
P2CDS-622 System

Release 1.0

FACTS Engineering

May 08, 2024

RELEASE NOTES:

1	Version Info	3
1.1	P2CDS-622 Firmware Version	3
1.2	Website Version	4
2	Preparation	7
2.1	CODESYS IDE Install	7
2.2	P2CDS-622 Firmware Image	7
2.3	P2CDS-622 Package Install	9
2.4	Configure the Ethernet Port(s)	10
2.5	IIoT License Install	10
3	P2CDS-622 Starter Project	13
3.1	Introduction	13
3.2	System Requirements and Installation	13
3.3	Recommendation for Data Protection	14
3.4	Help	14
3.5	Creating a Project	14
3.6	Connect to PLC and Download	19
3.7	Run and Watch the Application	21
3.8	Debug the Application	23
4	Visualization: WebVisu	25
4.1	Quick Start	26
4.2	Detailed Help Areas	28
5	Creating a Boot Application	29
6	PC Based (Win PLC) Starter Project	31
6.1	Introduction	31
7	Codesys Development System	33
7.1	Introduction	33
7.2	Functionality	34
7.3	Other Resources	35
8	CPU P2CDS-622	37
8.1	Introduction	37
9	Power Supply	45
9.1	Introduction	45
9.2	Functionality	45

10 Bases	47
10.1 Introduction	47
10.2 Functionality	48
11 IO Modules	49
11.1 Introduction	49
11.2 Functionality and Purchase	50
12 Physical Ports	51
12.1 Micro SD Card/Unmount Button	52
12.2 USB-C Programming Port	52
12.3 RS-232/RS-485 Serial Port (RJ12)	53
12.4 RS-485/RS-232 Serial Port (TBLK)	53
12.5 10/100 MB Ethernet Port (x2)	54
13 Serial RS-232/RS-485	55
13.1 RS-232/RS-485 Serial Port (RJ-12)	55
13.2 RS-485/RS-232 Serial Port (TBLK)	56
14 Ethernet	59
14.1 Introduction	59
14.2 System Requirements and Installation	61
14.3 Create a Project	63
14.4 Ethernet Applications	67
15 Modbus	69
15.1 Introduction	69
15.2 General Specifications	69
15.3 Function Codes, Addresses and Data Types	70
15.4 Modbus TCP	70
15.5 Modbus RTU	72
15.6 Example Projects	74
16 EtherNet/IP	75
16.1 General Specifications	75
16.2 Scanner	76
16.3 Adapter	78
16.4 Example Projects	80
17 IO Module Configuration and Status	81
17.1 Discrete Input	81
17.2 Discrete Output	85
17.3 Analog Input	93
17.4 Analog Output	106
17.5 Analog Combo	116
17.6 Temperature	119
17.7 Relay Output	124
17.8 Other	126
18 Intro to Projects	129
18.1 Security/Data Protection	129
18.2 Project Variable Nomenclature	129
19 Data Type Conversion	131
19.1 Introduction	131

19.2	System Requirements and Installation	132
19.3	Recommendation for Data Protection	132
19.4	Help	132
19.5	Creating a Project	133
19.6	Connect to PLC and Download	140
19.7	Run and Watch the Application	142
19.8	Debug the Application	143
20	Analog Input Module	145
20.1	Introduction	145
20.2	System Requirements and Installation	145
20.3	Recommendation for Data Protection	146
20.4	Help	146
20.5	Create a Project	146
20.6	Sensor/Input Wiring	152
20.7	Connect to PLC and Download	153
20.8	Run and Watch the Application	154
20.9	Debug the Application	156
20.10	Additional Resources	158
21	Analog In/Out Voltage Module	159
21.1	Introduction	159
21.2	System Requirements and Installation	159
21.3	Recommendation for Data Protection	159
21.4	Help	160
21.5	Create a Project	160
22	Thermocouple Input Module	165
22.1	Introduction	165
22.2	System Requirements and Installation	165
22.3	Recommendation for Data Protection	165
22.4	Help	166
22.5	Create a Project	166
23	Modbus TCP - Master Mode	171
23.1	Introduction	171
23.2	System Requirements and Installation	171
23.3	Recommendation for Data Protection	172
23.4	Help	172
23.5	Installation of Additional Software/Tools	172
23.6	Creating a Project	172
24	Modbus RTU - Master Mode	181
24.1	Introduction	181
24.2	System Requirements and Installation	181
24.3	Recommendation for Data Protection	182
24.4	Help	182
24.5	Installation of Additional Software/Tools	182
24.6	Creating a Project	182
25	Serial Communications	189
25.1	Introduction	189
25.2	System Requirements and Installation	189
25.3	Recommendation for Data Protection	189
25.4	Help	190

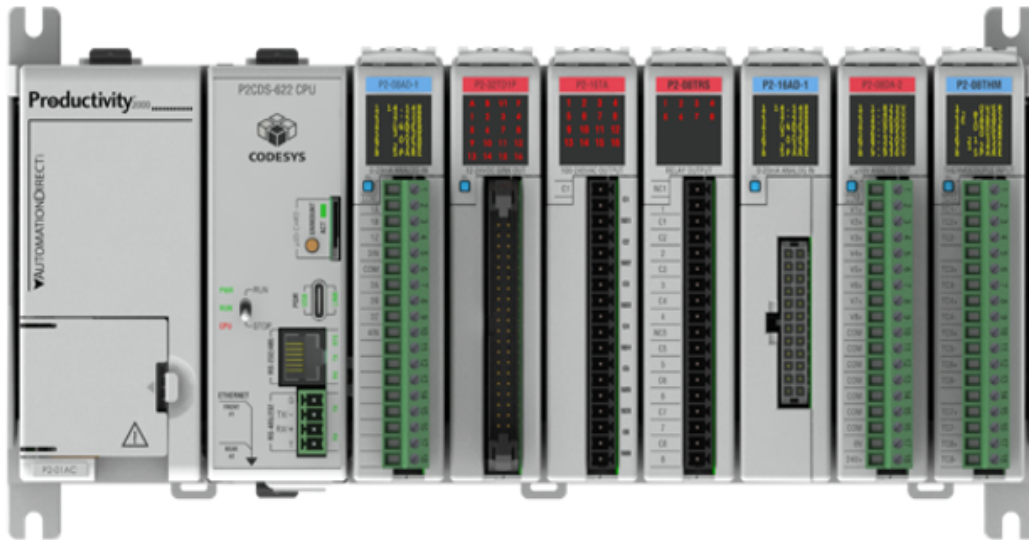
25.5	Import the Archived Project	190
26	EtherNet/IP Scanner - Explicit Mode	191
26.1	Project Scope	191
26.2	Create/Import the Scanner P2CDS-622 Project	192
26.3	Create/Import the Adapter Raspberry PI Project	195
27	EtherNet/IP Scanner - Implicit Mode	197
27.1	Project Scope	197
27.2	Create/Import the Scanner P2CDS-622 Project	198
27.3	Create/Import the Adapter Raspberry PI Project	201
28	EtherNet/IP Scanner - Multi-Adapters	203
28.1	Devices	203
29	IIoT - MQTT	205
29.1	Introduction	205
29.2	System Requirements and Installation	206
29.3	Recommendation for Data Protection	206
29.4	Help	207
29.5	Install IIoT Library	207
29.6	Creating a Project	207
29.7	Connect to PLC and Download	211
29.8	Run and Watch the Application	213
30	IIoT - Web Client Support	215
30.1	Introduction	215
30.2	Install IIoT Library	216
30.3	Project	216
31	IIoT - Mail Support	219
31.1	Introduction	219
31.2	Install IIoT Library	220
31.3	Project	221
32	IIoT - SNTP Support	225
32.1	Introduction	225
32.2	Install IIoT Library	226
32.3	Project	226
33	Weather Forecast Library	227
33.1	Introduction	227
33.2	Install API Library	228
33.3	Project	228
34	Data Logger/CSV Write	229
34.1	System Requirements and Installation	229
34.2	Install IIoT Library	230
34.3	Project Generation	230
34.4	Closing Comments	232
35	Object Oriented Industrial Programming	233
36	Firmware Update/Factory Reset	235
36.1	Firmware Update	235
36.2	Factory Reset	238

37 Power Supply H/W Revision Number	239
37.1 Nomenclature	239
37.2 Accessing the Information	240
38 P2-Base H/W Revision Number	241
38.1 Nomenclature	241
38.2 Accessing the Information	242
39 Firmware Revision	243
39.1 Device Connection Summary	243
39.2 Read Variable Access	244
40 SD Card Usage	245
40.1 Memory Card Specifications	245
41 Battery and Voltage	247
41.1 Variable Access	247
42 Real Time Clock (RTC)	249
42.1 Operation	249
43 Security and User Management	253
43.1 Areas of Security	253
44 System Error Flags Defined	259
44.1 Severity Type	259
44.2 Error List	260
45 System Errors Accessed	271
45.1 Variable Access	271
46 Watchdog/Outputs	273
47 Recipe Manager	275
47.1 Alternate Approach - CSV File	276
48 Scan Base for Modules	277
48.1 Operation	277
49 Memory Utilization	279
50 Retentive Memory	281
51 Login-Online Change	283
52 Trace	285
52.1 The Trace Object	285
53 Jitter Aspects	287
53.1 Display of Metrics	288
54 Modbus - ASCII Mode	289
54.1 Description	289
54.2 Workaround	289
54.3 Future Resolution	289

55 EtherNetIP - ACD	291
55.1 Description	291
55.2 Workaround	291
55.3 Future Resolution	291
56 “Retain-Persistent” NV Storage	293
56.1 Description	293
56.2 Workaround	293
56.3 Future Resolution	293
57 Task Times > 1 second	295
57.1 Description	295
57.2 Workaround	295
57.3 Future Resolution	295
58 “Symbol Configuration”	297
58.1 Description	297
58.2 Workaround	297
58.3 Future Resolution	297
59 Rapid Switching of Run/Stop	299
59.1 Description	299
59.2 Workaround	299
59.3 Future Resolution	299
60 Implicit Checks “CheckBounds”	301
60.1 Description	301
60.2 Workaround	301
60.3 Future Resolution	305
61 Introduction	307
62 P2CDS-622 System Hardware Centric	309
62.1 P2CDS-622 Support Forum	309
63 CODESYS Centric	311
63.1 CODESYS Textbook	311
63.2 CODESYS Online Help	311
63.3 CODESYS Support Forum (Forge)	311
63.4 CODESYS Official YouTube	311
63.5 CODESYS Users YouTube	311
63.6 CODESYS Trainings/Classes	312
63.7 CODESYS Store	312

Welcome to the P2CDS-622 System

World Class Industry Proven Hardware & **IEC 61131-3!**



Multiple Topologies and IO Module Options

- ✓ 600MHz Arm A5 Core CPU
- ✓ 4, 7, 11 and 15 Slot Base Options
- ✓ 52 Different IO Module Types
 - Analog Input and Output
 - Discrete Input and Output
 - Temperature and more!
- ✓ 3 Power Supply Options



New to CODESYS? Click here- [CODESYS Start](#).

Ready for First P2CDS-622 Design? Click here- [First Design](#).

Example Projects? Click here- [Example Projects](#).

No Hardware Yet? Click here- [WinPLC Emulation](#).

VERSION INFO

This section covers some key items that are updated on every new release of the Website, Runtime Software, etc.

Tip: All **screenshot images** in the website can be clicked on to be read more easily.

Tip: A **PDF** of the website can be generated from the lower left *Read the Docs* tab.

1.1 P2CDS-622 Firmware Version

1.1.1 Current

Version: 1.1.9

- Date of Release: 05/08/2024
- Items Updated:
 - NVL Max number errata resolved

1.1.2 Previous

Version: 1.1.8

- Date of Release: 09/25/2023
- Items Updated:
 - Firmware Update revision

1.2 Website Version

1.2.1 Current

Version: 1.5

- Date of Release: 5/08/24
 - “NVL Errata” removed.
 - FW 1.1.9 updated.

1.2.2 Previous

Version: 1.4

- Date of Release: 4/23/24
 - “Symbol Configuration” unsupported entry.

Version: 1.3

- Date of Release: 2/1/24
 - WebVisu clarification of features/limits.
 - Errata re. “Network Variables” and maximum number of variables allowed.

Version: 1.2

- Date of Release: 1/25/24
 - P2-8AD4DA-2 Voltage correction.
 - Add USB Device connection image.

Version: 1.1

- Date of Release: 1/24/24
 - Add additional reference to Modbus Simulator

Version: 1.0

- Date of Release: 1/23/24
 - Add PDF generation verbiage.
 - Add Status Signal e.g. “Module Not Ready” descriptors to RTD/NTC/THM.
 - Various Grammar/typos. Regen PDF to check.
 - TE updates incorp’d. Project screenshot typos in later revision.
 - Add “CheckBounds” errata.
 - Removed “License to Install” in Readme
 - Removed Previous history items
 - Add screenshot of Windows/USB device.
 - Update WebVisu section.
 - Update WebVisu Cert items.

- Update WebsVisu/Netconfig
- WebVisu browser cert items.

Version: 0.2 (Pre-Production release)

- Date of Release: 12/05/23
 - Add IEC61131 to Welcome page

PREPARATION

Caution: The only Fieldbuses and Add-ons that the P2CDS-622 will support are the Modbus TCP, Modbus RTU, EtherNet/IP Scanner and Adapter, the IIoT Library and WebVisu. Trying to install **other licenses** from the CODESYS Website, e.g OPC-UA **will not work**.

Tip: A PDF of the website can be generated under the lower left *Read the Docs* tab.

To begin using your P2CDS-622 PLC system, you will need to do the following steps.

- 1). Install the **CODESYS IDE**.
- 2). Install the latest **P2CDS-622 Firmware Image** file.
- 3). Install the **P2CDS-622 Device Package (.package)** file.
- 4). Configure the **Ethernet ports**
- 5). Install the **IIoT Library** file (optional if needed).

2.1 CODESYS IDE Install

The first task is to load and install the CODESYS Development system.

This can be found at the following CODESYS Store location: [S/W Download](#)

The bottom of the CODESYS download page links to a data sheet describing the main features of the CODESYS IDE.

2.2 P2CDS-622 Firmware Image

This step will initially use the USB Port of the PLC. To verify that your Host PC can enumerate the PLC's USB port do the following steps.

2.2.1 Host PC to P2CDS-622 connection

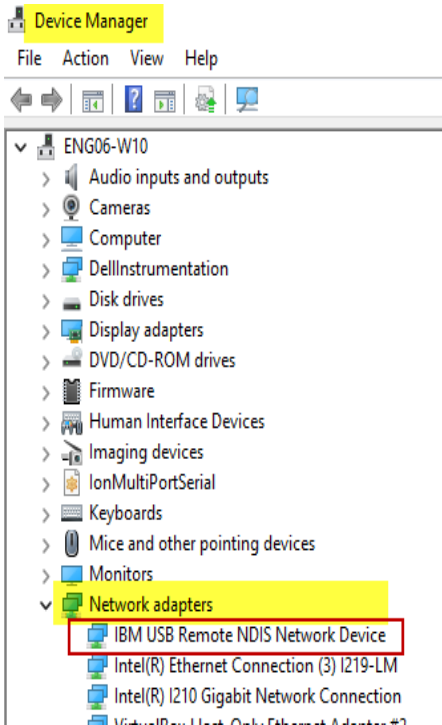
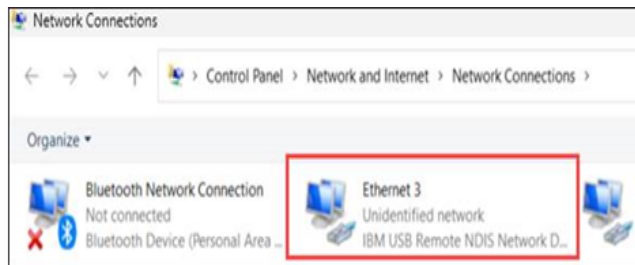
Go to the “Network and Sharing” section of your OS and note the Ethernet connections that are present.

We are going to use the “Virtual Ethernet over USB (RNDIS)” feature in the P2CDS-622 to use the USB communications port to connect.

- (1). With the P2CDS-622 powered off, connect the Host PC USB port to the P2CDS-622 front panel USB-C.
- (2). Power-up the P2CDS-622, and note in the PC’s “Network and Sharing” that a new Ethernet connection (P2CDS-622 Ethernet) is present.

This is the **USB port** of the P2CDS-622 and we want to make sure the PLC can be seen on the network.

For example in *Windows 11*:



2.2.2 Installation of F/W Utility Tool

The Firmware image file is the Runtime software that enables the P2CDS-622 lower level drivers to interface to the CODESYS IDE.

The Firmware Update section referenced below will have a link for the latest Firmware version.

Go to section [Firmware Update](#) for instructions on how to do this step to update your Firmware.

Tip: There may be releases of new Firmware image files from time to time. See [Latest Firmware](#) for the current revision.

After completing this step, continue with the following procedures.

2.3 P2CDS-622 Package Install

The P2CDS-622 CPU “Device” needs to be added to the CODESYS Library of available CPUs.

2.3.1 Installation

(1). Download the P2CDS-622 Package from [P2CDS-622 Package file](#).

Note: The file may download as a **.zip** e.g. P2CDS-622-Software-Package_vxxx.zip. If this occurs, you will need to rename the file extension to **.package** (P2CDS-622-Software-Package_vxxx.package).

(2). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

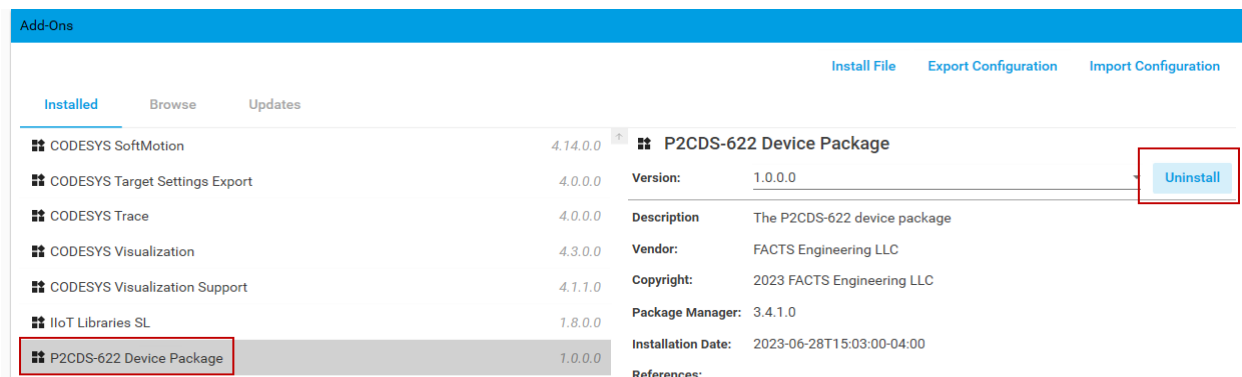
You can also click the CODESYS icon that is located on the desktop after installation.

2.3.2 Uninstall Current P2CDS-622 Device Library Package

Caution: If there is a **previous version** of the P2CDS-622 *.package* listed, it must be **removed** before installing new version.

(3). Select from the drop-down menus- **Tools > CODESYS Installer**

(4). Under *Add-ons* find the current package and click **Uninstall**. For example:



Note: You may have to **close CODESYS IDE** for the uninstall to complete.

2.3.3 Install the New P2CDS-622 Device Package

(5). If the Installer is not currently open, select from the drop-down menus- **Tools > CODESYS Installer**

(6). In the **CODESYS Installer** pop-up view, click on **Install File** under *AddOns*.

Note: The file may have downloaded as a **.zip** e.g. P2CDS-622-Software-Package_vxxx.zip. If this occurred, you will need to **rename the file extension** to a “.package” (P2CDS-622-Software-Package_vxxx.package) if so. You may need to enable extensions in Windows in order to do this.

(7). Browse to the **P2CDS-622-Software-Package_vxxx.package** file you just downloaded and double-click on it.

The install will start. You may have to **close the CODESYS IDE** to complete the installation.

(8). After the installation is complete, close the Installer and restart CODESYS.

Note: You will get “No signature was found in package” warning. Disregard and click lower left box to continue.

2.3.4 Verify the Device installed

(9). Open the CODESYS IDE and after creating/opening a project, and select **Tools > Device Repository...** Expand under the **PLCs** section, confirm the **P2CDS622** is present along with the associated revision you installed. If not, uninstall this version and repeat the P2CDS622 package installation.

2.4 Configure the Ethernet Port(s)

This will let you assign/edit the IP Address, etc. for the Ethernet ports on the P2CDS-622.

(10). Follow the steps defined in this link [Comms Setup](#).

Tip: After the Ethernet ports have been **initially configured** via the following steps, the Ethernet ports can be used in lieu of the USB-C for Firmware Update.

2.5 IIoT License Install

The P2CDS-622 CPU comes preinstalled with the license keys to enable the functions such as EtherNet/IP Scanner, Modbus TCP, etc. These are already included in the CODESYS library.

Some functions like MQTT require you to **install the associated IIoT library** from the CODESYS Store.

This is a **FREE library** but is not part of the default CODESYS IDE installation. The IIoT license key is preinstalled with the P2CDS-622.

If you do not require IIoT functions in your application, then there is no need to install this library.

2.5.1 IIoT Library

Download the library from this link- [IIoT Library Download](#)

The following are a few examples of the IIoT functions included in this library.

- Web Client
- MQTT Client
- Mail Service
- SNTP Service
- CSV Service

To install this library use the **CODESYS Installer** in the CODESYS IDE or the standalone Installer utility.

To use the Installer refer to the following link- [Installer](#) and the section near the bottom entitled **Installing an add-on package**.

P2CDS-622 STARTER PROJECT

This is an introductory project targeting the P2CDS-622 hardware and associated CODESYS Device package.

Tip: If you **do not yet have a P2CDS-622 system** and still want to learn the CODESYS IDE, there is an excellent tutorial referenced in [WinPLC Starter Project](#).

3.1 Introduction

The P2CDS-622 Intro project will default to using the four(4) slot Base P2-04B unit. If your system has a different Base, this can be changed within your project.

This project will not be using IO Modules. Its purpose is to demonstrate using the CODESYS debugger on the P2CDS-622 hardware.

The programming language chosen for this project will be “Structured Text” and will be a very basic counter.

The main purpose for this project is to introduce you to the P2CDS-622 CODESYS related items like installing the Device package, etc. There are many resources available online that explain the IEC61131-3 programming languages and the CODESYS IDE tool usage.

A few are referenced in the Support section- [CODESYS Support](#).

3.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 Power Supply (any variant)
- Latest version of CODESYS installed on Host PC
- Ethernet port(s) configured per [Ethernet](#) section
- Familiarity with the “Structured Text” programming language.

3.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and the Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any default passwords and update regularly.

If you want to publish a web visualization, we strongly recommend utilizing HTTPS and assign simple password protection to prevent access to the functionality of your PLC over the Internet.

3.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not.

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: In order to begin this project, the steps outlined in the *Preparation* section **must be completed** successfully.

3.5 Creating a Project

When creating a project, the main items that need to be done initially are to install the target device's (P2CDS-622) *Package* and pick the language that the program file will use.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU's.

3.5.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu. By default, the path is **Programs > CODESYS** (version).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

With the IDE open,

- (2). Click *File > New Project*

The example project here will be named **ExampleProj1**

- (3). In the **New Project** window, in the **Templates** section, select the **Standard Project** template.

- (4). Specify a name and a storage location for the project and click **OK**

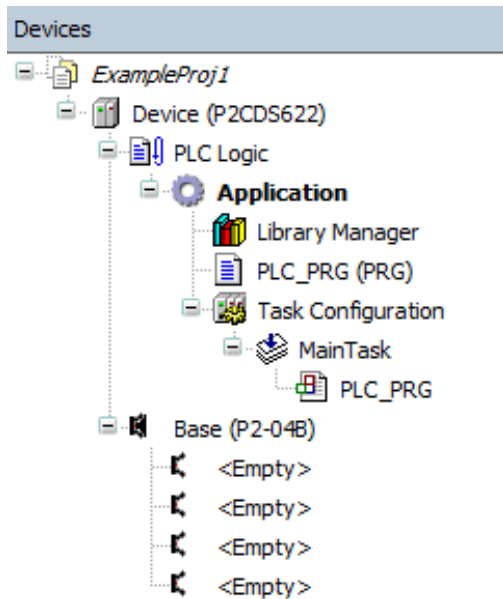
The project opens in the CODESYS **Standard Project** frame window.

- (5). In the **Device** drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create. The default name for this program is **PLC_PRG** which we will rename later to “**Main**”.

- (6). In the **PLC_PRG in** drop-down location, select **Structured Text (ST)**. This will be the language used in the first program.

Your Project Tree view should look similar to the image below with the *Name* of your project and the associated *Device* (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

Caution: If Ethernet ports have not been configured yet, it is required that they be setup per the *Ethernet* section!

Rename Design File and add Fieldbuses

- (7). Right Click on *PLC_PRG (PRG)* (not under *MainTask*) and in the drop-down select **Properties**. In the **Common** dialog box, rename *PLC_PRG* to *Main*. Click **OK**.
- (8). A dialog box will prompt regarding the name change to be applied over the entire project. Click **Yes**.
Another dialog box entitled **Refactoring** will pop up displaying all the areas in the project where this new name will be applied. Click **OK**.
- (9). If a Fieldbus (Modbus or EtherNet/IP) is to be used, it should be added at this point.

Add Fieldbus

(Not applicable for this project. Refer to the *Ethernet* section for more information if desired).

Change Base Type (if req'd)

If you are using a Base other than the P2-04B, you can change it now.

- (10). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**. The *Plug Device* dialogue box should appear.
- (11). Under the *Miscellaneous* name, find the desired Base unit device and double-click. Close the box.
The Device Tree should now show the updated Base.

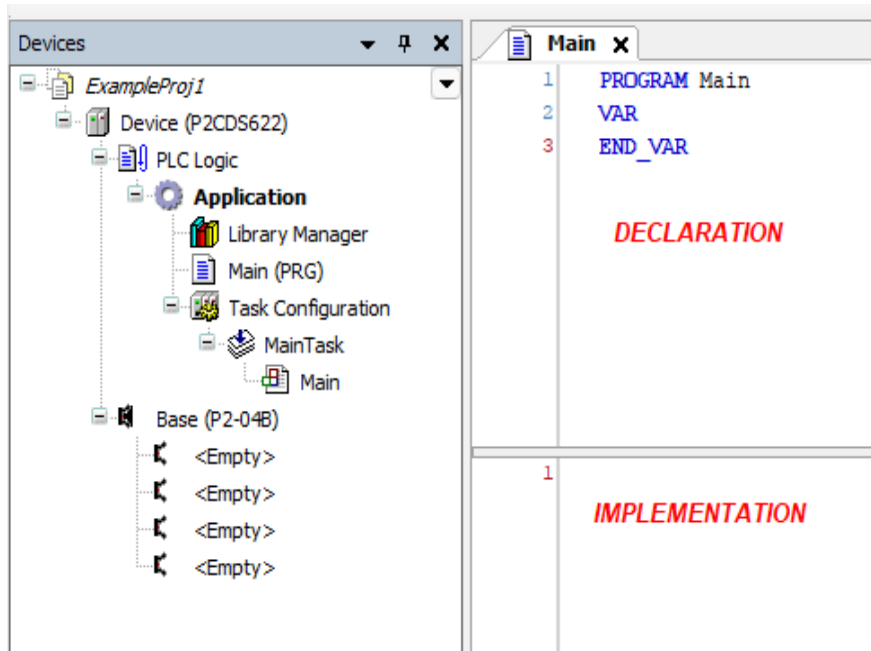
3.5.2 Write the Control Program

This simple program will be made up of two parts. One is a Function Block that will be the core counter function and the top-level code or the “Main” that calls the counter function.

Below is a view of the project with the Programmable Object Unit (POU) entitled *Main*.

- (12). Double-click on *Main* to view the program file.

All POU's are comprised of two sections, the *Declaration* part that on the top defines the variables and their associated Data Types and the *Implementation* section where the code is entered.



Variables can be declared first and then referenced in the code. If the code is written first, the user will be prompted to declare the variables before the project will successfully compile.

This project will declare the variables first then reference them.

Create a Counter Function Block

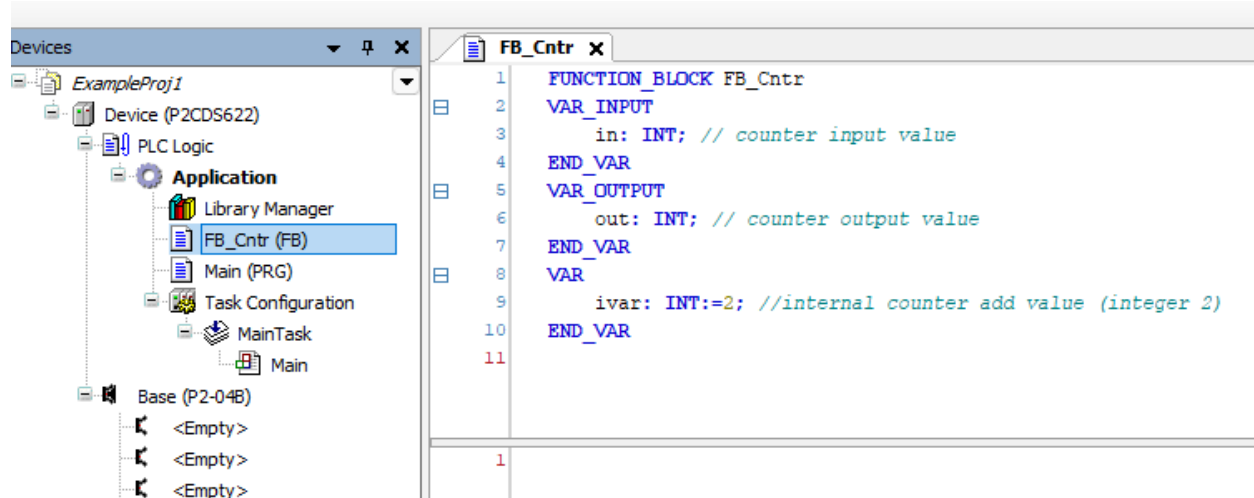
Add an additional POU that will be a Function Block using Structured Text.

This function will be a counter that adds the value '2' to the value contained in the input variable `in` and puts the result in the output variable called `out`.

(13). In the device tree, select the **Application** object. Select the **Project** menu drop-down, click **Add Object > POU**. An *Add POU* pop-up will appear.

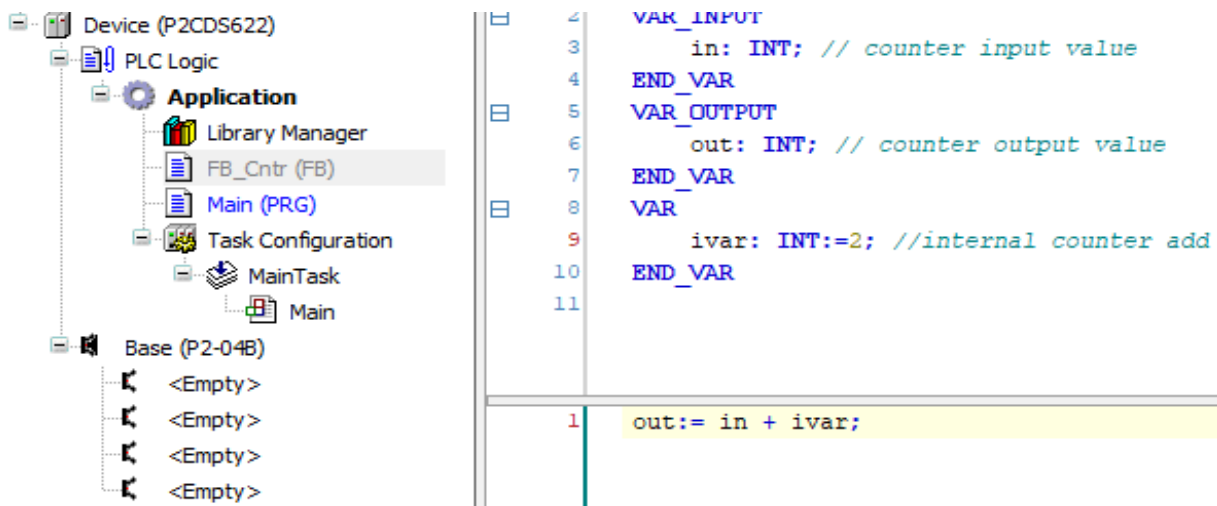
(14). In the *Name* area, type in the name "FB_Cntr" and for the *Type* of POU, select **Function Block**. For the **Implementation language**, select *Structured Text*. Click the **Add** button to confirm.

(15). Another editor window will open where the function `FB_Cntr` can be edited. Declare the following variables there. Comments may be added using the `"//"` delimiter (as shown below).



Enter the Counter Function Block Code

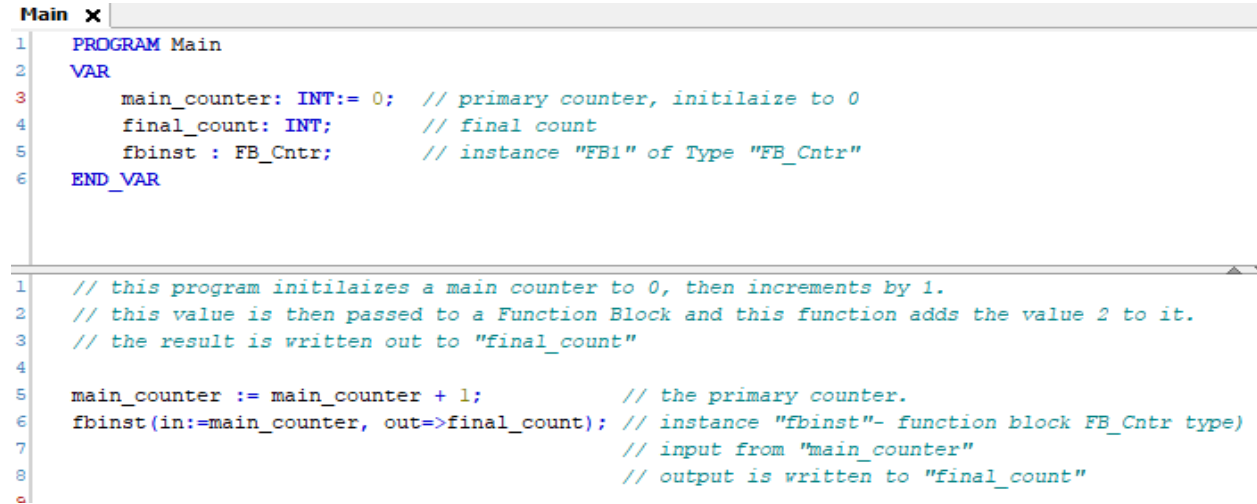
(16). In the *Implementation* for FB_Cntr, enter the following code that counts up by two (the declared value of internal variable ivar).



Type the Program Code into the Main Program

This main program will have a primary counter `main_counter` that increments by 1. This count then is fed into the function Block `FB_cntr` that will take this count and add 2 to it. The result will be put in `final_count`.

(17). In the Main program, enter the following variables and code.



```

Main x
1  PROGRAM Main
2  VAR
3      main_counter: INT:= 0; // primary counter, initilaize to 0
4      final_count: INT;      // final count
5      fbinst : FB_Cntr;      // instance "FB1" of Type "FB_Cntr"
6  END_VAR

1  // this program initilaizes a main counter to 0, then increments by 1.
2  // this value is then passed to a Function Block and this function adds the value 2 to it.
3  // the result is written out to "final_count"
4
5  main_counter := main_counter + 1; // the primary counter.
6  fbinst(in:=main_counter, out=>final_count); // instance "fbinst"- function block FB_Cntr type)
7                                          // input from "main_counter"
8                                          // output is written to "final_count"

```

A View of the “Task”

Another very important area in the Device Tree is the *Task Configuration* section. This defines the way in which the program tasks are setup (which POU's are to run on a scan) and the characteristics of the task (priority, speed etc.).

The online help goes into more detail for this area, but for this project, the default values are acceptable.

Compile the Project

Next, we will compile and build the project to download to the P2CDS-622 target.

(18). Select from the drop-down menus- **Build > Generate Code**.

3.6 Connect to PLC and Download

The project needs to be downloaded to the target, but we must first connect to it.

(19). Physically connect the Host PC to one of the Ethernet ports on the P2CDS-622.

3.6.1 Configure Connection Channel to PLC

(20). In the Device Tree, Double-click on the **Device(P2CDS622)**.

A **Device** tab should appear with the first tab on the left selected entitled *Communication Settings* selected. Now we need to scan the network to find our device.

(21). In the *Communication Settings* section, click on the **Scan network**.

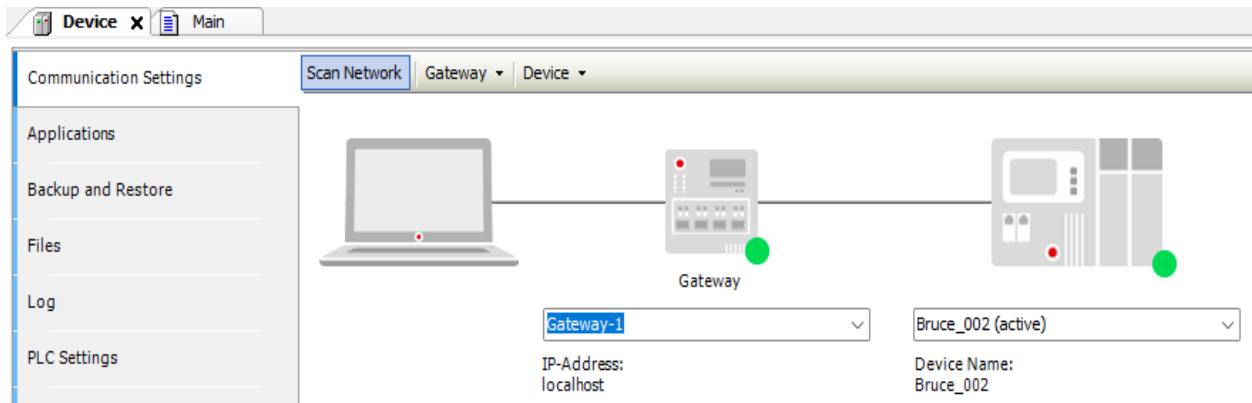
A dialog box should pop up with all the CODESYS PLCs available to connect to on this network.

Note: If you are unsure which unit is the one you want to connect to, click on the **Wink** button on the right side of the select box. This should cause the P2CDS-622 front panel **RUN** light to flash for 5 seconds.

(22). Select the applicable P2CDS-622 unit and Click **OK**.

(23). Toggle the **Run/Stop** switch.

Now the following should be visible- both the Gateway and the P2CDS-622 have the green indicators present indicating the IDE is connected to the P2CDS-622.



3.6.2 Set the CPUs Real Time Clock

(24). In the Device Tree, Double-click on the **Device(P2CDS622)** if the **Device** tab is closed.

(25). Select the *PLC Shell* label. Type in the lower dialogue the following command and the current local time as **rtc-set YYYY-MM-DDThh:mm:ss**.

For example, March 4th, 2023 at 1:07pm and 44 seconds → **rtc-set 2023-03-04T13:07:44**.

The **Device connection will disconnect** after this command, so reconnect again to the P2CDS-622.

To verify the time was set correctly, type **rtc-get** in the *PLC Shell* dialogue.

3.6.3 Download Application to PLC


(26). Click **Online > Login**.

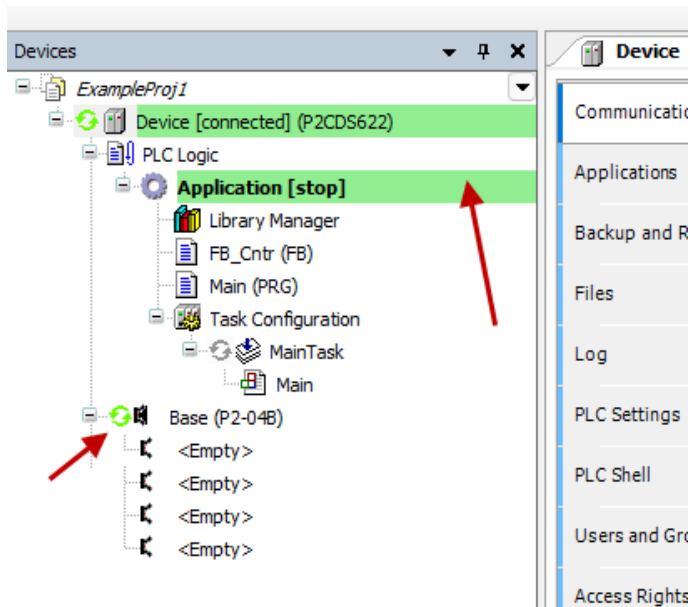
A dialog opens up if there is an existing application on the PLC, if so Click **Yes** to continue.

3.6.4 Successful Download. Ready to Run

The system should show green highlighted *Device* and *Application* as shown below:

Verify that the Run/Stop switch is in the **RUN** position.

The  icon indicates connection is active (but program is not executing).




3.7 Run and Watch the Application

This section will walk through starting the application and using the debugger features, which allow a user to watch the variables in real time, set breakpoints and force variables to a specific value.

3.7.1 Start the Application

(27). Click **Debug > Start**.

The IDE should now be showing a green “Run” in the lower portion of the status area. Also, the *Main Task* should have a green *Running* icon  next to it.

3.7.2 Watch the Application

There are several ways to monitor the variables of the application program and to influence them in the watch view:

- Online views of individual POU's
- Write and force variable values
- Defined variable lists in separate watch lists

Online View of the POU

This view of the POU shows the current values of the watchable expressions in a table format located in the *Declaration* section, and if enabled, also in the implementation section in the form of “inline monitoring”.

(28). To open the online view, double-click **Main** in the Device Tree, or right-click **Main** and select click *Edit Object* from the context menu.

In the bottom part of the view, you see the lines of code as specified in offline mode. They are supplemented by the inline monitoring views after each variable which show the variable’s current value.

In the top section, a table shows the watchable expressions (variables with Type and current Value) of the POU.

The screenshot shows the online view of the POU. At the top, there is a table titled "Device.Application.Main" with the following columns: Expression, Type, Value, Prepared value, Address, and Comment.

Expression	Type	Value	Prepared value	Address	Comment
main_counter	INT	19708			primary counter, initialize to 0
final_count	INT	19710			final count
fbinst	FB_Cntr				instance "FB1" of Type "FB_Cntr"

Below the table, there is a code editor showing the implementation of the POU. The code includes comments and inline monitoring values for variables:

```

1 // this program initializes a main counter to 0, then increments by 1.
2 // this value is then passed to a Function Block and this function adds the value 2 to it.
3 // the result is written out to "final_count"
4
5 main_counter := main_counter + 1; // the primary counter.
6 fbinst(in := main_counter, out => final_count); // instance "fbinst"- function block FB_Cntr type)
7 // input from "main_counter"
8 // output is written to "final_count"
9 RETURN
  
```

Writing and Forcing Variables

You can write or force a *Prepared Value* in the PLC’s variable. The dialog is used to prepare or set a value that will be used when a *force* command of that value is initiated.

For example *ivar*, write or force the value of this variable to 777d (decimal),

(29). In the **Prepared Value** column in the top section of the applicable program for the variable of interest e.g. *ivar*, select the field, press the spacebar to open an input field. Specify an integer value (777) and press the enter key or click outside of the field to close the field.

(30). In the menu, click **Debug > Write Values** or **Force Values**. You see the corresponding result in the *Value* column

Tip: The *Write Value* command writes the value for one(1) cycle then releases it (can be overwritten). The *Force* command will hold the value until the *Force* is removed. See the associated Online Help for further details.

Using Watch Lists

Watch lists can be used to organize the expressions/variables from the application that you want view together. This can be useful for debugging purposes when specific variables need to be checked at a glance.









(31). In the menu, click **View > Watch - Watch 1**. The view opens.

(32). In the *Expression* column, click in the first row to open an input field. Specify the path for the variable to be watched. We recommend that you use the Input Assistant for this by right-clicking in the blank expression field, and selecting **Input Assistant**. For this example, select **Device > Application > Main - main_counter**. Press the enter key to close the input field.

The data type of the variable is configured automatically in the row. Insert more rows for other variables if desired.

