRAY** | **Format:**<br>LSB_REAL_TO_ARRAY(*ArrayAnyType, DINToffset, REALval*);<br>**Description:**<br>The LSB_REAL_TO_ARRAY function converts the *REALval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *REALval* supports REAL variable type.<br>**Example:**<br>LSB_REAL_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_UDINT_TO_ARRAY** | **Format:**<br>LSB_UDINT_TO_ARRAY(*ArrayAnyType, DINToffset, UDINTval*);<br>**Description:**<br>The LSB_UDINT_TO_ARRAY function converts the *UDINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *UDINTval* supports UDINT variable type.<br>**Example:**<br>LSB_UDINT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_UINT_TO_ARRAY** | **Format:**<br>LSB_UINT_TO_ARRAY(*ArrayAnyType, DINToffset, UINTval*);<br>**Description:**<br>The LSB_UINT_TO_ARRAY function converts the *UINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *UINTval* supports UINT variable type.<br>**Example:**<br>LSB_UINT_TO_ARRAY(TXBUFF, OFST, VLIN); |

| Function Name | Function Details |
|---|---|
| **LSB_ULINT_TO_ARRAY** | **Format:**<br>LSB_ULINT_TO_ARRAY(*ArrayAnyType, DINToffset, ULINTval*);<br>**Description:**<br>The LSB_ULINT_TO_ARRAY function converts the *ULINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *ULINTval* supports ULINT variable type.<br>**Example:**<br>LSB_ULINT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LSB_WORD_TO_ARRAY** | **Format:**<br>LSB_WORD_TO_ARRAY(*ArrayAnyType, DINToffset, WORDval*);<br>**Description:**<br>The LSB_WORD_TO_ARRAY function converts the *WORDval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in LSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *WORDval* supports WORD variable type.<br>**Example:**<br>LSB_WORD_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **LWORD_TO_BYTE** | **Format:**<br>*AnyByte* := LWORD_TO_BYTE(*LWORD*);<br>**Description:**<br>The LWORD_TO_BYTE function converts the *LWORD* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type. *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := LWORD_TO_BYTE(CDE); |
| **LWORD_TO_DINT** | **Format:**<br>*DINT* := LWORD_TO_DINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_DINT function converts the *LWORD* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type. *DINT* supports DINT variable type.<br>**Example:**<br>ABC := LWORD_TO_DINT(CDE); |
| **LWORD_TO_DWORD** | **Format:**<br>*DWORD* := LWORD_TO_DWORD(*LWORD*);<br>**Description:**<br>The LWORD_TO_DWORD function converts the *LWORD* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *DWORD* supports DWORD variable type.<br>**Example:**<br>MNO := LWORD_TO_DWORD(CDE); |

| Function Name | Function Details |
|---|---|
| LWORD_TO_INT | **Format:**<br>*INT* := LWORD_TO_INT(*LWORD*);<br>**Description:**<br>The LWORD_TO_INT function converts the *LWORD* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type. *INT* supports INT variable type.<br>**Example:**<br>DEF := LWORD_TO_INT(CDE); |
| LWORD_TO_LINT | **Format:**<br>*LINT* := LWORD_TO_LINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_LINT function converts the *LWORD* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type,.*LINT* supports LINT variable type.<br>**Example:**<br>KLM := LWORD_TO_LINT(CDE); |
| LWORD_TO_LREAL | **Format:**<br>*LREAL* := LWORD_TO_LREAL(*LWORD*);<br>**Description:**<br>The LWORD_TO_LREAL function converts the *LWORD* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>GHI := LWORD_TO_LREAL(CDE); |
| LWORD_TO_REAL | **Format:**<br>*REAL* := LWORD_TO_REAL(*LWORD*);<br>**Description:**<br>The LWORD_TO_REAL function converts the *LWORD* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>PBX := LWORD_TO_REAL(CDE); |
| LWORD_TO_SINT | **Format:**<br>*SINT* := LWORD_TO_SINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_SINT function converts the *LWORD* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>RDS := LWORD_TO_SINT(CDE); |

| Function Name | Function Details |
|---|---|
| **LWORD_TO_UDINT** | **Format:**<br>*UDINT* := LWORD_TO_UDINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_UDINT function converts the *LWORD* (Passed in) value into an un-signed double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>BXD := LWORD_TO_UDINT(CDE); |
| **LWORD_TO_UINT** | **Format:**<br>*UINT* := LWORD_TO_UINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_UINT function converts the *LWORD* (Passed in) value into an un-signed integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>RSA := LWORD_TO_UINT(CDE); |
| **LWORD_TO_ULINT** | **Format:**<br>*ULINT* := LWORD_TO_ULINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_ULINT function converts the *LWORD* (Passed in) value into an un-signed long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *LWORD* supports LWORD variable type.<br>**Example:**<br>CBA := LWORD_TO_ULINT(CDE); |
| **LWORD_TO_USINT** | **Format:**<br>*USINT* := LWORD_TO_USINT(*LWORD*);<br>**Description:**<br>The LWORD_TO_USINT function converts the *LWORD* (Passed in) value into an un-signed short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type.  *LWORD* supports LWORD variable type.<br>**Example:**<br>DGA := LWORD_TO_USINT(CDE); |
| **LWORD_TO_WORD** | **Format:**<br>*WORD* := LWORD_TO_WORD(*LWORD*);<br>**Description:**<br>The LWORD_TO_WORD function converts the *LWORD* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type.  *LWORD* supports LWORD variable type.<br>**Example:**<br>LKT := LWORD_TO_WORD(CDE); |

| Function Name | Function Details |
|---|---|
| **MAX** | **Format:**<br>*AnyNumMax* := MAX(*AnyNumIn1, AnyNumIn2....AnyNuminX*);<br>**Description:**<br>The MAX function evaluates all the number variables passed into it (*AnyNumIn1, AnyNu-mIn2....AnyNuminX*) and returns the maximum (largest) of them and stores it in *AnyNum-Max.* The number of variables to input must be a minimum of two, but can have more.<br>**Supported Variable Types:**<br>*AnyNumMax*, *AnyNumIn1*, *AnyNumIn2* and *AnyNuminX* supports variable types LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.<br>**Example:**<br>RTNABC:= MAX(V1,V2,V3,V4); |
| **MID** | **Format:**<br>*STRINGout* := MID(*AnyStringIN, AnyIntLgth, AnyIntPos*);<br>**Description:**<br>The MID function searches the *AnyStringIN* string and returns the number of charac-ters (length) specified in *AnyIntLgth*, starting at *AnyIntPos* character of the string. The returned string is stored in *STRINGout.*<br>**Supported Variable Types:**<br>*STRINGout and AnyStringIN1* supports STRING variable type. *AnyIntLgth* and *AnyInt-Pos* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>STRRSLT := MID(STR1, A, B); |
| **MIN** | **Format:**<br>*AnyNumMin* := MIN(*AnyNumIn1, AnyNumIn2....AnyNuminX*);<br>**Description:**<br>The MIN function evaluates all the number variables passed into it (*AnyNumIn1, Any-NumIn2....AnyNuminX*) and returns the minimum (smallest) of them and stores it in *AnyNumMin.* The number of variables to input must be a minimum of two, but can have more.<br>**Supported Variable Types:**<br>*AnyNumMin*, *AnyNumIn1*, *AnyNumIn2* and *AnyNuminX* supports variable types LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.<br>**Example:**<br>RTNABC:= MIN(V1,V2,V3,V4); |
| **MSB_DINT_TO_ARRAY** | **Format:**<br>MSB_DINT_TO_ARRAY(*ArrayAnyType, DINToffset, DINTval*);<br>**Description:**<br>The MSB_DINT_TO_ARRAY function converts the *DINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* and *DINTval* supports DINT variable type.<br>**Example:**<br>MSB_DINT_TO_ARRAY(TXBUFF, OFST, VLIN); |

| Function Name | Function Details |
|---|---|
| **MSB_DWORD_TO_ARRAY** | **Format:**<br>MSB_DWORD_TO_ARRAY(*ArrayAnyType, DINToffset, DWORDval*);<br>**Description:**<br>The MSB_DWORD_TO_ARRAY function converts the *DWORDval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *DWORDval* supports DWORD variable type.<br>**Example:**<br>MSB_DWORD_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_INT_TO_ARRAY** | **Format:**<br>MSB_INT_TO_ARRAY(*ArrayAnyType, DINToffset, INTval*);<br>**Description:**<br>The MSB_INT_TO_ARRAY function converts the *INTval* and stores it into an array (*Ar-rayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *INTval* supports INT variable type.<br>**Example:**<br>MSB_INT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_LINT_TO_ARRAY** | **Format:**<br>MSB_LINT_TO_ARRAY(*ArrayAnyType, DINToffset, LINTval*);<br>**Description:**<br>The MSB_LINT_TO_ARRAY function converts the *LINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *LINTval* supports LINT variable type.<br>**Example:**<br>MSB_LINT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_LREAL_TO_ARRAY** | **Format:**<br>MSB_LREAL_TO_ARRAY(*ArrayAnyType, DINToffset, LREALval*);<br>**Description:**<br>The MSB_LREAL_TO_ARRAY function converts the *LREALval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *LREALval* supports LREAL variable type.<br>**Example:**<br>MSB_LREAL_TO_ARRAY(TXBUFF, OFST, VLIN); |