Any set of variables from all POUs of the project can be combined. Note in the case of instance variables, such as for those of `fbinst` instances, it is sufficient to specify the expression as `Main.fbinst`.

The individual parameters are inserted automatically and the corresponding rows can be opened by means of the plus sign at the front of each row. The current value of each expression is displayed in the value column.

Watch 1				
Expression	Application	Type	Value	Prepare
 Main.main_counter	Device.Application	INT	-28709	
  Main.fbinst	Device.Application	FB_Cntr		
  in		INT	-28709	
  out		INT	-28707	
 ivar		INT	2	

3.8 Debug the Application

One of the most powerful features of CODESYS is its debugging support capabilities. You can set multiple breakpoints and use various step-through the code commands.

3.8.1 Set Breakpoints

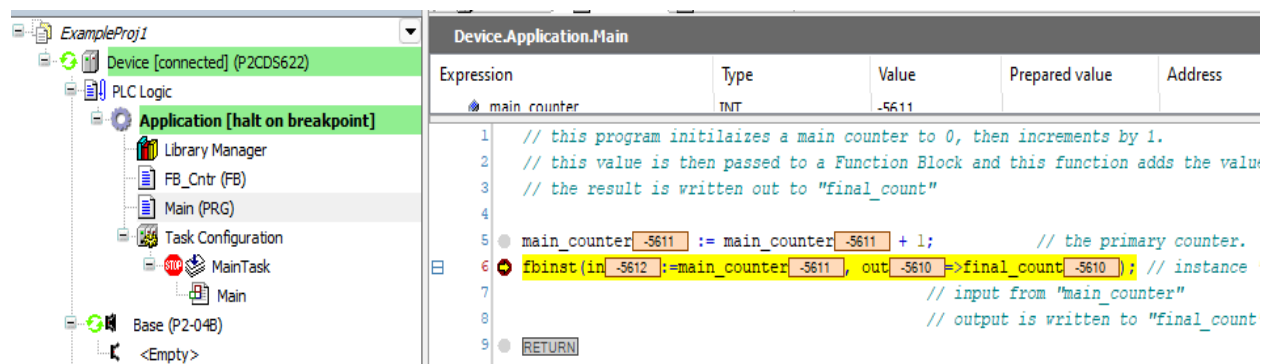
In Online/Login mode (not “Started” yet), you can define breakpoints where any program execution should be halted. When a breakpoint is reached, the program can be processed in steps. At each breakpoint and in each step, you can check the current value of the variables in the watch views.

(33). To demonstrate, in `Main`, set the cursor to line 6. In the menu, select **Debug > Toggle Breakpoint** to insert a Breakpoint where your cursor is.

The breakpoint is displayed in the program.

(34). Click **Debug > Start** to run the program.

The application will run to line 6 where the breakpoint is and halt. It will look like something like the following:



3.8.2 Stepping Through the Program

Now you can press <F8> repeatedly to execute the **Step Into** command in the Debug menu and run the program in single-step mode. This also steps into the function block instance.

(35). To skip the execution of a function block press <F10> to execute the **Step Over** command.

The current value of each variable is displayed at the processing position just reached.

Tip: The debugger can be single stepped also with the use of the Hot Keys that can be found in the following link: [Hot Keys](#)

(36). See also the Breakpoints dialog **View > Breakpoints** The currently defined breakpoints are listed here and can be edited or new ones can be added.

Note that the breakpoint positions are saved even when you log out of the PLC. The next time you log in, they will be displayed as light red markers and can be reactivated.

3.8.3 Execute Code Back to Beginning

(37). To restart debugging from the beginning of your program while still *Online*, select **Online > Reset Warm**.

This will take you to the beginning of the program in *Stop* mode and wait for you to initiate another *Start*.

CONGRATULATIONS! You just completed your first program on the P2CDS-622!

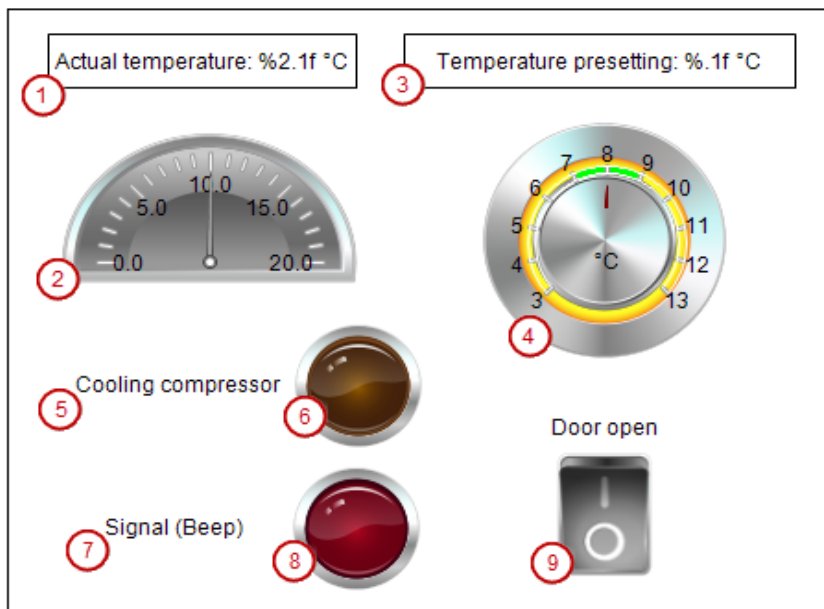
VISUALIZATION: WEBVISU

The P2CDS-622 contains advanced visualization/graphics generation capability to create an HMI called “WebVisu”.

Note: The P2CDS-622 WebVisu solution is not an “Application Based” CODESYS license and as such, does **not** have any maximum variable/data point restrictions. In addition, the P2CDS-622 supports all the WebVisu functionality and features.

It is an integrated web server that makes image and variable data available in a data network. This data can be displayed on any HTML5-capable display devices with a standard web browser.

There are numerous options to visualize static and animated images, some examples are shown below:



1. Numeric display of the actual temperature
2. Pointer to display of the actual temperature
3. Numeric display of the set temperature
4. Potentiometer for setting the set temperature
5. Label for compressor lamp
6. Lamp for compressor on
7. Label for signal lamp
8. Lamp for signal "Close doors"
9. Switch for opening and closing the refrigerator door

The WebVisu generates an IP Address for P2CDS-622 device and “serves up” a web page at that address. This URL can be HTTP or secure HTTPS. The following are some examples.

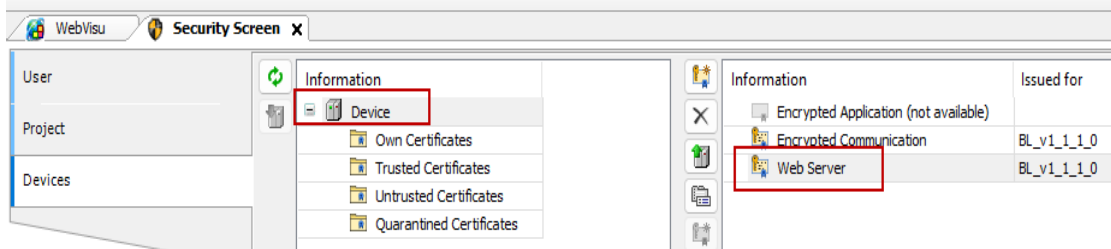
4.1 Quick Start

Caution: The P2CDS-622 is an entry level CODESYS PLC that runs on a single 600MHz core. As such, adding complex Visualizations may affect performance.

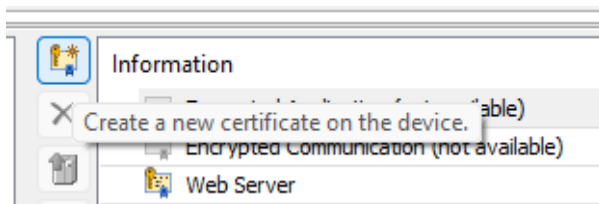
More detailed information is listed in the links below, but for a “Quick Start” to generate a secure HTTPS page, after adding a Visualization Object in the Application, follow the steps below.

- (1). If you do not have a project created, create one in any language.
- (2). Add the **NetConfig** device. This is used to identify the PLC's IP Address when needed later. Select **Device** in the project tree and Right-click. Select **Add Device > Miscellaneous > P2CDS_NetConfig**.
- (3). Connect to the P2CDS-622 via the **Scan Network > Select Device**.
- (4). Generate the security certificate that HTTPS will utilize. Select **View > Security Screen > Devices** and click on the green recycle to generate keys.

You should see something similar to the image below:

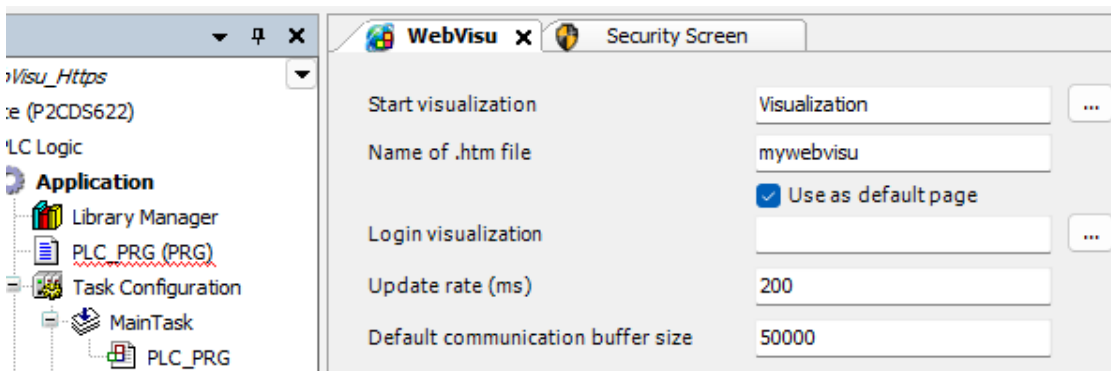


If you do not see the **Web Server** with a Certificate issued, double click on the symbol shown below, use default key lengths and click ok. It may take a minute, but a Certificate issued for the Web Server should appear.



Note: Some browsers e.g. Chrome **requires you to import** the WebVisu self-signed certificate into its browser certificates. Refer to online Chrome tutorials on how to do this. Firefox lets you continue to the WebVisu site without requiring this step.

- (5). Add the **Visualization** object by right-clicking on **Application** in the Device Tree, selecting **Add Object > Visualization**. Use default name and select **VisuSymbols** as **Active**.
- (6). Setup the name of WebVisu htm file by expanding the Visualization Manager and double-clicking on **WebVisu**. The image below shows the WebVisu name changed to *mywebvisu*. This name needs to be all lower case.



4.1.1 Obtain IP Address of PLC

In the next steps, you will identify the IP Address for the PLC. This will be the IP Address for the Web Server- *mywebvisu.htm*.

- (7). Build the project. Login and Download into PLC. Verify the green Run Icons are in the device tree.
- (8). Double-click on **P2CDS622_NetConfig**. Confirm which Ethernet port you are connected to and verify the IP Address desired. You will need the IP Address to be able to connect to the WebVisu from an external device.

4.1.2 Create a Visualization

- (9). Double-click on **Visualization** and create your HMI. Refer to the links below for detailed help.

The Web Server generated certificates will be used to enable an *HTTPS* connection.

Access your HMI via the secure URL HTTPS by using port 443. Note- 8080 is used for non-secure for HTTP.

- (10). In a browser type **https://x.x.x.x:443/mywebvisu.htm** where x.x.x.x = your IP address e.g. 192.168.5.67 from previous.

Your page should be displayed in the browser.

Note: The browser establishes a connection. If the certificate is not rated as trusted (due to be “self signed”), then a **security notice appears**. Click okay, etc. to continue. Some browsers like Chrome **may require installing this certificate** into it’s security section. Firefox does not require installation.

4.2 Detailed Help Areas

General Tutorial: [General Visu](#)

Multiple Data Types from Arrays, to Strings, etc. can be displayed. The following link gives an example of displaying some text and an associated integer value (%i Data Type) [Visu Mixed Text/Data](#)

The links below give additional information for incorporating Security into WebVisu utilizing HTTPS and security certificates:

[Secure1](#)

[Secure2](#)

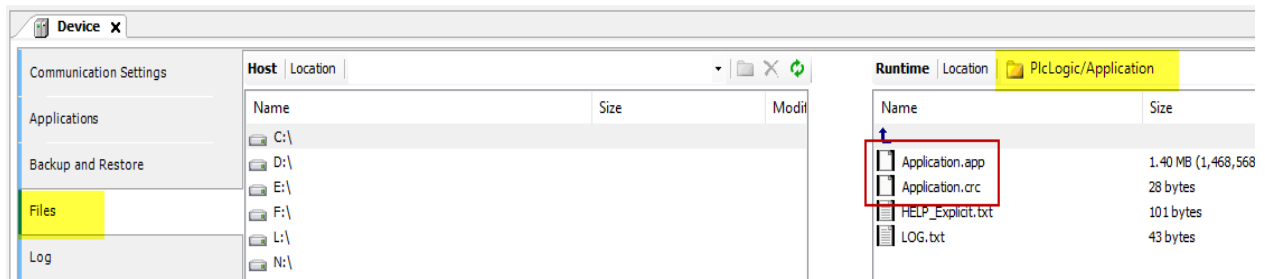
There are also many example videos online illustrating the WebVisu application.

CREATING A BOOT APPLICATION

A boot application (stored in non-Volatile Flash Memory) is the application version, which is started automatically when the controller is switched on or started. For this to happen, the application on the PLC has to exist as a <application name>.app file.

By default, CODESYS generates the boot application automatically when an application is downloaded and transfers them to the PLC.

The image below shows a PLC with the directory structure.



The Host section on the left is the Windows PC that the CODESYS IDE is running on and Runtime section on the right is the P2CDS-622 directory structure.

The files of interest are **Application.app** and **Application.crc**.

In the CODESYS IDE you can create a Boot application via a drop-down menu- **Online > Create Boot Application**. This is discussed further in [Boot Help](#).

A Boot application can also be generated with the *Login with Online Change* option, which writes the Boot code into **non-volatile Flash memory**. This is discussed further in the [Login-Online Change](#) section.

PC BASED (WIN PLC) STARTER PROJECT

This project can be used if you **do not have any PLC Hardware** (P2CDS-622 system) available and you want to learn/write some CODESYS code and run it. This is referred to as “Control WinP PLC” or a Windows PC based PLC that runs almost all the features of CODESYS within your PC.

The main item missing is the IO Modules the P2CDS-622 system would offer, but you can emulate these with some dedicated Input and Output variables you create.

6.1 Introduction

CODESYS has already prepared an excellent WinPLC “First Project” that is very comprehensive.

Click on this link [First Program](#) to get to the external website.

CODESYS DEVELOPMENT SYSTEM



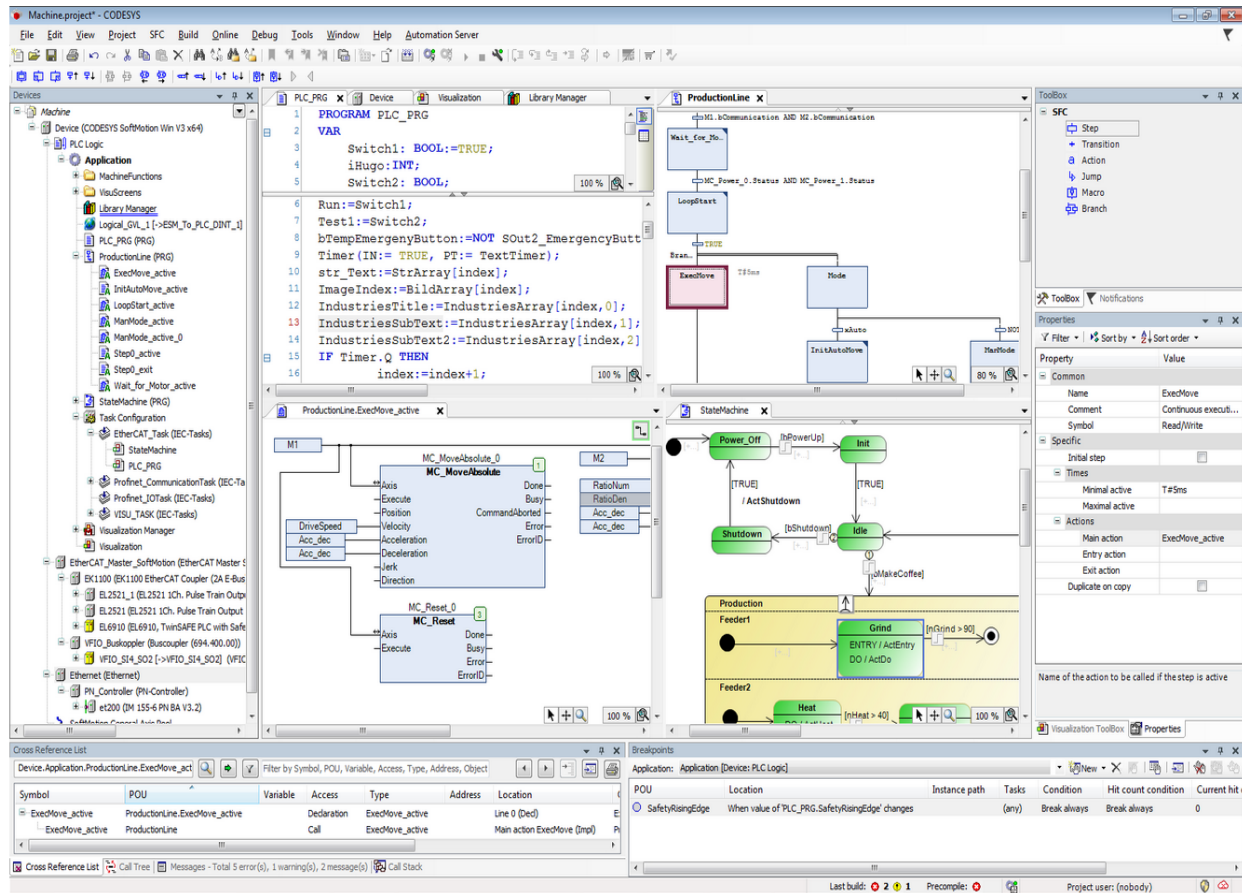
7.1 Introduction

What Exactly is CODESYS?

CODESYS is a comprehensive software suite used by automation specialists as a development environment for programming controller applications. It is an IEC 61131-3 programming tool that was developed by the Germany-based company, 3S (Smart Software Solutions) in order to provide users with integrated solutions that make the engineering of automated solutions more convenient.

Why Use CODESYS?

As systems get more complex and a single language (Ladder Diagram) does not adequately support these complexities, more optimized languages are required/demanded i.e. the IEC 61131 standard. Every one of the five PLC programming languages covered by the IEC 61131-3 standard — Ladder Diagram, Function Block Diagram, Sequential Function Chart, Structured Text, Instruction List and an additional highly powerful additional language called Continuous Function Chart — is supported by CODESYS.



CODESYS has become the dominant development environment for 61131 PLCs. There are over a million different devices — spread across 1,000 different device types — from over 400 manufacturers that are compatible with CODESYS. CODESYS effectively allows for systems designers to easily integrate the most complex automation components into their design using the IEC 61131-3 standard programming language, and customize their setup based on their application’s specific needs. In addition, utilizing Object Oriented Coding practices, the design becomes almost “self documenting” for ease of trouble shooting and handing off to future support personnel.

7.2 Functionality

The P2CDS-622 system, in addition to the full IEC61131-3 language support, has built-in the popular Modbus and EtherNet/IP Fieldbus systems, an HMI for visualizations (WebVisu) and the IIoT Library.

Languages supported in this package:

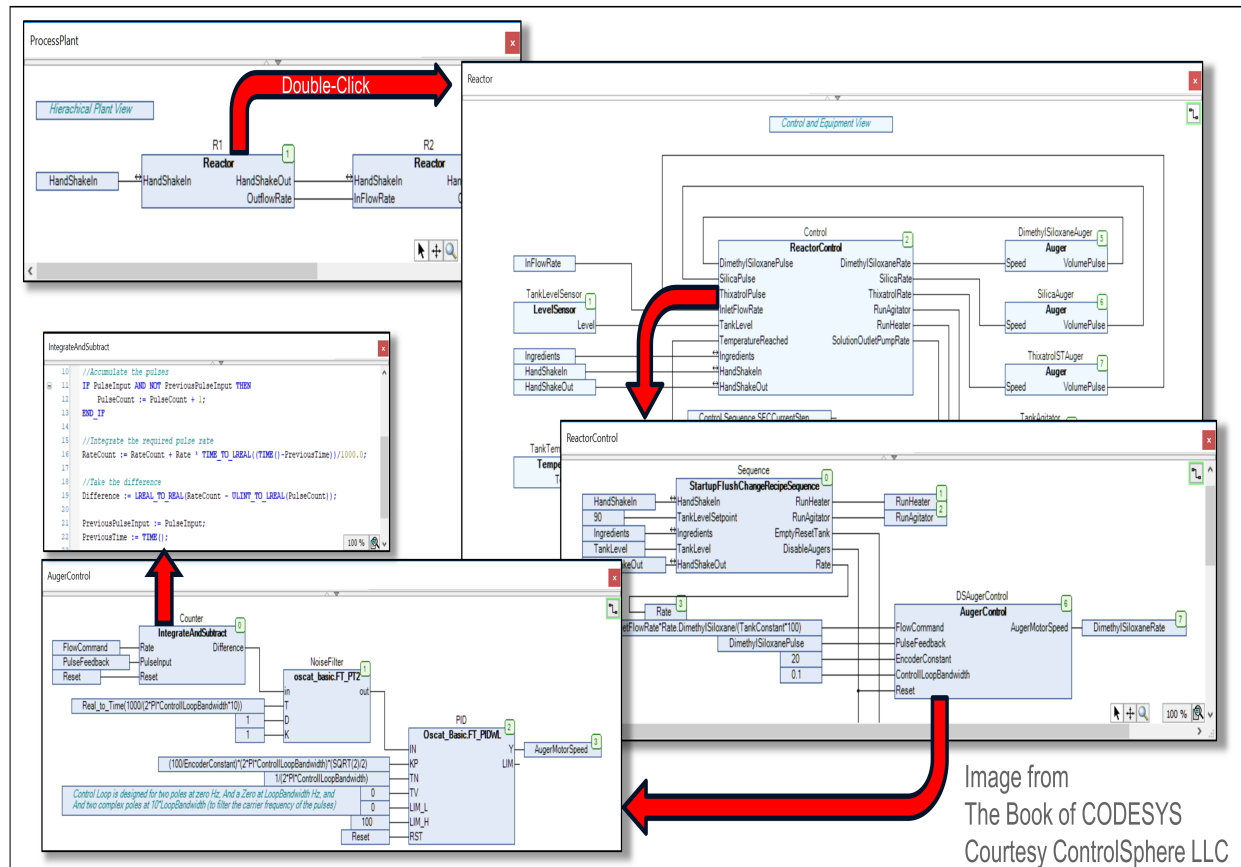
Language	Description	Ideal Applications
LD	Ladder Logic	Discrete Logic, Simple Combinatorial Logic
SFC	Sequential Function Chart	State Machines
ST	Structured Text	Math, Bitwise Logic, Complex Logic
FBD	Function Block Diagram	Reusable Complex Math and Logic
CFC	Continuous Function Chart	Block Diagrams, Hierarchical Object Oriented Designs

Note: IL (Instruction List) language is discontinued.

7.2.1 CFC Example

The following illustrates the Hierarchical/Block Diagramming capability of the CODESYS system that not only supports an **Object Oriented** design approach, but makes the overall system design significantly easier to discern the data flows, functionality, etc.

This is especially critical for Plant Maintenance personnel.

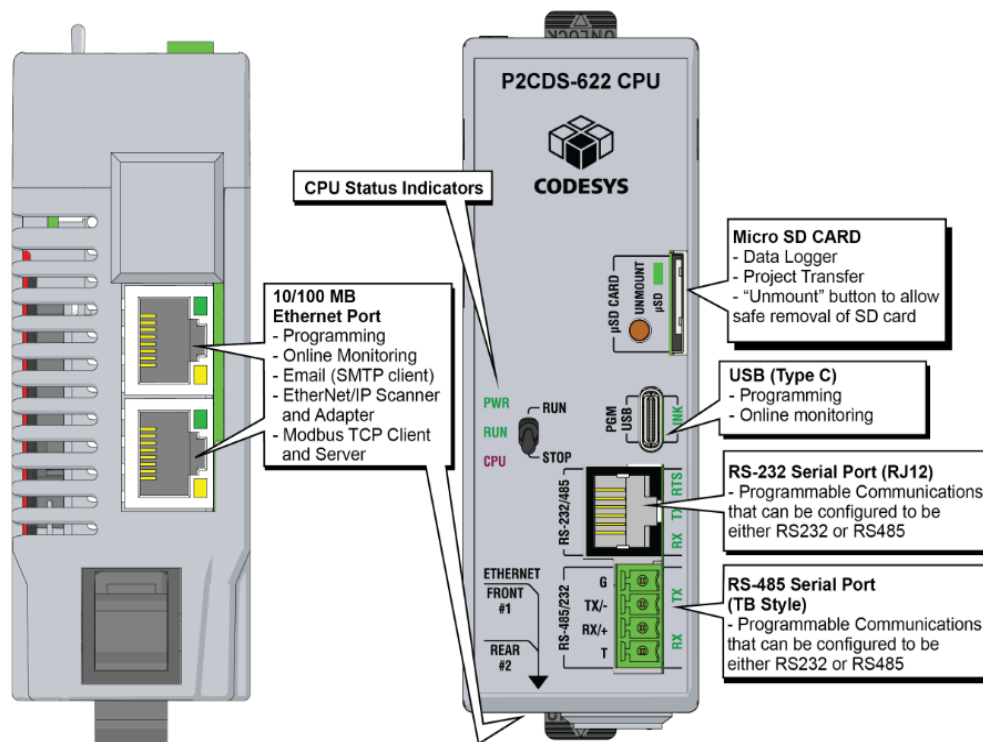


Caution: The only Fieldbuses and Add-ons that the P2CDS-622 will support are the Modbus TCP, Modbus RTU, EtherNet/IP Scanner and Adapter, the IIoT Library and WebVisu. Trying to install **other licenses** from the CODESYS Website, e.g OPC-UA **will not work**.

7.3 Other Resources

Learn more about CODESYS: [CODESYS Main](#)

CPU P2CDS-622



8.1 Introduction

The P2CDS-622 CPU is based on a 600MHz ARM A5 core and is built upon the Green Hills Software "INTEGRITY" Real-Time Operating System (RTOS)

The INTEGRITY RTOS is built around a partitioning architecture to provide embedded systems with total reliability, optimal security, and maximum real-time performance,

8.1.1 Specifications

Tip: A complete CPU datasheet can be downloaded here: [P2CDS-622 Datasheet](#)

General System Specs

General Specifications	
Operating Temperature	0° to 60°C (32° to 140°F)
Storage Temperature	-20° to 70°C (-4° to 158°F)
Humidity	5 to 95% (non-condensing)
Altitude	2,000 meters max
Pollution Degree	2
Environmental Air	No corrosive gases permitted
Vibration	IEC60068-2-6 (Test Fc)
Shock	IEC60068-2-27 (Test Ea)
Overvoltage Category	II
Heat Dissipation	4800mW
Enclosure Type	Open Equipment
Module Location	Controller slot in the local base in a Productivity2000 System
Weight	TBD
Agency Approvals	UL 61010-1 and UL 61010-2-201 File E139594, Canada and USA CE (EN 61131-2 EMC, EN 61010-1 and EN 61010-2-201 Safety)*

*Meets EMC and Safety requirements. See the D.O.C. for details.

CPU Specific Specs

CPU Specifications	
User Memory	50MB (Includes program, data and documentation)
Memory Type	Flash and Battery Backed RAM
Retentive Memory	1MB (Retain 800KB / Retain-Persistent 200KB)
Scan Time	550 μ s (5K Boolean Logic)
Interfaces	USB IN: USB 2.0 (single port), Program, Monitor, Debug, Firmware Update ETHERNET: Two independent 10/100Mbps RJ-45 connectors PROTOCOLS: Modbus TCP Master/Slave, Modbus Serial Master/Slave, EtherNet/IP Scanner/Adapter VISUALIZATION: "WebVisu" (Web Server) RS-232/485: RJ12 connector RS-232/485: 4-position Terminal Block
Data Logging	Micro SD card slot
Hardware Topologies	Four (4) Base Groups: Four (4), Seven (7), Eleven (11), and Fifteen (15) Slot Bases
IEC 61131-3 Supported Editor Types	Functional Block Diagram (FBD) Structured Text (ST) Sequential Function Charts (SFC) Ladder Diagram (LD) Continuous Function Chart (CFC)
Real Time Clock Accuracy	± 2 s per day typical at 25°C ± 10 s per day maximum at 60°C

Danger: The CPU can NOT be removed or inserted while **power is being applied!**

8.1.2 Functionality

RUN/STOP Switch

This switch puts the CPU in *Run* mode (executing the program) or in a halted *Stop* mode.



CPU Status Indicators

The following LEDs indicators next to the switch are used to display the states of the CPU:

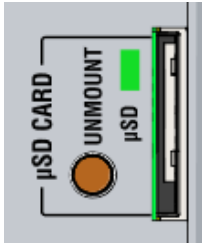
- **PWR** - CPU has power applied to it. Green = Powered, Off = Unpowered.
- **RUN** - CPU is in “Run” mode (executing the program tasks). Green = Run, Off = Stopped.
- **CPU** - CPU is in a reset condition or watch-dog timeout has occurred (Red) . Off = Normal operation.

Micro SD Card/Unmount Button

The SD Card can be used for Data Logging in the project.

When an SD Card is inserted, the “uSD” LED will flash green a few times then stay on steady green.

The “Unmount” button is pressed prior to removing the SD Card. When pressed, the **uSD** LED will flash momentarily during the unmounting process. When the uSD LED turns off, this indicates it is safe to remove the SD Card.



For technical specifications of the Memory Card refer to the section *SD Card Usage*.

USB - Type C

This is a standard USB-C Slave input for programming and online monitoring, with built-in surge protection.

The transfer rate is 480Mbps.

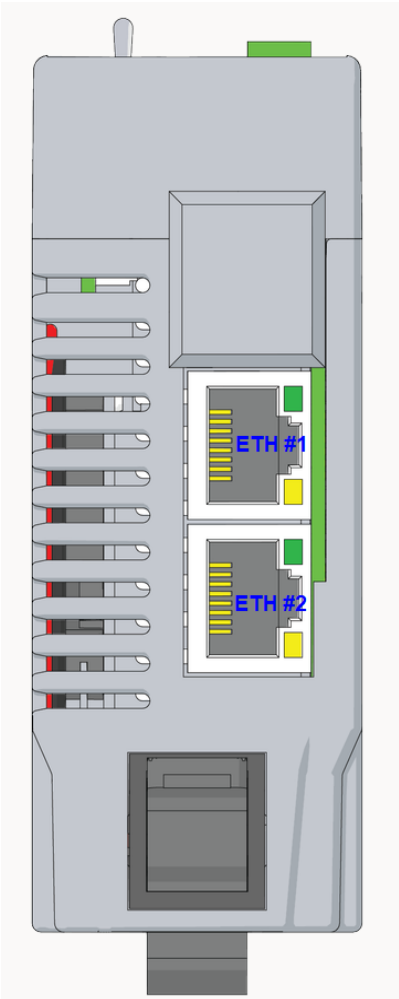


- **LINK** - USB connection status. Green LED illuminated when connected (LINK established).

Ethernet

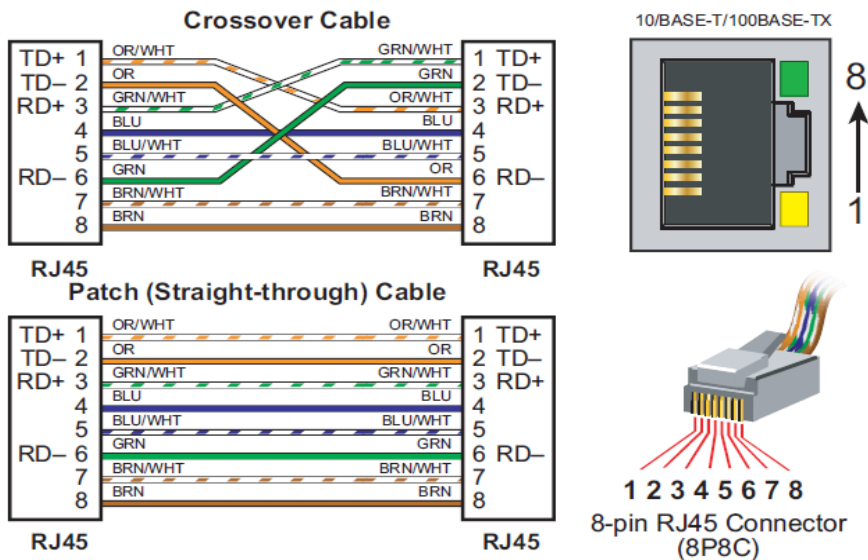
There are two (2) 10/100MB Ethernet ports located on the bottom of the CPU that are run independently from each other.

See- *Ethernet* for more information on how to configure the Ethernet ports.



Bottom View of CPU

The RJ-45 pinout is shown below:

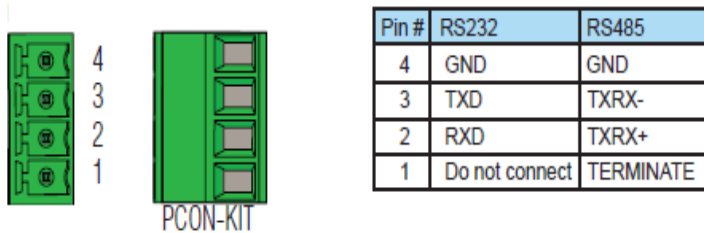


The section in *CPU Communications* entitled *Ethernet* describes these ports in more detail and how to configure them in the CODESYS IDE.

Tip: There is **automatic crossover detection** of which cable is inserted, so either cable will work.

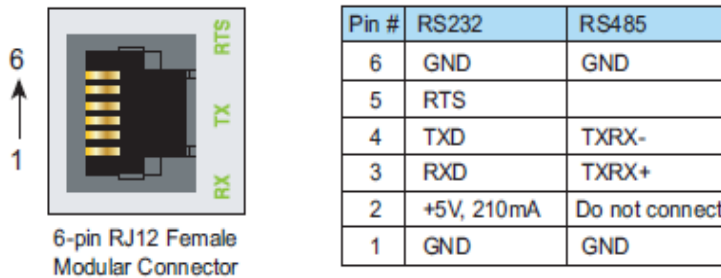
RS-485 Serial Port (TBLK)

The 4-position Terminal Block shown below supports RS-485 communications and is detailed in the section *CPU Communications* under *Serial RS-232/RS-485*.



RS-232 Serial Port (RJ-12)

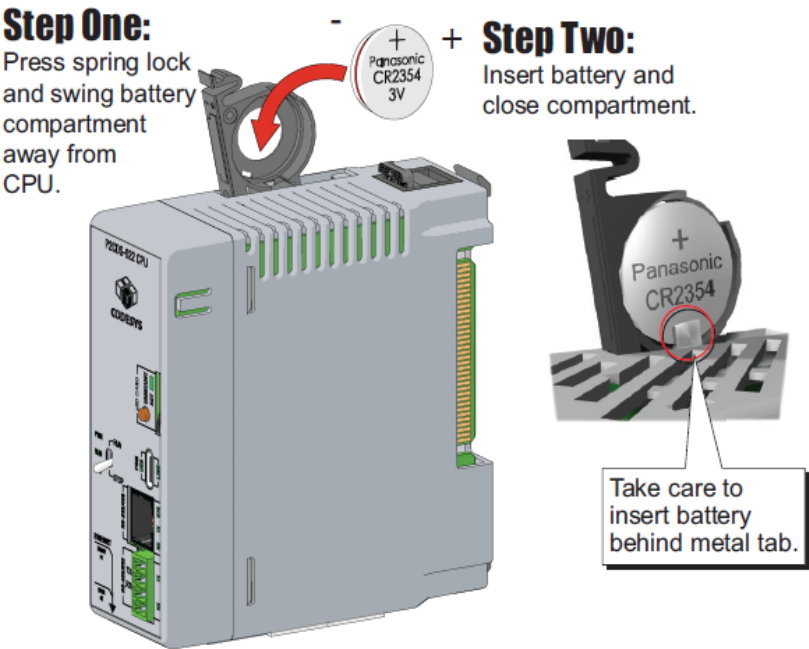
The RJ-12 connector is shown below and supports RS-232 or RS-485 communications (selected within the CODESYS IDE) and is detailed in the section *CPU Communications* under *Serial RS-232/RS-485*.



Battery

The Battery-backed SRAM is supplied the backup power via a coin cell battery in the unit as shown below.

Tip: There is a *Super-Cap* device that backs up the SRAM data when the Battery is removed. The backup time is a **minimum of 8 hours** with no loss of SRAM data.



Battery (Optional)

D2-BAT-1	Coin type, 3.0 V Lithium battery, 560mA, battery number CR2354
----------	--

Note: Although not needed for program backup, an uninstalled battery is included with the P2CDS-622. Install this battery if you want the CPU to retain the Time and Date along with any tags you have configured as retentive.

POWER SUPPLY



9.1 Introduction

The P2CDS-622 CPU system utilizes the Productivity2000 power supplies that provide the DC power needed for proper operation of CPU and I/O bases in a system.

Models are available as shown below with various input voltage topologies, and can support any I/O module combination in a Base with no power budgeting requirements.

Nomenclature	Type	Description
P2-01AC	Productivity2000 AC base power supply	100-240 VAC/125VDC Input
P2-01DC	Productivity2000 DC base power supply	24-48 VDC Input
P2-02DC	Productivity2000 DC base power supply	24 VDC Input
P2-01DCAC	Productivity2000 AC/DC base power supply	12-24 VDC/24 VAC Input

9.2 Functionality

The functionality and associated spec sheets for each Power Supply can be obtained from the following links:

- AC Power Supply: [P2-01AC](#)
- DC Power Supply: [P2-01DC](#)
- DC Power Supply: [P2-02DC](#)
- AC/DC Power Supply: [P2-01DCAC](#)

Caution: The “Download Software” links in these pages **do NOT apply** to the P2CDS-622 CPU system.

BASES

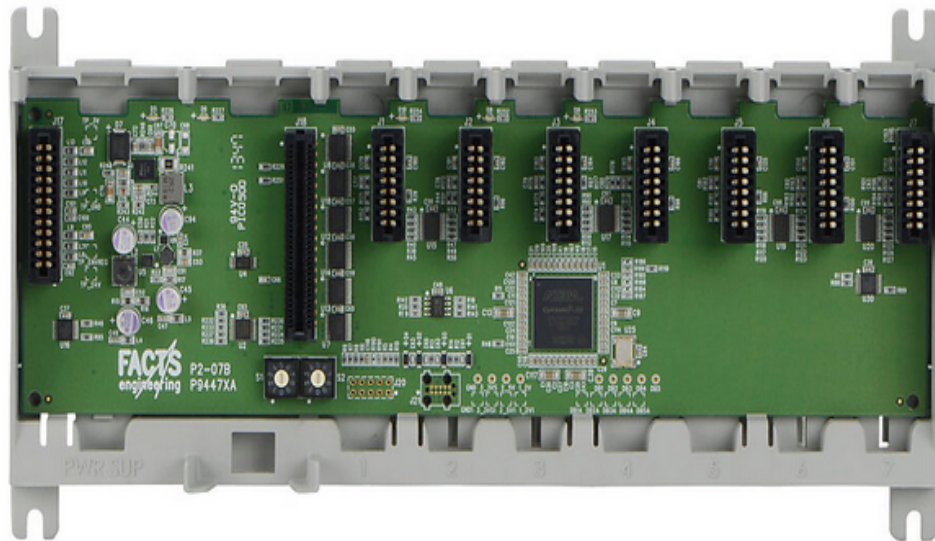


Fig. 1: 7-Slot Base Example

10.1 Introduction

The P2CDS-622 CPU system utilizes the Productivity2000 Base units and provide the means of housing and connecting the installed P2CDS-622 CPU module, power supply, and selected I/O modules.

These modular bases are available in the following topologies depending on the number of IO Modules a system requires.

Nomenclature	Type	Description
P2-04B	Productivity2000 4-Slot Base Unit	Hold up to Four(4) I/O Modules
P2-07B	Productivity2000 7-Slot Base Unit	Hold up to Seven(7) I/O Modules
P2-11B	Productivity2000 11-Slot Base Unit	Hold up to Eleven(11) I/O Modules
P2-15B	Productivity2000 15-Slot Base Unit	Hold up to Fifteen(15) I/O Modules

Note: Any of the desired Power Supplies that are listed in the P2000 Power Supply system will work in any of the

Bases.

Power Supply Reference Section: [Introduction](#)

10.2 Functionality

The functionality and associated spec sheets for each Base unit can be obtained from the following links:

- 4-Slot Base: [P2-04B](#)
- 7-Slot Base: [P2-07B](#)
- 11-Slot Base: [P2-11B](#)
- 15-Slot Base: [P2-15B](#)

IO MODULES



11.1 Introduction

The following P2000 system IO Modules are supported in the P2CDS-622 ecosystem.

Note: All Analog modules are **16-bit resolution** unless otherwise noted.

Nomenclature	Type	Description
P2-08ND3-1	Discrete Input - 8 point DC	12-24VDC, Sink/Source
P2-16ND3-1	Discrete Input - 16 point DC	12-24VDC, Sink/Source
P2-32ND3-1	Discrete Input - 32 point DC	12-24VDC, Sink/Source
P2-08NE3	Discrete Input - 8 point DC	24VAC/VDC, Sink/Source
P2-16NE3	Discrete Input - 16 point DC	24VAC/VDC, Sink/Source
P2-32NE3	Discrete Input - 32 point DC	24VAC/VDC, Sink/Source
P2-08NAS	Discrete Input- 8 point AC	120VAC
P2-16NA	Discrete Input - 16 point AC	120-240VAC
P2-08TD1S	Discrete Output - 8 point DC	3.3-24VDC, Sink
P2-08TD2S	Discrete Output - 8 point DC	12-24VDC, Source
P2-08TD1P	Discrete Output - 8 point DC	12-24VDC, Sink
P2-16TD1P	Discrete Output - 16 point DC	12-24VDC, Sink
P2-32TD1P	Discrete Output - 32 point DC	12-24VDC, Sink
P2-08TD2P	Discrete Output - 8 point DC	24VDC, Source
P2-16TD2P	Discrete Output - 16 point DC	24VDC, Source
P2-32TD2P	Discrete Output - 32 point DC	24VDC, Source
P2-15TD1	Discrete Output - 15 point DC	3.3-24VDC, Sink
P2-15TD2	Discrete Output - 15 point DC	12-24VDC, Sink
P2-08TAS	Discrete Output - 8 point AC	120VAC

continues on next page

Table 1 – continued from previous page

Nomenclature	Type	Description
P2-16TA	Discrete Output - 16 point AC	120-240VAC
P2-04AD	Analog Input - 4 Ch. Current/Voltage	0-20mA, 0-5V/10V, +/-5V/10V
P2-04AD-1	Analog Input - 4 Ch. Current	0-20mA
P2-08AD-1	Analog Input - 8 Ch. Current	0-20mA
P2-16AD-1	Analog Input - 16 Ch. Current	0-20mA
P2-08ADL-1	Analog Input - 8 Ch. Current	0-20mA (13bit resolution)
P2-16ADL-1	Analog Input - 16 Ch. Current	0-20mA (13bit resolution)
P2-04AD-2	Analog Input - 4 Ch. Voltage	0-10V
P2-08AD-2	Analog Input - 8 Ch. Voltage	0-10V
P2-16AD-2	Analog Input - 16 Ch. Voltage	0-10V
P2-08ADL-2	Analog Input - 8 Ch. Voltage	0-10V (13bit resolution)
P2-16ADL-2	Analog Input - 8 Ch. Voltage	0-10V (13bit resolution)
P2-04DA-1	Analog Output - 4 Ch. Current	4-20mA
P2-08DA-1	Analog Output - 8 Ch. Current	4-20mA
P2-16DA-1	Analog Output - 16 Ch. Current	4-20mA
P2-04DAL-1	Analog Output - 4 Ch. Current	4-20mA (12bit resolution)
P2-08DAL-1	Analog Output - 8 Ch. Current	4-20mA (12bit resolution)
P2-16DAL-1	Analog Output - 16 Ch. Current	4-20mA (12bit resolution)
P2-04DA-2	Analog Output - 4 Ch. Voltage	+/-10V
P2-08DA-2	Analog Output - 8 Ch. Voltage	+/-10V
P2-16DA-2	Analog Output - 16 Ch. Voltage	+/-10V
P2-04DAL-2	Analog Output - 4 Ch. Voltage	+/-10V (12bit resolution)
P2-08DAL-2	Analog Output - 8 Ch. Voltage	+/-10V (12bit resolution)
P2-16DAL-2	Analog Output - 16 Ch. Voltage	+/-10V (12bit resolution)
P2-8AD4DA-1	Analog Combo - Current 8 Ch. In/4 Ch. Out	4-20mA
P2-8AD4DA-2	Analog Combo - Voltage 8 Ch. In/4 Ch. Out	0-5V, 0-10V
P2-08THM	Temperature - 8 Ch. Thermocouple	Thermocouple, Voltage Input
P2-08NTC	Temperature - 8 Ch. Thermistor	Thermistor Sensor Input
P2-06RTD	Temperature - 6 Ch. RTD	RTD Sensor Input
P2-08TRS	Relay Output - 8 point	6-24Vdc/6-120Vac Isolated
P2-16TR	Relay Output - 16 point	6-24Vdc/6-120Vac
P2-04PWM	PWM Output - 4 Ch.	5-24Vdc, Sink/Source
P2-08SIM	Simulator Inputs - 8 point	Logic switch 1 or 0 Input

11.2 Functionality and Purchase

The functionality and specification sheets for each IO Module can be obtained from the following links.

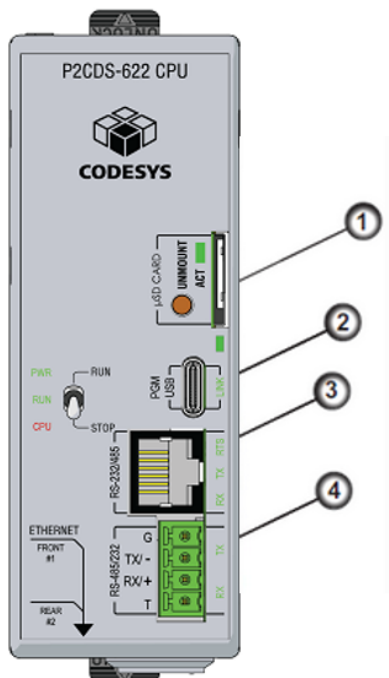
- [Discrete Input](#)
- [Discrete Output](#)
- [Analog Combo](#)
- [Analog Input](#)
- [Analog Output](#)
- [Relay Output](#)
- [PWM Output](#)
- [Simulator Inputs](#)

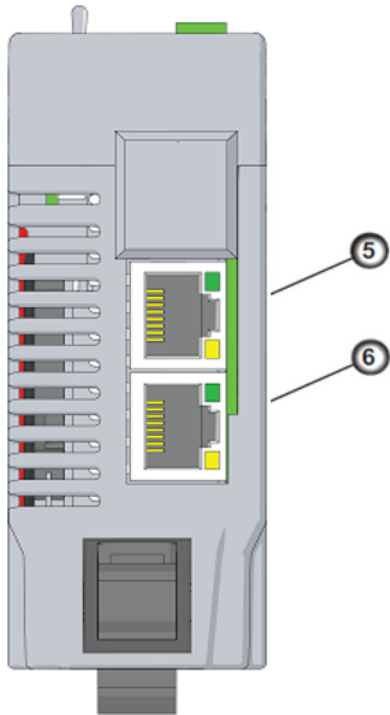
PHYSICAL PORTS

The P2CDS-622 CPU is provided with several Communications Ports. A detailed description of each of these ports are in the sections below.

The Communication Ports are:

Item #	Communication Port
1	MicroSD Card
2	USB-C Programming Port
3	RS-232 or RS-485 Serial Port (RJ12)
4	RS-485 or RS-232 Serial Port (TBLK)
5	10/100 MB Ethernet Port #1
6	10/100 MB Ethernet Port #2



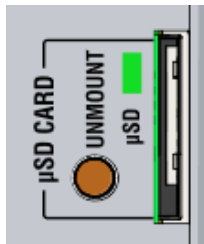


12.1 Micro SD Card/Unmount Button

The SD Card can be used for Data Logging in the project.

When an SD Card is inserted, the “uSD” LED will flash green a few times then stay on steady green.

The “Unmount” button is pressed prior to removing the SD Card. When pressed, the **uSD** LED will flash momentarily during the unmounting process. When the uSD LED turns off, this indicates it is safe to remove the SD Card.



12.2 USB-C Programming Port

This is a standard USB-C Slave input for programming and online monitoring, with built-in surge protection.

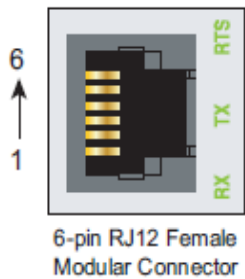
The transfer rate is 480Mbps.



- **LINK** - USB connection status. Green LED illuminated when connected (LINK established).

12.3 RS-232/RS-485 Serial Port (RJ12)

The RJ-12 connector is shown below and supports RS-232 or RS-485 communications (programmable in the IDE) and is detailed in the section *CPU Communications* under *Serial RS-232/RS-485*.



Pin #	RS232	RS485
6	GND	GND
5	RTS	
4	TXD	TXRX-
3	RXD	TXRX+
2	+5V, 210mA	Do not connect
1	GND	GND

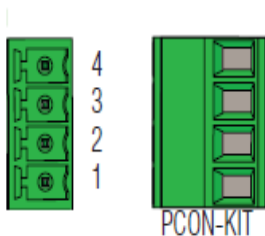
This port can be used for:

- Modbus RTU Master Connections
- Modbus RTU Slave Connections

Modbus detail functionality is covered in the Modbus Section.

12.4 RS-485/RS-232 Serial Port (TBLK)

The 4-position Terminal Block shown below supports RS-485 communications and is detailed in the section *CPU Communications* under *Serial RS-232/RS-485*.



Pin #	RS232	RS485
4	GND	GND
3	TXD	TXRX-
2	RXD	TXRX+
1	Do not connect	TERMINATE

This port can be used for:

- Modbus RTU Master Connections
- Modbus RTU Slave Connections

Modbus RTU detail functionality is covered in the Modbus Section here- [Modbus](#).

12.5 10/100 MB Ethernet Port (x2)

The Ethernet ports are based on 10/100Base-T Ethernet with an RJ-45 style connector.

This port can be used for:

- Connection to a PC running the CODESYS Suite Development System.
- Modbus TCP Client connections (Modbus requests sent from the CPU).
- Modbus TCP Server connections (Modbus requests received by the CPU).
- EtherNet/IP Scanner (32 Adaptors)
- EtherNet/IP Adapter (4 scanners) with 8 connections per device.
- MQTT protocol
- Outgoing Email
- WebVisu enabled HMI

Ethernet Port Configuration detail in *Ethernet*.

Modbus TCP detail functionality is covered in the Modbus Section- *Modbus*.

EtherNet/IP detail functionality is covered in the EtherNet/IP Section- *EtherNet/IP*.

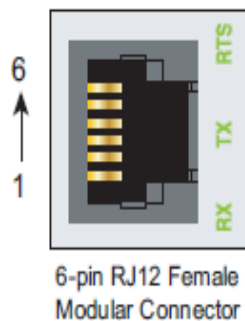
SERIAL RS-232/RS-485

Note:

For future reference, in the CODESYS IDE, **RJ-11 connector** corresponds to COM port #1, **4-pin TBLK** is Com port #2.

Refer to Modbus RTU example project for more details at *Modbus RTU*.

13.1 RS-232/RS-485 Serial Port (RJ-12)

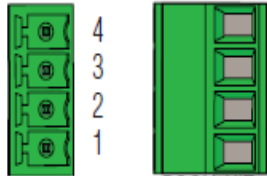


Pin #	RS232	RS485
6	GND	GND
5	RTS	
4	TXD	TXRX-
3	RXD	TXRX+
2	+5V, 210mA	Do not connect
1	GND	GND

RS-232 Specifications	
TxD	RS-232 Transmit output
RxD	RS-232 Receive input
RTS	“Request To Send” Handshake for modem control (RJ12 only).
GND	Logic ground
Maximum Output Load (TXD/RTS)	3Kohm, 1000pF
Maximum Output Voltage Swing	+/-5V
Output Short Circuit Protection	+/-15mA

RJ12 Connector	
Description	Programmable RS232/485 Port - Non-isolated RS-232 DTE port connects the CPU as a Modbus master or slave to a peripheral device. Includes ESD and built in surge protection - Non-isolated RS-485 port connects the CPU as a Modbus master or slave to a peripheral device. Includes ESD/EFT protection and automatic echo
Data Rates (Baud)	Selectable: 1200, 2400, 4800, 9600, 19200, 33600, 38400, 57600, and 115200.
+5V Cable Power	210mA maximum at 5V, $\pm 5\%$. Reverse polarity and overload protected.
Port Status LED	Green LEDs illuminated when active for TXD, RXD and RTS
Cable Options (AutomationDirect.com)	EA-MG-PGM-CBL D2-DSCBL USB-RS232 with D2-DSCBL FA-CABKIT

13.2 RS-485/RS-232 Serial Port (TBLK)

 PCON-KIT	4	Pin #	RS232	RS485
	3	4	GND	GND
	2	3	TXD	TXRX-
	1	2	RXD	TXRX+
		1	Do not connect	TERMINATE

RS-485 Specifications	
TXD+/RXD-	RS-485 transceiver high
TXD-/RXD-	RS-485 transceiver low
GND	Logic Ground
Termination Resistance (TBLK Jumper Wire "Terminate" to "TXRX+")	120 Ω . To use, add jumper between Pin 1 and Pin2 (enables internally connected 120 Ω resistors)
Maximum Load	Fifty (50) transceivers, 19K Ω ea. 60 Ω termination
Output Short Circuit Protection	± 250 mA, thermal shut-down protection
ESD Protection	Contact ± 4 KV, Air ± 8 KV per IEC1000-4-2 Cable is installed for testing
Electrical Fast Transient Protection	± 1 KV per IEC1000-4-4
Minimum Differential Output Voltage	1.5V with 60 Ω load
Fail Safe Outputs	Logic High input state if inputs are not connected
Maximum Common Mode Voltage	-7.5V to 12.5V

4 Position Terminal Block	
Description	Programmable RS232/485 Port - Non-isolated RS-232 DTE port connects the CPU as a Modbus/ASCII master or slave to a peripheral device. Includes ESD and built in surge protection. - Non-isolated RS-485 port connects the CPU as a Modbus master or slave to a peripheral device. Includes ESD/EFT protection and automatic echo cancellation when transmitter is active.
Data Rates	Selectable, 1200, 2400, 4800, 9600, 19200, 33600, 38400, 57600, and 115200
+5V Cable Power	210mA max. at 5V, $\pm 5\%$. Reverse polarity and overload protection.
Port Status LED	Green LED illuminated when active for TXD, RXD and RTS.
Cable Options (AutomationDirect.com)	EA-MG-PGM-CBL D2-DSCBL USB-RS232 with D2-DSCBL FA-CABKIT

ETHERNET

There are two 10/100MB Ethernet ports on the P2CDS-622 that need to be **setup via the USB port** before the Ethernet ports can be used.

This chapter discusses how to setup the ports in the CODESYS IDE.

The P2CDS-622 Ethernet Ports support different two different operating modes:

1). Information Technology (IT) Mode:

- LANs with access to Internet
- Default Gateway only allowed to be setup here.
- Recommended applications- MQTT, WebVisu, etc.

2). Operational Technology (OT) Mode:

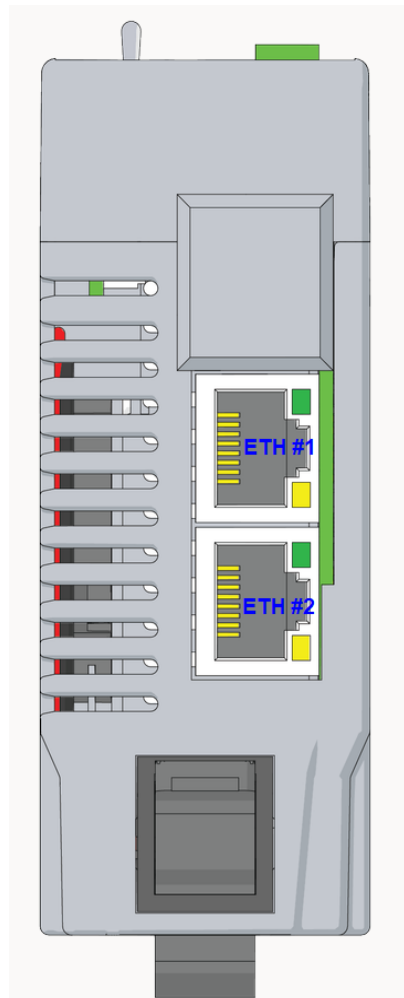
- Internal (non-Internet) networks
- Default Gateway is not applicable
- Recommended applications- on premise or local device like EtherNet/IP, Modbus TCP, etc.

Note: Ethernet Port 1 (ETH1) is dedicated to **only the IT mode**, while ETH2 is dedicated **only to the OT mode**.

14.1 Introduction

The *NetConfig* setup utility will be used in this configuration process.

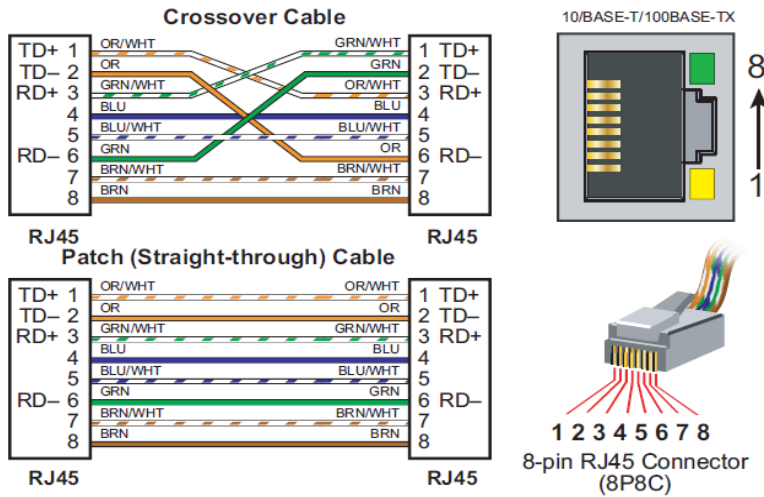
The Ethernet ports are located on the bottom of the CPU, *ETH 1* is located toward the **front** of the faceplate.



The specifications and functionality are listed below.

Ethernet Specifications	
Port Name	ETHERNET
Description	Standard transformer isolated Ethernet port with built-in surge protection for programming, online monitoring, firmware, MQTT, Email (SMTP client), Modbus/TCP client/server connections (fixed IP or DHCP) and Ethernet/IP Scanner/Adapter connections.
Transfer Rate	RJ45 Yellow LED Off = 10Mbps / On = 100 Mbps
Port Status LED	RJ45 Green LED Solid when network LINK is established. Flashes when port is active (ACT).

The pinout of the connectors are as follows:



The following sections will define how to configure the Ethernet ports in the CODESYS IDE.

Tip: There is **automatic crossover detection** of which cable is inserted, so either cable will work.

14.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 Power Supply (any variant)
- Latest version of CODESYS installed on Host PC
- USB TypeC cable

14.2.1 Initial Network Setup

Caution: To do this section, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

Host PC to P2CDS-622 connection

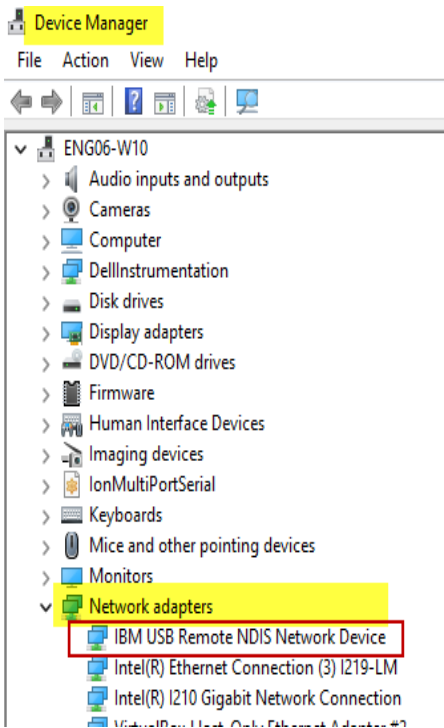
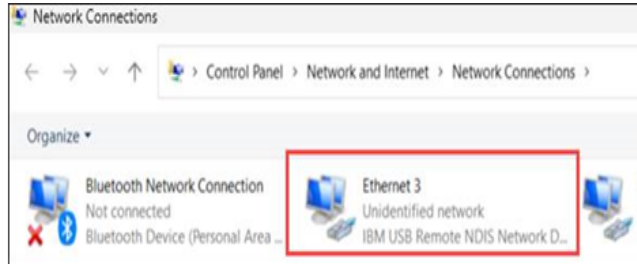
To get a baseline on the Host PC Ethernet connections you have on your Host PC, go to the “Network and Sharing” section of your OS and note the Ethernet connections that are present.

We are going to use the “Virtual Ethernet over USB (RNDIS)” feature in the P2CDS-622 to use the USB communications port to setup the device’s Ethernet ports.

First we want to verify the P2CDS-622 is on the network and can be seen by the Host PC.

P2CDS-622 onto the Network (via USB)

- (1). With the P2CDS-622 powered off, connect the Host PC USB port to the P2CDS-622 front panel USB-C.
- (2). Power-up the P2CDS-622, and note in the PC's "Network and Sharing" that a new Ethernet connection (P2CDS-622 Ethernet) is present. This is the **USB port** of the P2CDS-622.



We are not connected to the P2CDS-622 with the CODESYS IDE yet, but we want to make sure the PLC can be seen on the network.

14.3 Create a Project

We need to create a project and add the device (*P2CDS622_NetConfig*) that will allow the Ethernet ports to be setup.

14.3.1 Start CODESYS and Create a Project

Start CODESYS

(3). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Create the project

(4). Click **File > New Project**

The example project here will be named “*TestGenItems*”

(5). In the *New Project* window, in the *Templates* section, select the **Standard Project** template.

(6). Specify a name and a storage location for the project and click **OK**

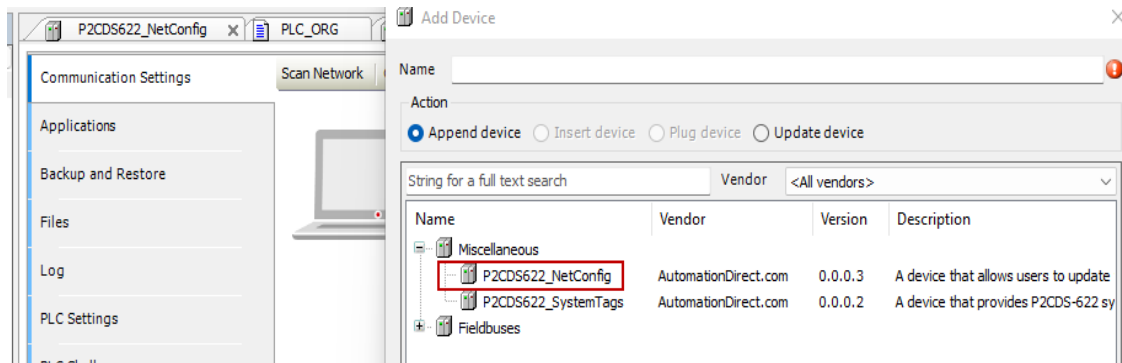
(7). In the **Device** drop-down location, select the **P2CDS622** device.

(8). In the **PLC_PRG in** drop-down location, leave default values.

Add the “P2CDS622_NetConfig” Device

(9). Right-click on *Device (P2CDS622)* in the device tree and select **Add Device**.

(10). In the pop-up under *Miscellaneous*, select **P2CDS622_NetConfig** as shown below.



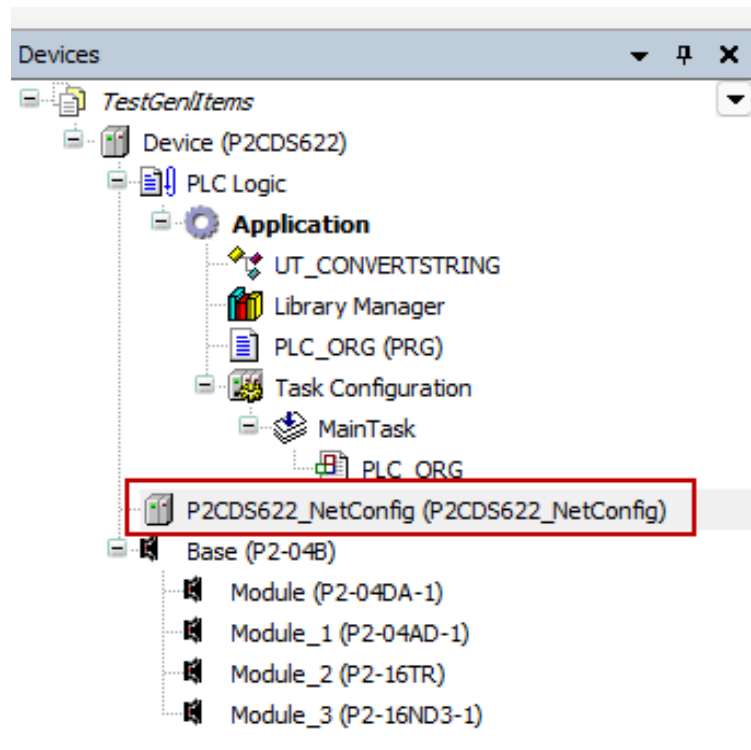
It now appears in the Device Tree and will be edited.

14.3.2 Configure the Ethernet Ports

“NetConfig” Device

The NetConfig device is an interface to enable the User to define the Ethernet Port’s IP Address, Subnet Mask, DHCP mode, etc. Both Ethernet ports on the P2CDS-622 can be configured in this area.

Note: This device is not associated with any System I/O or Fieldbuses, but is just used as a configuration “tool”



Caution: The Ethernet ports are independent and must operate on **separate** networks.

Internal Parameters

(11). To view the setup parameters, Click on *P2CDS622_NetConfig* and the following should be seen.

P2CDS622_NetConfig X						
Internal Parameters	Parameter	Type	Value	Default Value	Unit	Description
Internal I/O Mapping Internal IEC Objects Status Information	ET 1 Settings					
	ET1: DHCP	BOOL	FALSE	FALSE		Enable/Disable DHCP
	ET1: IP Address	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The desired IP address (e.g. [192,168,0,1])
	ET1: Subnet Mask	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The desired subnet mask (e.g. [255,255,255,0])
	ET1: Default Gateway	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The desired default gateway (e.g. [192,168,0,1])
	ET1: Adjust Ethernet Settings	BOOL	FALSE	FALSE		
	ET 2 Settings					
	ET2: DHCP	BOOL	FALSE	FALSE		Enable/Disable DHCP
	ET2: IP Address	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The desired IP address (e.g. [192,168,0,1])
	ET2: Subnet Mask	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The desired subnet mask (e.g. [255,255,255,0])
	ET2: Default Gateway	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The desired default gateway (e.g. [192,168,0,1])
	ET2: Adjust Ethernet Settings	BOOL	FALSE	FALSE		
	DNS Settings					
	DNS: Obtain from DHCP Server	BOOL	FALSE	FALSE		Enable/Disable obtain from DHCP Server
	DNS: Preferred DNS Address	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		The preferred DNS address (e.g. [192,168,0,1])
	DNS: Adjust Ethernet Settings	BOOL	FALSE	FALSE		

To setup Ethernet Port #1 (ETH1), follow the steps below.

Note that whenever you want to modify a parameter, you need to set the **Adjust** field to TRUE for the first download. After that, the profile is saved and it is recommended you set the **Adjust** field to FALSE so it is not continually rewritten.

(12). Edit the applicable fields under the *Value* column.

Setup DHCP IP Address Mode

- “ETH1: DHCP” > *TRUE*
- “ETH1: IP Address” (n/a)
- “ETH1: Subnet Mask” (n/a)
- “ETH1: Default Gateway” (n/a)
- “ETH1: Adjust Ethernet Settings” > *TRUE*

Setup Fixed IP Address Mode

- “ETH1: DHCP” > *FALSE*
- “ETH1: IP Address” > (set to Address desired, e.g. [192,168,5,25])
- “ETH1: Subnet Mask” (set to Mask desired)
- “ETH1: Default Gateway” (set to Gateway desired)
- “ETH1: Adjust Ethernet Settings” > *TRUE*

Setup DNS Settings

- “DNS: Obtain from DHCP Server” > (enable/disable)
- “DNS: Preferred DNS Address” > (set to Address desired)
- “DNS: Adjust Ethernet Settings” > *TRUE*

The Ethernet ports are now ready to be used to connect to the CODESYS IDE or be utilized to add an Ethernet connected fieldbus like Modbus TCP or EtherNet/IP.

Note: To reiterate, whenever you want to **modify a parameter**, you need to set the “Adjust...” field to TRUE for the **first** download. After that, the profile is saved and it is recommended you set the “Adjust...” field to FALSE to not be overwritten on subsequent downloads.

14.3.3 Verify Ethernet Settings

This section will show you how to see what the actual port parameters are based on your settings. For example if DHCP was chosen, you can see what the assigned Ethernet address.

Connect to PLC and Download Project

The project needs to be downloaded to the target and until now, we have not connected to it.

(13). In the Device Tree, Double-click on the **Device(P2CDS622)**.

A **Device** tab should appear with the first tab on the left selected entitled *Communication Settings* selected. Now we need to scan the network to find our device.

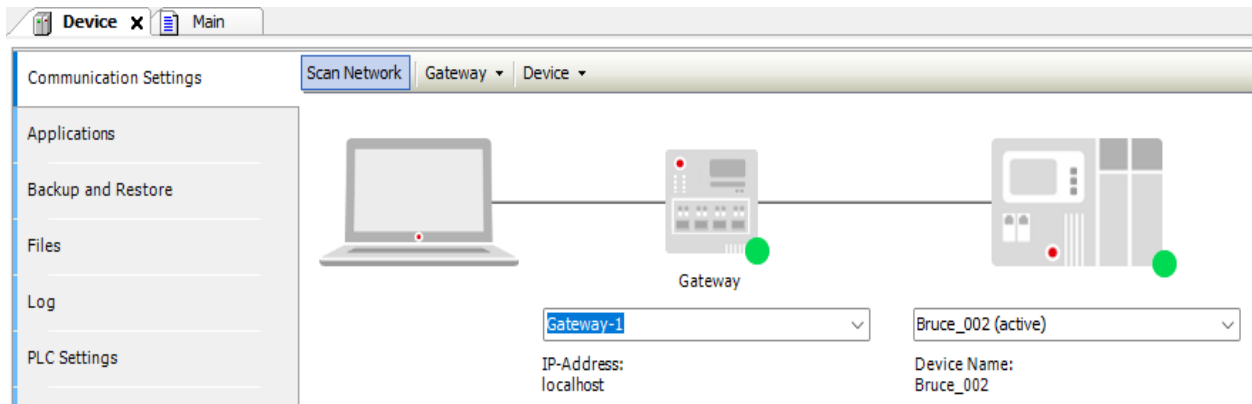
(14). In the *Communication Settings* section, click on the **Scan network**.

A dialog box should pop up with all the CODESYS PLCs available to connect to on this network.

Tip: If you are unsure which unit is the one you want to connect to, on the right side is a **Wink** select box. Click on this and this P2CDS-622 front panel **RUN** light should start to flash for 5 seconds.

(15). Select the applicable P2CDS-622 unit and Click **OK**.

Now the following should be visible. Both the Gateway and the P2CDS-622 should have the green indicators present indicating the CODESYS IDE is connected to the P2CDS-622.



Download Application to PLC

(16). Click **Online > Login**.

If there is an existing application on the PLC, a dialog opens up. Click **Yes** to continue.

The Project is now in the PLC.

(17). Click **Online > Logout** to disconnect.

(18). Remove the USB cable.

Connect to PLC via the Ethernet Port

(19). Plug in Ethernet cable to the ETH1 port.

(20). In the *Communication Settings* section, click on the **Scan network** and connect to the PLC (this may take several seconds).

Inspect the Ethernet Port Settings

(21). Login to the PLC. Click **Online > Login**.

(22). Verify the setup parameters, Click on *P2CDS_622_NetConfig*. The parameters should be visible under *Current Value*.

If they are satisfactory, you are good to go. If not, logout, edit NetConfig and repeat the process.

14.4 Ethernet Applications

The two communication applications that can be run on the Ethernet Ports simultaneously are *Modbus-TCP* and *EtherNet/IP*.

14.4.1 Modbus TCP

Refer to this section *Modbus* for detail information.

14.4.2 EtherNet/IP

Refer to this section *EtherNet/IP* for detail information.

MODBUS

The Modbus fieldbus modes included with the P2CDS-622 system are the Modbus TCP Master, TCP Slave, RTU Master and RTU Slave topologies.

15.1 Introduction

The section overviews the solutions available, their capabilities and references some specific example projects.

The two physical interface modes are the *TCP*, which utilizes the Ethernet ports for the physical communications channel and the *RTU* mode which uses either the RS-232 or RS-485 serial ports as the communication connection.

15.2 General Specifications

- Modbus TCP Client can support: **32 Slaves**
- Modbus RTU Client can support: **32 Slaves**
- Modbus TCP Server can support: **64 Connections**
- Modbus RTU Server can support: **1 Connection**

Note: All the Modbus device libraries are included **free** in the P2CDS-622 system. The license is automatically enabled.

The CODESYS **Modbus TCP- Master Datasheet** can be downloaded from- [TCP Master](#)

The CODESYS **Modbus TCP- Slave Datasheet** can be downloaded from- [TCP Slave](#)

The CODESYS **Modbus RTU- Master Datasheet** can be downloaded from- [RTU Master](#)

The CODESYS **Modbus RTU- Slave Datasheet** can be downloaded from- [RTU Slave](#)

15.3 Function Codes, Addresses and Data Types

The following are the operational Function Codes (**FC**) supported, the applicable Address Range where they may reside (can be custom assigned) and the type of Data these FC's support.

FC	Name	Addressing	Operation	Data Type
01	Read Coils	00001 - 09999	Read	Byte/Bool
02	Read Discrete Inputs	10001 - 19999	Read	Byte/Bool
03	Read Holding Register	40001 - 49999	Read	Word (16 bits)
04	Read Input Register	30001 - 39999	Read	Word
05	Write Single Coil	00001 - 09999	Write	Byte/Bool
06	Write Single Register	40001 - 49999	Write	Word
15	Write Multiple Coils	00001 - 09999	Write	Word
16	Write Multiple Registers	40001 - 49999	Write	Word
23	Read/Write Multiple Registers	40001 - 49999	Read/Write	Word

15.4 Modbus TCP

The **TCP** topology utilizes the standard Ethernet Adapter supporting both the Modbus TCP Master configuration and the Slave/Device topology.

This configuration is dependent on the Modbus library device selected.

Detailed features and of **how to use the Modbus TCP version of the stack** are given in an example project in Section *Modbus TCP Project*

The Modbus TCP protocol uses a Client/Server architecture for data exchange.

Clients are devices that initiate any data exchange with other devices on the network, i.e. Masters. This applies to both I/O communications and service messaging.

Servers are devices that address any data requests generated by a Client. This applies to both I/O communications and service messaging.

The communication between the Modbus TCP IOScanner and the slave device is accomplished using Modbus TCP *Channels*.

15.4.1 TCP Master

The communication parameters are predefined in the configurator, such as the IP Address and port number settings of the Ethernet Adapter.

Modbus commands are defined in the *Configurator* and are oriented to a specific Modbus slave. The commands can be processed by the device at predefined intervals, or can be triggered programmatically. For predefined commands, I/O channels are generated automatically with variables that can be mapped (I/O mapping).

The following link provides a project example- *Modbus TCP Project*.

Master Configurator

The CODESYS Modbus TCP Master configurator defines the Modbus commands and are targeted to a specific Modbus slave.

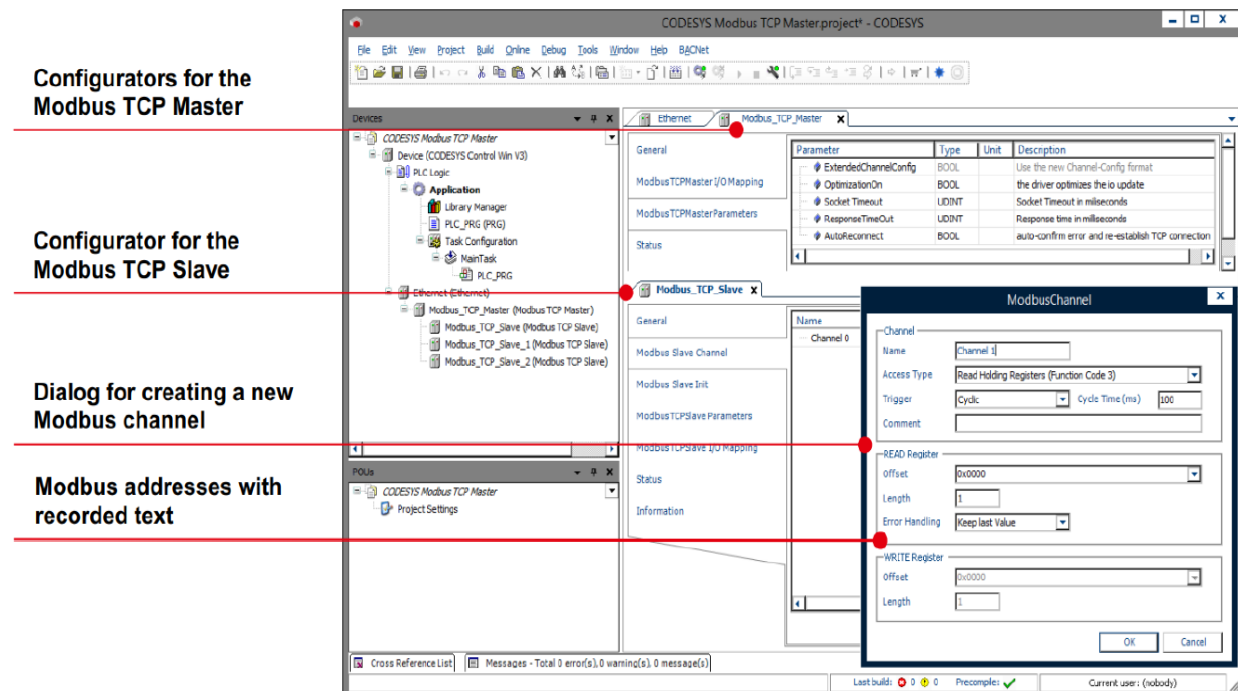
The commands can be processed by the device at predefined intervals, or can be triggered programmatically.

For predefined commands, I/O channels are generated automatically with variables that can be mapped (I/O mapping).

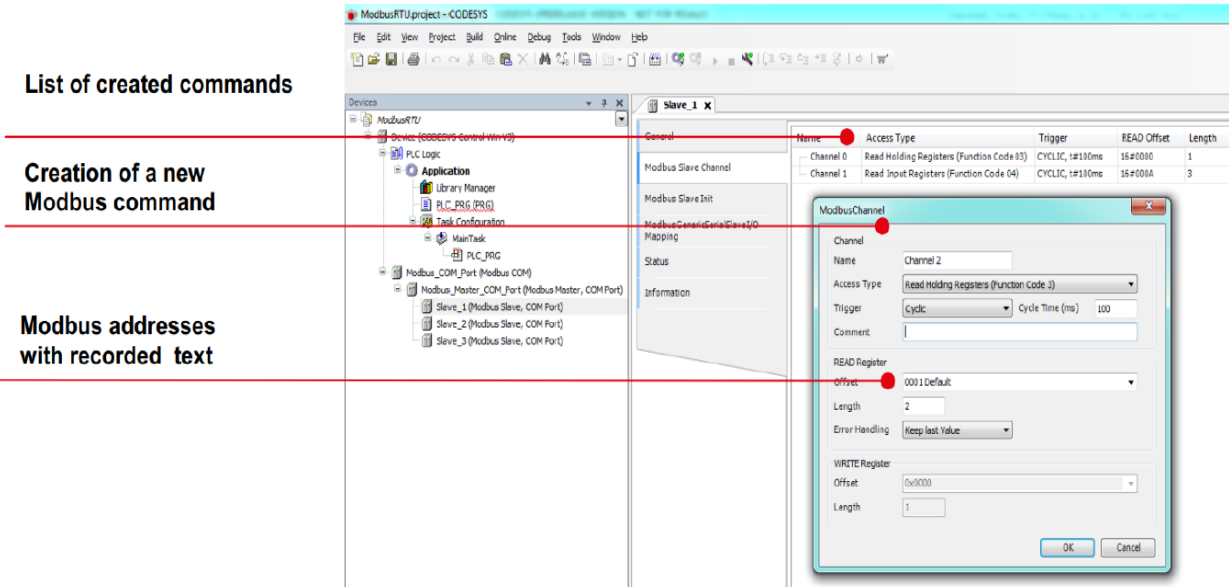
The CODESYS Modbus TCP Master configurator consists of editors for the following device categories that are inserted into the device tree hierarchy:

- Ethernet: The Ethernet adapter is configured here (IP address, subnet mask, etc.)
- Modbus TCP Master: A Modbus TCP master can be inserted below the Ethernet node. Communication settings can be defined specifically for Modbus, for example “Response Timeout” for defining the time to wait for a response from a Modbus TCP slave.
- Modbus TCP Slave: Multiple Modbus TCP slave devices can be inserted below a Modbus TCP master. The slave address is defined here, as well as a series of Modbus commands (incl. respective I/O mapping). These are processed by the driver and exchanged with the Modbus TCP slave.

The following is an example of the **Master Configurator for the TCP mode** within the tool.



Creation of **Modbus commands**, as shown below can be defined so that they are created automatically when devices are inserted into the project.



The example projects located here at (*Modbus TCP Project* and *Modbus RTU Project*) go into detail on how to implement the commands.

15.4.2 TCP Slave

The PLC can also be configured as a Modbus Slave device using the *TCP Slave* fieldbus.

15.5 Modbus RTU

Modbus RTU is an open serial protocol derived from the master/slave architecture (now client/server) originally developed by Modicon (now Schneider Electric). It is a widely accepted serial level protocol due to its ease of use and reliability.

Modbus RTU messages are a simple 16-bit structure with a Cyclic-Redundant Checksum. The simplicity of these messages ensures reliability. Due to this simplicity, the basic 16-bit Modbus RTU register structure can be used to pack in floating point, tables, ASCII text, queues, and other data.

The *RTU* topology utilizes the Serial Comm ports (RS-232 or RS-485) for the physical connections.

15.5.1 RTU Master

The communication parameters are predefined in the configurator, such as the settings of the Comm ports (baud rate, port number).

Modbus commands are defined in the *Configurator* and are oriented to a specific Modbus slave. The commands can be processed by the device at specific intervals, or can be triggered programmatically. For predefined commands, I/O channels are generated automatically with variables that can be mapped (I/O mapping).

Processing requires a protocol stack and CODESYS I/O driver that implements Modbus communication on the configured port. This I/O driver is supplied with the license with the P2CDS-622 package.

The CODESYS Modbus RTU Master is configured completely from within the CODESYS Development System.

The following link provides a project example

Modbus RTU Example Project.

Master Configurator

The CODESYS Modbus Master (RTU) configurator consists of editors for the following COM port device categories that are inserted into the device tree hierarchy:

- Modbus: The COM port settings are configured here, such as baud rate and parity.
- Modbus Master: A Modbus master can be inserted below the COM port.

Communication settings can be defined specifically for Modbus RTU, for example “Response Timeout” for defining the time to wait for a response from a Modbus slave.

- Modbus slave: Multiple Modbus slave devices can be inserted below a Modbus master. The slave address is defined here, as well as a series of Modbus commands (incl. respective I/O mapping). These are processed by the driver and exchanged with the Modbus slave.

Creation of *Modbus commands* can be defined to use that are created automatically when devices are inserted into the project as shown below.

List of created commands

Creation of a new Modbus command

Modbus addresses with recorded text

Name	Access Type	Trigger	READ Offset	Length
Channel 0	Read Holding Registers (Function Code 03)	CYCLIC, t#100ms	16#0000	1
Channel 1	Read Input Registers (Function Code 04)	CYCLIC, t#100ms	16#000A	3

ModbusChannel

Channel: Channel 2

Access Type: Read Holding Registers (Function Code 3)

Trigger: Cyclic Cycle Time (ms): 100

Comment:

READ Register

Offset: 0001 Default

Length: 2

Error Handling: Keep last Value

WRITE Register

Offset: 0x0000

Length: 1

OK Cancel

15.5.2 RTU Slave

The PLC can also be configured as a Modbus Slave device using the *Modbus Serial Slave* fieldbus.

15.6 Example Projects

Examples projects a located at these links for *Modbus TCP* and for *Modbus RTU*

ETHERNET/IP

EtherNet/IP™ (ENIP) is an Ethernet communication network that provides users with the tools to deploy standard Ethernet technology (IEEE 802.3 combined with the TCP/IP Suite) in industrial automation applications while enabling Internet and enterprise connectivity.

EtherNet/IP offers various network topology options including star or linear with standard Ethernet infrastructure devices utilizing the “Common Industrial Protocol” (CIP™).

Like all CIP Networks, EtherNet/IP utilizes CIP™ for its upper layers. CIP Networks follow the Open Systems Interconnection (OSI) model, which defines a framework for implementing network protocols in seven layers: physical, data link, network, transport, session, presentation and application. Networks that follow this model define a complete suite of network functionality from the physical implementation through the application or user interface layer.

Topologies are comprised of a Master or *Scanner* (also called the *Originator*) and an end-point device or slave called the *Adapter* (also called the *Target*).

The following gives a summary Table for the Messaging types and their characteristics:

Item	EXPLICIT Messaging	IMPLICIT Messaging
Characteristics	Requests & defines information	Data only, no protocol information
Form of Messaging	Unconnected or Class 3 Messaging	Connected or Class 1 Messaging
Typical Application	Diagnostic/Event/Config Data	Real-time Control Data
Originator (Master)	Controller = “Client”	Controller = “IO Scanner”
Target (Slave)	Field Device = “Server”	Field device = “IO Adapter”
Communications Layer	TCP/IP	UDP

16.1 General Specifications

- Maximum Adapter nodes supported per Scanner: **64 Adapters**
- Maximum Scanner Connections per Server/Adapter: **Up to eight(8) Class 1 Implicit or eight(8) Class 3 Explicit connections.**

Note: All the EtherNet/IP device libraries are included **free** in the P2CDS-622 system. The license is automatically enabled.

The CODESYS Scanner Datasheet can be downloaded from- [Scanner](#)

The CODESYS Adapter Datasheet can be downloaded from- [Adapter](#)

16.2 Scanner

The CODESYS EtherNet/IP Scanner is offered as part of the P2CDS-622 development environment and uses the standard Ethernet ports on the CPU. The bus is configured directly from within the CODESYS Development System. Devices that are configured in this way exchange Ethernet messages with interconnected EtherNet/IP Adapters.

Scanners create connections and scan I/O. The EtherNet/IP scanner can communicate in real time with connected adapters, read their inputs, and write to their outputs. All parameters for communication are predefined in the configurator. These parameters include IP addresses or network cards and adapters, the connections and their properties, the RPI (Requested Packet Interval), as well as the user parameters.

16.2.1 FEATURES

All parameters for communication are predefined in the configurator.