| Function Name | Function Details |
|---|---|
| **MSB_LWORD_TO_ARRAY** | **Format:**<br>MSB_LWORD_TO_ARRAY(*ArrayAnyType, DINToffset, LWORDval*);<br>**Description:**<br>The MSB_LWORD_TO_ARRAY function converts the *LWORDval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *LWORDval* supports LWORD variable type.<br>**Example:**<br>MSB_LWORD_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_REAL_TO_ARRAY** | **Format:**<br>MSB_REAL_TO_ARRAY(*ArrayAnyType, DINToffset, REALval*);<br>**Description:**<br>The MSB_REAL_TO_ARRAY function converts the *REALval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *REALval* supports REAL variable type.<br>**Example:**<br>MSB_REAL_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_UDINT_TO_ARRAY** | **Format:**<br>MSB_UDINT_TO_ARRAY(*ArrayAnyType, DINToffset, UDINTval*);<br>**Description:**<br>The MSB_UDINT_TO_ARRAY function converts the *UDINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *UDINTval* supports UDINT variable type.<br>**Example:**<br>MSB_UDINT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_UINT_TO_ARRAY** | **Format:**<br>MSB_UINT_TO_ARRAY(*ArrayAnyType, DINToffset, UINTval*);<br>**Description:**<br>The MSB_UINT_TO_ARRAY function converts the *UINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *UINTval* supports UINT variable type.<br>**Example:**<br>MSB_UINT_TO_ARRAY(TXBUFF, OFST, VLIN); |

| Function Name | Function Details |
|---|---|
| **MSB_ULINT_TO_ARRAY** | **Format:**<br>MSB_ULINT_TO_ARRAY(*ArrayAnyType, DINToffset, ULINTval*);<br>**Description:**<br>The MSB_ULINT_TO_ARRAY function converts the *ULINTval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *ULINTval* supports ULINT variable type.<br>**Example:**<br>MSB_ULINT_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MSB_WORD_TO_ARRAY** | **Format:**<br>MSB_WORD_TO_ARRAY(*ArrayAnyType, DINToffset, WORDval*);<br>**Description:**<br>The MSB_WORD_TO_ARRAY function converts the *WORDval* and stores it into an array (*ArrayAnyType*) beginning at the offset location specified with *DINToffset.* The values are converted and stored into the array in MSB order.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type, *DINToffset* supports DINT variable type. *WORDval* supports WORD variable type.<br>**Example:**<br>MSB_WORD_TO_ARRAY(TXBUFF, OFST, VLIN); |
| **MUX** | **Format:**<br>*AnyNumOut* := MUX(*AnyIntSelect,AnyNumIn1, AnyNumIn2....AnyNumInX*);<br>**Description:**<br>The MUX function acts as a selector based on the *AnyIntSelect* variable (passed in). This value selects from the other passed in variables (*AnyNumIn1, AnyNumIn2....AnyNumInX*) and returns the selected one and stores it in *AnyNumOut.*<br>**Supported Variable Types:**<br>*AnyNumOut*, *AnyNumIn1, AnyNumIn2* and *AnyNumInX* supports variable types LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT. *AnyIntSelect* supports variable types LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.<br>**Example:**<br>RTNABC:= MUX(K,V1,V2,V3); |
| **REAL_TO_BYTE** | **Format:**<br>*AnyByte* := REAL_TO_BYTE(*REAL*);<br>**Description:**<br>The REAL_TO_BYTE function converts the *REAL* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type. *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := REAL_TO_BYTE(PBX); |

| Function Name | Function Details |
|---|---|
| REAL_TO_DINT | **Format:**<br>*DINT* := REAL_TO_DINT(*REAL*);<br>**Description:**<br>The REAL_TO_DINT function converts the *REAL* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type. *DINT* supports DINT variable type.<br>**Example:**<br>ABC := REAL_TO_DINT(PBX); |
| REAL_TO_DWORD | **Format:**<br>*DWORD* := REAL_TO_DWORD(*REAL*);<br>**Description:**<br>The REAL_TO_DWORD function converts the *REAL* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type. *REAL* supports REAL variable type.<br>**Example:**<br>MNO := REAL_TO_DWORD(PBX); |
| REAL_TO_INT | **Format:**<br>*INT* := REAL_TO_INT(*REAL*);<br>**Description:**<br>The REAL_TO_INT function converts the *REAL* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type. *INT* supports INT variable type.<br>**Example:**<br>DEF := REAL_TO_INT(PBX); |
| REAL_TO_LINT | **Format:**<br>*LINT* := REAL_TO_LINT(*REAL*);<br>**Description:**<br>The REAL_TO_LINT function converts the *REAL* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type. *LINT* supports LINT variable type.<br>**Example:**<br>KLM := REAL_TO_LINT(PBX); |
| REAL_TO_LWORD | **Format:**<br>*LWORD* := REAL_TO_LWORD(*REAL*);<br>**Description:**<br>The REAL_TO_LWORD function converts the *REAL* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports the LWORD variable type. *REAL* supports REAL variable type.<br>**Example:**<br>CDE := REAL_TO_LWORD(PBX); |

| Function Name | Function Details |
|---|---|
| REAL_TO_SINT | **Format:**<br>*SINT* := REAL_TO_SINT(*REAL*);<br>**Description:**<br>The REAL_TO_SINT function converts the *REAL* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *REAL* supports REAL variable type.<br>**Example:**<br>RDS := REAL_TO_SINT(PBX); |
| REAL_TO_UDINT | **Format:**<br>*UDINT* := REAL_TO_UDINT(*REAL*);<br>**Description:**<br>The REAL_TO_UDINT function converts the *REAL* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type. *REAL* supports REAL variable type.<br>**Example:**<br>BXD := REAL_TO_UDINT(PBX); |
| REAL_TO_UINT | **Format:**<br>*UINT* := REAL_TO_UINT(*REAL*);<br>**Description:**<br>The REAL_TO_UINT function converts the *REAL* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *REAL* supports REAL variable type.<br>**Example:**<br>RSA := REAL_TO_UINT(PBX); |
| REAL_TO_ULINT | **Format:**<br>*ULINT* := REAL_TO_ULINT(*REAL*);<br>**Description:**<br>The REAL_TO_ULINT function converts the *REAL* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *REAL* supports REAL variable type.<br>**Example:**<br>CBA := REAL_TO_ULINT(PBX); |
| REAL_TO_USINT | **Format:**<br>*USINT* := REAL_TO_USINT(*REAL*);<br>**Description:**<br>The REAL_TO_USINT function converts the *REAL* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *REAL* supports REAL variable type.<br>**Example:**<br>DGA := REAL_TO_USINT(PBX); |

| Function Name | Function Details |
|---|---|
| **REAL_TO_WORD** | **Format:**<br>*WORD* := REAL_TO_WORD(*REAL*);<br>**Description:**<br>The REAL_TO_WORD function converts the *REAL* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *REAL* supports REAL variable type.<br>**Example:**<br>LKT := REAL_TO_WORD(PBX); |
| **REPLACE** | **Format:**<br>*STRINGout* := REPLACE(*AnyStringIN1, AnyStringIN2, AnyIntLgth, AnyIntPos*);<br>**Description:**<br>The REPLACE function replaces characters of the *AnyStringIN1* string with the *Any-StringIN2* ; replacing the number of characters (length) in *AnyStringIN1* specified in *AnyIntLgth*, starting at *AnyIntPos* character of the string. The returned string is stored in *STRINGout. STRINGout* must be large enough to handle the combined string. Addition-ally, the length and position must be within range for a replace to occur or the result returned will be the original *AnyStringIN1* string.<br>**Supported Variable Types:**<br>*STRINGout, AnyStringIN1 and AnyStringIN2* supports STRING variable type. *AnyIntLgth* and *AnyIntPos* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>STRRSLT := REPLACE(STR1, STR2 ,A, B); |
| **RIGHT** | **Format:**<br>*STRINGout* := RIGHT(*AnyStringIN, AnyInt*);<br>**Description:**<br>The RIGHT function returns the right most *AnyInt* number of characters of *AnyStringIN* and stores them in *STRINGout.*<br>**Supported Variable Types:**<br>*STRINGout and AnyStringIN* supports STRING variable type. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>STRRSLT := RIGHT(STR1, ABC); |
| **ROL** | **Format:**<br>*AnyBitOut* := ROL(*AnyBitIn, AnyInt*);<br>**Description:**<br>The ROL function left-rotates the *AnyBitIn* by *AnyInt* bits (Circular). The result is stored in *AnyBitOut.*<br>**Supported Variable Types:**<br>*AnyBitIn* and *AnyBitOut* supports LWORD, DWORD, WORD, BYTE and BOOL vari-able types. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>LKT := ROL(QRS, ABC); |