Configurator

Editors

- Ethernet Bus Editor
- EtherNet/IP Scanner Editor
- EtherNet/IP Remote Adapter Module Editor

Communication Settings

- IP Address
- Electronics Keying
- Connections and configuration data
- Input/Output Assembly layout

Diagnosis

- Display of device state
- Display of connection errors

EDS Import

- Supported

IEC Stack

Supported Platforms

- 32/64 bit
- Little/Big Endian

Specification

- CIP Networks Library Volume 1 and 2

Max. number of Slaves/Connections

- 64

Connection Types

- Class 1 (I/O Messaging)
- Class 3 (Explicit Messaging)
- UCMM

I/O Connection Types

- Point to Point and Multicast
- Cyclic Transmission
- Exclusive Owner, Listen Only, Input Only

Minimum RPI

- 1 ms

Supported Objects

- Identity Object
- Message Router Object
- Connection Manager Object
- TCP/IP Interface Object
- EtherNet Link Object

Large ForwardOpen

- (supported)

CIP Motion

- (not supported)

CIP Sync

- (not supported)

Device Level Ring (DLR)

- (not supported)

API**EtherNet/IP Services IEC Library (Function Blocks)**

- Get_Attributes_All, Get_Attributes_Single
- Set_Attributes_All, Set_Attributes_Single
- Start/Stop/Reset
- Apply_Attributes
- NOP
- Generic Service
- Visualization Templates

API IEC Stack (Function Blocks)

- State/Diagnosis variables for Scanner and Adapters
- Ethernet status information

- Reset of Scanner and Adapter
- Generic Device Diagnosis
- Reconfigure

16.3 Adapter

The CODESYS EtherNet/IP Adapter is an add-on for CODESYS compatible PLCs and uses the CPU Ethernet ports.

The bus is configured directly from within the CODESYS Development System. Adapters are End-point devices and are configured to exchange Ethernet messages with interconnected EtherNet/IP scanners.

16.3.1 FEATURES

The communication parameters are predefined in the configurator, such as the settings of the Ethernet adapter

Configurator

Editors

- Ethernet Bus Editor
- EtherNet/IP Adapter Editor
- EtherNet/IP Module Editor

Communication Settings

- IP Address
- Device Identification (Vendor ID, Product Code, ...)
- I/O Data Layout (module configuration)

Diagnosis

- Display of device state
- Display of connection errors

EDS Export

- Export of an EDS file suitable for the configuration.

IEC Stack

Supported Platforms

- 32/64 bit
- Little/Big Endian

Specification

- CIP Networks Library Volume 1 and 2

Max. Number of Connections

- Up to eight (8) Class 1 Connections and eight (8) Class 3 Connections.

I/O Connection Types

- Point to Point
- Cyclic Transmission
- Exclusive Owner, Listen Only, Input Only

Max. Number of I/O Assemblies

- The supplied EDS files support only one Input and Output Assembly by default. A custom EDS file can be generated via the CODESYS IDE.
- 4 Scanners with 8 Connections per. maximum.

Configuration Assembly

- (not supported)

Connection Types

- Class 1 (I/O Messaging)
- Class 3 (Explicit Messaging)
- UCMM

Maximum Connection Types

- Class 1 (I/O Messaging) - 8 total
- Class 3 (Explicit Messaging) - 8 total

Minimum RPI

- 1 ms

Supported Objects

- Identity Object
- Message Router Object
- Assembly Object
- Connection Manager Object
- TCP/IP Interface Object
- EtherNet Link Object

ACD

- (currently not supported, see *Errata* section)

CIP Motion

- (not supported)

CIP Sync

- (not supported)

Device Level Ring (DLR)

- (not supported)

16.4 Example Projects

Examples projects are located at these links for *Implicit Mode* and for *Explicit Mode*

IO MODULE CONFIGURATION AND STATUS

This section defines the Configuration parameters in the CODESYS IDE (if configuration is required) for each module and any associated status signals.

Status signals could be for the module as a whole e.g. *Missing 24V* or for an individual IO Channel basis e.g. *Under-Range*.

17.1 Discrete Input

The following modules are for Discrete (On/Off) type of modules.

17.1.1 P2-08ND3-1

This module supports up to eight(8) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.2 P2-16ND3-1

This module supports up to sixteen(16) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.3 P2-32ND3-1

This module supports up to thirty-two(32) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.4 P2-08NE3

This module supports up to eight(8) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.5 P2-16NE3

This module supports up to sixteen(16) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.6 P2-32NE3

This module supports up to thirty-two(32) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.7 P2-08NAS

This module supports up to eight(8) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.1.8 P2-16NA

This module supports up to sixteen(16) Discrete input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.2 Discrete Output

The following modules are for Discrete (On/Off) Output type of modules.

17.2.1 P2-08TD1S

This module supports up to eight(8) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.2.2 P2-08TD2S

This module supports up to eight(8) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.2.3 P2-08TD1P

This module supports up to eight(8) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Fault Output

- Output signal (by channel) is the fault state of overload.
- If TRUE, the channel is in an over current state.
- If FALSE, the channel is operating with range.

17.2.4 P2-16TD1P

This module supports up to sixteen(16) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Fault Output

- Output signal (by channel) is the fault state of overload.
- If TRUE, the channel is in an over current state.
- If FALSE, the channel is operating with range.

17.2.5 P2-32TD1P

This module supports up to thirty-two(32) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Fault Output

- Output signal (by channel) is the fault state of overload.
- If TRUE, the channel is in an over current state.
- If FALSE, the channel is operating with range.

17.2.6 P2-08TD2P

This module supports up to eight(8) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Channel:

Fault Output

- Output signal (by channel) is the fault state of overload.
- If TRUE, the channel is in an over current state.
- If FALSE, the channel is operating with range.

17.2.7 P2-16TD2P

This module supports up to sixteen(16) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Fault Output

- Output signal (by channel) is the fault state of overload.
- If TRUE, the channel is in an over current state.
- If FALSE, the channel is operating with range.

17.2.8 P2-32TD2P

This module supports up to thirty-two(32) Discrete Input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Channel:

Fault Output

- Output signal (by channel) is the fault state of overload.
- If TRUE, the channel is in an over current state.
- If FALSE, the channel is operating with range.

17.2.9 P2-15TD1

This module supports up to fifteen(15) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.2.10 P2-15TD2

This module supports up to fifteen(15) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.2.11 P2-08TAS

This module supports up to eight(8) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.2.12 P2-16TA

This module supports up to sixteen(16) Discrete Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.3 Analog Input

These modules are input only devices that are either Current, Voltage or both Current/Voltage types.

17.3.1 P2-04AD

This module supports up to four(4) of both Current and Voltage input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 4 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4

Channel Range

Ch1 Range, Ch2 Range, Ch3 Range, Ch4 Range

- Each channel is individually configurable.

- $\pm 5V$, $\pm 10V$, $0-5V$, $0-10V$, $0-20mA$

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.2 P2-04AD-1

This module supports up to four(4) Current input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 4 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.3 P2-08AD-1

This module supports up to eight(8) Current input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.4 P2-16AD-1

This module supports up to sixteen(16) Current input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8 Ch. Select 1-9, Ch. Select 1-10, Ch. Select 1-11, Ch. Select 1-12 Ch. Select 1-13, Ch. Select 1-14, Ch. Select 1-15, Ch. Select 1-16

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.5 P2-08ADL-1

This module supports up to eight(8) Current input (12-bit resolution) signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:***Under Range Errors***

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.6 P2-16ADL-1

This module supports up to sixteen(16) Current input (12-bit resolution) signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:***Ch. Select***

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8 Ch. Select 1-9, Ch. Select 1-10, Ch. Select 1-11, Ch. Select 1-12 Ch. Select 1-13, Ch. Select 1-14, Ch. Select 1-15, Ch. Select 1-16

Hot Swap Enable***True/False***

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.7 P2-04AD-2

This module supports up to four(4) Voltage input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 4 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4

Hot Swap Enable

True/False

- Module allowed to be removed.

- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.8 P2-08AD-2

This module supports up to eight(8) Voltage input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.9 P2-16AD-2

This module supports up to sixteen(16) Voltage input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8 Ch. Select 1-9, Ch. Select 1-10, Ch. Select 1-11, Ch. Select 1-12 Ch. Select 1-13, Ch. Select 1-14, Ch. Select 1-15, Ch. Select 1-16

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.10 P2-08ADL-2

This module supports up to eight(8) Voltage input (12-bit resolution) signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.3.11 P2-16ADL-2

This module supports up to sixteen(16) Voltage input (12-bit resolution) signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:***Ch. Select***

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8 Ch. Select 1-9, Ch. Select 1-10, Ch. Select 1-11, Ch. Select 1-12 Ch. Select 1-13, Ch. Select 1-14, Ch. Select 1-15, Ch. Select 1-16

Hot Swap Enable***True/False***

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:***Module Failed***

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.4 Analog Output

17.4.1 P2-04DA

This module supports up to four(4) of both Current and Voltage output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Channel Range

Ch1 Range, Ch2 Range, Ch3 Range, Ch4 Range

- Each channel is individually configurable.
- +/-10V, 4-20mA Source, 4-20mA Sink

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.2 P2-04DA-1

This module supports up to four(4) Current output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.

- If FALSE, the module has 24V applied within specification.

17.4.3 P2-08DA-1

This module supports up to eight(8) Current output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.4 P2-16DA-1

This module supports up to sixteen(16) Current output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.5 P2-04DAL-1

This module supports up to four(4) Current output signals (12-bit resolution).

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.6 P2-08DAL-1

This module supports up to eight(8) Current output signals (12-bit resolution).

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.

- If FALSE, the module has 24V applied within specification.

17.4.7 P2-16DAL-1

This module supports up to sixteen(16) Current output signals (12-bit resolution).

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.8 P2-04DA-2

This module supports up to four(4) Voltage output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.9 P2-08DA-2

This module supports up to eight(8) Voltage output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.10 P2-16DA-2

This module supports up to sixteen(16) Voltage output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.

- If FALSE, the module has 24V applied within specification.

17.4.11 P2-04DAL-2

This module supports up to four(4) Voltage output signals (12-bit resolution).

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the Module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.12 P2-08DAL-2

This module supports up to eight(8) Voltage output signals (12-bit resolution).

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.4.13 P2-16DAL-2

This module supports up to sixteen(16) Voltage output signals (12-bit resolution).

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

17.5 Analog Combo

The *Combo* modules support both Inputs and Outputs.

17.5.1 P2-8AD4DA-1

This module supports both input and output Current signals. Up to eight(8) inputs and four(4) outputs.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Channel Resolution

Input Ch1 Resolution - Input Ch8 Resolution

- Each channel is configurable for resolution (Coarse-12bit/Medium-14bit/Fine-16bit)
- Coarse, Medium, Fine

Hot Swap Enable

True/False

- Module allowed to be removed.

- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.5.2 P2-8AD4DA-2

This module supports both input and output Voltage signals. Up to eight(8) inputs and four(4) outputs.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Channel Range

Input Ch1 Range - , Input Ch2 Range

- Each channel is individually configurable.
- 0-5V, 0-10V

Channel Resolution

Input Ch1 Resolution - Input Ch8 Resolution

- Each channel is configurable for resolution (Coarse-12bit/Medium-14bit/Fine-16bit)
- Coarse, Medium, Fine

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Missing 24V

- External 24V power being applied or absent.
- If TRUE, the module does not have the 24V field power within specification connected.
- If FALSE, the module has 24V applied within specification.

Channel:

Under Range Errors

- Input signal (by channel) is below the allowable input range.

- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.6 Temperature

The following modules are used to measure temperature in a variety of applications.

17.6.1 P2-08THM

This module supports up to eight(8) Temperature or Voltage input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Channel Range

Range Ch1 - Range Ch8

- Each channel is individually configurable.
- Thermocouples: Type J, Type K, Type E, Type R, Type S, Type T, Type B, Type N, Type C
- Voltage: 0-39mV, +/-39mV, +/-78mV, 0-156mV, +/-156mV, 0-1.25V

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Burnout Enable/Temperature Scale

(various, see below)

- If Disable Burnout Detection/Deg C, Burnout OFF and Temp scale Celsius.
- If Low Side Burnout Detection/Deg C, Burnout ON and Temp scale Celsius.
- If High Side Burnout Detection/Deg C, Burnout ON and Temp scale Celsius.
- If Disable Burnout Detection/Deg F, Burnout OFF and Temp scale Fahrenheit.
- If Low Side Burnout Detection/Deg F, Burnout ON and Temp scale Fahrenheit.
- If High Side Burnout Detection/Deg F, Burnout ON and Temp scale Fahrenheit.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Module Not Ready

- First channel data conversions after a power-up or reconfiguration are not ready.
- If TRUE, the module has not processed the channel inputs yet.
- If FALSE, current channel input data reflects latest configuration.

Channel:

Burnout Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.6.2 P2-08NTC

This Thermistor module supports up to eight(8) Temperature input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1- 8 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6, Ch. Select 1-7, Ch. Select 1-8

Burnout Enable/Temperature Scale

(various, see below)

- If Disable Burnout Detection/Deg C, Burnout OFF and Temp scale Celsius.
- If Low Side Burnout Detection/Deg C, Burnout ON and Temp scale Celsius.
- If High Side Burnout Detection/Deg C, Burnout ON and Temp scale Celsius.
- If Disable Burnout Detection/Deg F, Burnout OFF and Temp scale Fahrenheit.
- If Low Side Burnout Detection/Deg F, Burnout ON and Temp scale Fahrenheit.
- If High Side Burnout Detection/Deg F, Burnout ON and Temp scale Fahrenheit.

Channel Range

Range

- The range selected applies to the all the module inputs.
- Thermistor Type: 2252, 10K-AN, 10K-CP, 5K, 3K, 1.8K,

Digital Conversion Filter

Digital Filter

- The speed at which the Converter runs. Faster speed is lower accuracy.
- Sample Rate: 61ms, 16ms, 4ms

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Module Not Ready

- First channel data conversions after a power-up or reconfiguration are not ready.
- If TRUE, the module has not processed the channel inputs yet.
- If FALSE, current channel input data reflects latest configuration.

Channel:

Burnout Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel sensor is open and not functioning.
- If FALSE, the channel sensor is operating correctly.

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.6.3 P2-06RTD

This RTD module supports up to six(6) Temperature or Resistance input signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Number of Channels Active:

Ch. Select

- Selects between 1-6 active input channels.
- Ch. Select 1, Ch. Select 1-2, Ch. Select 1-3, Ch. Select 1-4 Ch. Select 1-5, Ch. Select 1-6

Burnout Enable/Temperature Scale

(various, see below)

- If Disable Burnout Detection/Deg C, Burnout OFF and Temp scale Celsius.
- If Low Side Burnout Detection/Deg C, Burnout ON and Temp scale Celsius.
- If High Side Burnout Detection/Deg C, Burnout ON and Temp scale Celsius.
- If Disable Burnout Detection/Deg F, Burnout OFF and Temp scale Fahrenheit.
- If Low Side Burnout Detection/Deg F, Burnout ON and Temp scale Fahrenheit.
- If High Side Burnout Detection/Deg F, Burnout ON and Temp scale Fahrenheit.

Channel Range

Range

- The range selected applies to the all the module inputs.
- RTD Type: jPt100, Pt100, Pt1000, Cu10, Cu25, Ni120,
- Resistance: 0-10K, 0-6.25K, 0-3.125K, 0-1562.5K, 0-781.25, 0-390.625, 0-193.3125

Digital Conversion Filter

Digital Filter

- The speed at which the Converter runs. Faster speed is lower accuracy.
- Sample Rate: 16ms, 28ms, 56ms, 75ms, 88ms, 488ms

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

Module Not Ready

- First channel data conversions after a power-up or reconfiguration are not ready.
- If TRUE, the module has not processed the channel inputs yet.
- If FALSE, current channel input data reflects latest configuration.

Channel:

Burnout Errors

- Input signal (by channel) sensor is opened up and not functioning.
- If TRUE, the channel sensor is open and not functioning.
- If FALSE, the channel sensor is operating correctly.

Under Range Errors

- Input signal (by channel) is below the allowable input range.
- If TRUE, the channel input is too low.
- If FALSE, the channel input is within range.

Over Range Errors

- Input signal (by channel) is above the allowable input range.
- If TRUE, the channel input is too high.
- If FALSE, the channel input is within range.

17.7 Relay Output

The Relay modules provide surge-protected outputs to operate DC or AC rated devices (6-24 VDC / 6-120 (or 6-240) VAC).

17.7.1 P2-08TRS

This module supports up to eight(8) Relay Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.7.2 P2-16TR

This module supports up to sixteen(16) Relay Output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

17.8 Other

The following are specialty modules that do not fall under the previous categories.

17.8.1 P2-04PWM

This PWM module supports up to four(4) high speed output signals.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Channel Output Type

Ch1 Output Type - Ch4 Output Type

- The output type is channel configurable and deals with PWM speed and direction.
- PWM Type: PWM (x1), PWM (x0.1), PWM (x0.01)
- Direction: Dir

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the Module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

Module:

Module Failed

- Catastrophic failure state of the module.
- If TRUE, the module has a major failure and is unusable.
- If FALSE, the module is operating normally.

17.8.2 P2-08SIM

The P2-08SIM Input Simulator module provides 8 toggle switches to simulate input devices.

Configuration is found in the **Parameters** section of the IO Module.

Status is found in the **IO Mapping** section of the IO Module.

Configuration

The following items can be configured:

Hot Swap Enable

True/False

- Module allowed to be removed.
- If TRUE, the module can be removed and the system will not go into Stop mode. Task operations continue.
- If FALSE, removing the module will put the system into Stop mode.

Status

The following items give the specific status of the module and each channel:

(No status signals for this module)

INTRO TO PROJECTS

Caution: To do any project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

Some key notes are listed below that apply to all projects.

18.1 Security/Data Protection

- Use Ethernet Port 1 (ETH1) for all possible **external** public network applications, e.g. MQTT-TLS, SNTP, etc.
- Use Ethernet Port 2 (ETH2) for all **internal** network applications, e.g. Modbus, EtherNet/IP.
- With **WebVisu**, if you want to publish **external** a web visualization, utilize HTTPS and password protect.

18.2 Project Variable Nomenclature

- Example projects utilize “**Hungarian Notation**” for variable naming.
- In lieu of modern hierarchical Object Oriented Programming that is supported by the CODESYS IDE, it is recommended to **not use this format**. This will make the code more readable across boundaries.
- Future projects will be updated accordingly.

Caution: The only Fieldbuses and Add-ons that the P2CDS-622 will support are the Modbus TCP, Modbus RTU, EtherNet/IP Scanner and Adapter, the IIoT Library and WebVisu. Trying to install **other licenses** from the CODESYS Website, e.g OPC-UA **will not work**.

DATA TYPE CONVERSION

There are times when a user has to convert between *Data Types*. For example a sensor may have 16-bit data in the form of two(2) 16-bit Words that are to be read, yet the data eventually needs to be used in a *Float* or *Real* data type format. This could be in an HMI or maybe a SCADA system.

This project demonstrates how to easily convert the data via Unions, Arrays, Structures and IO Mapping techniques.

19.1 Introduction

This example is based upon an actual physical device, the *SOCOME DIRIS A-40 Meter*.

The *Current* readings (*I1 - I3*) for the various phases are shown below (in WORD format).

Dec address	Hex address	Words count	Description	Unit	Data type
18432	0x4800	1	Load status 0 : Disabled 1 : Enabled	-	U8
18433	0x4801	2	Date of last instance	s	DATETIME
18435	0x4803	1	Integration time	s / 5	U16
18436	0x4804	2	System Ph-N Voltage	V 10 ⁻²	U32
18438	0x4806	2	System Ph-Ph Voltage	V 10 ⁻²	U32
18440	0x4808	2	System Current	mA	U32
18442	0x480A	2	Frequency	mHz	U32
18444	0x480C	2	Ph-N Voltage : V1	V 10 ⁻²	U32
18446	0x480E	2	Ph-N Voltage : V2	V 10 ⁻²	U32
18448	0x4810	2	Ph-N Voltage : V3	V 10 ⁻²	U32
18450	0x4812	2	Ph-N Voltage : Vn	V 10 ⁻²	U32
18452	0x4814	2	Ph-Ph Voltage : U12	V 10 ⁻²	U32
18454	0x4816	2	Ph-Ph Voltage : U23	V 10 ⁻²	U32
18456	0x4818	2	Ph-Ph Voltage : U31	V 10 ⁻²	U32
18458	0x481A	2	Current : I1	mA	U32
18460	0x481C	2	Current : I2	mA	U32
18462	0x481E	2	Current : I3	mA	U32
18464	0x4820	2	Current : In	mA	U32

The sensor's total Current data field *Data Type* value is an *Unsigned 32-bit* value which is comprised of two(2) separate 16-bit Words.

In the program example, we will use the variable names CuAW1 and CuAW2 to hold the *Current A phase* two(2) Words of data.

This project will convert the data into a final 32-bit *Real* format that is required by our fictional SCADA system.

19.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- Latest version of CODESYS installed on Host PC
- Familiarity with the “Structured Text” programming language.

19.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, change any existing default passwords during the initial commissioning, and change passwords regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assigning simple password protection to prevent access to the functionality of your PLC over the Internet.

19.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not.

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

Tip: NOTE: There is a **completed project** that can be downloaded entitled **Demo_datatype_convert**. The following instructions on this page provide details to the generation of this project.

Download the project here: [Data Type Conversion project](#)

19.5 Creating a Project

The following sections outline the steps taken to create the downloadable project.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POUs.

19.5.1 Start CODESYS and Create a Project

Start CODESYS

- (1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS (version)**).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

- (2). Click **File > New Project**

The example project here will be named **datatype_convert**

- (3). In the **New Project** window, **Templates** section, select the **Standard Project** template.

- (4). Specify a name and a storage location for the project and click **OK**

The project opens in the CODESYS **Standard Project** frame window.

- (5). In the **Device** drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create (default name is **PLC_PRG**). We will rename this file later to my preference of **“Main”**.

- (6). In the **PLC_PRG in** drop-down location, select **Structured Text (ST)**. This will be the language we will use for the first program.

The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

Caution: If Ethernet ports have not been configured yet, it is required that they be setup per section the [Ethernet](#) section!

Rename Design File and Add Fieldbusses

It is not required, but for this example we will change the name of the *PLC_PRG* to *Main*.

- (7). Right Click on *PLC_PRG* and in the drop-down select **Properties**. In the **Common** dialog box, rename *PLC_PRG* to *Main*. Click **OK**.

- (8). A dialog box will prompt regarding the name change to be applied over the entire project. Click **Yes**.

Another dialog box entitled **Refactoring** will pop up displaying all the areas in the project where this new name will be applied. Click **OK**.

If a Fieldbus (Modbus or EtherNet/IP) is to be used, this is the point where you would add these.

Add Fieldbus

(Not required)

Change Base Type (if req'd)

If you are using a Base topology other than the P2-04B, you can change it now.

(9). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**. The *Plug Device* dialogue box should appear.

(10). Under the *Miscellaneous* name, find the Base unit device you want and double-click. Close the box. The updated Base should be showing.

19.5.2 Write the Control Program

This project will be comprised of the following key aspects:

1. Create a UNION that takes in the ARRAY values and converts to a Double Word (DWORD).
2. Create a UNION that takes in the DWORD values and converts them to a Real/Float (REAL).
3. Create an *Object* or Structure (STRUCT) that represents the Meter's registers and other variables desired.
4. Map the Meter values into WORD size variables.
5. Map these variables into an ARRAY so the Words can be appended together to create a DWORD.

Create a UNION to convert to DWORD

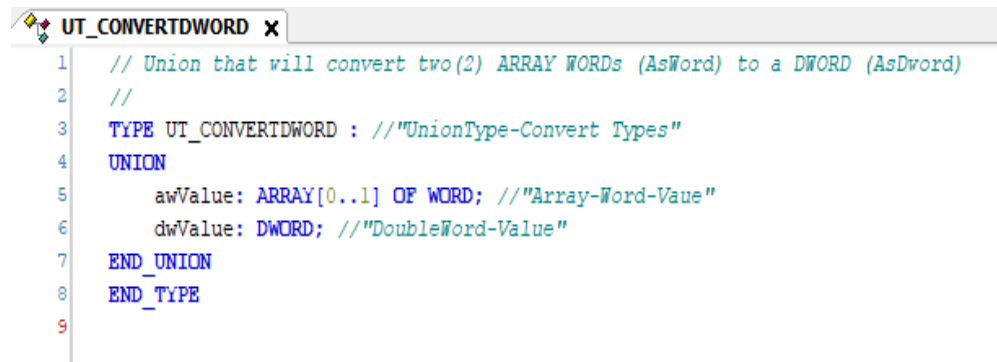
This Union is a custom, user defined *DataType* that will take in an ARRAY variable and create a DWORD representation of it.

(11). Add a Union by right-clicking on **Application > Add Object > DUT...**

A pop-up will appear entitled *Add DUT* (Device Unit Type)

(12). Select the **Union** item and name it UT_CONVERTDWORD ("Unit Type-Convert to DWORD")

(13). Fill in the file as shown below.



```
1 // Union that will convert two(2) ARRAY WORDs (AsWord) to a DWORD (AsDword)
2 //
3 TYPE UT_CONVERTDWORD : //"UnionType-Convert Types"
4 UNION
5     awValue: ARRAY[0..1] OF WORD; //"Array-Word-Vaue"
6     dwValue: DWORD; //"DoubleWord-Value"
7 END_UNION
8 END_TYPE
9
```

Create a UNION to convert to REAL

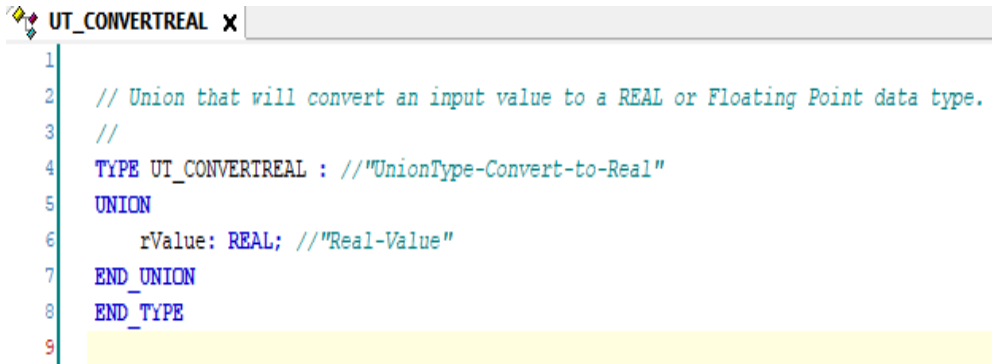
This Union is a custom, user defined *DataType* that will take in an **ARRAY** variable and create a **DWORD** representation of it.

(14). Add a Union by right-clicking on **Application > Add Object > DUT...**

A pop-up will appear entitled *Add DUT* (Device Unit Type)

(15). Select the **Union** item and name it **UT_CONVERTREAL** (“Union Type-Convert to REAL”)

(16). Fill in the file as shown below.



```

1 // Union that will convert an input value to a REAL or Floating Point data type.
2 //
3 TYPE UT_CONVERTREAL : //"UnionType-Convert-to-Real"
4 UNION
5     rValue: REAL; //"Real-Value"
6 END_UNION
7 END_TYPE

```

Create the STRUCTURE for the Powermeter Object

A *Structure* (**STRUCT**) is a user-defined data type, which combines multiple variables of any data type into a logical unit. The variables declared within a structure are called members.

For Object Oriented Programming, a *Structure* can be referred to as an *Object* for example a Motor and the attributes of the Motor like *Speed*, *RPM* and *Temperature* are all members of this object.

In our example of a Meter that measures 3-phase current, voltage, etc., we have have a structure that contains the Current related members only:

- Current Phase A Word 1 (**CuAW1**)
- Current Phase A Word 2 (**CuAW1**)
- Current Phase B Word 1 (**CuBW1**)
- Current Phase B Word 2 (**CuBW2**)
- Current Phase C Word 1 (**CuCW1**)
- Current Phase C Word 2 (**CuCW2**)
- Total Current Phase A (**CurrentA**) - Two Phase A WORDs → **DWORD**
- Total Current Phase B (**CurrentB**)
- Total Current Phase C (**CurrentC**)

(17). Add a Structure by right-clicking on **Application > Add Object > DUT...**

A pop-up will appear entitled *Add DUT* (Device Unit Type)

(18). Select the **Structure** item and name it **ST_POWERMETER** (“Structure Type-Power Meter”)

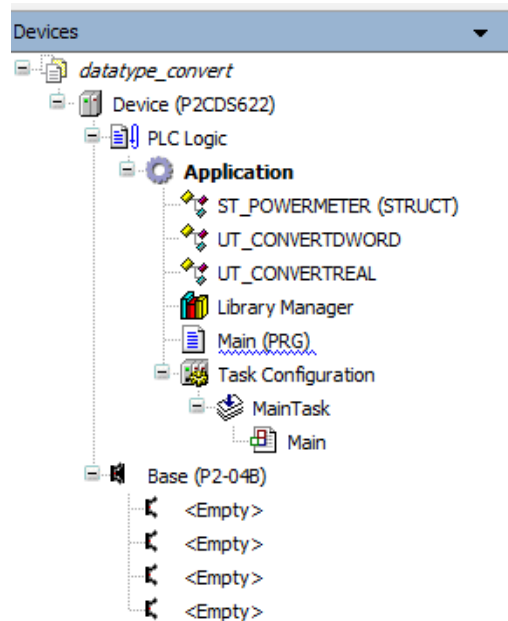
(19). Fill in the file as shown below.

```

1  // STRUCTURE that represnets the Meter as an OBJECT.
2  //
3  TYPE ST_POWERMETER : //"StructureType-Power Meter"
4  STRUCT
5      CuAW1  : WORD; // Current "I1", Word #1
6      CuAW2  : WORD; // Current "I1", Word #2
7      CuBW1  : WORD; // Current "I2", Word #1
8      CuBW2  : WORD; // Current "I2", Word #2
9      CuCW1  : WORD; // Current "I3", Word #1
10     CuCW2  : WORD; // Current "I3", Word #2
11     CurrentA : UT_CONVERTDWORD; //Current A is the Data Type "Union". Used to generate DWORD.
12     CurrentB : UT_CONVERTDWORD; //Current B is the Data Type "Union". Used to generate DWORD.
13     CurrentC : UT_CONVERTDWORD; //Current C is the Data Type "Union". Used to generate DWORD.
14 END_STRUCT
15 END_TYPE
16

```

Below is a view of the project tree with the Programmable Object Unit (POU) entitled Main and DUTs.



Create the Variables in Main

(20). In the Main program, enter the following variables:

```

Main x
1  PROGRAM Main
2  VAR
3      PowerMeter : ST_POWERMETER; //reference to STRUCTURE for elements
4      udwResult : UT_CONVERTDWORD; //Output of the Union "DWORD" conversion from WORD
5      urResult : UT_CONVERTREAL; //Output of the Union "REAL" conversion from DWORD
6
7      dwCurrentPhaseA: DWORD; //final DWORD value of the Meter current Phase A.
8      dwCurrentPhaseB: DWORD; //final DWORD value of the Meter current Phase B.
9      dwCurrentPhaseC: DWORD; //final DWORD value of the Meter current Phase C.
10     scadaCurrent : REAL; //SCADA currnet in Floating Point, AMPS (versus mA)
11 END_VAR

```

Create the Main Control Code

The main area of the code is explained in each functional section.

(21). In the Main program, enter the following control code.

The first section shown below, is emulating a possible current reading from the Meter's two data Words for *Current A - Word 1* and *Current A - Word 2*. The values chosen in Hex were selected so when the two Words are appended, it produces the 16-bit value 12345678 in decimal.

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two (2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1 := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2 := 16#00BC; //WORD size variable data for Currunt I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (urResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is deciaml "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue/1000;
21

```

The second section, takes each Word and assigns it to an ARRAY variable that can easily be appended to create a DWORD.

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1 := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2 := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue/1000;
21

```

The next line of code writes this Array variable into the Union (.awValue) location and in parallel a DWORD is generated in the Union as the .dwValue assignment.

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1 := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2 := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue/1000;
21

```

The fourth section copies this DWORD value and puts it into a new variable named dwCurrentPhaseA which gives us the decimal value of "12345678".

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1 := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2 := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue/1000;
21

```

To convert this value to a Real or Floating Point values, the urResult (datatype Union to convert to Real) is copied into the Union as shown below.

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1 := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2 := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue/1000;
21

```

And finally, the SCADA unit wants the data in Amps and since the value is in mA from the Meter i.e. 12345678 mA, we need to convert this to Amps by dividing by the mA value by 1000 to get 12345.678 amps.


```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1 := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2 := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue/1000;
21

```

A View of the “Task”

Another very important area in the Device Tree is the *Task Configuration* section which defines the program tasks are defined (which POU's are to run on a scan) and the characteristics of the task (priority, speed etc.).

The online help goes into detail for this area. The defaults are good for this project.

Compile the Project

Next, we will compile and build the project to download to the P2CDS-622 target.

(22). Select from the drop-down menus- *Build > Generate Code*.

19.6 Connect to PLC and Download

The project needs to be downloaded to the target but we must first connect to it.

(23). Physically connect the Host PC to one of the Ethernet ports on the P2CDS-622.

19.6.1 Configure Connection Channel to PLC

(24). In the Device Tree, Double-click on the **Device(P2CDS622)**.

A **Device** tab should appear with the first tab on the left selected entitled *Communication Settings* selected. Now we need to scan the network to find our device.

(25). In the *Communication Settings* section, click on the **Scan network**.

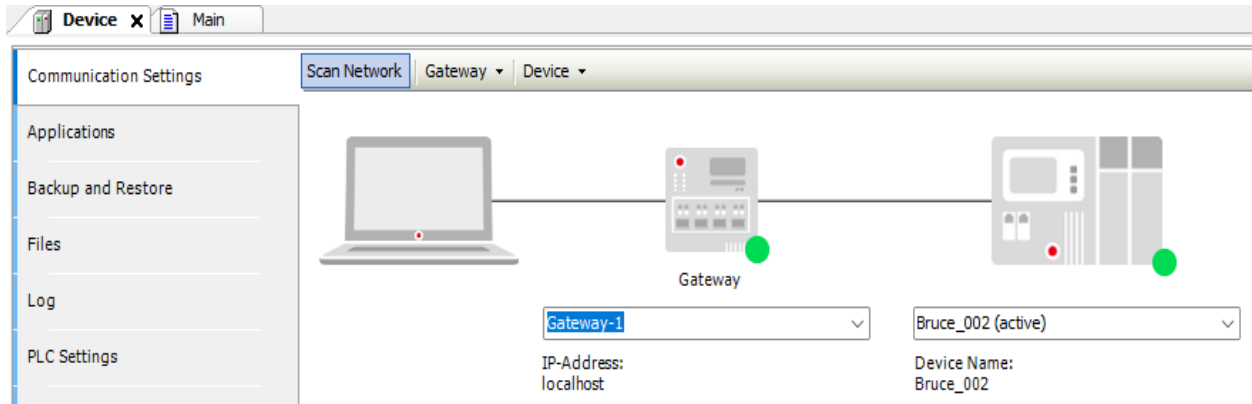
A dialog box should pop up with all the CODESYS PLCs available to connect to on this network.

Note: If you are unsure which unit is the one you want to connect to, on the right side is a **Wink** select box. Click on this and this P2CDS-622 front panel **RUN** light should start to flash for 5 seconds.

(26). Select the applicable P2CDS-622 unit and Click **OK**.

(27). Toggle the **Run/Stop** switch.

Now the following should be visible- both the Gateway and the P2CDS-622 have the green indicators present indicating the IDE is connected to the P2CDS-622.




19.6.2 Download Application to PLC

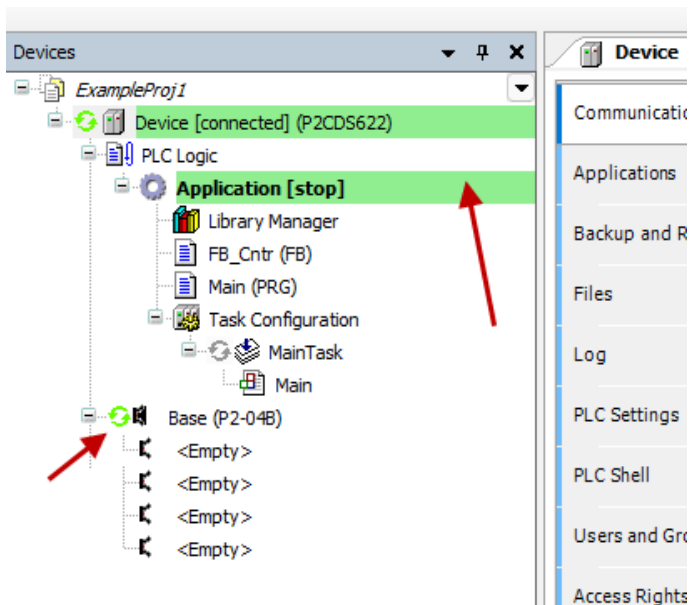
(28). Click **Online > Login**.

A dialog opens up if there is an existing application on the PLC, if so Click **Yes** to continue.

19.6.3 Successful Download. Ready to Run

The system should show green highlighted *Device* and *Application* as shown below:

The  icon indicates connection is active (but program is not executing).




19.7 Run and Watch the Application

This section will start the application running and using the debugger features, allow a user to watch the variables in real time, set breakpoints and force variables to a specific value.

19.7.1 Start the Application

(29). Click **Debug > Start**.

The IDE should now be showing a green “Run” in the lower portion of the status area. Also, the *Main Task* has the running icon  next to it.

19.7.2 Watch the Application

There are several ways to monitor the variables of the application program and to influence them in the watch view:

- Online views of individual POUs
- Write and force variable values
- Defined variable lists in separate watch lists

Online View of the POU

This view of a POU shows the current values of the watchable expressions contained in a table in the declaration part, and if enabled, also in the implementation section in the form of “inline monitoring”.

(30). To open the online view, in the device tree double-click **Main**, or select it and in the context menu click *Edit Object*.

In the center part of the view, you see the lines of code as specified in offline mode. They are supplemented by the orange inline monitoring views after each variable which show the current value.

In the top part, a table shows the watchable expressions of the POU- the corresponding variables with Type and current Value.

Main x

Device.Application.Main

Expression	Type	Value
PowerMeter	ST_POWERMETER	
udwResult	UT_CONVERTDWORD	
urResult	UT_CONVERTREAL	
dwCurrentPhaseA	DWORD	0
dwCurrentPhaseB	DWORD	0
dwCurrentPhaseC	DWORD	0
scadaCurrent	REAL	0

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1[0] := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2[0] := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0][0] := PowerMeter.CuAW1[0]; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1][0] := PowerMeter.CuAW2[0]; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA[0] := udwResult.dwValue[0]; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue[0] := dwCurrentPhaseA[0];
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent[0] := urResult.rValue[0]/1000;
21
22 RETURN

```

19.8 Debug the Application

One of the most powerful features of CODESYS is its debugging features. For example, you can set multiple breakpoints and use different step-through commands around lines of code.

19.8.1 Set Breakpoints

In Online/Login mode (not “Started” yet), you can set breakpoints where any program execution should be halted. When a breakpoint is reached, the program can be processed in steps. At each breakpoint and in each step, you can check the current value of the variables in the watch views.

(31). To demonstrate, “Main”, set the cursor in line 7. In the menu, select **Debug > Toggle Breakpoint** to insert a Breakpoint where your cursor is.

The breakpoint is displayed in the program.

(32). Click **Debug > Start** to run the program.

The application will run to line 7 where the breakpoint is and halt. It is currently in the stop status, it will look like:

```

1 //emulate the data coming from the Power meter
2 // Current A is made up of two(2) Words of Data for total of 32bits.
3 PowerMeter.CuAW1[24910] := 16#614E; //WORD size variable data for Current I1, Word 1 = 1234 decimal
4 PowerMeter.CuAW2[188] := 16#00BC; //WORD size variable data for Current I1, Word 2 = 5678h decimal
5
6 //write the WORD values into an Array format
7 PowerMeter.CurrentA.awValue[0] := PowerMeter.CuAW1[24910]; //Meter-CurrentA-WORDvalue-Element0
8 PowerMeter.CurrentA.awValue[1] := PowerMeter.CuAW2[188]; //Meter-CurrentA-WORDvalue-Element1
9
10 //write the array "WORDS" into the UNION (uResult) to do conversion (DWORD) store result in "udwResult"
11 udwResult.awValue := PowerMeter.CurrentA.awValue; //"Union-Result"
12
13 //Now grab the DWORD inside the UNION and store into final DWORD variable.
14 dwCurrentPhaseA := udwResult.dwValue; // result is decimal "12345678"
15
16 //write the DWORD value into the UNION (urResult) to do conversion (REAL) store result in "urResult"
17 urResult.rValue := dwCurrentPhaseA;
18
19 //SCADA system needs the current value /1000 (want data in Amps versus mA) and converted to Real/Float type.
20 scadaCurrent := urResult.rValue /1000;
21
22 RETURN

```

Note: The line where the breakpoint is executed/stops has NOT been executed yet.

19.8.2 Stepping Through the Program

Now you can press <F8> repeatedly to execute the *Step Into* command in the Debug menu and run the program in single-step mode. This also steps into the function block instance.

(33). To skip this function block processing, instead of <F8> you can press <F10> to execute the *Step Over* command.

The current variable values are displayed at the processing position just reached.

Tip: The debugger can be single stepped also with the use of the Hot Keys that can be found in the following link: [Hot Keys](#)

(34). See also the Breakpoints dialog **View > Breakpoints** The currently defined breakpoints are listed here and can be edited or new ones can be added.

Note: Note that the breakpoint positions are saved even when you log out of the PLC. The next time you log in, they will be displayed as light red markers and can be reactivated.

19.8.3 Execute Code Back to Beginning

If you hit a breakpoint and are stepping through and want to restart from the beginning of your program while still Online.

(35). Select **Online > Reset Warm**.

This will take you to the beginning of the program in *Stop* mode, and wait for you to initiate another *Start*.

ANALOG INPUT MODULE

This example project utilizes the four channel Analog 0-20mA Current input module- **P2-04AD-1**.

This project can also be easily modified to be used with an eight (8) channel analog input module, the P2-08AD-1.

Datasheets, specifications and setup for the module can be found here- [Current Input](#)

20.1 Introduction

The P2-04AD-1 Current Analog Input module provides four channels for receiving 0–20 mA signals for use with the P2CDS-622 system.

This module does not require any configuration of the input signal channels, i.e. each channels is fixed to a 0-20mA topology.

The programming language chosen for this project will be “Structured Text” and will be a very basic program that reads the mA Current value and sets some alarm variables based on the values.

The main purpose for this project is to introduce you to an Analog Input type modules and the associated configuration data flow and status signals.

20.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- P2-04AD-1 Current Module (or -08)
- Latest version of CODESYS installed on Host PC
- Familiarity with the “Structured Text” programming language.

Note: If Ethernet ports are not configured, see Refer to the [Ethernet](#) section.

20.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assigning simple password protection to prevent access to the functionality of your PLC over the Internet.

20.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not..

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

20.5 Create a Project

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU's.

20.5.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

(2). Click **File > New Project**

The example project here will be named *CurrentInput*

(3). In the *New Project* window, in the *Templates* section, select the **Standard Project** template.

(4). Specify a name and a storage location for the project and click **OK**

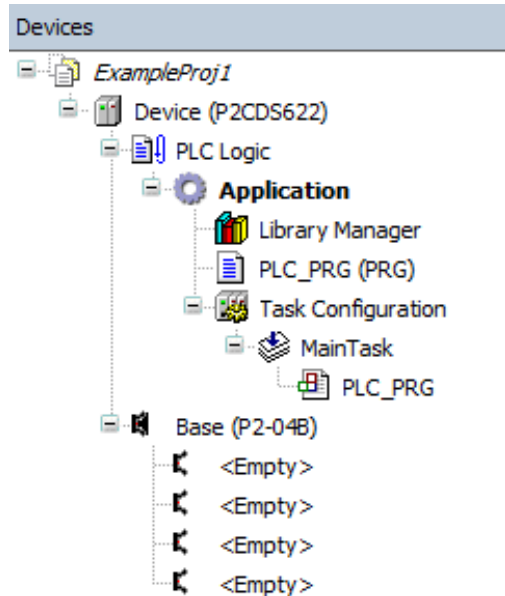
The project opens in the CODESYS *Standard Project* frame window.

(5). In the *Device* drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create (default name is *PLC_PRG*). We will rename this file later to my preference of *Main*.

- (6). In the *PLC_PRG* in drop-down location, select **Structured Text (ST)**. This will be the language we will use for the first program.

Your Project Tree view should look something like below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, we will change that later to match.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

Rename Design File and Add Fieldbusses

It is not required, but we will change the name of the *PLC_PRG* to *Main*.