| Function Name | Function Details |
|---|---|
| **ROR** | **Format:**<br>*AnyBitOut* := ROR(*AnyBitIn, AnyInt*);<br>**Description:**<br>The ROR function right-rotates the *AnyBitIn* by *AnyInt* bits (Circular). The result is stored in *AnyBitOut.*<br>**Supported Variable Types:**<br>*AnyBitIn* and *AnyBitOut* support LWORD, DWORD, WORD, BYTE and BOOL variable types. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>LKT := ROR(QRS, ABC); |
| **SEL** | **Format:**<br>*AnyVarOut* := SEL(*BOOLSelect, AnyVarIn1, AnyVarIn2*);<br>**Description:**<br>The SEL selects and returns the selection between *AnyVarIn1* and *AnyVarIn2.* When sel is 0 (false), then *AnyVarOut* equals *AnyVarIn1*. When sel is 1 (true), then *AnyVarOut* equals *AnyVarIn2*.<br>**Supported Variable Types:**<br>*AnyVarOut, AnyVarIn1* and *AnyVarIn2* support all variable types.  *BOOLSelect* support BOOL variable type.<br>**Example:**<br>PBX := SEL(A, QRS, TUV); |
| **SHL** | **Format:**<br>*AnyBitOut* := SHL(*AnyBitIn, AnyInt*);<br>**Description:**<br>The SHL function left-shifts the *AnyBitIn* by *AnyInt* bits (and fills zero on the right). The result is stored in *AnyBitOut.*<br>**Supported Variable Types:**<br>*AnyBitIn* and *AnyBitOut* supports LWORD, DWORD, WORD, BYTE and BOOL variable types. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>LKT := SHL(QRS, ABC); |
| **SHR** | **Format:**<br>*AnyBitOut* := SHR(*AnyBitIn, AnyInt*);<br>**Description:**<br>The SHR function right-shifts the *AnyBitIn* by *AnyInt* bits (and fills zero on the left). The result is stored in *AnyBitOut.*<br>**Supported Variable Types:**<br>*AnyBitIn* and *AnyBitOut* supports LWORD, DWORD, WORD, BYTE and BOOL variable types. *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>LKT := SHR(QRS, ABC); |

| Function Name | Function Details |
|---|---|
| **SIN** | **Format:**<br>*SINReal* := SIN(*AnyReal*);<br>**Description:**<br>The SIN function returns the Sine as *SINReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*SINReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= SIN(PB); |
| **SINT_TO_BYTE** | **Format:**<br>*AnyByte* := SINT_TO_BYTE(*SINT*);<br>**Description:**<br>The SINT_TO_BYTE function converts the *SINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := SINT_TO_BYTE(RDS); |
| **SINT_TO_DINT** | **Format:**<br>*DINT* := SINT_TO_DINT(*SINT*);<br>**Description:**<br>The SINT_TO_DINT function converts the *SINT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := SINT_TO_DINT(RDS); |
| **SINT_TO_DWORD** | **Format:**<br>*DWORD* := SINT_TO_DWORD(*SINT*);<br>**Description:**<br>The SINT_TO_DWORD function converts the *SINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type, *SINT* supports SINT variable type.<br>**Example:**<br>MNO := SINT_TO_DWORD(RDS); |
| **SINT_TO_INT** | **Format:**<br>*INT* := SINT_TO_INT(*SINT*);<br>**Description:**<br>The SINT_TO_INT function converts the *SINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *SINT* supports SINT variable type.<br>**Example:**<br>DEF := SINT_TO_INT(RDS); |

| Function Name | Function Details |
|---|---|
| **SINT_TO_LINT** | **Format:**<br>*LINT* := SINT_TO_LINT(*SINT*);<br>**Description:**<br>The SINT_TO_LINT function converts the *SINT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *SINT* supports SINT variable type.<br>**Example:**<br>KLM := SINT_TO_LINT(RDS); |
| **SINT_TO_LREAL** | **Format:**<br>*LREAL* := SINT_TO_LREAL(*SINT*);<br>**Description:**<br>The SINT_TO_LREAL function converts the *SINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *SINT* supports SINT variable type.<br>**Example:**<br>GHI := SINT_TO_LREAL(RDS); |
| **SINT_TO_LWORD** | **Format:**<br>*LWORD* := SINT_TO_LWORD(*SINT*);<br>**Description:**<br>The SINT_TO_LWORD function converts the *SINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *SINT* supports SINT variable type.<br>**Example:**<br>CDE := SINT_TO_LWORD(RDS); |
| **SINT_TO_REAL** | **Format:**<br>*REAL* := SINT_TO_REAL(*SINT*);<br>**Description:**<br>The SINT_TO_REAL function converts the *SINT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *SINT* supports SINT variable type.<br>**Example:**<br>PBX := SINT_TO_REAL(RDS); |
| **SINT_TO_UDINT** | **Format:**<br>*UDINT* := SINT_TO_UDINT(*SINT*);<br>**Description:**<br>The SINT_TO_UDINT function converts the *SINT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *SINT* supports SINT variable type.<br>**Example:**<br>BXD := SINT_TO_UDINT(RDS); |

| Function Name | Function Details |
|---|---|
| **SINT_TO_UINT** | **Format:**<br>*UINT* := SINT_TO_UINT(*SINT*);<br>**Description:**<br>The SINT_TO_UINT function converts the *SINT* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *SINT* supports SINT variable type.<br>**Example:**<br>RSA := SINT_TO_UINT(RDS); |
| **SINT_TO_ULINT** | **Format:**<br>*ULINT* := SINT_TO_ULINT(*SINT*);<br>**Description:**<br>The SINT_TO_ULINT function converts the *SINT* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *SINT* supports SINT variable type.<br>**Example:**<br>CBA := SINT_TO_ULINT(RDS); |
| **SINT_TO_USINT** | **Format:**<br>*USINT* := SINT_TO_USINT(*SINT*);<br>**Description:**<br>The SINT_TO_USINT function converts the *SINT* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *SINT* supports SINT variable type.<br>**Example:**<br>DGA := SINT_TO_USINT(RDS); |
| **SINT_TO_WORD** | **Format:**<br>*WORD* := SINT_TO_WORD(*SINT*);<br>**Description:**<br>The SINT_TO_WORD function converts the *SINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *SINT* supports SINT variable type.<br>**Example:**<br>LKT := SINT_TO_WORD(RDS); |
| **SQRT** | **Format:**<br>*SQReal* := SQRT(*AnyReal*);<br>**Description:**<br>The SQRT function returns the Square Root as *SQReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*SQReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= SQRT(PB); |

| Function Name | Function Details |
|---|---|
| **TAN** | **Format:**<br>*SQReal* := SQRT(*AnyReal*);<br>**Description:**<br>The SQRT function returns the Square Root as *SQReal* of the provided (passed in) *AnyReal* value.<br>**Supported Variable Types:**<br>*SQReal* and *AnyReal* supports variable types LREAL, REAL<br>**Example:**<br>TMY:= SQRT(PB); |
| **TO_LSB_DINT** | **Format:**<br>*DINT* := TO_LSB_DINT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_LSB_DINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a double integer (*DINT*) The values are converted in LSB order and stored into *DINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINT* and *DINToffset* supports DINT variable type.<br>**Example:**<br>ABC:=TO_LSB_DINT(TXBUFF, OFST); |
| **TO_LSB_DWORD** | **Format:**<br>*DWORD* := TO_LSB_DWORD(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_LSB_DWORD function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a double word (*DWORD*) The values are converted in LSB order and stored into *DWORD*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *DWORD* supports DWORD variable type.<br>**Example:**<br>MNO:=TO_LSB_DWORD(TXBUFF, OFST); |
| **TO_LSB_INT** | **Format:**<br>*INT* := TO_LSB_INT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_LSB_INT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an integer (*INT*) The values are converted in LSB order and stored into *INT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *INT* supports INT variable type.<br>**Example:**<br>DEF:=TO_LSB_INT(TXBUFF, OFST); |

| Function Name | Function Details |
|---|---|
| **TO_LSB_LINT** | **Format:**<br>*LINT* := TO_LSB_LINT(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_LINT function converts the *ArrayAnyType* (Passed in), beginning at the off-set location specified with *DINToffset* and stores it into a long integer (*LINT*) The values are converted in LSB order and stored into *LINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *LINT* supports LINT variable type.<br>**Example:**<br>KLM:=TO_LSB_LINT(TXBUFF, OFST); |
| **TO_LSB_LREAL** | **Format:**<br>*LREAL* := TO_LSB_LREAL(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_LREAL function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a long real (*LREAL*) The values are converted in LSB order and stored into *LREAL*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *LREAL* supports LREAL variable type.<br>**Example:**<br>GHI:=TO_LSB_LREAL(TXBUFF, OFST); |
| **TO_LSB_LWORD** | **Format:**<br>*LWORD* := TO_LSB_LWORD(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_LWORD function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a long word (*LWORD*) The values are converted in LSB order and stored into *LWORD*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>CDE:=TO_LSB_LWORD(TXBUFF, OFST); |
| **TO_LSB_REAL** | **Format:**<br>*REAL* := TO_LSB_REAL(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_REAL function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a real (*REAL*) The values are converted in LSB order and stored into *REAL*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *REAL* supports REAL variable type.<br>**Example:**<br>PBX:=TO_LSB_REAL(TXBUFF, OFST); |

| Function Name | Function Details |
|---|---|
| **TO_LSB_UDINT** | **Format:**<br>*UDINT* := TO_LSB_UDINT(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_UDINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an unsigned double integer (*UDINT*) The values are converted in LSB order and stored into *UDINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *UDINT* supports UDINT variable type.<br>**Example:**<br>BXD:=TO_LSB_UDINT(TXBUFF, OFST); |
| **TO_LSB_UINT** | **Format:**<br>*UINT* := TO_LSB_UINT(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_UINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an unsigned integer (*UINT*) The values are converted in LSB order and stored into *UINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *UINT* supports UINT variable type.<br>**Example:**<br>RSA:=TO_LSB_UINT(TXBUFF, OFST); |
| **TO_LSB_ULINT** | **Format:**<br>*ULINT* := TO_LSB_ULINT(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_ULINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an unsigned long integer (*ULINT*) The values are converted in LSB order and stored into *ULINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *ULINT* supports ULINT variable type.<br>**Example:**<br>CBA:=TO_LSB_ULINT(TXBUFF, OFST); |
| **TO_LSB_WORD** | **Format:**<br>*WORD* := TO_LSB_WORD(*ArrayAnyType, DINToffset*);<br>**Description:**<br>The TO_LSB_WORD function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a word (*WORD*) The values are converted in LSB order and stored into *WORD*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *WORD* supports WORD variable type.<br>**Example:**<br>LKT:=TO_LSB_WORD(TXBUFF, OFST); |

| Function Name | Function Details |
|---|---|
| **TO_MSB_DINT** | **Format:**<br>*DINT* := TO_MSB_DINT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_DINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a double integer (*DINT*) The values are converted in MSB order and stored into *DINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINT* and *DINToffset* supports DINT variable type.<br>**Example:**<br>ABC:=TO_MSB_DINT(TXBUFF, OFST); |
| **TO_MSB_DWORD** | **Format:**<br>*DWORD* := TO_MSB_DWORD(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_DWORD function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a double word (*DWORD*) The values are converted in MSB order and stored into *DWORD*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *DWORD* supports DWORD variable type.<br>**Example:**<br>MNO:=TO_MSB_DWORD(TXBUFF, OFST); |
| **TO_MSB_INT** | **Format:**<br>*INT* := TO_MSB_INT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_INT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an integer (*INT*) The values are converted in MSB order and stored into *INT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *INT* supports INT variable type.<br>**Example:**<br>DEF:=TO_MSB_INT(TXBUFF, OFST); |
| **TO_MSB_LINT** | **Format:**<br>*LINT* := TO_MSB_LINT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_LINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a long integer (*LINT*) The values are converted in MSB order and stored into *LINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *LINT* supports LINT variable type.<br>**Example:**<br>KLM:=TO_MSB_LINT(TXBUFF, OFST); |

| Function Name | Function Details |
|---|---|
| **TO_MSB_LREAL** | **Format:**<br>*LREAL* := TO_MSB_LREAL(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_LREAL function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a long real (*LREAL*) The values are converted in MSB order and stored into *LREAL*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *LREAL* supports LREAL variable type.<br>**Example:**<br>GHI:=TO_MSB_LREAL(TXBUFF, OFST); |
| **TO_MSB_LWORD** | **Format:**<br>*LWORD* := TO_MSB_LWORD(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_LWORD function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a long word (*LWORD*) The values are converted in MSB order and stored into *LWORD*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *LWORD* supports LWORD variable type.<br>**Example:**<br>CDE:=TO_MSB_LWORD(TXBUFF, OFST); |
| **TO_MSB_REAL** | **Format:**<br>*REAL* := TO_MSB_REAL(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_REAL function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a real (*REAL*) The values are converted in MSB order and stored into *REAL*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *REAL* supports REAL variable type.<br>**Example:**<br>PBX:=TO_MSB_REAL(TXBUFF, OFST); |
| **TO_MSB_UDINT** | **Format:**<br>*UDINT* := TO_MSB_UDINT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_UDINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an unsigned double integer (*UDINT*) The values are converted in MSB order and stored into *UDINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *UDINT* supports UDINT variable type.<br>**Example:**<br>BXD:=TO_MSB_UDINT(TXBUFF, OFST); |

| Function Name | Function Details |
|---|---|
| **TO_MSB_UINT** | **Format:**<br>*UINT* := TO_MSB_UINT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_UINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an unsigned integer (*UINT*) The values are converted in MSB order and stored into *UINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *UINT* supports UINT variable type.<br>**Example:**<br>RSA:=TO_MSB_UINT(TXBUFF, OFST); |
| **TO_MSB_ULINT** | **Format:**<br>*ULINT* := TO_MSB_ULINT(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_ULINT function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into an unsigned long integer (*ULINT*) The values are converted in MSB order and stored into *ULINT*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *ULINT* supports ULINT variable type.<br>**Example:**<br>CBA:=TO_MSB_ULINT(TXBUFF, OFST); |
| **TO_MSB_WORD** | **Format:**<br>*WORD* := TO_MSB_WORD(*ArrayAnyType, DINToffset)*;<br>**Description:**<br>The TO_MSB_WORD function converts the *ArrayAnyType* (Passed in), beginning at the offset location specified with *DINToffset* and stores it into a word (*WORD*) The values are converted in MSB order and stored into *WORD*.<br>**Supported Variable Types:**<br>*ArrayAnyType* supports an ARRAY[] OF of any variable type. *DINToffset* supports DINT variable type. *WORD* supports WORD variable type.<br>**Example:**<br>LKT:=TO_MSB_WORD(TXBUFF, OFST); |
| **TRUNC** | **Format:**<br>*AnyInt* := TRUNC(*AnyReal*);<br>**Description:**<br>The TRUNC function truncates (removes the decimal point) of the *AnyReal* varibale (Passed in) and returns (stores) the reult as *AnyInt*.<br>**Supported Variable Types:**<br>*AnyReal* supports variable types LREAL, REAL *AnyInt* supports LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT variable types.<br>**Example:**<br>ABC:= TRUNC(TMY); |

| Function Name | Function Details |
|---|---|
| **UDINT_TO_BYTE** | **Format:**<br>*AnyByte* := UDINT_TO_BYTE(*UDINT*);<br>**Description:**<br>The UDINT_TO_BYTE function converts the *UDINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := UDINT_TO_BYTE(BXD); |
| **UDINT_TO_DINT** | **Format:**<br>*DINT* := UDINT_TO_DINT(*UDINT*);<br>**Description:**<br>The UDINT_TO_DINT function converts the *UDINT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := UDINT_TO_DINT(BXD); |
| **UDINT_TO_DWORD** | **Format:**<br>*DWORD* := UDINT_TO_DWORD(*UDINT*);<br>**Description:**<br>The UDINT_TO_DWORD function converts the *UDINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type. *UDINT* supports UDINT variable type.<br>**Example:**<br>MNO := UDINT_TO_DWORD(BXD); |
| **UDINT_TO_INT** | **Format:**<br>*INT* := UDINT_TO_INT(*UDINT*);<br>**Description:**<br>The UDINT_TO_INT function converts the *UDINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>DEF := UDINT_TO_INT(BXD); |
| **UDINT_TO_LINT** | **Format:**<br>*LINT* := UDINT_TO_LINT(*UDINT*);<br>**Description:**<br>The UDINT_TO_LINT function converts the *UDINT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>RSA := UDINT_TO_LINT(BXD); |

| Function Name | Function Details |
|---|---|
| **UDINT_TO_LREAL** | **Format:**<br>*LREAL* := UDINT_TO_LREAL(*UDINT*);<br>**Description:**<br>The UDINT_TO_LREAL function converts the *UDINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>GHI := UDINT_TO_LREAL(BXD); |
| **UDINT_TO_LWORD** | **Format:**<br>*LWORD* := UDINT_TO_LWORD(*UDINT*);<br>**Description:**<br>The UDINT_TO_LWORD function converts the *UDINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>CDE := UDINT_TO_LWORD(BXD); |
| **UDINT_TO_REAL** | **Format:**<br>*REAL* := UDINT_TO_REAL(*UDINT*);<br>**Description:**<br>The UDINT_TO_REAL function converts the *UDINT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>PBX := UDINT_TO_REAL(BXD); |
| **UDINT_TO_SINT** | **Format:**<br>*SINT* := UDINT_TO_SINT(*UDINT*);<br>**Description:**<br>The UDINT_TO_SINT function converts the *UDINT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *UDINT* supports UDINT variable type.<br>**Example:**<br>RDS := UDINT_TO_SINT(BXD); |
| **UDINT_TO_UINT** | **Format:**<br>*UINT* := UDINT_TO_UINT(*UDINT*);<br>**Description:**<br>The UDINT_TO_UINT function converts the *UDINT* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>RSA := UDINT_TO_UINT(BXD); |

| Function Name | Function Details |
|---|---|
| **UDINT_TO_ULINT** | **Format:**<br>*ULINT* := UDINT_TO_ULINT(*UDINT*);<br>**Description:**<br>The UDINT_TO_ULINT function converts the *UDINT* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>CBA := UDINT_TO_ULINT(BXD); |
| **UDINT_TO_USINT** | **Format:**<br>*USINT* := UDINT_TO_USINT(*UDINT*);<br>**Description:**<br>The UDINT_TO_USINT function converts the *UDINT* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>DGA := UDINT_TO_USINT(BXD); |
| **UDINT_TO_WORD** | **Format:**<br>*WORD* := UDINT_TO_WORD(*UDINT*);<br>**Description:**<br>The UDINT_TO_WORD function converts the *UDINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *UDINT* supports UDINT variable type.<br>**Example:**<br>LKT := UDINT_TO_WORD(BXD); |
| **UINT_TO_BYTE** | **Format:**<br>*AnyByte* := UINT_TO_BYTE(*UINT*);<br>**Description:**<br>The UINT_TO_BYTE function converts the *UINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := UINT_TO_BYTE(RSA); |
| **UINT_TO_DINT** | **Format:**<br>*DINT* := UINT_TO_DINT(*UINT*);<br>**Description:**<br>The UINT_TO_DINT function converts the *UINT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := UINT_TO_DINT(RSA); |