- (7). Right Click on *PLC_PRG* and in the drop-down select **Properties**. In the **Common** dialog box, rename *PLC_PRG* to *Main*. Click **OK**.

- (8). A dialog box will prompt regarding the name change to be applied over the entire project. Click **Yes**.

Another dialog box entitled **Refactoring** will pop up displaying all the areas in the project where this new name will be applied. Click **OK**.

- (9). If a Fieldbus (Modbus or EtherNet/IP) was to be used, this is the point where you would add these. We do NOT require a Fieldbus for this project.

Refer to section [Fieldbus](#) for more information if desired.

Add Fieldbus

(Not applicable for this project. Refer to section *Ethernet* for more information if desired.)

Change Base Type (if req'd)

If you are using a Base topology other than the P2-04B, you can change it now.

(10). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**. The *Plug Device* dialogue box should appear.

(11). Under the *Miscellaneous* name, find the Base unit device you want and double-click. Close the box. The updated Base should be showing.

Add the IO Module P2-04AD-1

The IO Module will be pulled from the *Device Library* and inserted into Slot 1 of the Base (P2-04B).

(12). In the device tree, below the *Base (P2-04B)* element, in the top slot (Slot 1), highlight it with a single click, then Right-click. Select **Plug Device**.

The *Plug Device* dialogue box should appear.

(13). Under the *Miscellaneous* name, find the **P2-04AD-1** device and double-click on it. Close the box.

The module should now appear in the first Slot of the Base. Rename the current “Module” to “Slot1”.

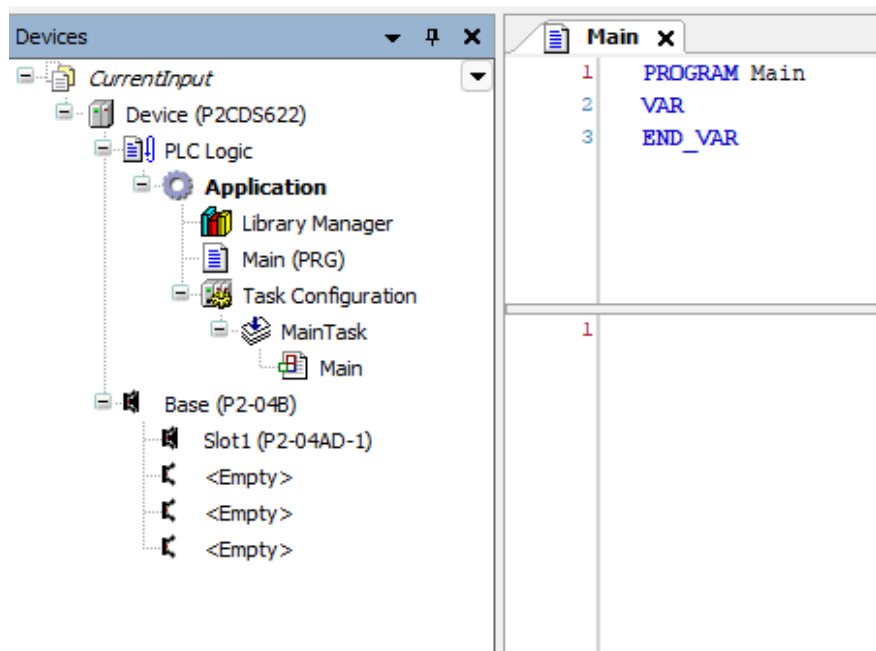
(14). Right Click on *Module* and in the drop-down select **Properties**. In the *Common* dialog box, rename *Module* to *Slot1*. Click **OK**. Save the project.

20.5.2 Write the Control Program

This simple program will read the Current input channels and based on certain values will illuminate ac certain LED that will be emulated in the visualization tool “WebVisu”.

(15). Double-click on *Main* to view the program file.

All POU's are comprised of two sections, the *Declaration* part that on the top defines the variables and their associated Data Types and the *Implementation* section where the code is entered.



A View of the Module

The parameters that require configuration and the associated status signals are defined in the IO Module Configuration section (P2-04AD-1)- *Configuration*

Parameters

See reference *Configuration - P2-04AD-1*

IO Mapping

The *IO Mapping* includes the Input Data and the Status signals.

Status

This indicates if the module is currently running.

Information

This gives general information regarding the module like description, revision, etc.

Create the Main Program

This program will first read the module status bits and determine if there are any error conditions.

Then the Current input channels are read and based on the Current range, an actionable step is taken.

In addition, a visualization will be created with some LEDs and some input text boxes to display the input channel current values.

IO Mapping

The Module has the following:

- Input current- Channels 1 -4
- Status signals: Channel Range Error and General Module status

(16). Name the Channel Input Data and Status signals as below.

P2_04AD_1 I/O Mapping		Variable	Mapping	Channel	Address	Type
Status		Ch1_Current_FI		Ch1	%ID0	DWORD
		Ch2_Current_FI		Ch2	%ID1	DWORD
		Ch3_Current_FI		Ch3	%ID2	DWORD
		Ch4_Current_FI		Ch4	%ID3	DWORD
Information		Status		Status	%IB16	
		Module Status		Module Status	%IB19	BYTE
		Status_Fail		Module Failed	%IX19.0	BOOL
		Status_M24v		Missing 24V	%IX19.1	BOOL
		Under Range Errors		Under Range Errors	%IB23	BYTE
		Status_Ch1_Under		Under Range Error Ch1	%IX23.0	BOOL
		Status_Ch2_Under		Under Range Error Ch2	%IX23.1	BOOL
		Status_Ch3_Under		Under Range Error Ch3	%IX23.2	BOOL
		Status_Ch4_Under		Under Range Error Ch4	%IX23.3	BOOL
		Over Range Errors		Over Range Errors	%IB27	BYTE
		Status_Ch1_Over		Over Range Error Ch1	%IX27.0	BOOL
		Status_Ch2_Over		Over Range Error Ch2	%IX27.1	BOOL
		Status_Ch3_Over		Over Range Error Ch3	%IX27.2	BOOL
		Status_Ch4_Over		Over Range Error Ch4	%IX27.3	BOOL

Implementation

Enter the following main functional code:

```

1  // "FI" indicates variable is from a Field device-Input into program
2  // "VO" indicates variable is the program out to an HMI (Visualization Out)
3
4  //Check the Module Status Flags. If any are in an error or out state, set the alarm
5  IF ( ( StatusFail_FI = TRUE ) OR ( StatusM24v_FI = TRUE ) OR ( StatusCh1Under_FI = TRUE ) OR ( StatusCh2Under_FI = TRUE ) OR
6      ( StatusCh3Under_FI = TRUE ) OR ( StatusCh4Under_FI = TRUE ) OR ( StatusCh1Over_FI = TRUE ) OR
7      ( StatusCh2Over_FI = TRUE ) OR ( StatusCh3Over_FI = TRUE ) OR ( StatusCh4Over_FI ) )
8  THEN
9      // alarm state
10     StatusBad := TRUE;
11     AlarmLED_VO := TRUE; //used to drive an alarm LED on an HMI panel.
12 ELSE
13     //no errors
14     StatusBad := FALSE;
15     AlarmLED_VO := FALSE;
16 END_IF
17
18 // Verify the Presure Sensors in valid range.
19 IF ( Ch1Current_FI >= SENSOR_MAX )
20 THEN
21     Ch1TooHigh := TRUE;
22 END_IF
23
24 // Verify the Presure Sensors in valid range.
25 IF ( Ch2Current_FI >= SENSOR_MAX )
26 THEN
27     Ch2TooHigh := TRUE;
28 END_IF
29
30 // Verify the Presure Sensors in valid range.
31 IF ( Ch3Current_FI >= SENSOR_MAX )
32 THEN
33     Ch3TooHigh := TRUE;
34 END_IF
35
36 // Verify the Presure Sensors in valid range.
37 IF ( Ch4Current_FI >= SENSOR_MAX )
38 THEN
39     Ch4TooHigh := TRUE;
40 END_IF
41

```

Declaration

These are the Variables and their associated Data Types that need to be entered:

```

1  PROGRAM Main
2  VAR
3      StatusBad: BOOL := FALSE; //internal alarm status variable.
4      AlarmLED_VO: BOOL := FALSE; //variable used to drive the HMI LED. "VO" = Visualation Out.
5      Ch1TooHigh: BOOL;
6      Ch2TooHigh: BOOL;
7      Ch3TooHigh: BOOL;
8      Ch4TooHigh: BOOL;
9  END_VAR
10
11 VAR CONSTANT
12     SENSOR_MAX: UINT := 48000; //Upper limit for readings for Pressure sensor. Max 19K.5 PSI
13 END_VAR

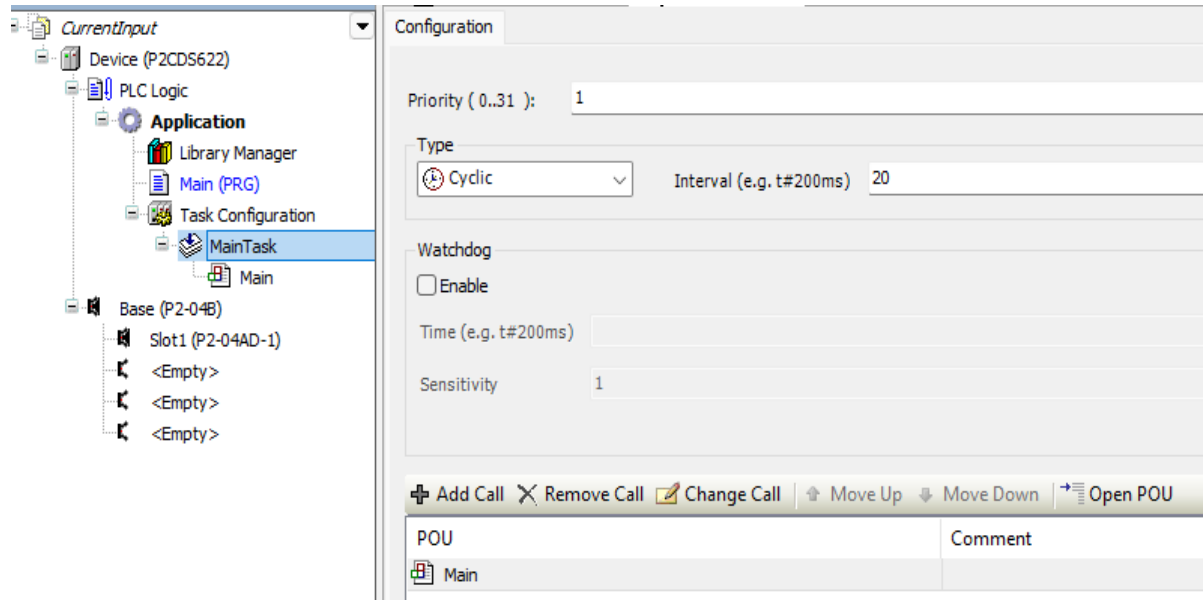
```

Tip: To help identify with variables are associated with an IO Module or a Fieldbus, you can append the suffix “_FI” (Field In) or “_FO” (Field output). This proves helpful when debugging your design.

The “Task” Configuration

Another very important area in the Device Tree is the *Task Configuration* section which defines the way in which the program tasks are defined (which POU's are to run on a scan) and the characteristics of the task (priority, speed etc.).

The online help goes into detail for this area. The defaults are good for this project.



Configuring the Module

The only item to configure in the *Parameters* section is the number of channels to be enabled.

Select the IO Module and in the **Parameters > Ch. Select** field Select Ch. Select 1-4.

Compile the Project

Next we will compile and build the project to download to the P2CDS-622 target.

(17). Select from the drop-down menus- **Build > Generate Code**.

20.6 Sensor/Input Wiring

The P2-04AD-1 is an input only, Current mode device. The inputs are driven by a 0-20mA source.

The resolution of the module is 16 bits, so the conversion counts will range from 0 - 65535d.

The following specification document goes into greater detail and shows the wiring requirements [P2-04AD-1](#)

20.7 Connect to PLC and Download

The project needs to be downloaded to the target and until now, we have not connected to it.

(18). Physically connect the Host PC to one of the Ethernet ports on the P2CDS-622.

20.7.1 Configure Connection Channel to PLC

(19). In the Device Tree, Double-click on the **Device(P2CDS622)**.

A **Device** tab should appear with the first tab on the left selected entitled *Communication Settings* selected. Now we need to scan the network to find our device.

(20). In the *Communication Settings* section, click on the **Scan network**.

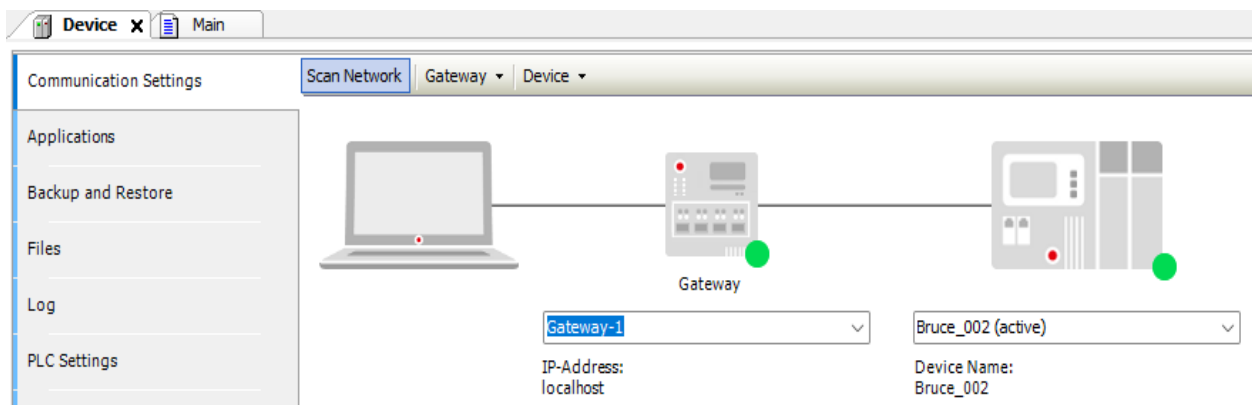
A dialog box should pop up with all the CODESYS PLCs available to connect to on this network.

Note: If you are unsure which CPU unit is the one you want to connect to, on the right side is a **Wink** select box. Click on this and this P2CDS-622 front panel **RUN** light should start to flash for 5 seconds.

(21). Select the applicable P2CDS-622 unit and Click **OK**.

(22). Toggle the **Run/Stop** switch.

Now the following should be visible- both the Gateway and the P2CDS-622 have the green indicators present indicating the IDE is connected to the P2CDS-622.




20.7.2 Download Application to PLC

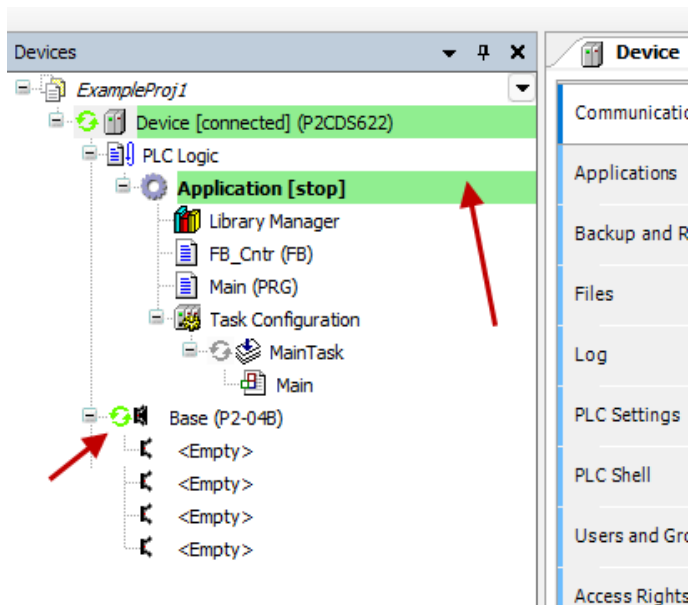
(23). Click **Online > Login**.

A dialog opens up if there is an existing application on the PLC, if so Click **Yes** to continue.

20.7.3 Successful Download. Ready to Run

The system should show green highlighted *Device* and *Application* as shown below:

The  icon indicates connection is active (but program is not executing).




20.8 Run and Watch the Application

This section will start the application running and using the debugger features, allow a user to watch the variables in real time, set breakpoints and force variables to a specific value.

20.8.1 Start the Application

(24). Click **Debug > Start**.

The IDE should now be showing a green “Run” in the lower portion of the status area. Also, the *Main Task* has the running icon  next to it.

20.8.2 Watch the Application

There are several ways to monitor the variables of the application program and to influence them in the watch view: 1. Online views of individual POU's 2. Write and force variable values 3. Defined variable lists in separate watch lists

Online View of the POU

This view of a POU shows the current values of the watchable expressions contained in it.

(25). To open the online view, in the device tree double-click **Main**, or select it and in the context menu click *Edit Object*.

In the bottom part of the view, you see the lines of code as specified in offline mode. They are supplemented by the little inline monitoring views after each variable which show the current value.

In the top part, a table shows the watchable expressions of the POU. This means the corresponding variables with Type and current Value.

Expression	Type	Value	Prepared value	Address	Comment
StatusBad	BOOL	TRUE			Internal alarm status variable.
AlarmLED_VO	BOOL	TRUE			variable used to drive ... HMI LED, "VO" = Vi...
Ch1TooHigh	BOOL	FALSE			
Ch2TooHigh	BOOL	FALSE			
Ch3TooHigh	BOOL	FALSE			
Ch4TooHigh	BOOL	FALSE			
SENSOR_MAX	UINT	48000			Upper limit for readings for Pressure sensor. ...

```

1 // "FI" indicates variable is from a Field device-Input into program
2 // "VO" indicates variable is the program out to an HMI (Visualization Out)
3
4 //Check the Module Status Flags. If any are in an error or out state, set the alarm
5 IF ( ( StatusFail_FI FALSE = TRUE ) OR ( StatusM24v_FI TRUE = TRUE ) OR ( StatusCh1Under_FI FALSE = TRUE ) OR ( StatusCh2Under_FI FALSE = TRUE ) OR
6   ( StatusCh3Under_FI FALSE = TRUE ) OR ( StatusCh4Under_FI FALSE = TRUE ) OR ( StatusCh1Over_FI FALSE = TRUE ) OR
7   ( StatusCh2Over_FI FALSE = TRUE ) OR ( StatusCh3Over_FI FALSE = TRUE ) OR ( StatusCh4Over_FI FALSE = TRUE ) )
8 THEN
9 // alarm state
10 StatusBad TRUE := TRUE;
11 AlarmLED_VO TRUE := TRUE; //used to drive an alarm LED on an HMI panel.
12 ELSE
13 //no errors
14 StatusBad TRUE := FALSE;
15 AlarmLED_VO TRUE := FALSE;
16 END_IF
17
18 // Verify the Presure Sensors in valid range.
19 IF ( Ch1Current_FI 0 >= SENSOR_MAX 48000 )
20 THEN

```

Writing and Forcing Variables

You can write or force a *Prepared Value* in the PLC's variable. The dialog is used to prepare or set a value that will be used when a *force* command of that value is initiated.

For example for the variable Ch1Current_FI, we want to write or force this variable to 50000d (decimal),

(26). In the **Prepared Value** column in the top section of the applicable program for the variable of interest e.g. Ch1Current_FI, select the field, press the spacebar to open an input field. Specify an integer value (50000d) and press the enter key or click outside of the field to close the field.

(27). In the menu, click **Debug > Write Values or Force Values**. You see the corresponding result in the *Value* column

Caution: Using the *Write Value* command writes the value for one(1) cycle then releases it (can be overwritten). The *Force* will always hold the value at that written value and cannot be overwritten until *Force* is removed. See the associated Online Help for further details.

Using Watch Lists

Watch lists can be used to compile in different lists the expressions from the application to be viewed. This can be useful for debugging purposes when specific variables need to be checked at a glance.

(28). In the menu, click **View > Watch - Watch 1**. The view opens.

(29). In the *Expression* column, click in the first row to open an input field. Specify the path for the variable to be watched. We recommend that you use the Input Assistant for this by right-clicking in the blank expression field, and select **Input Assistant**. Select for our example- *Device > Application > Main Ch1TooHigh*. Press the enter key to close the input field.

The data type of the variable is configured automatically in the row. Insert more rows for other variables if desired .

The screenshot shows the CODESYS Watch 1 window. The top part displays a ladder logic program with the following code:

```

19 IF ( Ch1Current_FI 0 >= SENSOR_MAX(48000) )
20 THEN
21   Ch1TooHighFALSE := TRUE;
22 END_IF
23
24 // Verify the Presusre Sensors in valid range.

```

Below the code is the Watch 1 table:

Expression	Application	Type	Value	Prepared value	Execution point
Main.Ch1TooHigh	Device.Application	BOOL	FALSE		Cyclic Monitoring

20.9 Debug the Application

One of the most powerful features of CODESYS is its debugging features. You can set multiple breakpoints and use a different step-through commands.

20.9.1 Set Breakpoints

In Online/Login mode (not “Started” yet), you can set breakpoints where any program execution should be halted. When a breakpoint is reached, the program can be processed in steps. At each breakpoint and in each step, you can check the current value of the variables in the watch views.

With the Ch1Current_FI sensor input forced to a 50000 (which is above the SENSOR_MAX threshold), the breakpoint should be hit.

(30). To demonstrate, select “Main” and set the cursor in line 21. In the menu, select **Debug > Toggle Breakpoint** to insert a Breakpoint where your cursor is.

The breakpoint is displayed in the program.

(31). Click **Debug > Start** to run the program.

The application will run to line 21 where the breakpoint is and halt. It is currently in the stop status, it will look like:


```
5 IF ( ( StatusFail_FI[FALSE] = TRUE ) OR ( StatusM24v_FI[TRUE] = TRUE ) OR ( StatusCh1Unde
6     ( StatusCh3Under_FI[FALSE] = TRUE ) OR ( StatusCh4Under_FI[FALSE] = TRUE ) OR ( Status
7     ( StatusCh2Over_FI[FALSE] = TRUE ) OR ( StatusCh3Over_FI[FALSE] = TRUE ) OR ( StatusCh
8 THEN
9     // alaram state
10    StatusBad[TRUE] := TRUE;
11    AlarmLED_VO[TRUE] := TRUE; //used to drive an alarm LED on an HMI panel.
12 ELSE
13    //no errors
14    StatusBad[TRUE] := FALSE;
15    AlarmLED_VO[TRUE] := FALSE;
16 END_IF
17
18 // Verify the Presusre Sensors in valid range.
19 IF ( Ch1Current_FI[50000] >= SENSOR_MAX[48000] )
20 THEN
21     Ch1TooHigh[FALSE] := TRUE;
22 END_IF
23
24 // Verify the Presusre Sensors in valid range.
```

Watch 1				
Expression	Application	Type	Value	Prepared value
Main.Ch1TooHigh	Device.Application	BOOL	FALSE	

Note: The line where the breakpoint is executed/stops has NOT been executed yet (Ch1TooHigh flag is still FALSE).

20.9.2 Stepping Through the Program

Now you can press <F8> repeatedly to execute the *Step Into* command in the Debug menu and run the program in single-step mode. This also steps into the function block instance.

(32). To skip this function block processing, instead of <F8> you can press <F10> to execute the *Step Over* command. The current variable values are displayed at the processing position just reached.

Tip: The debugger can be single stepped also with the use of the Hot Keys that can be found in the following link: [Hot Keys](#)

(33). See also the Breakpoints dialog **View > Breakpoints** The currently defined breakpoints are listed here and can be edited or new ones can be added.

Note: Note that the breakpoint positions are saved even when you log out of the PLC. The next time you log in, they will be displayed as light red markers and can be reactivated.

20.9.3 Execute Code Back to Beginning

So let's say you hit a breakpoint and are stepping through and want to restart from the beginning of your program while still Online.

(34). Select **Online > Reset Warm**.

This will take you to the beginning of the program in *Stop* mode, waiting for you to initiate another *Start*.

20.10 Additional Resources

- An excellent reference text in general for CODESYS development is entitled "*The Book of CODESYS*" by Pratt.

ANALOG IN/OUT VOLTAGE MODULE

This example project utilizes an Analog Voltage *Combo* (Input and Output) module.

Datasheets, specifications and setup for the module can be found here- [Voltage Combo Module](#)

21.1 Introduction

The P2-8AD4DA-2 supports eight(8) voltage inputs and four(4) voltage output signals.

The main purpose for this project is to introduce you to the Voltage Combo Module and its associated configuration, data flow and status signals.

21.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- P2-8AD4DA-2 Module
- Latest version of CODESYS installed on Host PC

Note: If Ethernet ports are not configured, see Refer to section [Ethernet](#).

21.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assign a simple password protection to prevent access to the functionality of your PLC over the Internet.

21.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not..

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

21.5 Create a Project

When creating a project, the main items that need to be done initially are to install the target device's (P2CDS-622) *Package* and pick the language that the program file will use.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU's.

21.5.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

(2). Click **File > New Project**

The example project here will be named *Demo_InOutVolt*

(3). In the *New Project* window, in the *Templates* section, select the **Standard Project** template.

(4). Specify a name and a storage location for the project and click **OK**

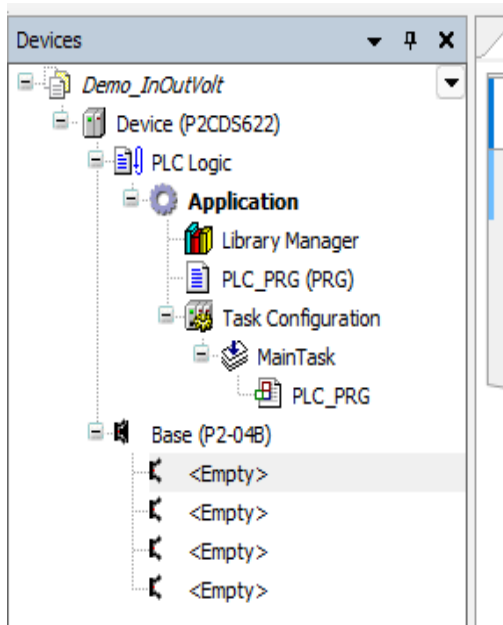
The project opens in the CODESYS *Standard Project* frame window.

(5). In the *Device* drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create (default name is *PLC_PRG*).

(6). In the *PLC_PRG in* drop-down location, select language of choice. In this example we chose **Structured Text (ST)**.

Your Project Tree view should look something like below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

Add Fieldbus

(Not applicable for this project. Refer to section *Ethernet* for more information if desired.)

Change Base Type (if req'd)

If you are using a Base topology other than the P2-04B, you can change it now.

(7). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**.

The *Plug Device* dialogue box should appear.

(8). Under the *Miscellaneous* name, find the Base unit device you want and double-click. Close the box.

The updated Base should be showing.

Add the IO Module P2-8AD2DA-2

The IO Module will be pulled from the *Device Library* and inserted into Slot 1 of the Base (P2-04B).

(9). In the device tree, below the *Base (P2-04B)* element, in the top slot (Slot 1), highlight it with a single click, then Right-click. Select **Plug Device**.

The *Plug Device* dialogue box should appear.

(10). Under the *Miscellaneous* name, find the **P2-8AD4DA-2** device and double-click on it and close the box.

The module should now appear in the first Slot of the Base.

21.5.2 Configure the Module

This can be done at any time during the project design.

(11). Double-click on the P2-P2-8AD4DA-2 module in the P2-04B device tree.

There are two main areas that require configuration- the *Parameters* section (number of channels, channel sensor type, etc.) and the *I/O Mapping* section which defines the variable names for the channels and status data.

Parameter	Type	Value	Default Value
Ch. Select	Enumeration of WORD	Ch. Select 1-8	Ch. Select 1-8
Input Ch1 Resolution	Enumeration of WORD	Fine	Fine
Input Ch1 Range	Enumeration of WORD	0-10V	0-10V
Input Ch2 Resolution	Enumeration of WORD	Fine	Fine
Input Ch2 Range	Enumeration of WORD	0-10V	0-10V
Input Ch3 Resolution	Enumeration of WORD	Fine	Fine
Input Ch3 Range	Enumeration of WORD	0-10V	0-10V
Input Ch4 Resolution	Enumeration of WORD	Fine	Fine
Input Ch4 Range	Enumeration of WORD	0-10V	0-10V
Input Ch5 Resolution	Enumeration of WORD	Fine	Fine
Input Ch5 Range	Enumeration of WORD	0-10V	0-10V
Input Ch6 Resolution	Enumeration of WORD	Fine	Fine
Input Ch6 Range	Enumeration of WORD	0-10V	0-10V
Input Ch7 Resolution	Enumeration of WORD	Fine	Fine
Input Ch7 Range	Enumeration of WORD	0-10V	0-10V
Input Ch8 Resolution	Enumeration of WORD	Fine	Fine
Input Ch8 Range	Enumeration of WORD	0-10V	0-10V
Output Ch1 Range	Enumeration of WORD	0-10V	0-10V
Output Ch2 Range	Enumeration of WORD	0-10V	0-10V
Output Ch3 Range	Enumeration of WORD	0-10V	0-10V
Output Ch4 Range	Enumeration of WORD	0-10V	0-10V
Hot Swap Enable	BOOL	FALSE	FALSE

A View of the Module

The parameters that require configuration and the associated status signals are defined in the IO Module Configuration section (P2-8AD4DA-2)- *Configuration*

Parameters

This section defines the Channel's Input and Output range and resolution of the conversions.

Module x

P2_8AD4DA_2 Parameters

P2_8AD4DA_2 I/O Mapping

Status

Information

Parameter	Type	Value	Default Value	Unit
Ch. Select	Enumeration of WORD	Ch. Select 1-8	Ch. Select 1-8	
Input Ch1 Resolution	Enumeration of WORD	Fine	Fine	
Input Ch1 Range	Enumeration of WORD	0-10V	0-10V	
Input Ch2 Resolution	Enumeration of WORD	0-10V	Fine	
Input Ch2 Range	Enumeration of WORD	0-5V	0-10V	
Input Ch3 Resolution	Enumeration of WORD	Fine	Fine	
Input Ch3 Range	Enumeration of WORD	0-10V	0-10V	
Input Ch4 Resolution	Enumeration of WORD	Fine	Fine	
Input Ch4 Range	Enumeration of WORD	0-10V	0-10V	
Input Ch5 Resolution	Enumeration of WORD	Fine	Fine	
Input Ch5 Range	Enumeration of WORD	0-10V	0-10V	
Input Ch6 Resolution	Enumeration of WORD	Fine	Fine	

IO Mapping

The *IO Mapping* area is where the user defines the variable names for the Input channel data and any associated Status signals they may want to monitor.

Channel Input Data

The data coming in on Channel One is named dwInput_Ch1, output data on channel 1 is dwOutput_Ch1. This is a Global variable type in this example, but you can limit its scope by referencing a variable assignment within your project and then assign that exact variable name in this section.

Module x

P2_8AD4DA_2 Parameters

P2_8AD4DA_2 I/O Mapping

Status

Information

Find

Filter






Show all

Variable	Mapping	Channel	Address	Type
		Output Channels	%QD0	
dwOutput_Ch1		Output Ch1	%QD0	DWORD
		Output Ch2	%QD1	DWORD
		Output Ch3	%QD2	DWORD
		Output Ch4	%QD3	DWORD
		Input Channels	%ID0	
dwInput_Ch1		Input Ch1	%ID0	DWORD
		Input Ch2	%ID1	DWORD
		Input Ch3	%ID2	DWORD
		Input Ch4	%ID3	DWORD
		Input Ch5	%ID4	DWORD
		Input Ch6	%ID5	DWORD
		Input Ch7	%ID6	DWORD
		Input Ch8	%ID7	DWORD
		Status	%IB32	

Status - General





















This indicates the overall Module status.

This includes if the Module has a fatal error `Module_Failed` or if there is valid data present based on the latest configuration settings `Missing 24V`.

			Module Status	%IB35	BYTE
	xComboVFail_Unit1		Module Failed	%IX35.0	BOOL
	xComboM24V_Unit1		Missing 24V	%IX35.1	BOOL

Status - Channel

This indicates if the channel is Over or Under the max/min range for the sensor type selected.

			Under Range Errors	%IB39	BYTE
	xUnder_Ch1		Under Range Error Ch1	%IX39.0	BOOL
			Under Range Error Ch2	%IX39.1	BOOL
			Under Range Error Ch3	%IX39.2	BOOL
			Under Range Error Ch4	%IX39.3	BOOL
			Under Range Error Ch5	%IX39.4	BOOL
			Under Range Error Ch6	%IX39.5	BOOL
			Under Range Error Ch7	%IX39.6	BOOL
			Under Range Error Ch8	%IX39.7	BOOL
			Over Range Errors	%IB43	BYTE
	XOver_Ch1		Over Range Error Ch1	%IX43.0	BOOL
			Over Range Error Ch2	%IX43.1	BOOL
			Over Range Error Ch3	%IX43.2	BOOL
			Over Range Error Ch4	%IX43.3	BOOL
			Over Range Error Ch5	%IX43.4	BOOL
			Over Range Error Ch6	%IX43.5	BOOL
			Over Range Error Ch7	%IX43.6	BOOL
			Over Range Error Ch8	%IX43.7	BOOL

Create the Main Program

Tip: For project creation, execution and debugging refer to example in *Analog Input Program*.

THERMOCOUPLE INPUT MODULE

This example project utilizes the eight(8) channel Thermocouple Temperature module- P2-08THM.

Datasheets, specifications and setup for the module can be found here- [Thermocouple Module](#)

22.1 Introduction

The P2-08THM Thermocouple Input Module provides eight differential channels for receiving thermocouple and voltage input signals.

The main purpose for this project is to introduce you to a P2-08THM Module and its associated configuration, data flow and status signals.

22.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- P2-08THM Temperature Module
- Latest version of CODESYS installed on Host PC

Note: If Ethernet ports are not configured, see Refer to section [Ethernet](#).

22.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assign a simple password protection to prevent access to the functionality of your PLC over the Internet.

22.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not.

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

22.5 Create a Project

When creating a project, the main items that need to be done initially are to install the target device's (P2CDS-622) *Package* and pick the language that the program file will use.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU's.

22.5.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

(2). Click **File > New Project**

The example project here will be named *Demo_P208THM*

(3). In the *New Project* window, in the *Templates* section, select the **Standard Project** template.

(4). Specify a name and a storage location for the project and click **OK**

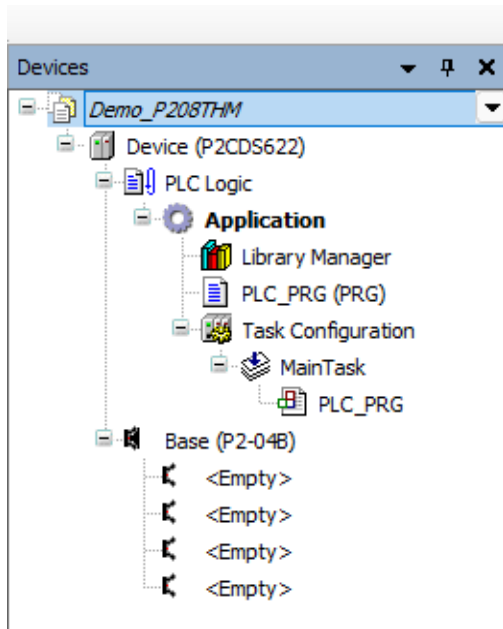
The project opens in the CODESYS *Standard Project* frame window.

(5). In the *Device* drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create (default name is *PLC_PRG*).

(6). In the *PLC_PRG in* drop-down location, select language of choice. In this example we chose **Structured Text (ST)**.

Your Project Tree view should look something like below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

Add Fieldbus

(Not applicable for this project. Refer to section *Ethernet* for more information if desired.)

Change Base Type (if req'd)

If you are using a Base topology other than the P2-04B, you can change it now.

(7). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**.

The *Plug Device* dialogue box should appear.

(8). Under the *Miscellaneous* name, find the Base unit device you want and double-click. Close the box.

The updated Base should be showing.

Add the IO Module P2-08THM

The IO Module will be pulled from the *Device Library* and inserted into Slot 1 of the Base (P2-04B).

(9). In the device tree, below the *Base (P2-04B)* element, in the top slot (Slot 1), highlight it with a single click, then Right-click. Select **Plug Device**.

The *Plug Device* dialogue box should appear.

(10). Under the *Miscellaneous* name, find the **P2-08THM** device and double-click on it and close the box.

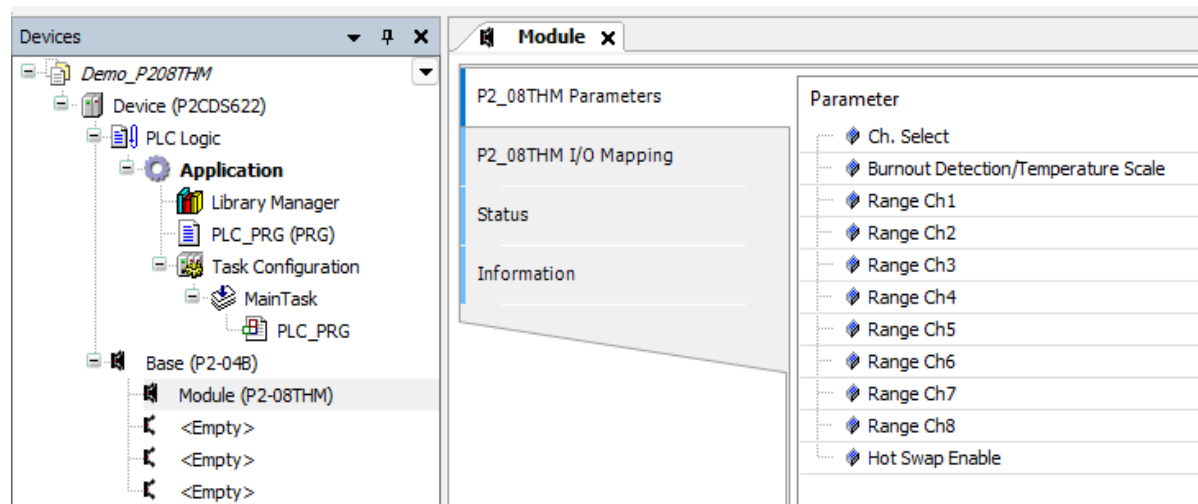
The module should now appear in the first Slot of the Base.

22.5.2 Configure the Module

This can be done at any time during the project design.

(11). Double-click on the P2-08THM module in the P2-04B device tree.

There are two main areas that require configuration- the *Parameters* section (number of channels, channel sensor type, etc.) and the *I/O Mapping* section which defines the variable names for the channels and status data.

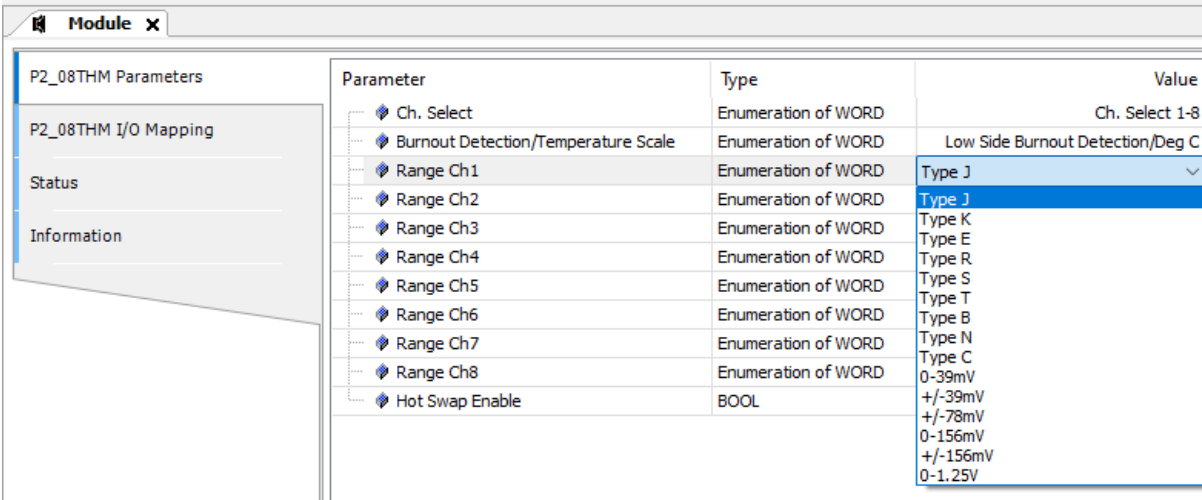


A View of the Module

The parameters that require configuration and the associated status signals are defined in the IO Module Configuration section (P2-08THM)- *Configuration*

Parameters

This section defines the Channel's Sensor type- Thermocouple or Voltage and associated range.

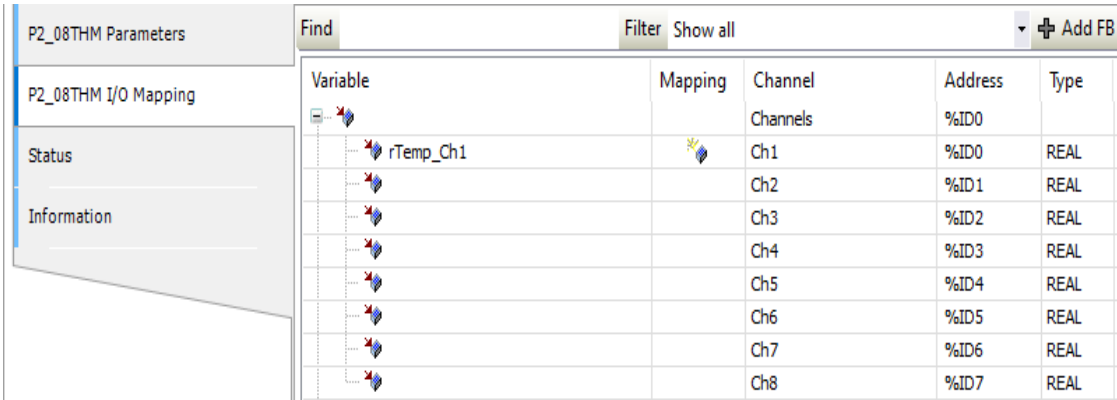


IO Mapping

The *IO Mapping* area is where the user defines the variable names for the Input channel data and any associated Status signals they may want to monitor.

Channel Input Data

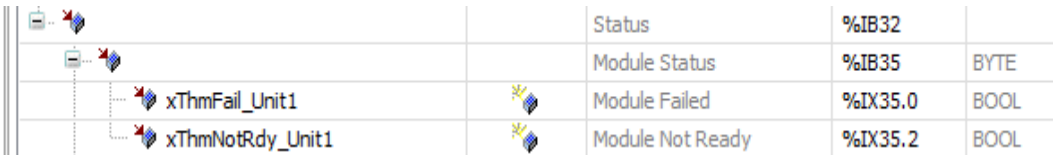
The data coming in on channel one is named *rTemp_Ch1*. This is a Global here, but you can also reference a variable assignment within your project by assigning that exact variable name in this section.



Status - General

This indicates the overall Module status.

This includes if the Module has a fatal error *Module_Failed* or if there is valid data present based on the latest configuration settings *Module Not Ready*.



Status - Channel

This indicates if the Thermocouple sensor has failed (burned out and open).

		Burnout Errors	%IB37	BYTE
		Burnout Error Ch1	%IX37.0	BOOL
		Burnout Error Ch2	%IX37.1	BOOL
		Burnout Error Ch3	%IX37.2	BOOL
		Burnout Error Ch4	%IX37.3	BOOL
		Burnout Error Ch5	%IX37.4	BOOL
		Burnout Error Ch6	%IX37.5	BOOL
		Burnout Error Ch7	%IX37.6	BOOL
		Burnout Error Ch8	%IX37.7	BOOL

This indicates if the channel is Over or Under the max/min range for the sensor type selected.

		Under Range Errors	%IB39	BYTE
		Under Range Error Ch1	%IX39.0	BOOL
		Under Range Error Ch2	%IX39.1	BOOL
		Under Range Error Ch3	%IX39.2	BOOL
		Under Range Error Ch4	%IX39.3	BOOL
		Under Range Error Ch5	%IX39.4	BOOL
		Under Range Error Ch6	%IX39.5	BOOL
		Under Range Error Ch7	%IX39.6	BOOL
		Under Range Error Ch8	%IX39.7	BOOL
		Over Range Errors	%IB43	BYTE
		Over Range Error Ch1	%IX43.0	BOOL
		Over Range Error Ch2	%IX43.1	BOOL
		Over Range Error Ch3	%IX43.2	BOOL
		Over Range Error Ch4	%IX43.3	BOOL
		Over Range Error Ch5	%IX43.4	BOOL
		Over Range Error Ch6	%IX43.5	BOOL
		Over Range Error Ch7	%IX43.6	BOOL
		Over Range Error Ch8	%IX43.7	BOOL

Create the Main Program

Tip: For Project Creation, Execution and Debugging refer to example in *Analog Input Project*.

MODBUS TCP - MASTER MODE

The Modbus TCP protocol uses a Client/Server (Master/Slave) architecture for data exchange.

Our example here is for a Modbus TCP implicit (cyclic) mode with data exchanges being managed by the Modbus TCP IOScanner. The Modbus TCP IOScanner is a service in the Device library that is based on Ethernet that polls slave devices continuously to exchange data, status, and diagnostic information.

For example, this is lieu of having specific (Explicit mode) Function Blocks in your program that generates the Modbus commands/data exchanges on demand.

This process monitors inputs and controls outputs of slave devices.

Clients (Masters) are devices that initiate any data exchange with other devices on the network. This applies to both I/O communications and service messaging.

The communication between the Modbus TCP IOScanner and the slave device is accomplished using Modbus TCP channels which are discussed in detail later.

23.1 Introduction

This project will utilize the Modbus TCP fieldbus device in the *Master* mode topology and requires a Modbus Slave device to be connected to the applicable Ethernet port.

The Slave device will be emulated using a Modbus TCP software application on a PC.

The P2CDS-622 Intro project will default to using the four(4) slot Base P2-04B unit. If your system has a different Base, this can be changed within your project.

The programming language chosen for this project will be “Structured Text”.

The main purpose for this project is to introduce you to the Modbus TCP Fieldbus and some Function Codes examples.

23.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- Latest version of CODESYS installed on Host PC
- Modbus Slave Simulator or Modbus Slave Device attached Example used in this project is “ModRSSinm2” found at: [Modbus Simulator](#)

- Familiarity with the “Structured Text” programming language.

23.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you still want to publish your web visualization, then we strongly recommend that you assign it at least a simple password protection to prevent access to the functionality of your PLC over the Internet.

23.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not.

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

23.5 Installation of Additional Software/Tools

To emulate an attached Modbus Slave, we will use a Modbus simulator tool- “ModRStsim2”.

This is located at: [Modbus Simulator](#) and needs to be installed.

Note: If this tool is unavailable, there are other Modbus emulation tools that should suffice.

23.6 Creating a Project

When creating a project, the main items that need to be done initially are to install the target device’s (P2CDS-622) *Package* and pick the language that the program file will use.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU’s.

Tip: NOTE: Below is a link to the **completed project** that can be downloaded entitled **Demo_ModbusTcpMaster**. The following instructions on this page provide details to the generation of this project.

Download the project here: [Modbus TCP project](#)

23.6.1 Start CODESYS and Create a Project

Tip: The following steps are for a guide to show you **how to create your own project** in lieu of the completed Demo project.

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

(6). Click *File > New Project*

The example project here will be named **ExampleProj1**.

(7). In the **New Project** window, in the **Templates** section, select the **Standard Project** template.

(8). Specify a name and a storage location for the project and click **OK**

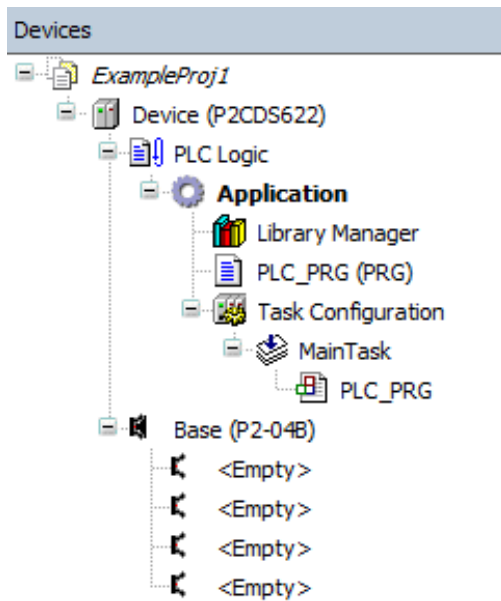
The project opens in the CODESYS **Standard Project** frame window.

(9). In the **Device** drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create (default name is **PLC_PRG**).

(10). In the **PLC_PRG in** drop-down location, select **Structured Text (ST)**. This will be the language we will use for the first program.

Your Project Tree view should look something like below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

Caution: If Ethernet ports have not been configured yet, it is required that they be setup per section [Ethernet](#) !

Add NetConfig

(11). Add this device as outlined in the section [Ethernet](#).

Add Fieldbus

The Modbus TCP device will reside under the **Ethernet** device in the *Device Tree*. The first step is to add an Ethernet device in the project.

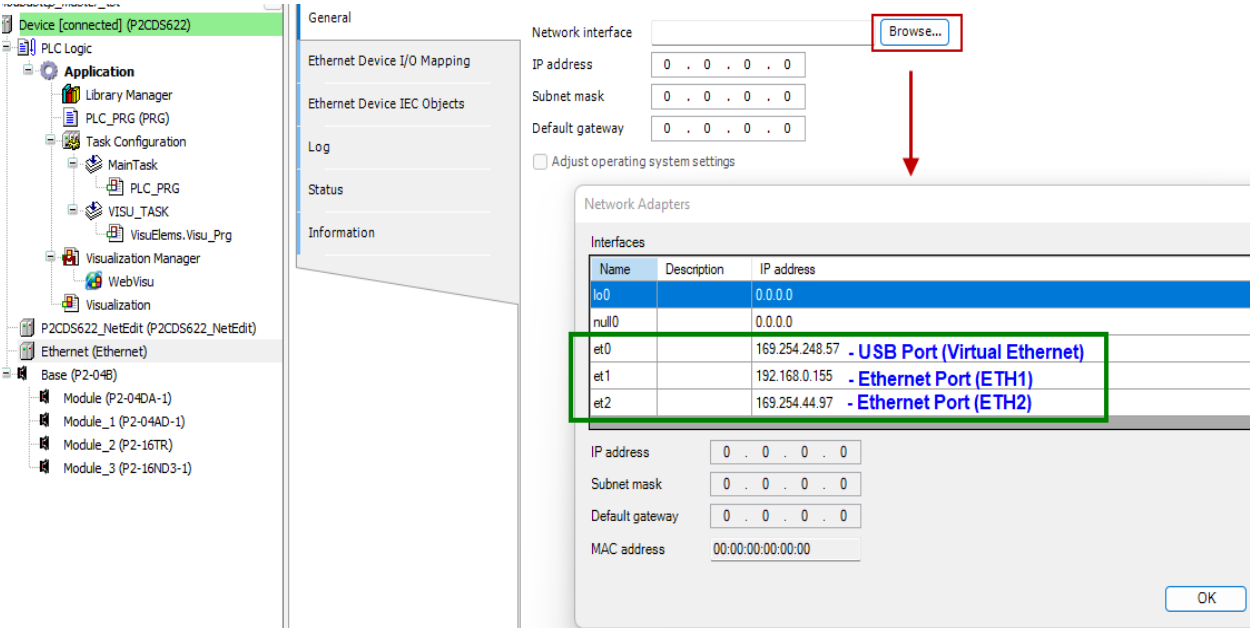
Add Ethernet Device

(12). Right Click on *Device (P2CDS622)* in the *Device Tree* and in the drop-down select **Add Device**. In the *Add Device* pop-up dialog box, Expand **Ethernet**, and select **Ethernet Adapter**. Double-click on **Ethernet** and this should add the Ethernet device to the Device Tree.

Assign/Configure Ethernet Device

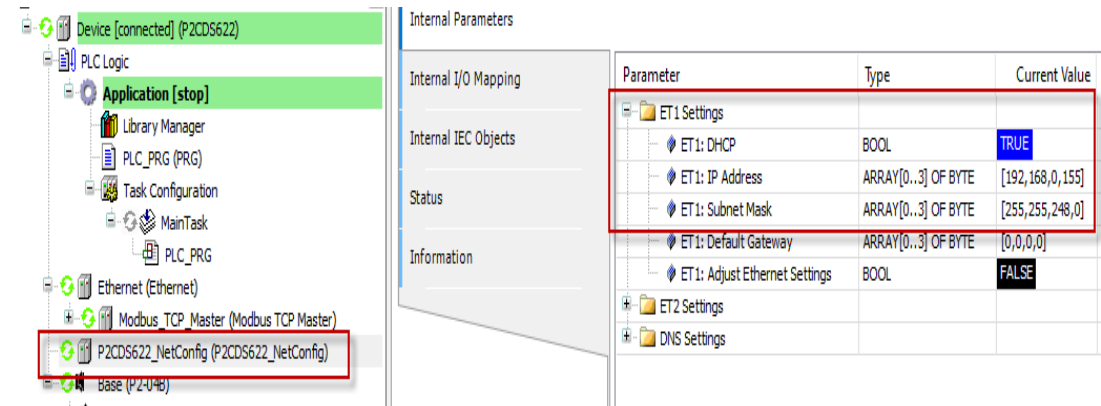
(13). Select the correct Ethernet port for this device by double-clicking on the **Ethernet** device and under the **General** > **Network interface** option area, browse to the applicable interface to use or you can define a *custom* IP Address, etc. by selecting the **Adjust operating system settings** and filling in accordingly.

Caution: The **Adjust operating system settings** is not recommended for most applications. This will override the settings for the Ethernet port that were defined in the setup with the *NetConfig* device and may cause undesired behavior for the Ethernet fieldbus port.



To verify that the assigned *Ethernet Device* IP address matches the Ethernet port, when the IDE is connected to the PLC and Logged in, you should see a IP address match.

For example the IP address of Ethernet port #1 (ET1) matches at 192.168.0.155.



Add Modbus-TCP Master

The Modbus TCP Master function will be assigned to this Industrial Ethernet channel.

(14). Right Click on *Ethernet (Ethernet)* and in the *Device Tree* drop-down select **Add Device**. In the *Add Device* pop-up dialog box, Expand **Fieldbusses**, expand **Modbus**, expand **Modbus_TCP_Master**. Double-click on **Modbus_TCP_Master** and this should add the Ethernet device to the Device Tree.

Configure Modbus-TCP Master

(15). In the **Modbus_TCP_Master** section, update any configuration parameters in the sections highlighted below.

General

- **Response timeout** (ms value) - Time interval (in milliseconds) for the master to wait for the response from a slave node.

If the nodes do not respond within this time span, then an error is issued for the implicit slave function block. The value specified for the time span is also the default value for each node. For each node, you can still set a specific value within its Modbus TCP slave configuration.

- **Socket timeout** (ms value) - Maximum time (in milliseconds) to wait for incoming TCP/IP packages

The bus cycle task can be blocked during this time. (Example: When a Modbus TCP slave is disconnected)

Modbus TCPMaster I/O Mapping

- **Bus Cycle task** (task) - Task associated with executing the Modbus cyclic data exchange.

Modbus TCPMaster IEC Objects

- (Do not modify)

Modbus TCPMaster Parameters

- **ExtendedChannelConfig** - (not modifiable).
- **OptimizationOn** - (not utilized, Vendor specific).
- **SocketTimeout** (ms value) - value reflects what was set in *General* > *Socket timeout* item.
- **ResponseTimeOut** (ms value) - value reflects what was set in *General* > *Response timeout* item.
- **AutoReconnect** - (TRUE, FALSE) - auto confirm connection error and reconnect.
- **ModbusTCP Slave Instance** - (not modifiable).

Add Modbus-TCP Slave

The Slave device that the Master will be connected to needs to be added under the current Master.

(16). Right Click on *Modbus_TCP_Master* and in the *Device Tree* drop-down select **Add Device**. In the *Add Device* pop-up dialog box, Expand **Fieldbuses**, expand **Modbus**, expand **Modbus_TCP_Slave**. Double-click on **Modbus_TCP_Slave** and this should add the Ethernet device to the Device Tree.

Configure Modbus-TCP Slave

(17). In the **Modbus_TCP_Slave** section, update any configuration parameters in the sections highlighted below.

General

- **Slave IP address** (ip addr) - the IP address assigned to the connected Modbus Slave.
- **Response timeout** (ms value) - Time interval (in milliseconds) for the master to wait for the response from a slave node.

If the nodes do not respond within this time span, then an error is issued for the implicit slave function block. The value specified for the time span is also the default value for each node. For each node, you can still set a specific value within its Modbus TCP slave configuration.

- **Port** (value) - Port number (TCP/IP) of the slave. Port 502 is the default for Modbus

The bus cycle task can be blocked during this time. (Example: When a Modbus TCP slave is disconnected)

Modbus Slave Channel

Each Modbus Channel needs to have its data exchanges specifically defined. Double-click on the Channel entry.

- **Channel - Name** (text) - optional string for naming the channel.
- **Channel - Access Type** (type) - type of Modbus request by Function Code (FC).
 1. *Read Coils* - (FC 1)
 2. *Read Discrete Inputs* - (FC 2)
 3. *Read Holding Registers* - (FC 3)
 4. *Read Input Registers* - (FC 4)
 5. *Write Single Coil* - (FC 5)
 6. *Write Single Register* - (FC 6)
 7. *Write Multiple Coils* - (FC 15)
 8. *Write Multiple Registers* - (FC 16)
 9. *Read/Write Multiple Registers* - (FC 23)
- **Channel - Trigger** (type) - when the request is to occur.
 1. *Cyclic* - request occurs periodically.
 2. *Rising Edge* - The request occurs as a reaction to a rising edge of the Boolean trigger variables. The trigger variable is defined on the *I/O Mapping* tab.
 3. *Application* - The Modbus request is triggered by the PLC application. This happens by means of the ModbusChannel function block (Explicit mode).
- **Channel - Cycle time** (ms value) - Request interval for a *Cyclic*. Should be the same as or a multiple of the application cycle time.
- **Channel - Comment** (text) - Description of the channel.
- **READ Register - Offset** (0-65535) - Start address where reading should start (value range 0-65535).
- **READ Register - Length** (value) - Number of registers to be read (for word access) or number of discrete inputs to be read (for bit access).
- **READ Register - Error Handling** (type) - If communication error, should previous data be kept (Keep...) or set to 0 (Set...).

- **WRITE Register - Offset** (0–65535) - Start address where writing should start (value range 0–65535).
- **WRITE Register - Length** (value) - Number of the register to be written to (value range 0–65535) .

Modbus Slave Init

You use this tab to define initialization commands. Initialization commands are executed one time when starting the bus or activating the slave (setting the “Enabled” flag of the slave instance). When setting up or editing a slave initialization value, the following parameters are available in the respective dialogs:

- **Move Up, Move Down** - the order of initialization.
- **New** - opens the Initialization Value dialog. The initialization commands are defined.
- **Access Type** (Write FC) - applicable Write Function Code (FC 05, FC 06, FC 15 or FC 16) to perform.
- **Register Offset** (0–65535) - actual number of the register to be written to (value range 0–65535).
- **Length** (value) - total number of registers to be written to (= words). The value range of the parameter depends on function code.
- **Initialization** (value) - initialization value for the register.
- **Comment** (text) -short description of the data.

ModbusTCPSlaveParameters

- **NewChannelConfig** (not modifiable).
- **Unit-ID** (1–255) - unit ID of the Modbus TCP slave device (by default 255).
- **ResponseTimeout** (ms value) - value reflects what was set in *General > Response timeout*
- **IPAddress** (IP ADDR) - value reflects what was set in *General > Slave IP Address*
- **Port** (value) - value reflects what was set in *General > Port* item.
- **Slave Diag** - multiple status items that are not modifiable.
- **Channel n** - multiple status items that are not modifiable.

ModbusTCPSlave IO Mapping

Assign the programs variables by Channel/Bit to be written out (WRITE FC's) or stored to (READ FC's).

ModbusTCPSlave IEC Objects

- (Do not modify)

Change Base Type (if req'd)

If you are using a Base topology other than the P2-04B, you can change it now.

(18). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**.

The *Plug Device* dialogue box should appear.

(19). Under the *Miscellaneous* name, find the Base unit device you want and double-click. Close the box.

The updated Base should be showing.

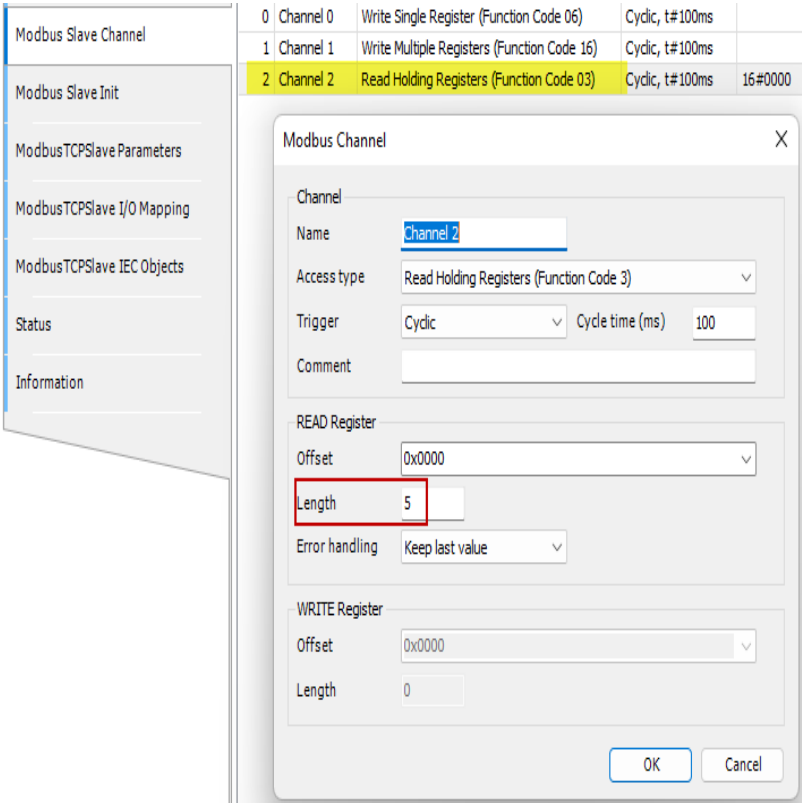
23.6.2 Example Program

The Archived project is located here: `Modbus TCP project`

(1). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. May have to add a Library, e.g. *Basic OS-CAT*.

This program will demonstrate writing a single value to the Slave Holding Register (FC-06), writing multiple values to the Holding Register (FC-16) and reading from the Slave a value with length five(5)- setup in the Channel configuration shown below.



Modbus TCP Master Emulator

To emulate an attached Modbus Slave, you can use a Modbus simulator tool like the one called “ModRSSim2”.

This is located at: [Modbus Simulator](#) and needs to be installed.

The following are the emulator registers in real time being displayed for this project.

- Holding Registers 400001d - 400005d are the values to be READ FROM the Slave (Function Code FC-03).
- Holding Register 400021d is the single value WRITTEN INTO the Slave (Function Code FC-06).
- Holding Registers 400051d - 400053d are the multiple values WRITTEN INTO the Slave (Function Code FC-16).

MODBUS Eth. TCP/IP PLC - Simulator (port: 502)

Connected (1/32) : (received/sent) (391505/391505) Serv. write Rx: Tx:

Address: ☐ H ☒ D I/O Holding Regs (400000) Fmt: decimal +/- Prot: MODBUS TCP/IP

Address	+0	+1	+2	+3	+4	+5
400001-400010	1	2	3	4	5	0
400011-400020	0	0	0	0	0	0
400021-400030	175	0	0	0	0	0
400031-400040	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0
400051-400060	79	158	237	0	0	0
400061-400070	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0

23.6.3 Closing Comments

Additional features like edge triggered enabling are available.

Help can be found at [Modbus Help](#)

For Modbus errors, this may be helpful [Modbus Errors](#)

MODBUS RTU - MASTER MODE

The Modbus RTU protocol uses a Client/Server (Master/Slave) architecture for data exchange. It utilizes the P2CDS-622 Serial Ports (RS-232 or RS-485 configurable).

Clients (Masters) are devices that initiate any data exchange with other devices on the network. This applies to both I/O communications and service messaging.

The communication parameters such as the settings of the serial ports (baud rate, port number), are predefined in the configurator, for example the settings of the serial ports (baud rate, port number). Modbus commands are defined in the configurator and are oriented to a specific Modbus slave.

The commands are processed by the device at specific intervals, or can be triggered programmatically. For predefined commands, I/O channels are generated automatically with variables that can be mapped (I/O mapping).

Our example here will illustrate the setup parameters for the Serial port and a few Modbus commands defined for the channels.

24.1 Introduction

This project will utilize the Modbus RTU fieldbus device in the *Master* mode topology and requires a Modbus Slave device to be connected to the applicable Serial port.

Serial #1 is the RJ-12 connector, while Comm port #2 is the 4-pin Terminal Block (TBLK) connector.

The P2CDS-622 Intro project will default to using the four(4) slot Base P2-04B unit. If your system has a different Base, this can be changed within your project.

The main purpose for this example is to introduce you to the Modbus RTU Fieldbus configuration and setup.

24.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- Latest version of CODESYS installed on Host PC

24.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assigning a simple password protection to prevent access to the functionality of your PLC over the Internet.

24.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not..

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

24.5 Installation of Additional Software/Tools

To emulate an attached Modbus Slave, you can use a Modbus simulator tool like “ModRSSim2”.

This is located at: [Modbus Simulator Utility](#) and needs to be installed.

Note: If this tool is unavailable, there are other Modbus emulation tools that should suffice.

24.6 Creating a Project

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

When creating a project, the main items that need to be done initially are to install the target device’s (P2CDS-622) *Package* and pick the language that the program file will use.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU’s.

24.6.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Create a project

(6). Click *File > New Project*

The example project here will be named **ExampleRtu**.

(7). In the **New Project** window, in the **Templates** section, select the **Standard Project** template.

(8). Specify a name and a storage location for the project and click **OK**

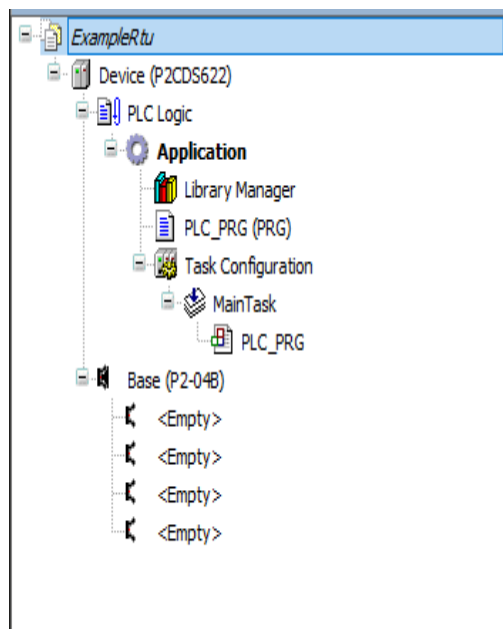
The project opens in the CODESYS **Standard Project** frame window.

(9). In the **Device** drop-down location, select the **P2CDS622** device.

The next step will be to define the language for the initial program you will create (default name is **PLC_PRG**).

(10). In the **PLC_PRG in** drop-down location, select your programming language of choice e.g. **Structured Text (ST)**.

Your Project Tree view should look something like below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing **<F1>** in the device tree area and searching for a specific item of interest.

Add Modbus-Serial Fieldbus

For this example configuration, we will use the RS-485 electrical interface over the 4-pin Terminal Block (Port2) Comm port.

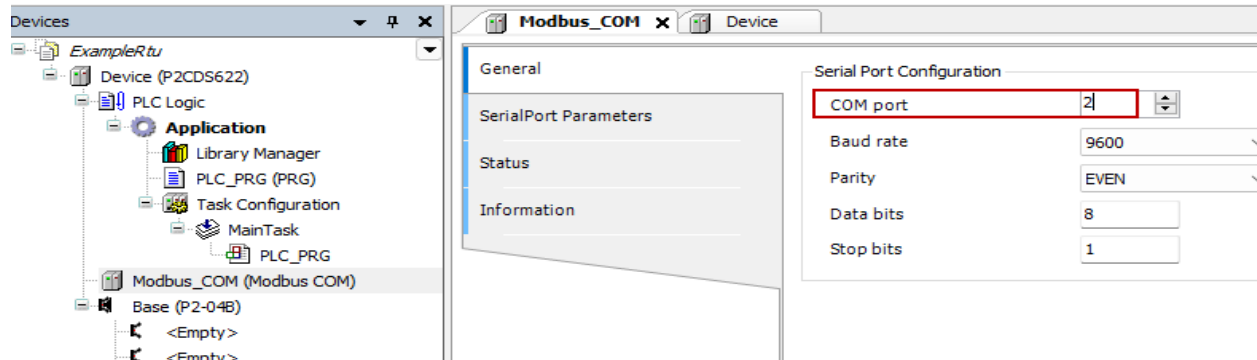
(11). Right -click on **Device (P2CDS622)** and select **Add Device > Fieldbuses > Modbus > Modbus Serial Port > Modbus COM**

Configure the Serial Port

Note: **RJ-11 connector** corresponds to Serial Com port #1, **4-pin TBLK** is Serial Com port #2.

To assign the Comm Port to “2”, setup the Baud Rate, etc. for this device,

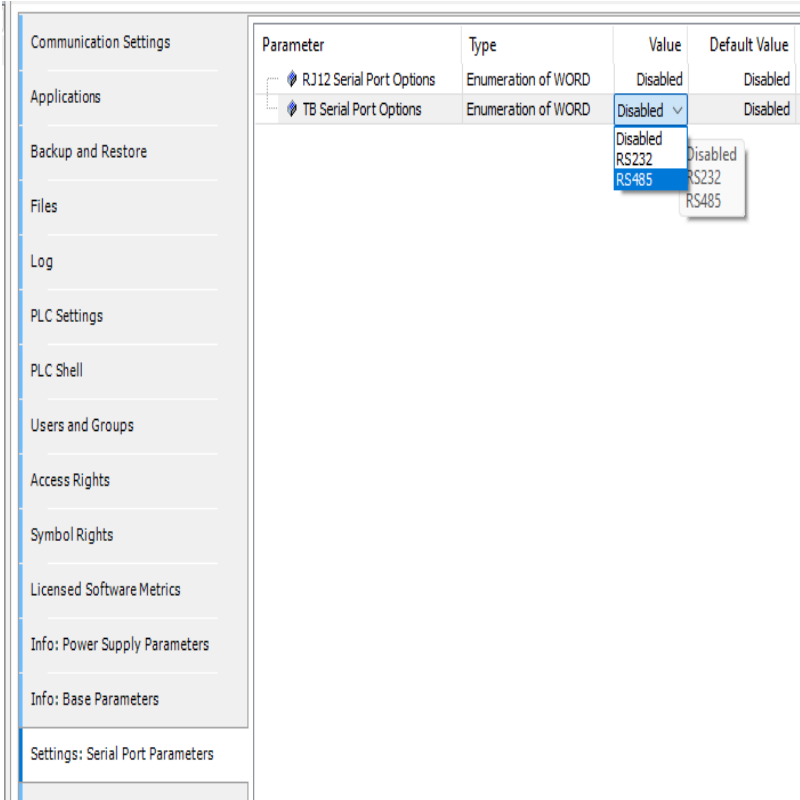
(12). Double-click on the **Modbus_COM** device just added.



To assign the Comm Port as a RS-485 type interface,

(13). Double-click on the **Device (P2CDS622)** in the device tree and look for **Settings: Serial Port Parameters**.

(14). In drop down for **Value** in the **TB Serial** section, select as shown:



No other settings here to change.

Add Modbus-Serial MASTER Type

(15). Right -click on **Modbus_COM** and select **Add Device > Fieldbuses > Modbus > Modbus Serial Master > Modbus Master, COM Port**

Configure the Master

(16). Double-click on the **Modbus_Master_COM** device just added and select **RTU** and any other parameters under **General** you want.

Note: No other tabs e.g. ModbusGeneric.. are to be modified.

Add Modbus-Serial SLAVE Type

(17). Right -click on **Modbus_Master_COM** and select **Add Device > Fieldbuses > Modbus > Modbus Serial Slave > Modbus Slave, COM Port**

Tip: There is a **downloadable example project** in the Modbus TCP-Master Mode section that may be a further help for **Channel Function setup**. See- *ModbusTCP project*

Configure the Slave

(18). Double-click on the **Modbus_Slave_COM** device just added modify parameters as desired under **General** you want.

General

- **Slave address** (1..247) - Address of a serial Modbus device (value between 1 and 247).
- **Response timeout** (ms value) - Maximum time (in milliseconds) to wait Time interval for the master to wait for the response from a slave node. This is configured especially for this slave node and overwrites the general response timeout setting of the respective master.

Modbus Slave Channel

Each Modbus Channel needs to have its data exchanges specifically defined.

(19). Click on the lower right hand button named **Add Channel...** setup the channels in your project as desired.

- **Channel - Name** (text) - optional string for naming the channel.
- **Channel - Access Type** (type) - type of Modbus request by Function Code (FC).
 1. *Read Coils* - (FC 1)
 2. *Read Discrete Inputs* - (FC 2)
 3. *Read Holding Registers* - (FC 3)
 4. *Read Input Registers* - (FC 4)
 5. *Write Single Coil* - (FC 5)
 6. *Write Single Register* - (FC 6)
 7. *Write Multiple Coils* - (FC 15)
 8. *Write Multiple Registers* - (FC 16)
 9. *Read/Write Multiple Registers* - (FC 23)
- **Channel - Trigger** (type) - when the request is to occur.
 1. *Cyclic* - request occurs periodically.
 2. *Rising Edge* - The request occurs as a reaction to a rising edge of the Boolean trigger variables. The trigger variable is defined on the *I/O Mapping* tab.
 3. *Application* -The Modbus request is triggered by the PLC application. This happens by means of the ModbusChannel function block (Explicit mode).
- **Channel - Cycle time** (ms value) - Request interval for a *Cyclic*. Should be the same as or a multiple of the application cycle time.
- **Channel - Comment** (text) - Description of the channel.
- **READ Register - Offset** (0-65535) - Start address where reading should start (value range 0-65535).
- **READ Register - Length** (value) - Number of registers to be read (for word access) or number of discrete inputs to be read (for bit access).
- **READ Register - Error Handling** (type) - If communication error, should previous data be kept (Keep...) or set to 0 (Set...).
- **WRITE Register - Offset** (0-65535) - Start address where writing should start (value range 0-65535).
- **WRITE Register - Length** (value) - Number of the register to be written to (value range 0-65535).

Modbus Slave Init

You use this tab to define initialization commands. Initialization commands are executed one time when starting the bus or activating the slave (setting the “Enabled” flag of the slave instance). When setting up or editing a slave initialization value, the following parameters are available in the respective dialogs:

- **Move Up, Move Down** - the order of initialization.
- **New** - opens the Initialization Value dialog. The initialization commands are defined.
- **Access Type** (Write FC) - applicable *Write* Function Code (FC 05, FC 06, FC 15 or FC 16) to perform.
- **Register Offset** (0–65535) - actual number of the register to be written to (value range 0–65535).
- **Length** (value) - total number of registers to be written to (= words). The value range of the parameter depends on function code.
- **Initialization** (value) - initialization value for the register.
- **Comment** (text) - short description of the data.

ModbusGenericSerialSlaveParameters

- (Do not modify)

These reflect any settings that have been made previously.

ModbusGenericSerialSlave IEC Objects

- (Do not modify)

Tip: There is a **downloadable example project** in the Modbus TCP-Master Mode section that may be a further help for **Channel Function setup**. See- *ModbusTCP project*

24.6.2 Closing Comments

At this point you should be ready to write your specific application code.

Additional features like edge triggered enabling are available.

Help can be found at [Modbus Help](#)

SERIAL COMMUNICATIONS

25.1 Introduction

The *Syscom* Library has several built-in functions to support communications.

This example project utilizes the *Sycom* Library to create a serial communications configuration to pass data such as ASCII character strings.

25.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- Latest version of CODESYS installed on Host PC

25.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assigning a simple password protection to prevent access to the functionality of your PLC over the Internet.

25.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not.

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

25.5 Import the Archived Project

Download the example project here- [Serial Communications](#)

25.5.1 Start CODESYS and Import the Project

Start CODESYS

(1). Start CODESYS from the Start menu by default, the path is **Programs > CODESYS** (version).

You can also click the CODESYS icon that is located on the desktop after installation.

Import the project

(2). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. You also may need to add a Library, e.g. *Basic OSCAT*.

The example project here will be named *Demo_SerialPassthrough*

Project Notes

- Functions are pulled from the *Syscom* library.
- Includes both the Transmit and Receive functions so you can do a loop-back test.
- All documentation is included in the *Syscom* library files.

ETHERNET/IP SCANNER - EXPLICIT MODE

Within EtherNet/IP (ENIP), the explicit message connection can be thought of as a client/server relationship. The client which is called a *Scanner*, such as the PLC controller, asks or requests the information from the server which is called an *Adapter*. The Adapter can be a VFD field device, and the adapter sends the requested information back to the controller.

Because the Scanner requests the information from the adapter via TCP/IP services, the request has all the information needed to respond explicitly to the message. The Scanner basically says, “Adapter, I need this information, formatted exactly as specified in this message, please send it.”

The Adapter responds with a message containing the formatted information, perhaps setting the VFD speed set point.

This configuring and monitoring ability, common to explicit messaging, works well for non-real time messaging as the Scanner can send a message request anytime, and the Adapter (field device) can respond when its available.

Note: Explicit messaging utilizes the *TCP* protocol and is typically used for client/server communication which is not time critical, but insuring that the communications packet is received is paramount.

26.1 Project Scope

This project will setup an ENIP Master or Scanner on the P2CDS-622 talking to a Raspberry Pi that is configured as the target or Adapter.

We will utilize the two explicit commands- *Get_Attribute_Single* to access data in the server and *Set_Attribute_Single* to control a variable in the Adapter.

This project will demonstrate:

- Adding a EtherNet/IP Master fieldbus to the project.
- Configuring the fieldbus parameters for connection.
- Example program code that will write and read access a servers/slave's *Object* in Explicit *Unconnected Messaging* mode.
- Illustrate the normal operation visual aspects.

The objects within the Server we will access are the *Identity Object* (Class 0x01h) and the *Assembly Object* (Class 0x04h).

The IDENTITY Class Object (0x01h) is defined as follows:

Instance

- Identity - ID 0x01h

Attributes

- Vendor ID - ID 0x01h
- Device Type - ID 0x02h
- Product Code - ID 0x03h

The ASSEMBLY Object (0x04) is defined as follows:

Instance

- Output Buffer - ID 0x64h
- Input Buffer - ID 0x65h

Attributes

- Get/Set - ID 0x03h

Services

- Get Single - ID 0x0Eh
- set Single - ID 0x10h

26.2 Create/Import the Scanner P2CDS-622 Project

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

Two projects are provided in an *Archive* format. One is the Scanner or Originator that will run on the P2CDS-622 and the Adapter that will be hosted by a Raspberry Pi unit.

The Scanner project provided is **downloadable below in an *Archive* format**. It is the Scanner or Originator that will run on the P2CDS-622.

Download here-ENIP ExplicitScanner project

The Adapter or Target is provided also for reference and was run on a Raspberry Pi 4 Model B using an evaluation license from CODESYS that can be found here <https://us.store.codesys.com/codesys-control-for-raspberry-pi-sl.html>.

The **downloadable RPi Adapter project**- is here- RPI Adapter.

26.2.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

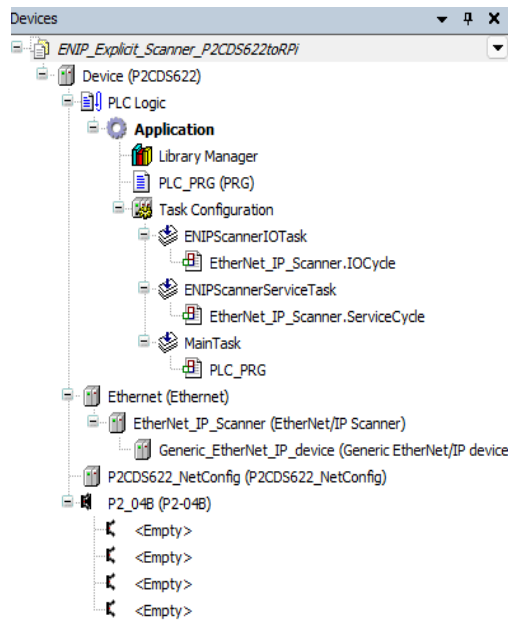
Import the project

Tip: If you want to start a project from scratch, do not import the project and just follow the steps in the next sections.

(2). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. You may need to add a Library, e.g. *Basic OSCAT*.

Your Project Tree view should look something like the image below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

26.2.2 Project Comments

Tip: The following steps are for a guide to show you **how to create your own project** in lieu of the completed Demo project.

Network Configuration “NetConfig”

(3). Setup Ethernet ports by adding the *NetConfig* functionality per *NetConfig*

Fieldbus Comm Port: “Ethernet”

The first step in adding the ENIP fieldbus is to instantiate an Ethernet port, assign it to the applicable P2CDS-622 port.

- Ethernet Port: Add Device > Fieldbus > Ethernet Adapter > Ethernet
- Assign IP Address: Ethernet > General > Browse

The edits that are required are as follows:

- Under the *General* tab, select the applicable Ethernet port out of the P2CDS-622. The IP address should match the expected.

EtherNet/IP Originator: “EtherNet_IP_Scanner”

Next the ENIP Master/Scanner is added.

(4). Right click on Ethernet: Add Device > Fieldbus > EtherNet/IP Scanner > EtherNet/IP Scanner

The edits that are required are as follows:

- (use defaults)

EtherNet/IP Target: “Generic_EtherNet_IP_device”

The Generic adapter will be added. NOTE: if you have the connected device/adapter’s *EDS* file it can be imported into the Device Directory and then added in lieu of “generic”

(5). Right click on Ethernet_IP_Scanner: Add Device > Fieldbus > EtherNet/IP Remote Adapter > Generic EtherNet/IP device

The edits that are required are as follows:

- Under *General* tab, assign the IP Address that corresponds to the connected Adapter.
- All other items can be default.

Program “PLC_PRG”

The program that has been provided is comprised of two sections.

Section 1 - Read the Adapter data

- Use the `getAttributeSingle` Explicit service (Code 0x0E) from the EtherNet/IP Library located at

The number of bytes to receive in this example is three(3) and are stored in local variables `bTempMotor1-3`.

The function is executed whenever `startEIP` is enabled.

Section 2 - Write out to the Adapter

- Use the `setAttributeSingle` Explicit service (Code 0x0E) from the EtherNet/IP Library.

The variable `bScannerWriteVar` is incremented to be able to see the variable changing and be verified at the Adapter also.

The function is executed whenever `startEIP2` is enabled.

26.3 Create/Import the Adapter Raspberry PI Project

The second project is the Adapter or Target and is provided also for reference. It is targeted for the Raspberry Pi 4 Model B using an evaluation license from CODESYS that can be found at [RPI License](#)

26.3.1 Start CODESYS and Create a Project

(As in the above example)

Fieldbus Comm Port: “Ethernet”

The first step in adding the ENIP fieldbus is to instantiate an Ethernet port, assign it to the applicable P2CDS-622 port.

- Ethernet Port: Add Device > Fieldbus > Ethernet Adapter > Ethernet
- Assign IP Address: Ethernet > General > Browse

The edits that are required are as follows:

(6). Under the *General* tab, select the applicable Ethernet port out of the P2CDS-622. The IP address should match the expected.

EtherNet/IP Target: “EtherNet_IP_Adapter”

Next the ENIP Target/Adapter is added.

(7). Right click on Ethernet: Add Device > Fieldbus > EtherNet/IP > EtherNet/IP Local Adapter Scanner > EtherNet/IP Adapter

The edits that are required are as follows:

- (use defaults or General can be modified as desired)

EtherNet/IP Target: “Generic_EtherNet_IP_device”

The Generic adapter will be added. NOTE: if you have the connected device/adapter’s *EDS* file it can be imported into the Device Directory and then added in lieu of “generic”

(8). Right click on Ethernet_IP_Adapter: Add Device > Fieldbus > EtherNet/IP > EtherNet/IP Module > EtherNet/IP Module

The edits that are required are as follows, under the “Assemblies” tab:

- The Consuming Assembly (Scanner data → Adapter) with Object ID of 0x64h. One(1) variable, single byte.
- The Producing assembly (Adapter data → Scanner) with Object ID of 0x65h. Three(3) variables, single byte.
- Additional items could be added if desired.

In the “EtherNet/IP Module I/O Mapping” tab, variables have been assigned to the respective assembly.

Program “PLC_PRG”

The program that has been provided reads in the Scanner data (*bMotor_VoltSet*) and writes three bytes out to the Scanner. (*bTemperature_Motor11-3*).

ETHERNET/IP SCANNER - IMPLICIT MODE

EtherNet/IP uses implicit messaging, sometimes called I/O messaging, for time-critical applications such as real-time control. Implicit messaging is often referred to as I/O messaging because it's frequently used for communication between a controller and remote I/O.

It's a much more efficient communication connection than explicit messaging as both the client and server ends are pre-configured to know implicitly or exactly what to expect in terms of communication.

Real-time, implicit messaging basically copies data with minimal additional information built into the message. Each end of the communication link doesn't need to be told much about what the information is for as the data has been predefined. The meaning of the data is "implied" or "implicit" so there is no extra baggage because both ends already know exactly what each bit and byte mean.

Note: Explicit messaging utilizes the *UDP* protocol and is typically used for continuous updating of data packets in time critical client/server communications.

27.1 Project Scope

This project will setup an ENIP Master or Scanner on the P2CDS-622 talking to a Raspberry Pi that is configured as the target or Adapter.

We will utilize the two explicit commands- *Get_Attribute_Single* to access data in the server and *Set_Attribute_Single* to control a variable in the Adapter.

This project will demonstrate:

- Adding a EtherNet/IP Master fieldbus to the project.
- Configuring the fieldbus parameters for connection.
- Example program code that will write and read access a servers/slave's *Object* in Explicit *Connected Messaging* mode.
- Illustrate the normal operation visual aspects.

The objects within the Server we will access are the *Identity Object* (Class 0x01h) and the *Assembly Object* (Class 0x04h).

The IDENTITY Class Object (0x01h) is defined as follows:

Instance

- Identity - ID 0x01h

Attributes

- Vendor ID - ID 0x01h
- Device Type - ID 0x02h
- Product Code - ID 0x03h

The ASSEMBLY Object (0x04) is defined as follows:

Instance

- Output Buffer - ID 0x64h
- Input Buffer - ID 0x65h

Attributes

- Get/Set - ID 0x03h (which is not used in this project).

Services

- Get Single - ID 0x0Eh (which is not used in this project).
- Set Single - ID 0x10h (which is not used in this project).

27.2 Create/Import the Scanner P2CDS-622 Project

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

Two projects are provided in an *Archive* format. One is the Scanner or Originator that will run on the P2CDS-622 and the Adapter that will be hosted by a Raspberry Pi unit.

This Scanner project is **downloadable in an *Archive* format**- ENIP Implicit Scanner project

It is the Scanner or Originator that will run on the P2CDS-622.

The Adapter or Target is provided also for reference and was run on a Raspberry Pi 4 Model B using an evaluation license from CODESYS that can be found here <https://us.store.codesys.com/codesys-control-for-raspberry-pi-sl.html>.

The **downloadable RPi Adapter project** is here- RPI Adapter.

27.2.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs > CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

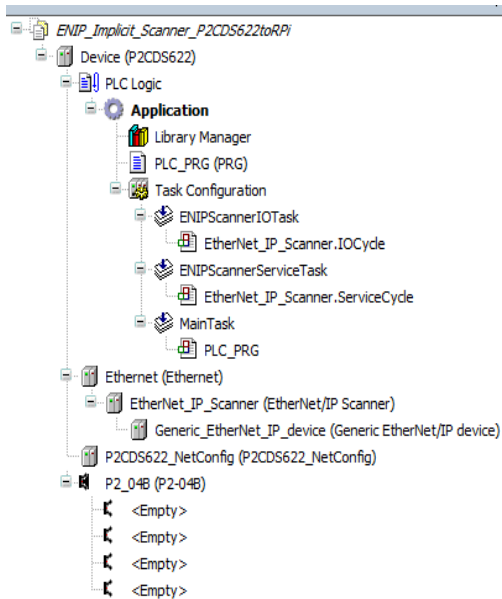
Import the project

Tip: If you want to start a project from scratch, do not import the projects and just follow the steps after the *Import*.