| Function Name | Function Details |
|---|---|
| **UINT_TO_DWORD** | **Format:**<br>*DWORD* := UINT_TO_DWORD(*UINT*);<br>**Description:**<br>The UINT_TO_DWORD function converts the *UINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type. *UINT* supports UINT variable type.<br>**Example:**<br>MNO := UINT_TO_DWORD(RSA); |
| **UINT_TO_INT** | **Format:**<br>*INT* := UINT_TO_INT(*UINT*);<br>**Description:**<br>The UINT_TO_INT function converts the *UINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *UINT* supports UINT variable type.<br>**Example:**<br>DEF := UINT_TO_INT(RSA); |
| **UINT_TO_LINT** | **Format:**<br>*LINT* := UINT_TO_LINT(*UINT*);<br>**Description:**<br>The UINT_TO_LINT function converts the *UINT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *UINT* supports UINT variable type.<br>**Example:**<br>RSA := UINT_TO_LINT(RSA); |
| **UINT_TO_LREAL** | **Format:**<br>*LREAL* := UINT_TO_LREAL(*UINT*);<br>**Description:**<br>The UINT_TO_LREAL function converts the *UINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *UINT* supports UINT variable type.<br>**Example:**<br>GHI := UINT_TO_LREAL(RSA); |
| **UINT_TO_LWORD** | **Format:**<br>*LWORD* := UINT_TO_LWORD(*UINT*);<br>**Description:**<br>The UINT_TO_LWORD function converts the *UINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *UINT* supports UINT variable type.<br>**Example:**<br>CDE := UINT_TO_LWORD(RSA); |

| Function Name | Function Details |
|---|---|
| UINT_TO_REAL | **Format:**<br>*REAL* := UINT_TO_REAL(*UINT*);<br>**Description:**<br>The UINT_TO_REAL function converts the *UINT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *UINT* supports UINT variable type.<br>**Example:**<br>PBX := UINT_TO_REAL(RSA); |
| UINT_TO_SINT | **Format:**<br>*SINT* := UINT_TO_SINT(*UINT*);<br>**Description:**<br>The UINT_TO_SINT function converts the *UINT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *UINT* supports UINT variable type.<br>**Example:**<br>RDS := UINT_TO_SINT(RSA); |
| UINT_TO_UDINT | **Format:**<br>*UDINT* := UINT_TO_UDINT(*UINT*);<br>**Description:**<br>The UINT_TO_UDINT function converts the *UINT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *UINT* supports UINT variable type.<br>**Example:**<br>BXD := UINT_TO_UDINT(RSA); |
| UINT_TO_ULINT | **Format:**<br>*ULINT* := UINT_TO_ULINT(*UINT*);<br>**Description:**<br>The UINT_TO_ULINT function converts the *UINT* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *UINT* supports UINT variable type.<br>**Example:**<br>CBA := UINT_TO_ULINT(RSA); |
| UINT_TO_USINT | **Format:**<br>*USINT* := UINT_TO_USINT(*UINT*);<br>**Description:**<br>The UINT_TO_USINT function converts the *UINT* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *UINT* supports UINT variable type.<br>**Example:**<br>DGA := UINT_TO_USINT(RSA); |

| Function Name | Function Details |
|---|---|
| **UINT_TO_WORD** | **Format:**<br>*WORD* := UINT_TO_WORD(*UINT*);<br>**Description:**<br>The UINT_TO_WORD function converts the *UINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *UINT* supports UINT variable type.<br>**Example:**<br>LKT := UINT_TO_WORD(RSA); |
| **ULINT_TO_BYTE** | **Format:**<br>*AnyByte* := ULINT_TO_BYTE(*ULINT*);<br>**Description:**<br>The ULINT_TO_BYTE function converts the *ULINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := ULINT_TO_BYTE(CBA); |
| **ULINT_TO_DINT** | **Format:**<br>*DINT* := ULINT_TO_DINT(*ULINT*);<br>**Description:**<br>The ULINT_TO_DINT function converts the *ULINT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := ULINT_TO_DINT(CBA); |
| **ULINT_TO_DWORD** | **Format:**<br>*DWORD* := ULINT_TO_DWORD(*ULINT*);<br>**Description:**<br>The ULINT_TO_DWORD function converts the *ULINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type. *ULINT* supports ULINT variable type.<br>**Example:**<br>MNO := ULINT_TO_DWORD(CBA); |
| **ULINT_TO_INT** | **Format:**<br>*INT* := ULINT_TO_INT(*ULINT*);<br>**Description:**<br>The ULINT_TO_INT function converts the *ULINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>DEF := ULINT_TO_INT(CBA); |

| Function Name | Function Details |
|---|---|
| **ULINT_TO_LINT** | **Format:**<br>*LINT* := ULINT_TO_LINT(*ULINT*);<br>**Description:**<br>The ULINT_TO_LINT function converts the *ULINT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>RSA := ULINT_TO_LINT(CBA); |
| **ULINT_TO_LREAL** | **Format:**<br>*LREAL* := ULINT_TO_LREAL(*ULINT*);<br>**Description:**<br>The ULINT_TO_LREAL function converts the *ULINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>GHI := ULINT_TO_LREAL(CBA); |
| **ULINT_TO_LWORD** | **Format:**<br>*LWORD* := ULINT_TO_LWORD(*ULINT*);<br>**Description:**<br>The ULINT_TO_LWORD function converts the *ULINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>CDE := ULINT_TO_LWORD(CBA); |
| **ULINT_TO_REAL** | **Format:**<br>*REAL* := ULINT_TO_REAL(*ULINT*);<br>**Description:**<br>The ULINT_TO_REAL function converts the *ULINT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>PBX := ULINT_TO_REAL(CBA); |
| **ULINT_TO_SINT** | **Format:**<br>*SINT* := ULINT_TO_SINT(*ULINT*);<br>**Description:**<br>The ULINT_TO_SINT function converts the *ULINT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *ULINT* supports ULINT variable type.<br>**Example:**<br>RDS := ULINT_TO_SINT(CBA); |

| Function Name | Function Details |
|---|---|
| **ULINT_TO_UDINT** | **Format:**<br>*UDINT* := ULINT_TO_UDINT(*ULINT*);<br>**Description:**<br>The ULINT_TO_UDINT function converts the *ULINT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>BXD := ULINT_TO_UDINT(CBA); |
| **ULINT_TO_UINT** | **Format:**<br>*UINT* := ULINT_TO_UINT(*ULINT*);<br>**Description:**<br>The ULINT_TO_UINT function converts the *ULINT* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>RSA := ULINT_TO_UINT(CBA); |
| **ULINT_TO_USINT** | **Format:**<br>*USINT* := ULINT_TO_USINT(*ULINT*);<br>**Description:**<br>The ULINT_TO_USINT function converts the *ULINT* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>DGA := ULINT_TO_USINT(CBA); |
| **ULINT_TO_WORD** | **Format:**<br>*WORD* := ULINT_TO_WORD(*ULINT*);<br>**Description:**<br>The ULINT_TO_WORD function converts the *ULINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *ULINT* supports ULINT variable type.<br>**Example:**<br>LKT := ULINT_TO_WORD(CBA); |
| **USINT_TO_BYTE** | **Format:**<br>*AnyByte* := USINT_TO_BYTE(*USINT*);<br>**Description:**<br>The USINT_TO_BYTE function converts the *USINT* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := USINT_TO_BYTE(DGA); |

| Function Name | Function Details |
|---|---|
| **USINT_TO_DINT** | **Format:**<br>*DINT* := USINT_TO_DINT(*USINT*);<br>**Description:**<br>The USINT_TO_DINT function converts the *USINT* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := USINT_TO_DINT(DGA); |
| **USINT_TO_DWORD** | **Format:**<br>*DWORD* := USINT_TO_DWORD(*USINT*);<br>**Description:**<br>The USINT_TO_DWORD function converts the *USINT* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type. *USINT* supports USINT variable type.<br>**Example:**<br>MNO := USINT_TO_DWORD(DGA); |
| **USINT_TO_INT** | **Format:**<br>*INT* := USINT_TO_INT(*USINT*);<br>**Description:**<br>The USINT_TO_INT function converts the *USINT* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *USINT* supports USINT variable type.<br>**Example:**<br>DEF := USINT_TO_INT(DGA); |
| **USINT_TO_LINT** | **Format:**<br>*LINT* := USINT_TO_LINT(*USINT*);<br>**Description:**<br>The USINT_TO_LINT function converts the *USINT* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *USINT* supports USINT variable type.<br>**Example:**<br>RSA := USINT_TO_LINT(DGA); |
| **USINT_TO_LREAL** | **Format:**<br>*LREAL* := USINT_TO_LREAL(*USINT*);<br>**Description:**<br>The USINT_TO_LREAL function converts the *USINT* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *USINT* supports USINT variable type.<br>**Example:**<br>GHI := USINT_TO_LREAL(DGA); |