(2). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. You may need to add a Library, e.g. *Basic OSCAT*.

Your Project Tree view should look something like the image below with the Name of your project and the associated Device (P2CDS622) as the target.



The default Base (*P2-04B*) is inserted in the tree with no modules added yet, i.e. *<Empty>*. If your hardware topology is a different Base, see the section below on changing the Base type.

The elements of the project tree can be investigated further by going to the **Help** section online by pressing <F1> in the device tree area and searching for a specific item of interest.

27.2.2 Project Comments

Tip: The following steps are for a guide to show you **how to create your own project** in lieu of the completed Demo project.

Network Configuration “NetConfig”

(3). Setup Ethernet ports by adding the *NetConfig* functionality per *NetConfig Device*

Fieldbus Comm Port: “Ethernet”

The first step in adding the ENIP fieldbus is to instantiate an Ethernet port, assign it to the applicable P2CDS-622 port.

- Ethernet Port: Add Device > Fieldbus > Ethernet Adapter > Ethernet
- Assign IP Address: Ethernet > General > Browse

The edits that are required are as follows:

- Under the *General* tab, select the applicable Ethernet port out of the P2CDS-622. The IP address should match the expected.

EtherNet/IP Originator: “EtherNet_IP_Scanner”

Next the ENIP Master/Scanner is added.

(4). Right click on Ethernet: Add Device > Fieldbus > EtherNet/IP Scanner > EtherNet/IP Scanner

The edits that are required are as follows:

- (use defaults)

EtherNet/IP Target: “Generic_EtherNet_IP_device”

The Generic adapter will be added. NOTE: if you have the connected device/adapter’s *EDS* file it can be imported into the Device Directory and then added in lieu of “generic”

(5). Right click on Ethernet_IP_Scanner: Add Device > Fieldbus > EtherNet/IP Remote Adapter > Generic EtherNet/IP device

The edits that are required are as follows:

- Under *General* tab, assign the IP Address that corresponds to the connected Adapter.
- Under the *Connections* tab, add a connection that looks like the following (these define the Assembly IDs for the Scanner and Adapter):

This “connection” will now show up in the **Assemblies** tab and you can name accordingly.

In the **EtherNet/IP I/O Mapping** tab section, variables are assigned to these IOs.

Program “PLC_PRG”

The program is very simple- Scanner output variables are set to values to send out and Adapter input variables are received.

27.3 Create/Import the Adapter Raspberry PI Project

The second project is the Adapter or Target and is provided also for reference. It is targeted for the Raspberry Pi 4 Model B using an evaluation license from CODESYS that can be found at [RPI License](#)

27.3.1 Start CODESYS and Create a Project

(As in the above example)

Fieldbus Comm Port: “Ethernet”

The first step in adding the ENIP fieldbus is to instantiate an Ethernet port, assign it to the applicable P2CDS-622 port.

- Ethernet Port: Add Device > Fieldbus > Ethernet Adapter > Ethernet
- Assign IP Address: Ethernet > General > Browse

The edits that are required are as follows:

(6). Under the *General* tab, select the applicable Ethernet port out of the P2CDS-622. The IP address should match the expected.

EtherNet/IP Target: “EtherNet_IP_Adapter”

Next the ENIP Target/Adapter is added.

(7). Right click on Ethernet: Add Device > Fieldbus > EtherNet/IP > EtherNet/IP Local Adapter Scanner > EtherNet/IP Adapter

The edits that are required are as follows:

- (use defaults or General can be modified as desired)

EtherNet/IP Target: “Generic_EtherNet_IP_device”

The Generic adapter will be added. NOTE: if you have the connected device/adapter’s *EDS* file it can be imported into the Device Directory and then added in lieu of “generic”

(8). Right click on Ethernet_IP_Adapter: Add Device > Fieldbus > EtherNet/IP > EtherNet/IP Module > EtherNet/IP Module

The edits that are required are as follows, under the “Assemblies” tab:

- The Consuming Assembly (Scanner data → Adapter) with Object ID of 0x64h. One(1) variable, single byte.
- The Producing assembly (Adapter data → Scanner) with Object ID of 0x65h. Three(3) variables, single byte.
- Additional items could be added if desired.

In the “EtherNet/IP Module I/O Mapping” tab, variables have been assigned to the respective assembly.

Program “PLC_PRG”

The program that has been provided takes reads in the Scanner data (*bMotor_VoltSet*) and writes out to the Scanner the three bytes (*bTemperature_Motor11-3*).

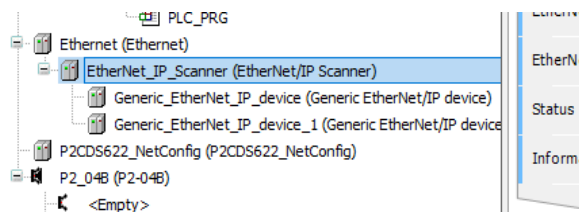
ETHERNET/IP SCANNER - MULTI-ADAPTERS

If the PLC is utilizing the EtherNet/IP Scanner fieldbus and is connected to **multiple** Adapters (with each Adapter required to have its own unique IP address), the Device Tree should have only **one(1) Scanner assigned per Ethernet port**.

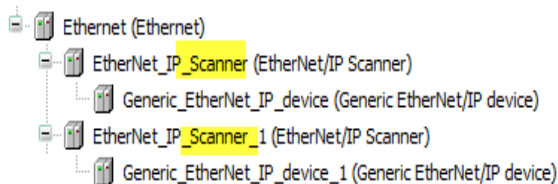
Then the Adapters are inserted under this Scanner.

28.1 Devices

The correct Device Tree is shown below for multiple Adapters. One Scanner, multiple Adapters under:



Caution: You do **not** want the topology below where there are **more than one(1)** Scanners under a single Ethernet port.



IIoT - MQTT

MQTT (Message Queuing Telemetry Transport) is an open-source IoT protocol that allows for the transmission of telemetry data as messages between devices. Device communication always takes place by means of an MQTT broker (e.g. Mosquitto <https://mosquitto.org/>).

Messages are sent and subscribed to based on topic names. A topic corresponds to a path (e.g. device1/temperature). Subscribing to messages is done by specifying a topic filter. Wildcards are also permitted (+ for one level and # for multiple levels).

The message format is not fixed, which means that a JSON string or any data structure can be transmitted. With the “MQTT Client SL” library, messages can be sent from a CODESYS controller to an MQTT broker, and messages can be subscribed to based on topics of interest.

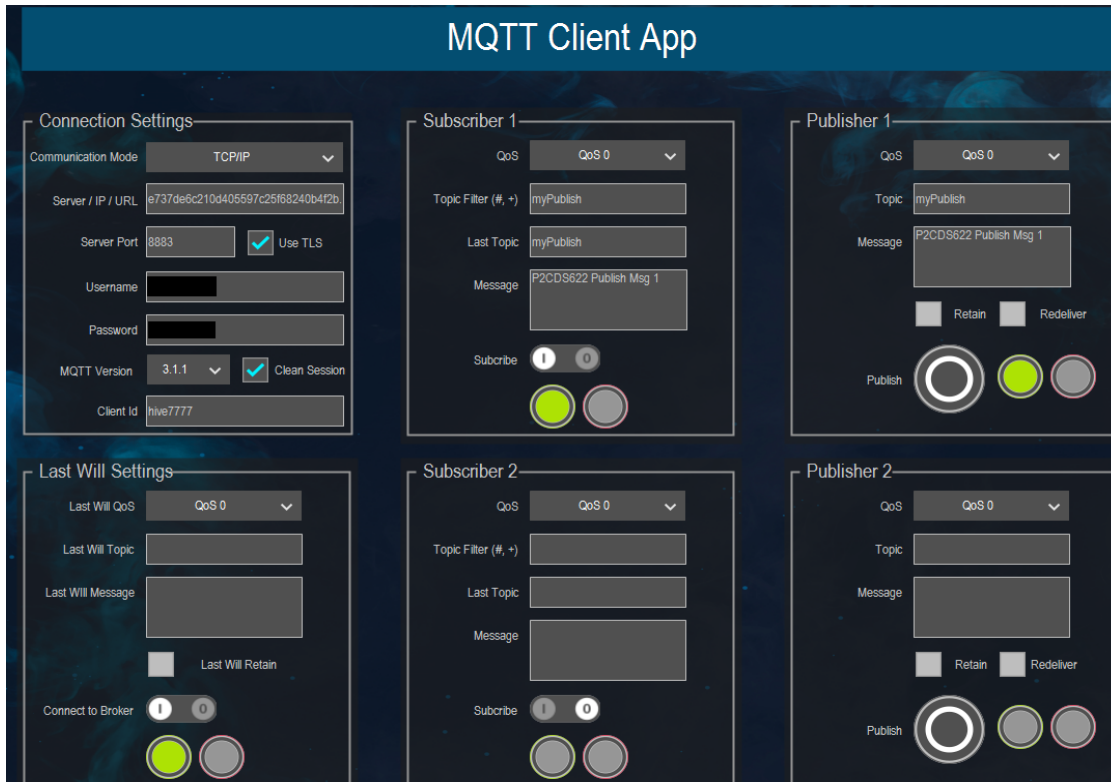
29.1 Introduction

This project will utilize the IIoT Library MQTT functions.

The P2CDS-622 Intro project will default to using the four(4) slot Base P2-04B unit. If your system has a different Base, this can be changed within your project.

This project will not be using any IO Modules, but will be used to demonstrate using the CODESYS debugger on the P2CDS-622 hardware.

The main purpose for this project is to introduce you to the MQTT capabilities based on the IIoT library that is included.



29.2 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- Latest version of CODESYS installed on Host PC
- Familiarity with the “Structured Text” and Continuous Function Chart (CFC)” programming languages.

29.3 Recommendation for Data Protection

In order to minimize the risk of data security breaches, we recommend the following organizational and technical measures for the system that will run your applications:

- Whenever possible, avoid exposing PLCs and controller networks to public networks and Internet.
- Use additional data link layers for protection, such as a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

If you want to publish a web visualization, then we strongly recommend utilizing HTTPS and assigning a simple password protection to prevent access to the functionality of your PLC over the Internet.

29.4 Help

Help is provided in the **Help** menu, as well as the context-sensitive <F1> key when in the IDE whether a project is open or not.

Currently, the web-based online help opens by default. This can be disabled in the CODESYS options so that the offline help (CHM format) is used.

29.5 Install IIoT Library

To access the free IIoT Library functions e.g. Web Client functions, you must install the IIoT Single license with the CODESYS Installer.

This license is located on the CODESYS website store at [IIoT License**](#)

29.6 Creating a Project

**A completed example project was created based on a known good MQTT Broker and can be downloaded here-
MQTT project**

Note: It is possible you will need to update the P2CDS-622 CPU Device in this project to the latest version. Right-click **Device** > **Update Device** and select latest version.

The following steps illustrate how this project was developed.

When creating a project, the main items that need to be done initially are to install the target device's (P2CDS-622) *Package* and pick the language that the program file will use.

The project can be made up of many program/design *PRG* files. These are referred to as *Programmable Object Units* or POU's.

29.6.1 Start CODESYS and Create a Project

Start CODESYS

(1). Start CODESYS from the Start menu (by default, the path is **Programs** > **CODESYS** (version)).

You can also click the CODESYS icon that is located on the desktop after installation.

Install the "IIoT Libraries SL" Package

(2). The IIoT Library should be available in the Installer, if not Download the IIoT package from the CODESYS Store: [IIoT](#)

Note: This Library does **NOT require any payment** to download! Just click on the **Download** button.

CODESYS Installer

- (3). Select from the drop-down menus- *Tools > CODESYS Installer*
- (4). In the **CODESYS Installer** pop-up view, click on *Install File* under **AddOns**.
- (5). Browse to the **IIoT Libraries SL x.x.x.x.pkg** file and double click.
- (6). The install will start. After completed, close the IDE and restart CODESYS.

Create a project

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

We are going to use an example project that is based on the project in installed with the IIoT library to modify.

For reference, this can be found at the install directory on your PC at **\$USER\$MQTT Client SL(version)**

A completed example project was created based on a known good MQTT Broker and can be downloaded here-
MQTT project

- (7). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.
- (8). Update **everything (including Library’s)** to latest revisions. May have to add a Library, e.g. *Basic OSCAT*.
- (9). Setup Ethernet ports by adding the *NetConfig* functionality per [NetConfig Device](#)

Add Fieldbus

(Not required)

Add Ethernet Device

(Not required)

Change Base Type (if req'd)

If you are using a Base topology other than the P2-04B, you can change it now.

- (10). In the device tree, highlight with a single click the **Base (P2-04B)** element, then Right-click. Select **Plug Device**. The *Plug Device* dialogue box should appear.
- (12). Under the *Miscellaneous* name, find the Base unit device you want and double-click. Close the box. The updated Base should be showing.

29.6.2 Review the Control Program

This

(13). Double-click on *PRG* to view the program file.

The top level program is done in the *Continuous Function Chart* language and supports hierarchical structure.

MQTT Broker

The MQTT broker is the backend system which coordinates messages between the different clients. Responsibilities of the broker include receiving and filtering messages, identifying clients subscribed to each message, and sending them the messages. It is also responsible for other tasks such as:

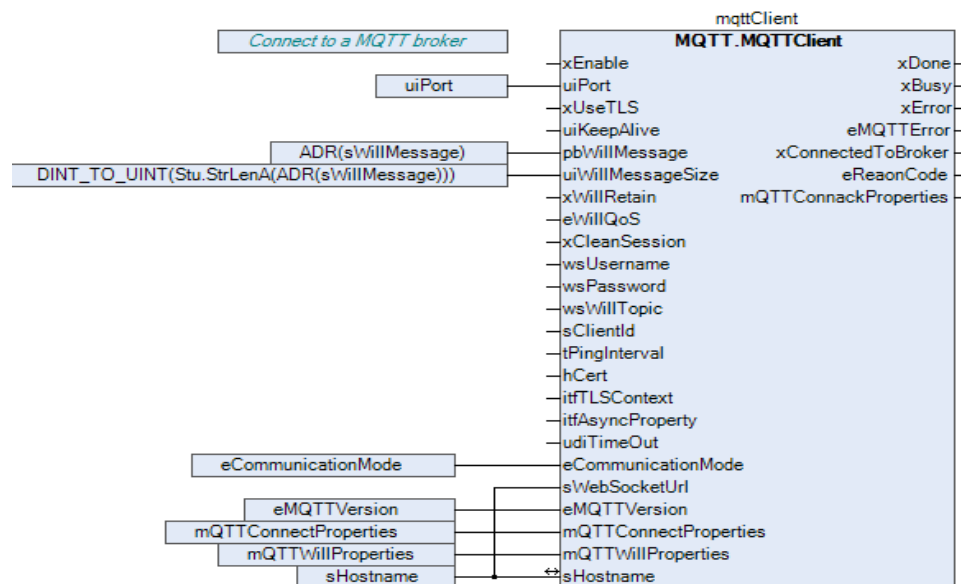
- Authorizing and authenticating MQTT clients
- Passing messages to other systems for further analysis
- Handling missed messages and client sessions

MQTT Client

An MQTT client is any device from a server to a microcontroller that runs an MQTT library. If the client is sending messages,

it acts as a publisher, and if it is receiving messages, it acts as a receiver. Basically, any device that communicates using MQTT over a network can be called an MQTT client device.

This Function Block `MQTTClient` will be used to establish an MQTT link with MQTT Broker and provide security aspects if needed.

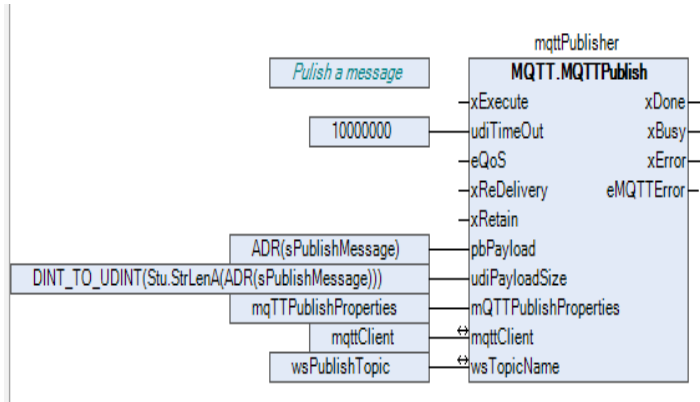


Details of this function can be found in the Help file located in the install directory `$USER$MQTT Client SL(version)` entitled `MQTT Client SL.chm`

MQTT Publish

MQTT clients publish messages that contain the topic and data in byte format. The client determines the data format such as text data, binary data, XML, or JSON files. For example, a lamp in the smart home system may publish a message on for the topic *livingroom/light*.

The Function Block `MQTTPublish` will be used to publish data

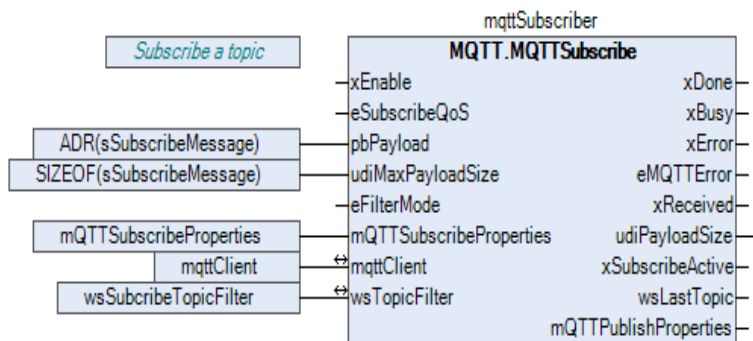


Details of this function can be found in the Help file located in the install directory `$USER$MQTT Client SL(version)` entitled `MQTT Client SL.chm`

MQTT Subscribe

MQTT clients send a SUBSCRIBE message to the MQTT broker, to receive messages on topics of interest. This message contains a unique identifier and a list of subscriptions. For example, the smart home app on your phone wants to display how many lights are on in your house. It will subscribe to the topic *light* and increase the counter for all on messages.

The Function Block `MQTTSubscribe` will be used to access or subscribe to data



Details of this function can be found in the Help file located in the install directory `$USER$MQTT Client SL(version)` entitled `MQTT Client SL.chm`

29.6.3 Utilize the Control Program

To use this project, data is entered into the Visualization fields.

Since all the items required for a secure MQTT communications are visible with this example program, it can be utilized as a starting template for your own design. You can then use a separate program file e.g. in Structured Text that will access/populate the variables with your own data in the order it is required.

29.6.4 A View of the “Task”

Another very important area in the Device Tree is the *Task Configuration* section which defines the way in which the program tasks are defined (which POU's are to run on a scan) and the characteristics of the task (priority, speed etc.).

The online help goes into detail for this area. The defaults are good for this project.

29.6.5 Compile the Project

Next we will compile and build the project to download to the P2CDS-622 target.

(14). Select from the drop-down menus- *Build > Generate Code*.

29.7 Connect to PLC and Download

The project needs to be downloaded to the target and but we must first connect to it.

(15). Physically connect the Host PC to one of the Ethernet ports on the P2CDS-622.

29.7.1 Configure Connection Channel to PLC

(16). In the Device Tree, Double-click on the **Device(P2CDS622)**.

A **Device** tab should appear with the first tab on the left selected entitled *Communication Settings* selected. Now we need to scan the network to find our device.

(17). In the *Communication Settings* section, click on the **Scan network**.

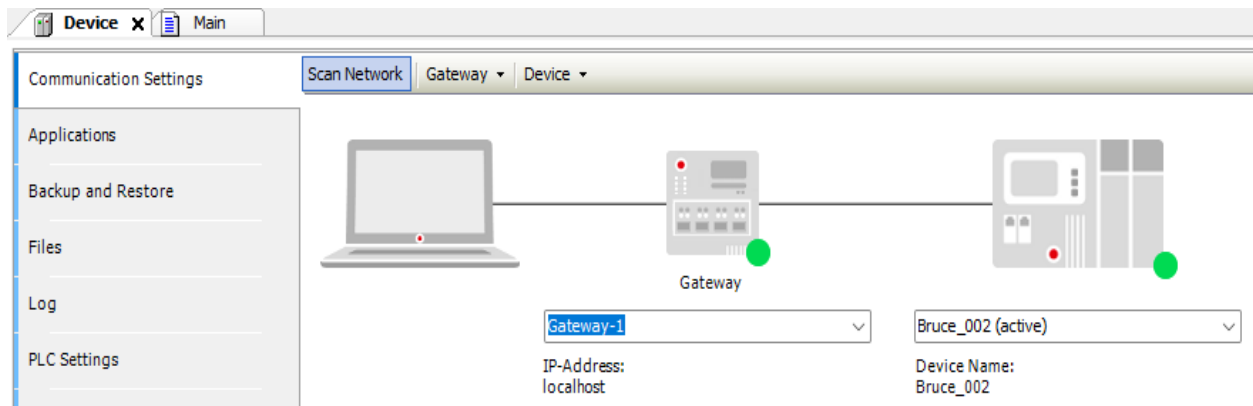
A dialog box should pop up with all the CODESYS PLCs available to connect to on this network.

Note: If you are unsure which unit is the one you want to connect to, on the right side is a **Wink** select box. Click on this and this P2CDS-622 front panel **RUN** light should start to flash for 5 seconds.

(18). Select the applicable P2CDS-622 unit and Click **OK**.

(26). Toggle the **Run/Stop** switch.

Now the following should be visible- both the Gateway and the P2CDS-622 have the green indicators present indicating the IDE is connected to the P2CDS-622.




29.7.2 Download Application to PLC

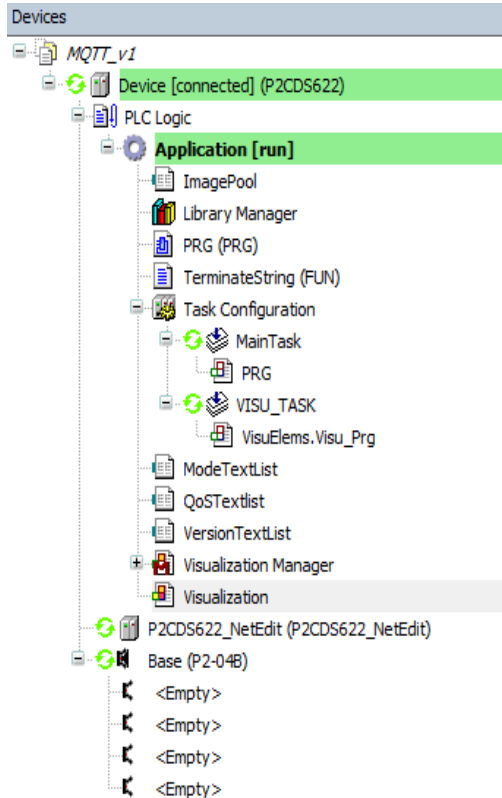
(19). Click **Online > Login**.

A dialog opens up if there is an existing application on the PLC, if so Click **Yes** to continue.

29.7.3 Successful Download. Ready to Run

The system should show green highlighted *Device* and *Application* as shown below:

The  icon indicates connection is active (but program is not executing).




29.8 Run and Watch the Application

This section will start the application running and using the debugger features, allow a user to watch the variables in real time, set breakpoints and force variables to a specific value.

29.8.1 Start the Application

(20). Click **Debug > Start**.

The IDE should now be showing a green “Run” in the lower portion of the status area. Also, the *Main Task* has the running icon  next to it.

Double-click on the **Visualization** file in the *Application* section.

In these fields, enter the following:

CONNECTION SETTINGS

- Communication Mode: **TCP/IP**
- Server/IP/URL: **broker.hivemq.com**
- Server Port: **1883**
- Use TLS: (unchecked)
- Username: (unused)
- Password: (unused)
- MQTT Version: **3.1.1**
- Clean Session: (**checked**)
- Clean Id: (unused)

LAST WILL SETTINGS

- Last Will QoS: **QoS 0**
- Last Will Topic: **/igotthis/fromFACTS**
- Last Will Message:(unused)
- Last Will Retain:(unchecked)
- Server Port: **1883**
- Connect to Broker: (select the **1** icon)

SUBSCRIBER 1

- QoS: **QoS 0**
- Topic Filter: **/FACTS/justtryitout**
- Last Topic:(from Publisher)
- Message:(from Publisher)
- Subscribe: (select the **1** icon)

PUBLISHER 1

- QoS: **QoS 0**

- Topic: **/FACTS/justtryitout**
- Message:(**message to send**)
- Retain: (unchecked)
- Redeliver: (unchecked)
- Publish: (click icon to send message)

29.8.2 Watch the Application

Click on the Visualization- Visualization in the device tree to view the options to enter data.

The Publish/Subscribe items can be tested after the applicable fields are setup and the Broker of choice enabled.

For our example using HiveMQ:



29.8.3 Closing Comments

It is strongly recommended to use MQTT with security. Security (TLS) can be easily incorporated in this design allowing for a Username and Password.

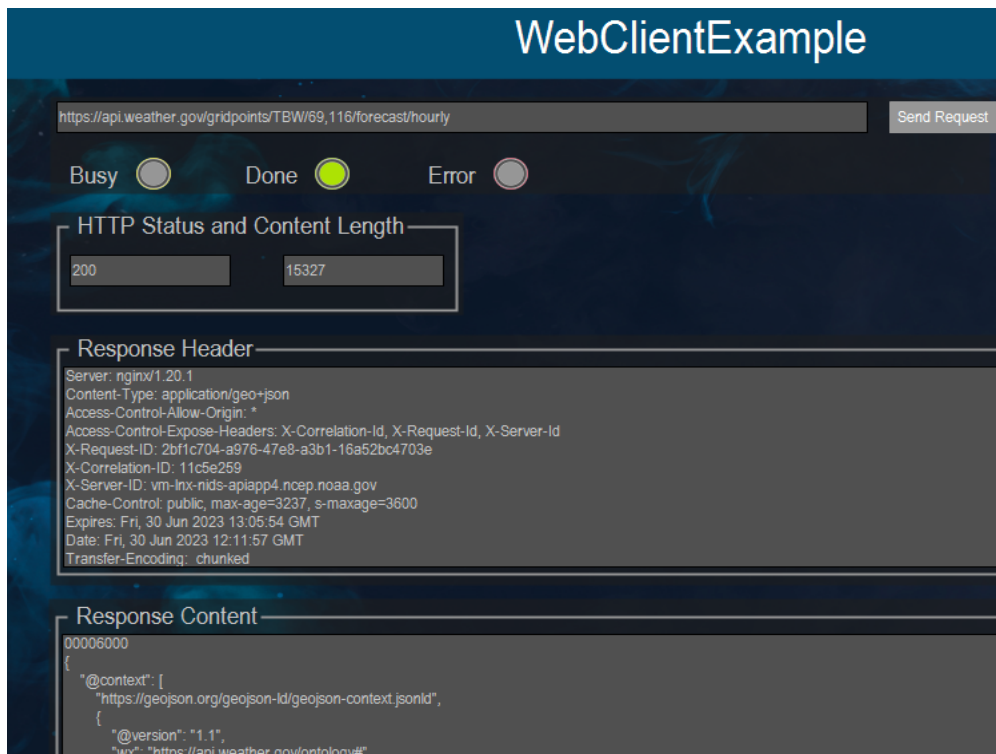
IIOT - WEB CLIENT SUPPORT

30.1 Introduction

The CODESYS Development System package includes a library with function blocks for communicating with a web server via HTTP or HTTPS.

- The HTTP method GET requests data from a web server.
- The HTTP methods POST and PUT submit data from the controller to a web server.
- When using DELETE, data is deleted from a web server.
- The HTTP method HEAD returns the HTTP header.

The HTTP header of a request can be extended without any restrictions. In addition, function blocks are included for HTTP basic authentication, digest access authentication and authentication via OAuth (Version 1a and 2).



Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

30.2 Install IIoT Library

To access the free IIoT Library functions e.g. Web Client functions, you must install the IIoT Single license with the CODESYS Installer.

This license is located on the CODESYS website store at [IIoT License](#)

Note: This Library does **NOT** require any payment to download! Just click on the **Download** button.

30.3 Project

The following project is a modified version taken from the IIoT Library example located in your install directory. This location for example would be at *C:\Users<username>\Web Client SL* along with an associated Help file.

The Web Client example project can be **downloaded in Archive format here**- [Web Client project](#).

(1). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. You may need to add a Library, e.g. *Basic OSCAT*.

The contents/raw data in the project, is located in the variable `httpResult` as shown below.

Variable	Type	Value
webClient	WEB_CLIENT.WebCl...	
xExecute	BOOL	FALSE
udTimeOut	UDINT	1000000
sURL	STRING(1024)	https://
eRequestType	REQUEST_TYPE	GET
eContentType	CONTENT_TYPE	APPLICA
pwsAdditionalHeader	POINTER TO WSTRING	16#01D
pwsPostValue	POINTER TO WSTRING	16#01D
xCloseConnection	BOOL	FALSE
hCert	POINTER TO BYTE	16#FFFF
hTlsContext	NBS.ITLSContext	16#000
hTlsAsyncProperty	NBS.IAsyncProperty	16#000
xDone	BOOL	FALSE
xBusy	BOOL	FALSE
xError	BOOL	FALSE
eError	ERROR	NO_ERROR
httpResult	HttpResult	
iStatus	INT	0
wsHeader	WSTRING(g_udMax...	Server:
wsContent	WSTRING(g_udMax...	1000060
sHeader	STRING(g_udMaxH...	Server:
sContent	STRING(g_udMaxR...	1000060
diContentLength	DINT	12490

```

"name": "",
"startTime": "2023-06-30T13:00:00-04:00",
"endTime": "2023-06-30T14:00:00-04:00",
"isDaytime": true,
"temperature": 93,
"temperatureUnit": "F",
"temperatureTrend": null,
"probabilityOfPrecipitation": {
  "unitCode": "wmoUnit:percent",
  "value": 0
},
"dewpoint": {
  "unitCode": "wmoUnit:degC",
  "value": 21.111111111111111
},
"relativeHumidity": {
  "unitCode": "wmoUnit:percent",
  "value": 48
},
"windSpeed": "8 mph",
"windDirection": "NE",
"icon": "https://api.weather.gov/icons/land/day/few,0?size=small",
"shortForecast": "Sunny",
"detailedForecast": ""
}
{
  "number": 7,
  "name": "",
  "startTime": "2023-06-30T14:00:00-04:00",
  "endTime": "2023-06-30T15:00:00-04:00",

```

30.3.1 Items of Note

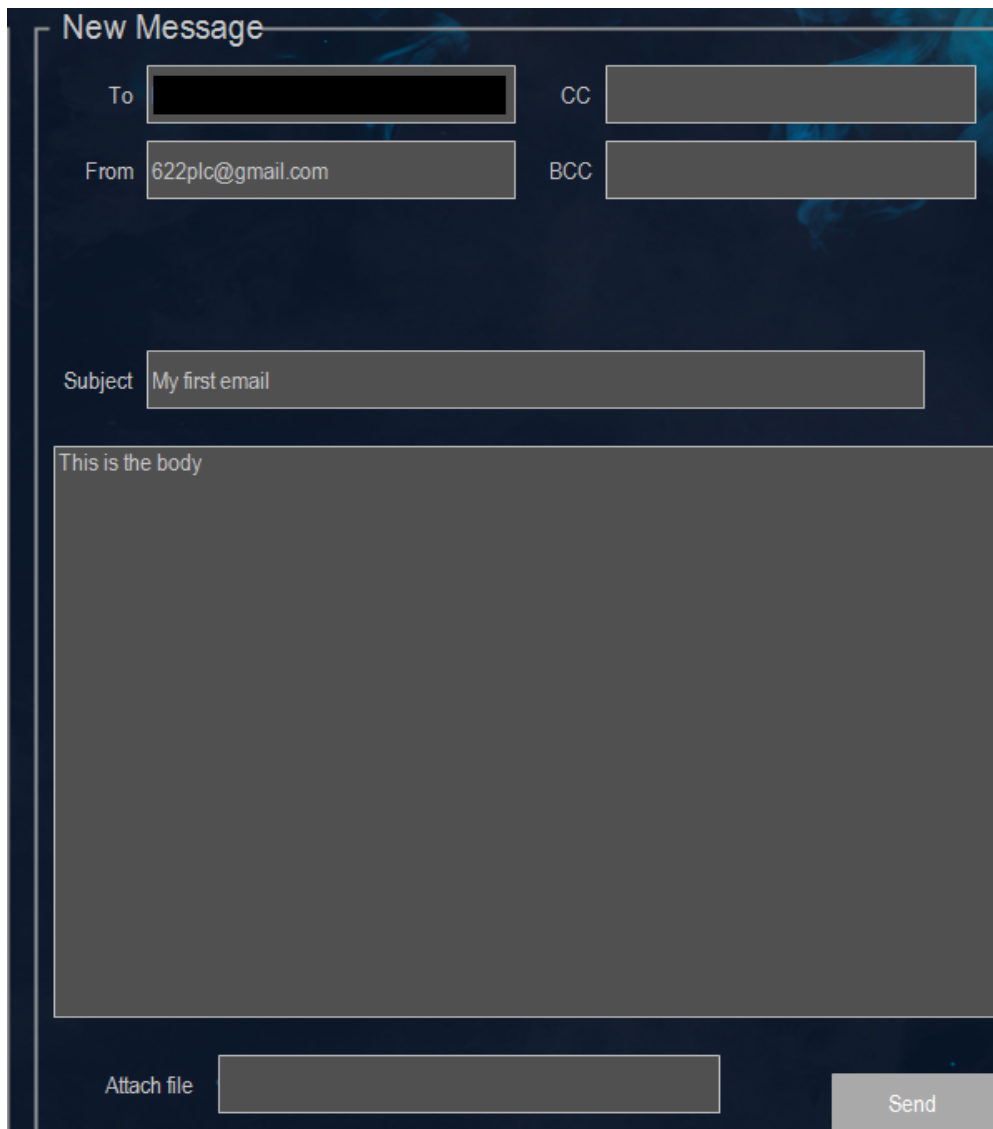
- This project uses the `WebClientExample` within Application.
- This project uses the link <https://api.weather.gov/gridpoints/TBW/69,116/forecast/hourly> to demonstrate the HTTP GET functionality.
- The above screenshot shows a successful (no errors) time access via the green icon **Done**.

IIOT - MAIL SUPPORT

31.1 Introduction

This library contains function blocks for sending, receiving, and deleting emails by means of SMTP and POP3 protocols.

Communication with the mail server can be established either encrypted (TLS) or unencrypted. A sample project demonstrates how to use the function blocks.



Note: The P2CDS-622 will only support **Outbound** emails and does NOT accept inbound for security reasons. The example project show both.

31.2 Install IIoT Library

To access the free IIoT Library functions e.g. Web Client functions, you must install the IIoT Single license with the CODESYS Installer.

This license is located on the CODESYS website store at [IIoT License](#)

Note: This Library does **NOT require any payment** to download! Just click on the **Download** button.

31.3 Project

Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

The following project is a modified version taken from the IIoT Library example located in your install directory. This location for example would be at *C:\Users<username>\Mail Service SL* along with an associated Help file.

The Mail project can be **downloaded in *Archive* format here-** Mail project.

(1). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

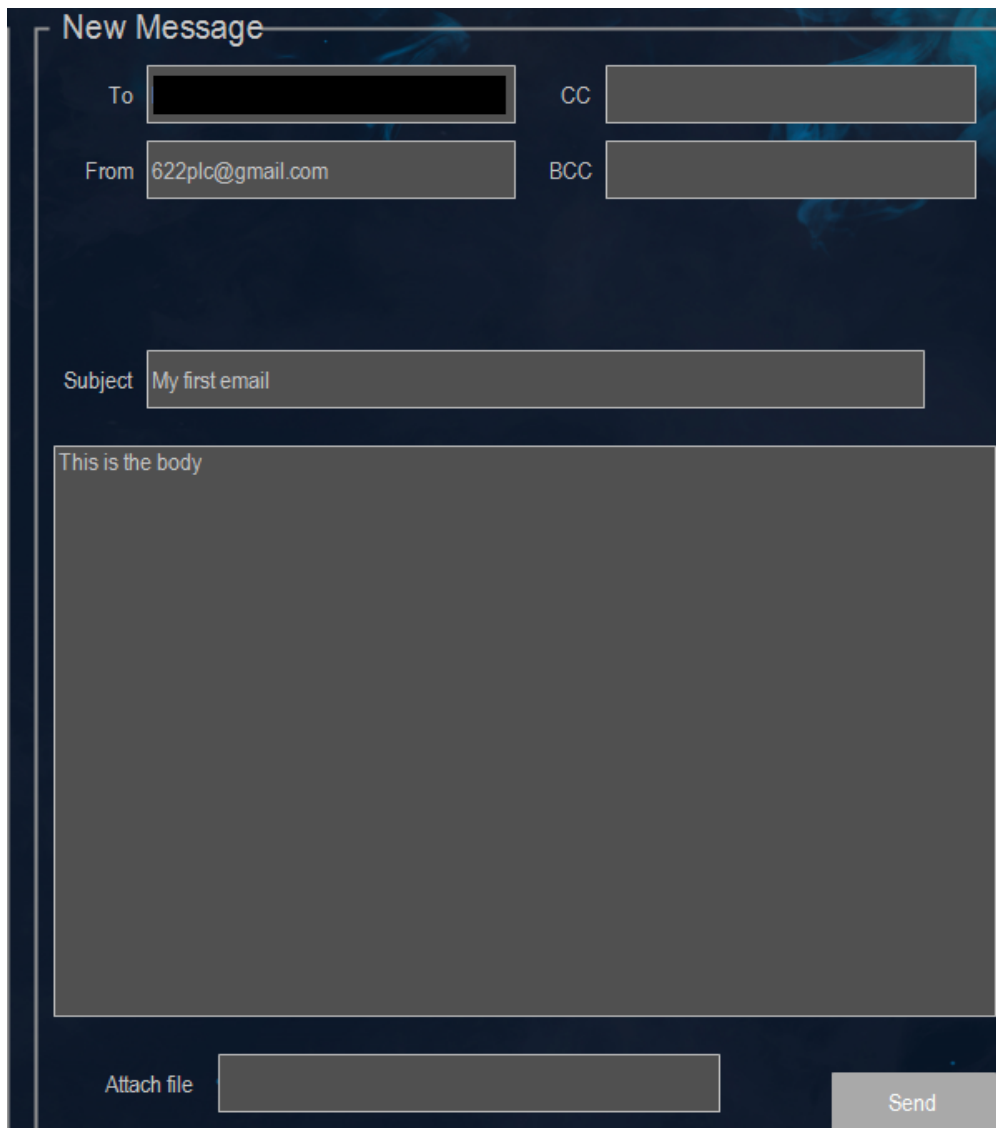
Update **everything (including Library’s)** to latest revisions. You may need to add a Library, e.g. *Basic OSCAT*.

The areas for the user to setup are shown below which include:

- SMTP UserName
- SMTP Server address
- SMTP Password

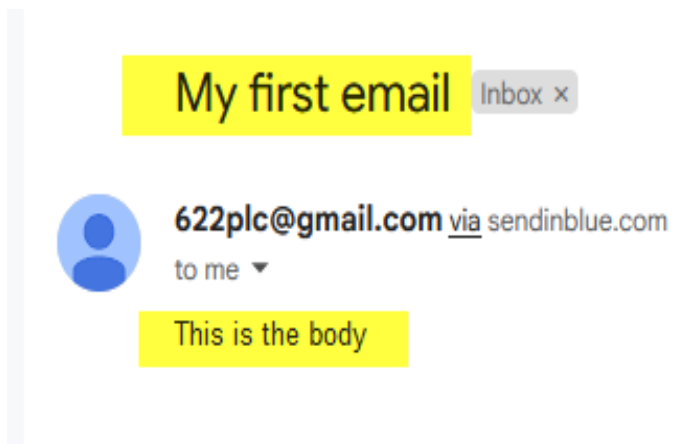
The screenshot shows the 'Settings' dialog box. It contains two main sections: 'POP3 Login' and 'Server Config'. In the 'POP3 Login' section, the 'User Name' field is filled with '622plc@gmail.com' and the 'Password' field is masked with asterisks. In the 'Server Config' section, the 'Smtp Server' field is filled with 'smtp-relay.sendinblue.co' and the 'Port' field is filled with '587'. The 'Pop3 Server' field is filled with 'pop.gmail.com' and the 'Port' field is filled with '995'. There are two checkboxes: 'TLS (SMTP, POP3)' which is unchecked, and 'STARTTLS (SMTP)' which is checked. At the bottom of the 'Server Config' section is a 'Save Files to' field. At the very bottom of the dialog are 'Abort' and 'Save' buttons.

Mail send example:



The screenshot shows a 'New Message' window with a dark blue background. It contains several input fields: 'To' (empty), 'CC' (empty), 'From' (622plc@gmail.com), 'BCC' (empty), and 'Subject' (My first email). Below these is a large text area for the body, which contains the text 'This is the body'. At the bottom, there is an 'Attach file' button and a 'Send' button.

Received email example Gmail account (622plc@gmail.com)



31.3.1 Items of Note

- An email account for the P2CDS-622 named *622plc@gmail.com* was created along with the associated password in sPassword to be used in the project. You will need to create your own email account.
- This project uses Brevo for the SMTP server support- <https://app.brevo.com/>.
- SMTP Port address is 587.
- Error codes can be found in the Library documentation for this function.

IIOT - SNTP SUPPORT

SNTP or Simple Network Time Protocol (SNTP) is an Internet Protocol (IP) used to synchronize the clocks of networks of computers.

This project will go out to a dedicated website that provides the current time in *Current UTC* time (previously Greenwich Mean Time) and use that to synchronize the P2CDS-622 clock.

For example, if the local time is EST at 7:30am, UTC is 4 hour faster at 11:30am.

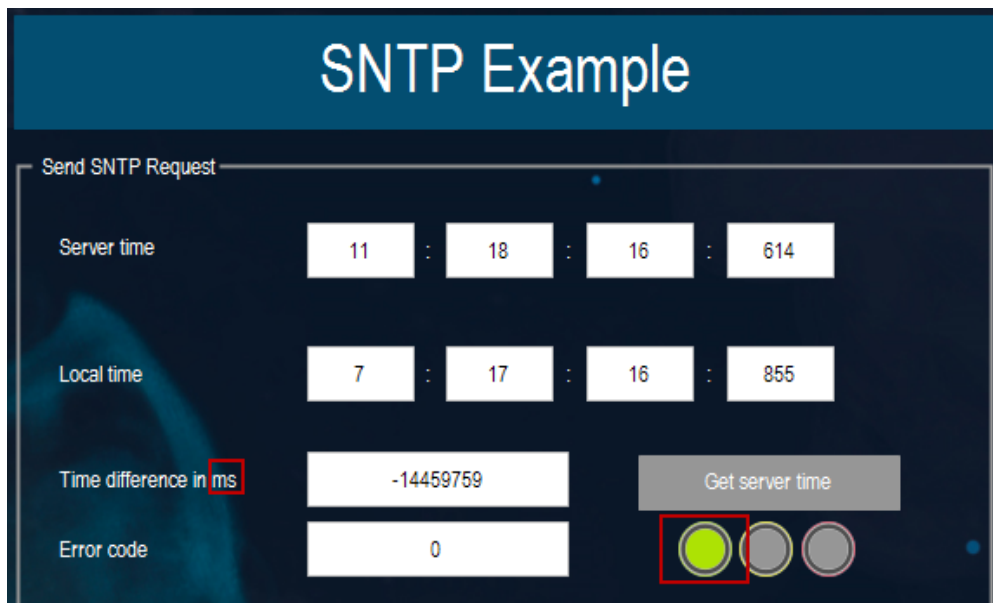
32.1 Introduction

The library *SNTP Service SL* contains function blocks for easy implementation of SNTP client and server components on a CODESYS control (SNTP V3, SNTP V4). The function block `SNTPGetUTCTime` can be used to request the UTC time of a SNTP/NTP server.

The function block `SNTPServer` can be used to build a simple SNTP server.