| Function Name | Function Details |
|---|---|
| **USINT_TO_LWORD** | **Format:**<br>*LWORD* := USINT_TO_LWORD(*USINT*);<br>**Description:**<br>The USINT_TO_LWORD function converts the *USINT* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *USINT* supports USINT variable type.<br>**Example:**<br>CDE := USINT_TO_LWORD(DGA); |
| **USINT_TO_REAL** | **Format:**<br>*REAL* := USINT_TO_REAL(*USINT*);<br>**Description:**<br>The USINT_TO_REAL function converts the *USINT* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *USINT* supports USINT variable type.<br>**Example:**<br>PBX := USINT_TO_REAL(DGA); |
| **USINT_TO_SINT** | **Format:**<br>*SINT* := USINT_TO_SINT(*USINT*);<br>**Description:**<br>The USINT_TO_SINT function converts the *USINT* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *USINT* supports USINT variable type.<br>**Example:**<br>RDS := USINT_TO_SINT(DGA); |
| **USINT_TO_UDINT** | **Format:**<br>*UDINT* := USINT_TO_UDINT(*USINT*);<br>**Description:**<br>The USINT_TO_UDINT function converts the *USINT* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *USINT* supports USINT variable type.<br>**Example:**<br>BXD := USINT_TO_UDINT(DGA); |
| **USINT_TO_UINT** | **Format:**<br>*UINT* := USINT_TO_UINT(*USINT*);<br>**Description:**<br>The USINT_TO_UINT function converts the *USINT* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *USINT* supports USINT variable type.<br>**Example:**<br>RSA := USINT_TO_UINT(DGA); |

| Function Name | Function Details |
|---|---|
| **USINT_TO_ULINT** | **Format:**<br>*ULINT* := USINT_TO_ULINT(*USINT*);<br>**Description:**<br>The USINT_TO_ULINT function converts the *USINT* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *USINT* supports USNT variable type.<br>**Example:**<br>CBA := USINT_TO_ULINT(DGA); |
| **USINT_TO_WORD** | **Format:**<br>*WORD* := USINT_TO_WORD(*USINT*);<br>**Description:**<br>The USINT_TO_WORD function converts the *USINT* (Passed in) value into a word and stores it in *WORD.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *USINT* supports USINT variable type.<br>**Example:**<br>LKT := USINT_TO_WORD(DGA); |
| **WORD_TO_BYTE** | **Format:**<br>*AnyByte* := WORD_TO_BYTE(*WORD*);<br>**Description:**<br>The WORD_TO_BYTE function converts the *WORD* (Passed in) value into a byte and stores it in *AnyByte.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *AnyByte* supports BYTE variable type.<br>**Example:**<br>XYZ := WORD_TO_BYTE(LKT); |
| **WORD_TO_DINT** | **Format:**<br>*DINT* := WORD_TO_DINT(*WORD*);<br>**Description:**<br>The WORD_TO_DINT function converts the *WORD* (Passed in) value into a double integer and stores it in *DINT.*<br>**Supported Variable Types:**<br>*WORD* supports WORD variable type, *DINT* supports DINT variable type.<br>**Example:**<br>ABC := WORD_TO_DINT(LKT); |
| **WORD_TO_DWORD** | **Format:**<br>*DWORD* := WORD_TO_DWORD(*WORD*);<br>**Description:**<br>The WORD_TO_DWORD function converts the *WORD* (Passed in) value into a double word and stores it in *DWORD.*<br>**Supported Variable Types:**<br>*DWORD* supports DWORD variable type. *WORD* supports WORD variable type.<br>**Example:**<br>MNO := WORD_TO_DWORD(LKT); |

| Function Name | Function Details |
|---|---|
| **WORD_TO_INT** | **Format:**<br>*INT* := WORD_TO_INT(*WORD*);<br>**Description:**<br>The WORD_TO_INT function converts the *WORD* (Passed in) value into an integer and stores it in *INT.*<br>**Supported Variable Types:**<br>*INT* supports INT variable type, *WORD* supports WORD variable type.<br>**Example:**<br>DEF := WORD_TO_INT(LKT); |
| **WORD_TO_LINT** | **Format:**<br>*LINT* := WORD_TO_LINT(*WORD*);<br>**Description:**<br>The WORD_TO_LINT function converts the *WORD* (Passed in) value into a long integer and stores it in *LINT.*<br>**Supported Variable Types:**<br>*LINT* supports LINT variable type, *WORD* supports WORD variable type.<br>**Example:**<br>RSA := WORD_TO_LINT(LKT); |
| **WORD_TO_LREAL** | **Format:**<br>*LREAL* := WORD_TO_LREAL(*WORD*);<br>**Description:**<br>The WORD_TO_LREAL function converts the *WORD* (Passed in) value into a long real and stores it in *LREAL.*<br>**Supported Variable Types:**<br>*LREAL* supports LREAL variable type, *WORD* supports WORD variable type.<br>**Example:**<br>GHI := WORD_TO_LREAL(LKT); |
| **WORD_TO_LWORD** | **Format:**<br>*LWORD* := WORD_TO_LWORD(*WORD*);<br>**Description:**<br>The WORD_TO_LWORD function converts the *WORD* (Passed in) value into a long word and stores it in *LWORD.*<br>**Supported Variable Types:**<br>*LWORD* supports LWORD variable type, *WORD* supports WORD variable type.<br>**Example:**<br>CDE := WORD_TO_LWORD(LKT); |
| **WORD_TO_REAL** | **Format:**<br>*REAL* := WORD_TO_REAL(*WORD*);<br>**Description:**<br>The WORD_TO_REAL function converts the *WORD* (Passed in) value into a real and stores it in *REAL.*<br>**Supported Variable Types:**<br>*REAL* supports REAL variable type, *WORD* supports WORD variable type.<br>**Example:**<br>PBX := WORD_TO_REAL(LKT); |

| Function Name | Function Details |
|---|---|
| **WORD_TO_SINT** | **Format:**<br>*SINT* := WORD_TO_SINT(*WORD*);<br>**Description:**<br>The WORD_TO_SINT function converts the *WORD* (Passed in) value into a short integer and stores it in *SINT.*<br>**Supported Variable Types:**<br>*SINT* supports SINT variable type. *WORD* supports WORD variable type.<br>**Example:**<br>RDS := WORD_TO_SINT(LKT); |
| **WORD_TO_UDINT** | **Format:**<br>*UDINT* := WORD_TO_UDINT(*WORD*);<br>**Description:**<br>The WORD_TO_UDINT function converts the *WORD* (Passed in) value into an unsigned double integer and stores it in *UDINT.*<br>**Supported Variable Types:**<br>*UDINT* supports UDINT variable type, *WORD* supports WORD variable type.<br>**Example:**<br>BXD := WORD_TO_UDINT(LKT); |
| **WORD_TO_UINT** | **Format:**<br>*UINT* := WORD_TO_UINT(*WORD*);<br>**Description:**<br>The WORD_TO_UINT function converts the *WORD* (Passed in) value into an unsigned integer and stores it in *UINT.*<br>**Supported Variable Types:**<br>*UINT* supports UINT variable type, *WORD* supports WORD variable type.<br>**Example:**<br>RSA := WORD_TO_UINT(LKT); |
| **WORD_TO_ULINT** | **Format:**<br>*ULINT* := WORD_TO_ULINT(*WORD*);<br>**Description:**<br>The WORD_TO_ULINT function converts the *WORD* (Passed in) value into an unsigned long integer and stores it in *ULINT.*<br>**Supported Variable Types:**<br>*ULINT* supports ULINT variable type, *WORD* supports WORD variable type.<br>**Example:**<br>CBA := WORD_TO_ULINT(LKT); |
| **WORD_TO_USINT** | **Format:**<br>*USINT* := WORD_TO_USINT(*WORD*);<br>**Description:**<br>The WORD_TO_USINT function converts the *WORD* (Passed in) value into an unsigned short integer and stores it in *USINT.*<br>**Supported Variable Types:**<br>*USINT* supports USINT variable type, *WORD* supports WORD variable type.<br>**Example:**<br>DGA := WORD_TO_USINT(LKT); |

DIVELBISS
SOFTWARE LICENSE AGREEMENT

This Software License Agreement (the "Agreement") sets forth the terms by which Divelbiss Corporation, an Ohio corporation having a principal place of business at 9778 Mt. Gilead Road, Fredericktown, Ohio ("Divelbiss"), authorizes its bona fide licensees who have paid all applicable fees and accepted the terms of this Agreement (each a "Licensee") to use the Licensed Software (as defined below) provided herewith. Installing, using or attempting to install or use such Licensed Software or otherwise expressing assent to the terms herein constitutes acceptance of this Agreement. Any installation, use or attempted installation or use of such Licensed Software by any party other than a Licensee or otherwise in violation of this Agreement is expressly prohibited.

**Introduction**
Whereas Divelbiss has developed certain modules of computer software known as "PLC ON A CHIP Kernel" and "EZ LADDER Toolkit"; and Licensee wishes to secure certain rights to use such software ; and Divelbiss is prepared to license such rights, subject to the terms and conditions of this Agreement; therefore, in consideration of the mutual covenants contained herein and intending to be legally bound hereby, Divelbiss and Licensee agree as follows:

**1. Licensed Software**

The PLC ON A CHIP Kernel and EZ LADDER Toolkit software, whether in source code or object code format, and all related documentation and revisions, updates and modifications thereto (collectively, "Licensed Software"), is licensed by Divelbiss to Licensee strictly subject to the terms of this Agreement.

**2. License Grant**

Divelbiss hereby grants to Licensee a non exclusive, non-transferable license to use the Licensed Software as follows.

(a) Except as otherwise provided herein, one (1) user may install and use on one (1) desktop personal computer and on one (1) portable personal computer the EZ LADDER Toolkit (i) to develop, test, install, configure and distribute certain applications on certain hardware devices such as programmable logic controllers (each a "Resulting Product"), and (ii) to configure the PLC ON A CHIP Kernel on designated processors, which shall constitute Resulting Products.

(b) Licensee may copy the EZ LADDER Toolkit only for backup purposes.

(c) Licensee may not amend, modify, decompile, reverse engineer, copy (except as expressly authorized in Section 2 of this Agreement), install on a network, or permit use by more than a single user, in whole or in part, the Licensed Software, or sublicense, convey or purport to convey any such right to any third party.

(d) Licensee, Licensee's customers and others who obtain Resulting Products are expressly prohibited from using, in whole or in part, the Licensed Software and any Resulting Product, in any use or application (i) intended to sustain or support life; (ii) for surgical implant; (iii) related to the operation of nuclear facilities; (iv) in which malfunction or failure could result in death or personal injury; or (v) in environments otherwise intended to be fault-tolerant.

**3. License Fee**

(a) Except when Licensee obtains the EZ LADDER Toolkit from an approved distributor or OEM pursuant to other fee arrangements, Licensee will pay to Divelbiss the license fee for the EZ LADDER Toolkit specified in the applicable Divelbiss price list, which is due and payable upon delivery of same.

(b) If Licensee fails to make any payment when due, Divelbiss may, at its sole option, terminate Licensee's rights under this Agreement to use the Licensed Software. If Licensee fails to pay any balance within thirty (30) days after being notified by Divelbiss that payment is overdue, Divelbiss may take whatever steps it deems necessary to collect the balance, including referring the matter to an agency and/or suing for collection. All expenses and fees associated with the collection of an overdue balance, including costs and fees of collection and attorney's fees, shall be paid by Licensee. Overdue balances are subject to a monthly finance charge equal to the greater of [1.5]% or the maximum interest rate permitted by law times the unpaid balance.

**4. Reporting**

(a) Upon request of Divelbiss, Licensee will provide a written report each quarter showing the number of Resulting Products produced, distributed or sold by Licensee during the previous calendar quarter, the parties (identified by name, address, etc.) to which they were distributed or sold, and the revenue received therefor.

(b) Divelbiss shall be entitled to commission or to conduct an audit of Licensee's books and records twice per year in order to verify the accuracy of reports regarding resulting Products made by Licensee to Divelbiss. Such audit shall be conducted during regular business hours at Licensee's facilities, and Licensee shall cooperate fully with in connection with such audit, making all facilities, records and personnel available upon request by Divelbiss or its representative.

**5. Divelbiss Warranties**

(a) Divelbiss represents and warrants that (i) it is the owner of the Licensed Software, and (ii) this Agreement violates no previous agreement between Divelbiss and any third party.

(b) Divelbiss further warrants that for a period of 90 days from the date this Agreement is accepted by Licensee, the EZ LADDER Toolkit will perform substantially in accordance with the accompanying documentation provided by Divelbiss, provided that the EZ LADDER Toolkit (i) has not been modified, (ii) has been maintained according to all applicable maintenance recommendations, (iii) has not been used with hardware or software or installed or operated in a manner inconsistent with any manuals or relevant system requirements provided by Divelbiss, and (iv) has not been subjected to abuse, negligence or other improper treatment, including, without limitation, use outside the operating environment or range of applications prescribed in any manuals or relevant system requirements provided by Divelbiss by Divelbiss. Provided that Licensee gives prompt written notice to Divelbiss of any alleged breach of the foregoing warranty and that such alleged breach can be reproduced by Divelbiss, Divelbiss will use commercially reasonable efforts to repair or replace the EZ LADDER Toolkit so that it performs as warranted, or , at its sole option, refund to Licensee a prorated share of the license fee paid by Licensee for the portion of the EZ LADDER Toolkit which caused the alleged breach of warranty. Licensee acknowledges that the foregoing represents Divelbiss's sole obligation and Licensee's sole remedy for any alleged breach of warranty regardingthe EZ LADDER Toolkit.

(c) Divelbiss expressly disclaims any and all warranties concerning any Resulting Products and any applications developed, tested, installed or distributed by Licensee using the Licensed Software, and Licensee expressly ac knowledges that it is solely responsible for any and all Resulting Products and applications developed, tested, installed or distributed using the Licensed Software, and for any and all claims, damages, settlements, expenses and attorney's fees arising from the distribution or use of the PLC ON A CHIP Kernel or Resulting Products by Licensee, Licensee's customers or others.

(d) DIVELBISS MAKES NO OTHER WARRANTIES OF ANY KIND WITH RESPECT TO THE LICENSED SOFTWARE OR THIS AGREEMENT, AND EXPRESSLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

**6. Licensee Warranties**

Licensee represents, warrants and covenants that:

(a) Licensee has all necessary authority to enter into and to fulfill its obligations under this Agreement;

(b) Licensee will comply with all federal, state and local laws and regulations applicable to the use or disposition of the Licensed Software, including without, limitation all export laws and regulations;

(c) Licensee shall be solely liable for all Resulting Products, any and all warranties on Resulting Products shall be made only by and on behalf of Licensee, and Licensee shall make NO representations or warranties on behalf of Divelbiss.

(d) For the term of this Agreement and any renewal thereof, and for one (1) year thereafter, Licensee will not solicit or hire any of Divelbiss's employees.

**7. Limitation of Liability**

LICENSEE ACKNOWLEDGES AND AGREES THAT NEITHER DIVELBISS NOR ITS SUPPLIERS, EMPLOYEES OR AFFILIATES WILL BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS OR GOODWILL, LOSS OF DATA OR USE OF DATA, INTERRUPTION OF BUSINESS, NOR FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND UNDER, ARISING OUT OF, OR RELATED TO THE SUBJECT MATTER OF THIS AGREEMENT (SPECIFICALLY INCLUDING ANY LOSS TO OR DAMAGES OF LICENSEE'S CUSTOMERS, OF ANY SORT WHATSOEVER), HOWEVER CAUSED, WHETHER ANY SUCH CLAIM SOUNDS IN CONTRACT, TORT, STRICT LIABILITY OR OTHER LEGAL OR EQUITABLE THEORY, EVEN IF DIVELBISS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS. IN NO EVENT WILL DIVELBISS'S LIABILITY UNDER, ARISING OUT OF OR RELATED TO THE SUBJECT MATTER OF THIS AGREEMENT EXCEED THE AMOUNT RECEIVED BY DIVELBISS FROM LICENSEE UNDER THIS AGREEMENT DURING THE NINETY (90) DAY PERIOD PRECEDING THE EVENT GIVING

RISE TO SUCH LIABILITY, OR THE AMOUNT OF A SINGLE-USER LICENSE FEE FOR THE EZ LADDER TOOLKIT, WHICHEVER IS GREATER.

**8. Indemnification**

(a) Subject to the limitations of Section 7 of this Agreement, Divelbiss will indemnify Licensee from and against liability for any judgment finally awarded by a court of competent jurisdiction against Licensee based upon a claim that the EZ LADDER Toolkit infringes any current U.S. patent or copyright of a third party, provided that Divelbiss is promptly notified of any such threats, claims or proceedings, afforded the opportunity to intervene in any such proceeding and given sole control over the defense of such claim, including all negotiations of any prospective settlement or compromise, and that Licensee gives all cooperation and assistance requested by Divelbiss in connection with same; and provided further that the foregoing obligation of Divelbiss does not apply with respect to any Resulting Products or any hardware, software (including the Licensed Software) or components thereof (i) not supplied by Divelbiss, (ii) made or modified in whole or in part by Licensee or according to Licensee's specifications, (iii) otherwise modified after delivery, (iv) combined with other hardware, software, products or processes by Licensee (including in creating Resulting Products) where such claim could have been avoided absent such combination, (v) insofar as Licensee continues allegedly infringing activity after being notified thereof or informed of steps or modifications that would have avoided the alleged infringement, or (vi) used by Licensee in violation of the terms of this Agreement.

(b) Licensee will defend, indemnify and hold Divelbiss harmless from and against any and all losses, liabilities, judgments, damages and claims against Divelbiss obtained or asserted by any third party (including any allegation of infringement or violation of proprietary rights), and all related costs, including attorney fees, incurred by Divelbiss, arising or resulting from or related to i) Licensee's use, modification or adaptation of the Licensed Software, including to create any application or any Resulting Product, ii) the operation or performance, or Licensee's or any third party's use, of any Resulting Product, iii) any breach by Licensee of any representation or warranty made by Licensee related to the Licensed Software or any Resulting Product, or iv) any breach by Licensee of any of its obligations under this Agreement.

(c) In the event that any claim of infringement under Section 8(a) above is, or in Divelbiss's sole judgment is likely to be, substantiated, Divelbiss may, at its sole discretion, use commercially reasonable efforts to i) obtain a license from the third party for Licensee to continue using the allegedly infringing feature or aspect of the EZ LADDER Toolkit; ii) replace or modify the allegedly infringing feature or aspect of the EZ LADDER Toolkit to avoid such infringement; or iii) terminate this Agreement and the license hereunder and refund a prorated portion of the initial license fee paid by Licensee for the allegedly infringing feature or aspect of the EZ LADDER Toolkit .

**9. Modification of Licensed Software**

(a) Divelbiss may, from time to time, at its sole discretion and without further notice to Licensee, make, and at its further discretion distribute to Licensee, modifications to the Licensed Software. In the event that Licensee fails to install such a modification when so advised by Divelbiss, Divelbiss shall be relieved of any obligation pursuant to the limited warranty set forth in Section 5 hereof. Should Licensee request modifications to the Licensed Software, Divelbiss may charge for and make such changes subject to the terms of a separate agreement between the parties.

(b) Licensee may not modify the Licensed Software or engage any third party to modify the Licensed Software without the express, written consent of Divelbiss. Any and all modifications made to the Licensed Software, whether by Licensee or any third party, and all rights therein are hereby assigned to and shall be the sole and exclusive property of Divelbiss.

**10. Ownership of Licensed Software**

(a) Licensee acknowledges that, subject only to the license specifically granted herein, all right, title, and interest in and to the Licensed Software, all revisions and copies thereof provided to or created by Licensee and all modifications thereof, by whomever made, are and shall remain the sole and exclusive property of Divelbiss.

(b) LICENSEE ACKNOWLEDGES THAT VARIOUS ASPECTS AND FEATURES OF THE LICENSED SOFTWARE MAY BE PROTECTED UNDER APPLICABLE PATENT, COPYRIGHT, TRADEMARK AND TRADE SECRET LAW AND THAT, EXCEPT AS EXPRESSLY AUTHORIZED IN WRITING BY DIVELBISS, LICENSEE MAY NOT USE, DISCLOSE OR REPRODUCE OR DISTRIBUTE ANY COPIES OF THE LICENSED SOFTWARE, IN WHOLE OR IN PART, NOR AUTHORIZE OR PERMIT OTHERS TO DO SO.

(c) Licensee further acknowledges that any applications made by Licensee using the Licensed Software, including any incorporated into Resulting Products, are derivative works made solely with the authorization of Divelbiss, in consideration for which Licensee agrees to provide, upon request from Divelbiss, copies of all such applications to

Divelbiss and grants to Divelbiss a perpetual, irrevocable, royalty-free license to copy and use such applications so long as Divelbiss is not competing with Licensee.

(d) Licensee shall not, nor will it assist others in attempting to, decompile, reverse engineer or otherwise re-create the source code for or functionality of the Licensed Software. Licensee shall not use the Licensed Software for the purpose of developing any similar or competing product, or assisting a third party to develop a similar or competing product.

(e) At no expense to Divelbiss, Licensee will take any action, including executing any document, requested by Divelbiss in order to secure, perfect or protect the rights of Divelbiss in the Licensed Software or Confidential Information (as hereinafter defined).

## 11. Confidentiality

Except as expressly provided in this Agreement, Licensee shall not disclose or permit disclosure to any third parties the Licensed Software (including object code, source code and documentation) or any other confidential information provided by Divelbiss ("Confidential Information"). Further, Licensee will use all reasonable precautions and take all steps necessary to prevent any Confidential Information from being acquired, in whole or in part, by any unauthorized party, will use Confidential Information solely in furtherance of this Agreement, and will permit access to any Confidential Information only by those employees of Licensee with a legitimate "need to know." In the event that Licensee learns or has reason to believe that Confidential Information has been disclosed or is at risk of being disclosed to any unauthorized party, Licensee will immediately notify Divelbiss thereof and will cooperate fully with Divelbiss in seeking to protect Divelbiss's rights in the Confidential Information.

## 12. Term and Termination

(a) This Agreement shall remain in effect from the date it is accepted until terminated as provided below.

(b) Divelbiss may terminate this Agreement and all license rights hereunder upon the occurrence of any of the following:

(i) Licensee fails to cure any material breach of this Agreement within thirty (30) days after receiving notice of such breach;

(ii) Licensee becomes insolvent or unable to pay its debts, makes an assignment for the benefit of creditors, ceases to be a going concern, files for protection of the bankruptcy laws, becomes the subject of any involuntary proceeding under federal bankruptcy laws or has a receiver or trustee appointed to manage its assets;

(iii) Licensee consolidates or merges into or with any other entity or entities or sells or transfers all or substantially all of its assets; or

(iv) Following ninety (90) days written notice of termination to Licensee.

(c) Licensee may terminate this Agreement and all licenses hereunder in the event that Divelbiss fails to cure any material breach of this Agreement within thirty (30) days after receiving notice of such breach.

(d) Any fees or expenses payable by Licensee to Divelbiss shall not be reduced or otherwise affected by termination of this Agreement. In the event of termination of this Agreement for any reason, neither party shall be liable to the other on account of loss of prospective profits or anticipated sales, or on account of expenditures, inventories, investments, or commitments.

(e) Upon termination of this Agreement for any reason, Licensee will immediately return to Divelbiss or, upon instruction from Divelbiss, destroy all copies of the Licensed Software (including all code, documentation, manuals, etc.) and all Confidential Information in its possession, and will certify in writing to Divelbiss that it has done so.

(f) All provisions regarding ownership, confidentiality, proprietary rights, payment of fees and royalties, indemnification, disclaimers of warranty and limitations of liability will survive termination of this Agreement.

## 13. Assignment and Sublicensing

This Agreement, the license granted hereunder and the Licensed Software may not be assigned, sublicensed or otherwise transferred or conveyed by Licensee to any third party without the express, written consent of Divelbiss.

**14. Dispute Resolution**

(a) In the event of any dispute arising between the parties related to the subject matter of this Agreement, except regarding the payment of fees under Sections 3 or 15 of this Agreement or as provided in Subsection (b) below ("Dispute"), the parties agree to attempt to resolve such Dispute according to the procedures set forth below.

(i) In the event either Divelbiss or Licensee notifies the other party of a Dispute, representatives of each party with adequate authority to settle such Dispute will promptly engage in direct negotiations. If such representatives are unable to resolve such Dispute within ten (10) business days after commencing negotiations, or twenty (20) business days after the initial notice of Dispute, then either party may initiate mediation of the Dispute as provided in Subsection (a)(ii) below.

(ii) In the event either party initiates mediation of the Dispute (by sending a written notice of mediation to the other party), then the Dispute shall be subject to mediation in Mt. Vernon, Ohio before a single mediator (to be proposed, in the first instance, by the party initiating mediation) who will be reasonably familiar with the computer industry and mutually acceptable to the parties. The parties agree to participate in such mediation in good faith, through representatives with due authority to settle any such Dispute. If such representatives are unable to resolve such Dispute within twenty (20) business days after commencing mediation, then each party may pursue whatever further recourse it deems necessary to protect its rights under this Agreement.

(b) Licensee agrees that any violation of this Agreement related to the Licensed Software or Confidential Information, specifically including Divelbiss's proprietary rights therein, is likely to result in irreparable injury to Divelbiss. Accordingly, notwithstanding any other provision of this Agreement to the contrary, Licensee agrees that Divelbiss shall be entitled to all appropriate relief from any court of competent jurisdiction, whether in the form of injunctive relief and/or monetary damages, to protect its proprietary rights in the Licensed Software and Confidential Information.

**15. Maintenance and Support**

(a) In consideration of the payment of annual maintenance and support fees by or on behalf of Licensee (payable for the first year with the license fee, and thereafter annually at least thirty (30) days before the anniversary date of this Agreement),

Divelbiss will provide maintenance and support of the EZ LADDER Toolkit, in the form of (i) such periodic corrections, updates and revisions to the EZ LADDER Toolkit as Divelbiss, in its sole discretion, may from time to time elect to release, and (ii) responses to inquiries submitted by Licensee by email to Divelbiss at sales@divelbiss.com.

(b) The maintenance and support fee is specified in the applicable Divelbiss price list.

**6. General**

(a) This agreement constitutes the entire agreement between the parties relating to the Licensed Software and the subject matter hereof, supersedes all other proposals, quotes, understandings or agreements, whether written or oral, and cannot be modified except by a writing signed by both Licensee and Divelbiss.

(b) In the event of any conflict between the terms of this Agreement and any purchase order or similar documentation prepared by Licensee in connection with the transactions contemplated herein, this Agreement shall govern and take precedence, notwithstanding Divelbiss's failure to object to any conflicting provisions.

(c) Notwithstanding anything to the contrary herein, except for payment obligations under Sections 3 or 15, neither party shall be liable for any failure of performance beyond its reasonable control.

(d) Except as otherwise provided, this Agreement will be subject to and construed in accordance with the laws of the State of Ohio (U.S.A.) without regard to its conflict of laws provisions. Exclusive venue for any legal action between the Parties arising out of or related to this Agreement or the subject matter hereof will be in the state or federal courts located or having jurisdiction in Knox County, Ohio (U.S.A.), which the Parties expressly acknowledge to have personal jurisdiction over them. The 1980 UN Convention on the International Sale of Goods (CISG) will not apply hereto.

(e) No waiver by either party of a breach of this Agreement shall operate or be construed as a waiver of any subsequent breach.

(f) The invalidity, illegality or unenforceability of any provision of this Agreement shall not affect the remainder of the Agreement, and this Agreement shall be construed and reformed without such provision, provided that the ability of neither party to obtain substantially the bargained for performance of the other shall have thereby been impaired.

(g) All notices, consents and other communications between the parties shall be in writing and shall be sent by (i) first class mail, certified or registered, return receipt requested, postage prepaid, (ii) electronic facsimile transmission, (iii) overnight courier service, (iv) telegram or telex or (v) messenger, to the respective addresses that the parties may provide.

(h) Licensee shall be deemed an independent contractor hereunder, and as such, shall not be deemed, nor hold itself out to be, an agent or employee of Divelbiss. Under no circumstances shall any of the employees of a party hereto be deemed to be employees of the other party for any purpose. This Agreement shall not be construed as authority for either party to act for the other party in any agency or other capacity, or to make commitments of any kind for the account of or on behalf of the other except to the extent and for the purposes provided herein.

(i) LICENSEE ACKNOWLEDGES THAT IT HAS READ THIS AGREEMENT, UNDERSTANDS IT, AND AGREES TO BE BOUND BY ITS TERMS AND CONDITIONS.