Function blocks (FBs) of the SNTP library:

- `SNTPGetUTCTime`: FB to request the time of a SNTP server (SNTP client)
- `SNTPServer`: FB to send the local server time (SNTP server)



Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

32.2 Install IIoT Library

To access the free IIoT Library functions e.g. Web Client functions, you must install the IIoT Single license with the CODESYS Installer.

This license is located on the CODESYS website store at [IIoT License](#)

Note: This Library does **NOT** require any payment to download! Just click on the **Download** button.

32.3 Project

The following project is a modified version of the IIoT Library example located in your install directory. This example should be located at *C:\Users<username>\SNTP Service SL* along with an associated Help file.

The SNTP example project can be **downloaded in Archive format here-** SNTP project.

(1). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. May have to add a Library, e.g. *Basic OS-CAT*.

Caution: It is provided that the provided example URL for accessing hourly time may not always be accessible. If not, pick a URL of your choosing.

32.3.1 Items of Note

- This project uses a NIST server to obtain the time at NIST Maryland. Defined at <https://tf.nist.gov/tf-cgi/servers.cgi>.
- Server time is in UTC time,
- Error codes obtained by double-clicking on the Function Block `SNTPGetUTCTime` and inspecting **SNTP Service > Enums > ERROR**.
- The above screenshot shows the time difference in milliseconds (ms). We would expect a 4-hour difference from local time. This example, 4.02 hours error.
- The above screenshot shows a successful (no errors) time access via the green icon under **Get server time**.

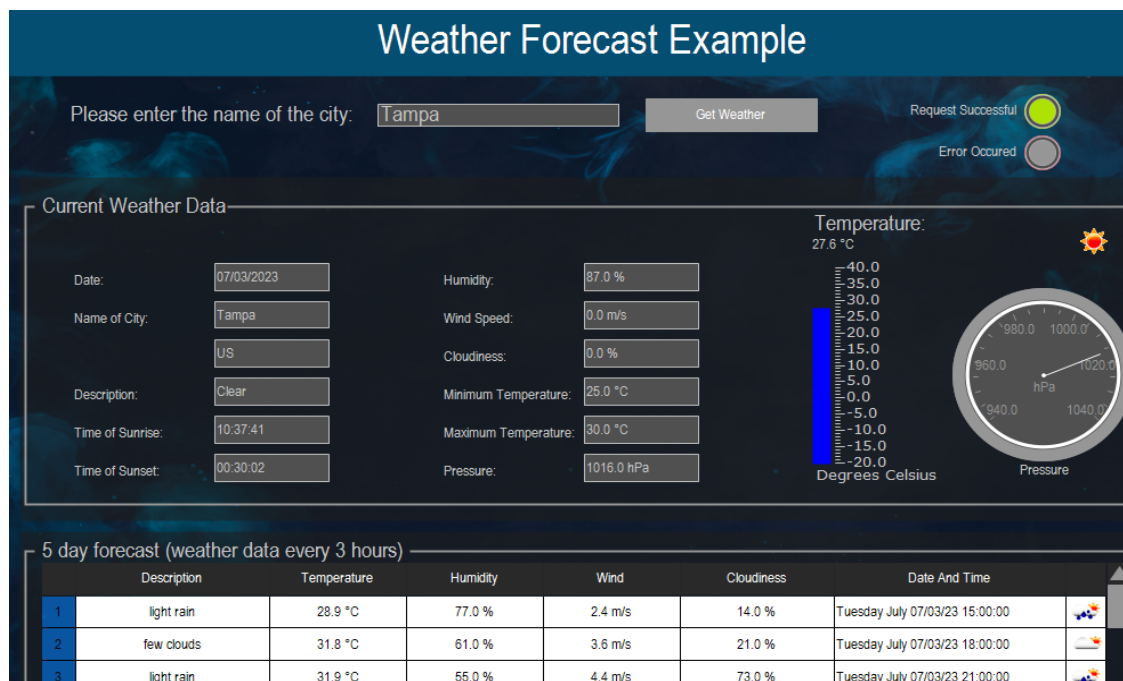
WEATHER FORECAST LIBRARY

33.1 Introduction

With the Weather Forecast Library, current weather data and forecasts can be queried for any city by means of the “OpenWeatherMap” online service.

OpenWeatherMap is a service that provides an open API for weather data and forecasts.

Using “OpenWeatherMap” requires an API key (APPID) that you must be setup.



Caution: To do this project, the steps outlined in the previous section *Preparation* **have to be completed** successfully.

33.2 Install API Library

To access the free Weather Forecast Library functions, you must install the Weather Forecast Library package and the IIoT Library with the CODESYS Installer.

The packages and associated datasheets are located on the CODESYS website store at

Click here- [Weather App License](#)

Click here- [IIoT Library License](#)

33.3 Project

The CODESYS example Weather project will be located at your install location, for example **C:\Users<User name>\CODESYS Examples\Weather Forecast Library** .

The following project is a modified version of the example project and **can be downloaded here** [API Weather project](#).

(1). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

Update **everything (including Library’s)** to latest revisions. You may need to add a Library, e.g. *Basic OSCAT*.

DATA LOGGER/CSV WRITE

This example project incorporates the IIoT .CSV Library functions to generate a data logger that takes in various Tag data types i.e. *REAL* , *WORD*, etc. stores them in a format and writes these Tag values out to a .CSV file on command or automatically.

Due to the complexity of the project, an example can be downloaded from the website at [datalogger project](#) .

The various parts of the project are described below and this can be used as a starting point to create your own solution.

The data logging project utilizes the SD Card for the storage location.

34.1 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 PowerSupply (any variant)
- Latest version of CODESYS installed on Host PC
- OSCAT “Basic” library installed.
- “Util” library installed.
- “SysTime” installed.
- Familiarity with the “Continuous Function Chart” programming language.

<p>Caution: To do this project, the steps outlined in the previous section <i>Preparation</i> have to be completed successfully.</p>
--

34.2 Install IIoT Library

To access the free IIoT Library functions e.g. CSV functions, you must install the IIoT Single license with the CODESYS Installer.

This license is located on the CODESYS website store at [IIoT License](#)

34.3 Project Generation

34.3.1 Open Archived Project

(1). Download the archived CODESYS project from below link to your local working directory.

Download here- Datalogger Project

(2). With CODESYS IDE launched, click **File > Project Archive > Extract Archive**. Select all options in pop-up. Click “yes” on next pop-ups.

(3). Update **everything (including Library’s)** to latest revisions. You may have to add a Library, e.g. *Basic OSCAT*.

Caution: When the SD Card is utilized continually as in a Data Logger application, it is highly recommended that the SD Card updates/writes are assigned its **own Task**. This is due to the possibility that, depending on size, the SD Card can take up to 100ms to update.

34.3.2 Inspect and Setup Project

This project has three(3) separate applications that can be run. Select the **CSVWriterCFCExample** as the active application if not active already.

(4). Right Click on **CSVWriterCFCExample > Set Active Application**.

Looking at the Top Level of the design (**WriteValues**), you can see the example was created using the CFC programming language.

The project has a Visualization that can be used to generate new writes into the .CSV file using Visu buttons.

The default .CSV file generated first is called **CSVWriterCFCExampleData.csv** after this a new file can be generated that is discussed below.

34.3.3 Elements of the Design

The following describe each block of the design. For further documentation, click on the Block of interest.

Various data types are stored by command i.e. **xExecute** driven high, into a particular function block. These blocks are buffers for the logging process and are stored internally in the library element, then written out to a file by command **CSV.WriteFile** and the **xExecute** strobe.

The design will write all the Tags or Data one defines on a single row of the CSV file until a **CSV.NewLine** command is initiated. Then the next logged values will be written on the next row.

Frequency of logging data is determined when the **xExecute** is strobed for a particular function block. In this project the first block’s logging is started with the press of the Visu tag button into the *xExecute* input, then that blocks **xDone** is daisy-chained into the next log block’s **xExecute** so it they are all done in sequence.

NOTE: there are numerous Tags (up to 64) that can be logged by inserting as many logging function blocks as required.

CSV Library File Related

To see all the logging functions available, go to the Library Manager and select **CSV_Utility_SL > CSVWriter**.

- **Initialize CSV function** (CSV.init) - This Function Block sets up the use of the CSV library. In this example we are using the CSV Write functionality CSVWriter. In addition, the file path and filename is defined here.
- **Add a DATE/TIME Type value** (CSV.AddDATE_AND_TIME) - The first data log entry on the row will be the Date. This data includes the Year/Month/Day/Hour/Second type of format.
- **Add a REAL Data Type value** (CSV.AddREAL) - The second value or Tag stored on this first line is a REAL data type.
- **Add a WORD Data Type value** (CSV.AddWORD) - The third value is a WORD type that is the output of a counter CTU, that is counting the number of presses of the logger in the Visu (actually corresponds to the row of the file also).
- **Add a STRING Data Type value** (CSV.AddSTRING) - The next item is a text String which can be used to label a Tag name, etc.
- **Add a new line/row for next Data log** (CSV.NewLine) - The function ends the current row of logging in the CSV file and adds a new row/line.
- **Write the logged values into CSV file** (CSV.WriteFile) - After the user is completed logging all the data for a specific row, executing the function writes this row/line into the CSV file. The CSV file is to be written to the SD Card by defining the location as “/sdc”.
- **Create another (New) .CSV file** (CSV.NewFile) - After the user is completed logging all the data for a specific file, executing the function creates a new CSV filename to start logging into. The previous file is then saved for that session.

Caution: You want to make sure you are logging data to the SD Card (/sdc for the file directory path). The local PLC Flash memory is not the correct location to store these files.

Support Functionality (non-CSV Library)

- **Get the current Time** (GetPlcTime) - This function incorporates the SysTimeRtcGet library call and generates the current local PLC time.
- **Obtain the Timestamp (Minutes)** (OSCAT_Basic_DT_STRF) - This OSCAT library function extracts the corresponding time element out of the at/time string depending on the value into the FMT input. In this case, I am extracting the *minutes* value to be used in the name or the CSV file. This gives it a timestamp in the name.
- **Concatenate Strings for a filename** (CONCAT) - To create the final text string for the filename, I have to concatenate the various text pieces together to get final name *LogSnapshot_minutes.csv*. Then when the CSV.NewFile is executed, this is the filename used.

34.4 Closing Comments

This project shows how to utilize the aspects of the IIoT CSV library functionality in the *CSV WRITE* mode and also implementing some of the functions associated with the OSCAT library.

A basic datasheet for the CVS complete functionality of the CVS solutions can be found in the IIoT install directory.

In addition, the Library Manager under the *CVS Utility* section lists the Function Blocks available.

OBJECT ORIENTED INDUSTRIAL PROGRAMMING

This approach is the next generation in Industrial Automation programming design methodology that is enabled fully with the CODESYS IDE.

If you have not done so already, it is **highly** recommended that you review this material and begin to apply the concepts even if you do this on a very basic level.

Analog Input Object used in Plant Design

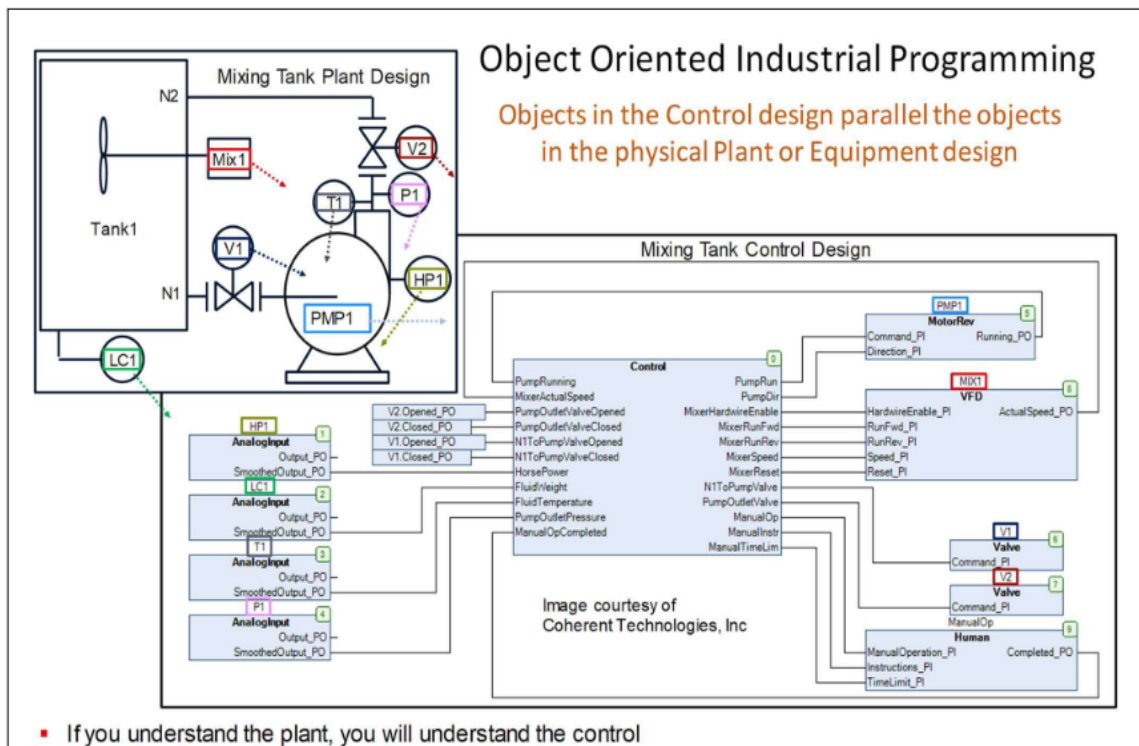


Image taken from “The Book of CODESYS” by Pratt

Here are some details of OOIP taken from the text “The Book of CODESYS” by Pratt- [OOIP Overview](#)

The following videos are an excellent resource on this topic:

- [OOIP Video 1](#)
- [OOIP Video 2](#)

FIRMWARE UPDATE/FACTORY RESET

There is a Firmware Update Utility tool that can be used to update your CPU Firmware and also can be used to reset the unit back to original factory settings.

36.1 Firmware Update

This section details how to upgrade the current CPU Firmware.

The current published Firmware revision can be found at *Firmware Revision*.

To read the installed Firmware on your CPU, see *Firmware Revision read*.

36.1.1 System Requirements and Installation

This section requires the following:

- P2CDS-622 CPU
- P2000 Base (any variant)
- P2000 Power Supply (any variant)
- Latest version of CODESYS installed on Host PC
- USB Type C cable or Ethernet cable

Installation

Caution: The Firmware Update Utility file is a zipped up **.EXE** type and may be flagged by antivirus software. A future signed version is currently being developed to avoid being flagged by antivirus software. If this is an issue, please contact tech support.

Download the Firmware Update tool from FW Update Tool

Unzip the file and it should have the .exe file. **Ignore the .cdfw file for now.**

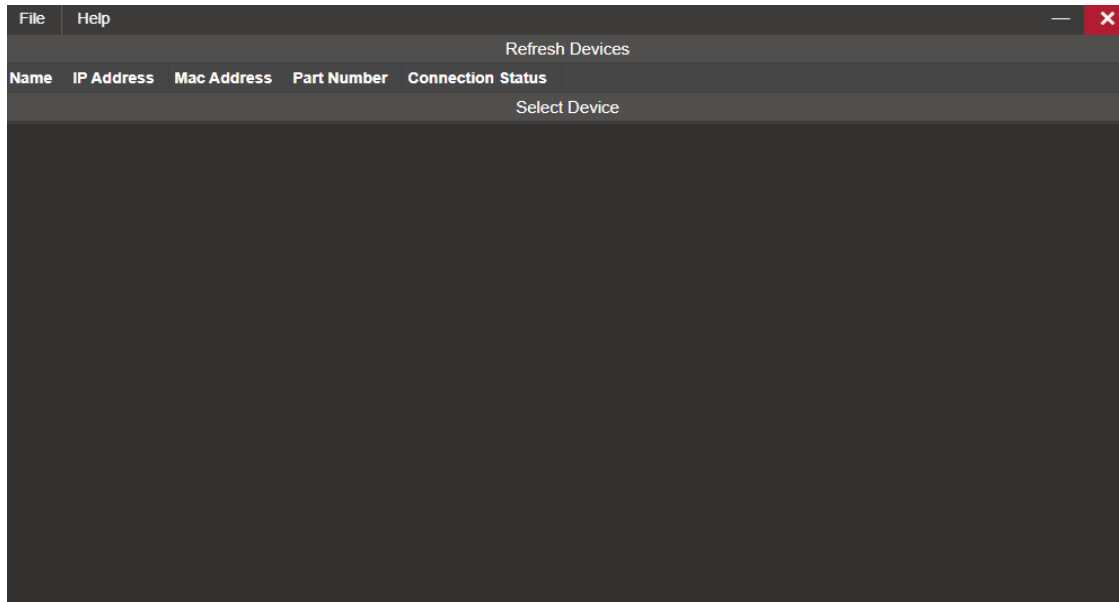
PLC On the Network

(1). With the CPU powered off, plug in the Host PC to the CPU's **USB-C port** (if this is the first time updating the Firmware) or into the Ethernet port **ETH1** (if Ethernet ports previously configured). Power-up the CPU.

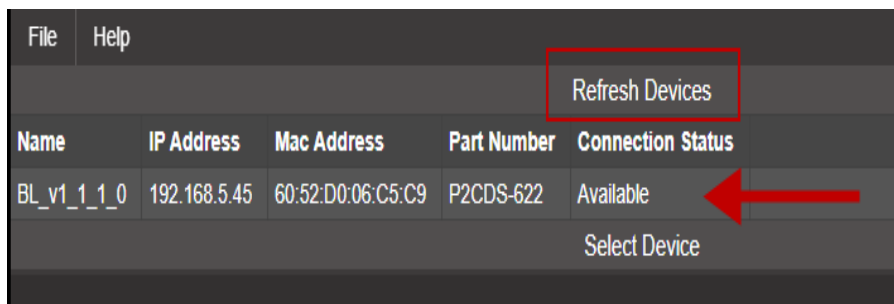
Update Procedure

(2). Inside the downloaded .zip file, double-click on the file **P2CDS-622 Firmware Updater-x.x.x Setup.exe**.

A screen should appear similar to the one below.



(3). Click on **Refresh Devices** to discover your P2CDS-622 CPU to update. You should see something like the following:



If it CPU is not listed, recheck the connection to USB-C (or to the Ethernet port).

(4). Double-click on the CPU desired, and the following should appear.

The screenshot shows the configuration interface for the P2CDS-622 system. It is divided into two main panels. The left panel, titled 'CPU Settings', has two tabs: 'Port 1' (selected) and 'Port 2'. Under 'Port 1', there are checkboxes for 'Obtain from DHCP Server' (unchecked), 'IP Address' (192.168.0.1), 'Subnet Mask' (255.255.248.0), and 'Default Gateway' (192.168.0.1). Below these are 'DNS Settings' with 'Obtain from DHCP Server' (unchecked), 'Preferred DNS Address' (96.187.214.4), and 'Alternate DNS Address' (0.0.0.0). At the bottom of the left panel is the 'CPU Name' section with a text box containing 'BL_v1_1_1_0' and an 'Update Device with Settings' button. The right panel has two sections: 'Update Firmware' with a 'Firmware File' text box, a 'Browse' button, and an 'Update Firmware' button; and 'Factory Reset' with a 'Factory Reset' button.

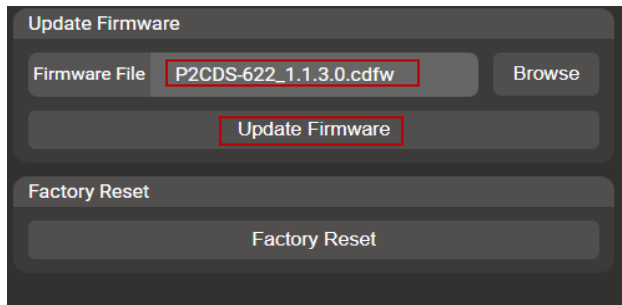
If this is a **first time USB-C type setup**, ignore the *Port* items.

Note: When updating reference in an Ethernet connection, the *Port 1* and *Port 2* tabs correspond to the P2CDS-622 ETH1 and ETH2 connectors discussed in the [Ethernet](#) section.

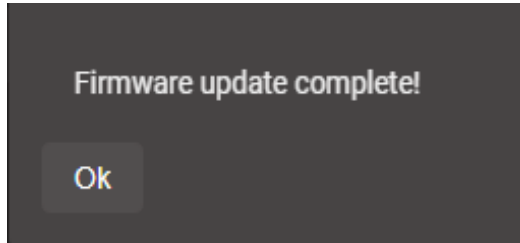
(5). Rename the CPU as desired. In the box for **CPU Name**, type in the name of your device, then click on **Update Device with Settings**.

This screenshot shows the same configuration interface as the previous one, but with the 'CPU Name' text box highlighted by a red rectangle. The text in the box has been changed to 'BL_1.1.3.0'. A red arrow points from the bottom right towards the 'Update Device with Settings' button. The 'IP Address' field now shows '192.168.5.45'. The 'DNS Address' field shows '96.155.213.4'. The 'Selected Device' at the top right is 'BL_1.1.3.0'.

(6). Browse to the update file (P2CDS-622_x.x.x.x.cdfw) and click on **Update Firmware**



The pop-up below should appear once the firmware update process is complete:



36.2 Factory Reset

A Factory Reset restores the unit to the Factory defaults. It removes all passwords, projects and the *Retain Memory* variables.

Note: To perform a Factory Reset, the user must be physically **present** with the CPU to toggle the P2CDS-622 *Run/Stop* switch.

36.2.1 Operation

- (1). In the Firmware Update Utility tool, select the **Factory Reset** button and toggle the Run/Stop switch.
- (2). Power cycle the unit. The system should be erased and ready for a Firmware Update.

<p>Caution: ALL DATA on the PLC will be erased with this operation.</p>
--

POWER SUPPLY H/W REVISION NUMBER

The Power Supply hardware revision number can be used for assistance with field failures or replacements.

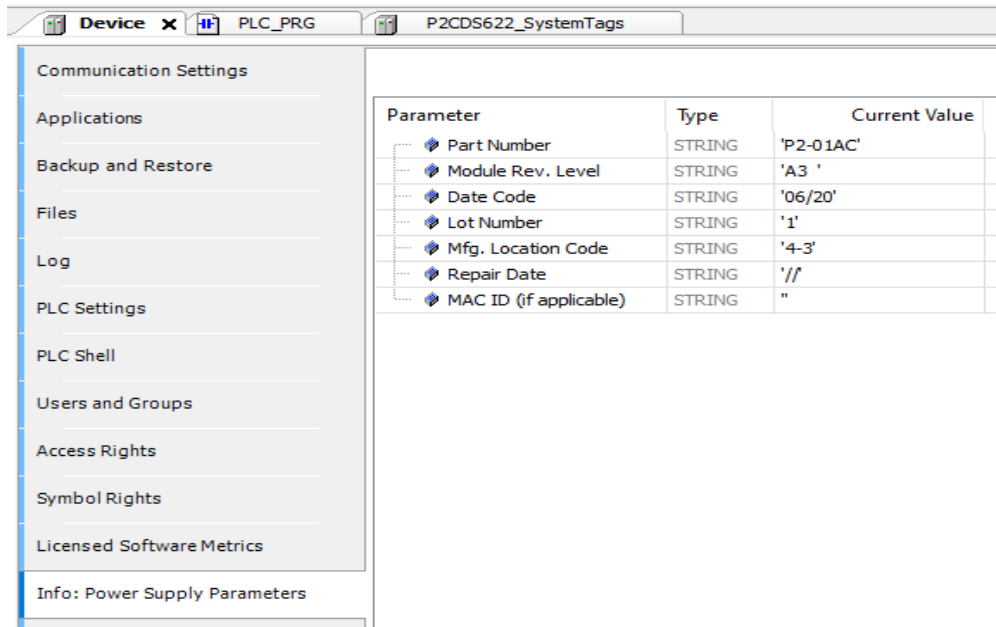
37.1 Nomenclature

The following are the items defined:

- Part Number - ordering part number.
- Module Rev. Level - revision hardware build.
- Date Code - when the unit was produced.
- Lot Number - group associated with the build.
- Mfg Location Code - what production facility was used.
- Repair Date - date when any repairs made.
- MAX ID - (not applicable)

37.2 Accessing the Information

After connecting to the P2CDS-622, these values are made available in the *Device* area- **Device > Info: Power Supply Parameters**.



The screenshot displays the P2CDS-622 System interface. The top navigation bar includes tabs for 'Device', 'PLC_PRG', and 'P2CDS622_SystemTags'. The left sidebar lists various settings categories, with 'Info: Power Supply Parameters' selected at the bottom. The main content area shows a table of parameters.

Parameter	Type	Current Value
Part Number	STRING	'P2-01AC'
Module Rev. Level	STRING	'A3 '
Date Code	STRING	'06/20'
Lot Number	STRING	'1'
Mfg. Location Code	STRING	'4-3'
Repair Date	STRING	'//'
MAC ID (if applicable)	STRING	' '

P2-BASE H/W REVISION NUMBER

The Base hardware revision can be used for assistance with debug field failures or replacements.

38.1 Nomenclature

The following are the items defined:

- Part Number - ordering part number.
- Module Rev. Level - revision hardware build.
- Date Code - when the unit was produced.
- Lot Number - group associated with the build.
- Mfg Location Code - what production facility was used.
- Repair Date - date when any repairs made.
- MAX ID - (not applicable)

38.2 Accessing the Information

After connecting to the P2CDS-622, these values are made available in the *Device* area- **Device > Info: Base Parameters**.

The screenshot shows the P2CDS-622 System interface. The top bar contains three tabs: 'Device' (selected), 'PLC_PRG', and 'P2CDS622_SystemTags'. On the left, a vertical menu lists various settings categories. The 'Info: Base Parameters' option is selected at the bottom of this menu. The main area on the right displays a table of parameters.

Parameter	Type	Current Value
Part Number	STRING	'P2-04B'
Module Rev. Level	STRING	'XA '
Date Code	STRING	'01/14'
Lot Number	STRING	'1'
Mfg. Location Code	STRING	'4-1'
Repair Date	STRING	'//'
MAC ID (if applicable)	STRING	' '

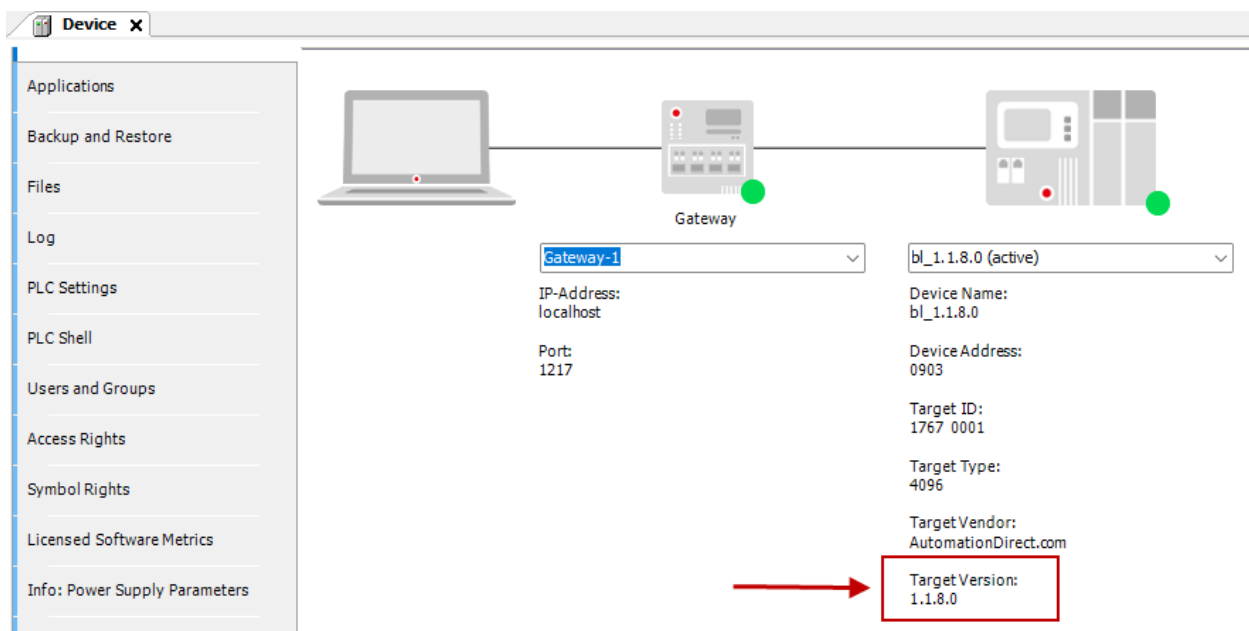
FIRMWARE REVISION

The version of firmware that is loaded into the P2CDS-622 can be viewed in the CODESYS IDE device connection area or read by the project.

Use the variable `sFirmwareVersion` to read the firmware revision of the CPU in the project. See below for more details.

39.1 Device Connection Summary

In a project, when connected to a device (see *Verify Device*) the Firmware Revision for the CPU will be listed in the image below. This example shows version 1.1.8.



39.2 Read Variable Access

To access the `sFirmwareVersion` variable, install the device `P2CDS622_SystemTags`, by right-clicking on **Device** and **Add Device > Miscellaneous**.

Then add the device.

After connecting to the P2CDS-622, these values will be available in the *Device* area- **Device > Info: Base Parameters**.

The variable in the IO Mapping space is of an *Array of Bytes*. This **ARRAY** can be converted to a **STRING** by creating a variable in your project of type **STRING** and assigning this variable in the IO Mapping. This assignment will automatically convert the array to a string.

See the images below for an example of how to do this. The string value in the example below is the Firmware revision “1.1.8.0”.

The top screenshot shows a ladder logic program for PLC_ORG. The variable declaration is as follows:

```

1 PROGRAM PLC_ORG
2 VAR
3   sFwVersion: STRING (16);
4   sMyFwVersion: STRING (16);
5
6 END_VAR
7
8

```

The bottom screenshot shows the 'Internal I/O Mapping' table. The mapping for `sFwVersion` is highlighted with a red box.

Variable	Mapping	Channel	Address	Type
		byBatteryIsVoltageTooLow	%IB0	BYTE
		uiBatteryCurrentVoltage	%IW1	UINT
		byErrorIsCritical	%IB4	BYTE
		sErrorMessageCritical	%IB5	ARRAY [1..60] OF BYTE
		byErrorIsNonCritical	%IB65	BYTE
		sErrorMessageNonCritical	%IB66	ARRAY [1..60] OF BYTE
sFwVersion		sFirmwareVersion	%IB126	ARRAY [1..16] OF BYTE

SD CARD USAGE

The removable Flash Memory device (MicroSD Card) can be used to store data in a data logging application such as the *Data Logger* example- [Data Logger](#)

The MicroSD (uSD) Card is accessed by setting the directory path defined as `/sdc` (see above project).

There is also an “unmount” button that when pressed ensures that the uSD Card is not being accessed and is safe to remove. When pressed, the uSD LED will flash momentarily during the unmounting process. When the uSD LED turns off, this indicates it is safe to remove the uSD Card.

40.1 Memory Card Specifications

The flash memory characteristics of the SD Card:

- Form Factor: MicroSD (uSD)
- Capacity 32GB
- Transfer Rate - 14Mbps Read / 5Mbps Write
- Endurance > 100K Write cycles

Caution: When the uSD Card is utilized continually as in a Data Logging application, it is highly recommended that the uSD Card update/write process is assigned its **own dedicated Task**, due to the possibility that the uSD Card can take up to 100ms to update.

BATTERY AND VOLTAGE

The Coin Cell battery (CR2354) holds up the contents of the Battery-backed SRAM and the Real-time clock for at least 5 years. It is discussed in section *CPU-Battery*.

The voltage level of this battery can be read by the project.

In addition, a variable that is a flag will trip when the Battery is low, which occurs at 2.9 volts. Even in a low voltage condition, the battery will continue to support the Battery Backed SRAM for a few days after this flag goes active.

The variables are- `byBatteryIsVoltageTooLow`, `uiBatteryCurrentVoltage`.

41.1 Variable Access

To access these variables, install the device *P2CDS622_SystemTags*, by right-clicking on **Device** and **Add Device > Miscellaneous**.

Add this device and then connect to the P2CDS-622.

To access these variables you can assign them as Global Vars by naming the variable in the IO Mapping sections as shown below. Then access the variables as you see fit.

Find	Filter	Show all			
Variable	Mapping	Channel	Address	Type	
BattVoltLowFlag		byBatteryIsVoltageTooLow	%IB0	BYTE	
BattVoltage		uiBatteryCurrentVoltage	%IW1	UINT	
		byErrorIsCritical	%IB4	BYTE	

Alternatively, you can create a variable in your program/Application and assign that variable in the IO Mapping sections as shown below.

Find	Filter	Show all			
Variable	Mapping	Channel	Address	Type	
		byBatteryIsVoltageTooLow	%IB0	BYTE	
Application.PLC_ORG.myBattVolt		uiBatteryCurrentVoltage	%IW1	UINT	
		byErrorIsCritical	%IB4	BYTE	

REAL TIME CLOCK (RTC)

The P2CDS-622 has a battery-backed RTC that will maintain the time for many months without power being applied to the system.

42.1 Operation

42.1.1 Set the CPUs Real Time Clock

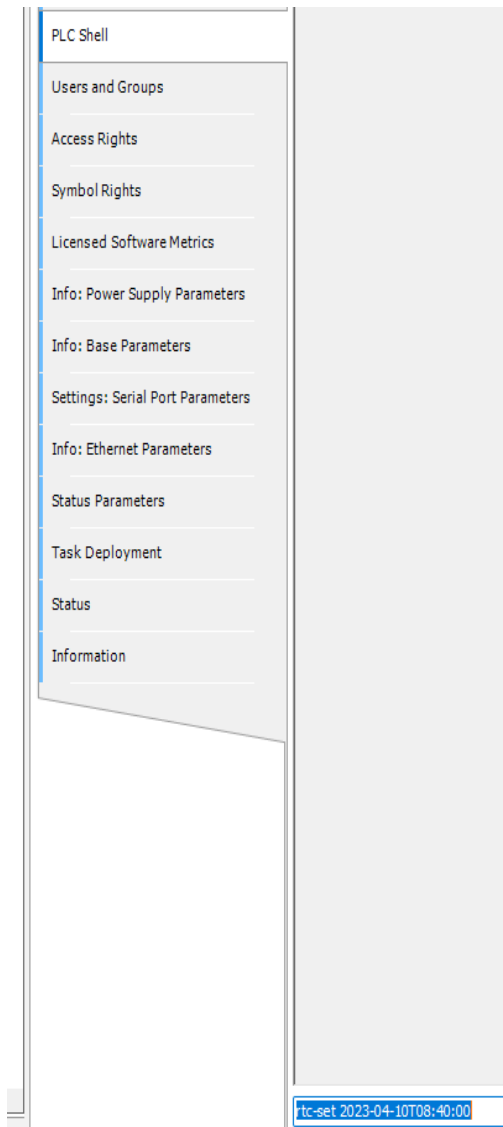
(1). In the Device Tree, Double-click on the **Device(P2CDS622)**.

A **Device** tab should appear with the first tab on the left selected entitled *Communication Settings* selected. Now we need to scan the network (**Scan Network**) to find our device.

Connect to the device of interest.

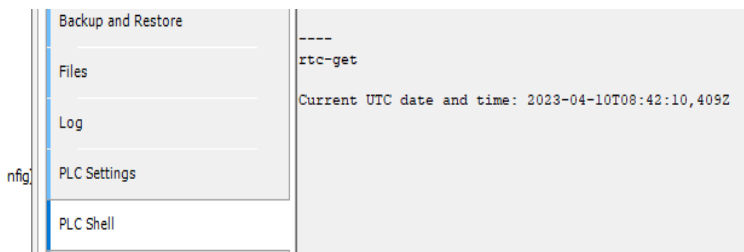
(2). In the *PLC Shell* section of that same view, type in the lower dialogue the command and the current local time as follows: **rtc-set YYYY-MM-DDThh:mm:ss**.

For example, to set the time to March 4th, 2023 at 1:07pm and 44 seconds, type the following dialog: **rtc-set 2023-03-04T13:07:44**.



The **Device connection to the CPU will reset/disconnect** after this command, so you will need to reconnect to the CPU.

To verify the time got set correctly, type **rtc-get** in the dialogue.



42.1.2 Access Time in Program

To be able to access the Time-of-Day and do other time measurements, you can install the SYSTimerRTC library.

Further information, refer to [CODESYS Help](#) section .

An example project is located at [CODESYS site](#) .

Overall, an **Excellent detailed reference and example** can be found in “SYSTimeRTC” section of [Textbook: “The Book of CODESYS”](#)

SECURITY AND USER MANAGEMENT

The P2CDS-622 has five areas of security provided by via the CODESYS IDE and the Runtime software.

43.1 Areas of Security

- 1). Device Security
- 2). Project/Application Permissions
- 3). Project File Encryption
- 4). IDE to Device (P2CDS-622) Connection Encryption
- 5). Visualization Security

Tip: Much of this material was generated with guidance from “Chapter 18-Security” in the [Textbook: “The Book of CODESYS”](#) Much more detail is in this text. It is highly recommended that you add this book to your library!

Also, you can refer to [CODESYS Online Help-Security](#)

43.1.1 Device Security

If not installed yet, install the **Security Agent** add-on from the CODESYS site- ****Security Agent**** This will enable a new tab called “Devices” in the **Security Screen**.

The first time the P2CDS-622 is powered up, there is no password associated with the device.

A password can be setup for the device (you are the default Admin) by clicking on **Device** and in the project tree, select **Users and Groups** and then **Add**.

Pick the **Default Group > Administrator**.

Define your **Name** and **Password**. Select *Password can be changed by user*.

Other Groups/Users outside of the Administrator can be given device passwords also and are setup in this area.

Groups and Users can be removed by the Administrator.

Note: Once a password is setup for an *Admin*, there must always be one setup for the Device. It can be changed but the unit will always require a defined *Device* password.

Danger: If the Device password is lost. The only way to recover is to perform a *Factory Reset* with the Firmware Update Utility. See *F/W Update- Factory Reset*.

Access control i.e. to get logged into the device and the associated privileges, can be setup initially by some default groups (discussed in more detail below).

This is accomplished with the **Device > Users and Groups** (synchronize to refresh) section.

43.1.2 Project/Application Permissions

To steps to enable *Project Level* security and set the permissions to edit a program, logon to the CPU, etc. are defined below. To do this an *Owner* must be defined, who “owns the project” and handles all further project setup permissions.

Initial Project Security

To create a password, open the project in the CODESYS IDE, go to **Project > Project Settings > Security** and select the **Encryption, Password** options and define a password.

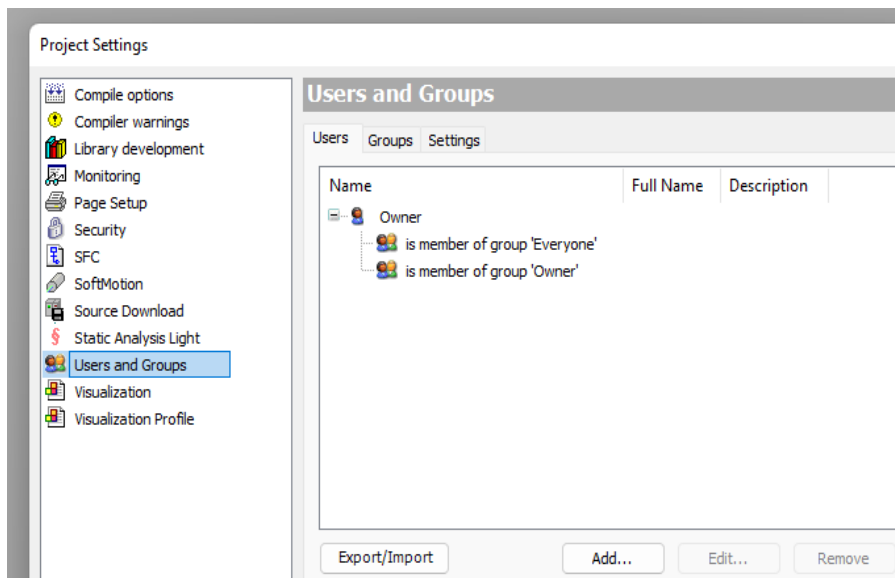
Once a password is defined, the CODESYS IDE, will prompt you to login every time the project is opened.

Configure an Owner

An *Owner* of a project has full access to everything associated with the project. *Owner* credentials are required to create other users and their associated permissions.

First you need to setup the *Owner* for the project or Application. When the project is first created there is a default *Owner* with no password. A password must be setup and it is recommended that you also create a new Username by doing an **Edit**.

Select **Project > Project Settings > Users and Groups** and you should see something similar to the following,

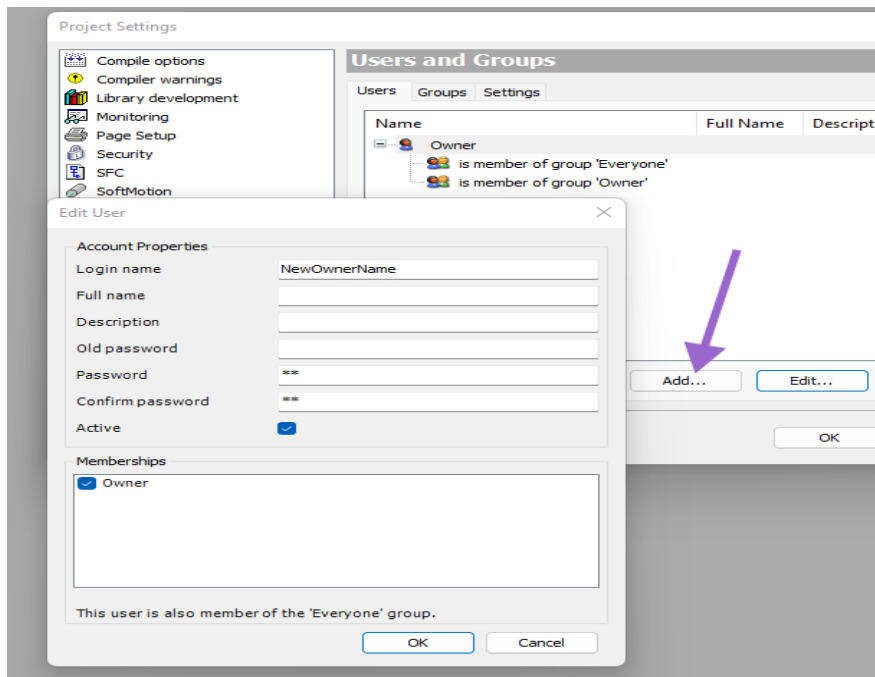


Select the *Owner* and click **Edit**.

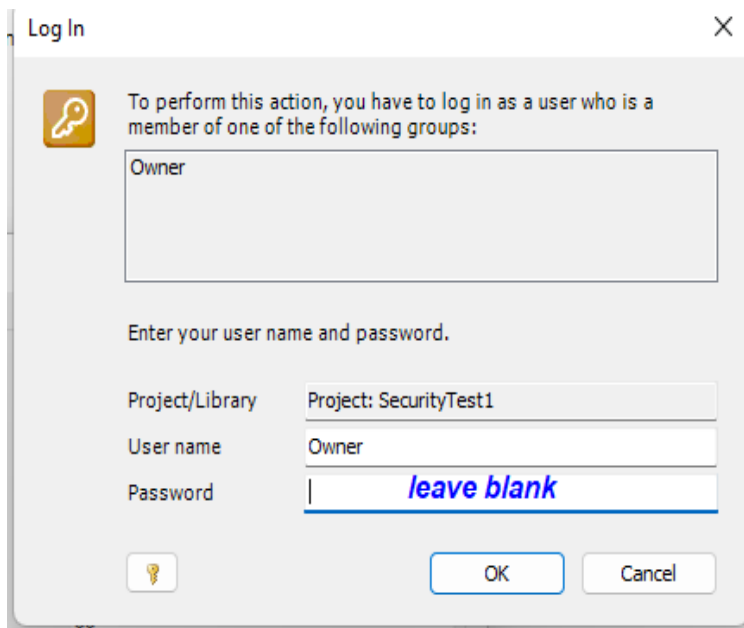
A pop-up will appear and edit it as follows:

- LOGIN NAME - name you want for Owner.
- FULL NAME - additional name info (optional).
- DESCRIPTION - additional info regarding owner (optional).
- OLD PASSWORD - delete all characters so it is blank.
- PASSWORD - password for this owner.
- CONFIRM PASSWORD - reenter

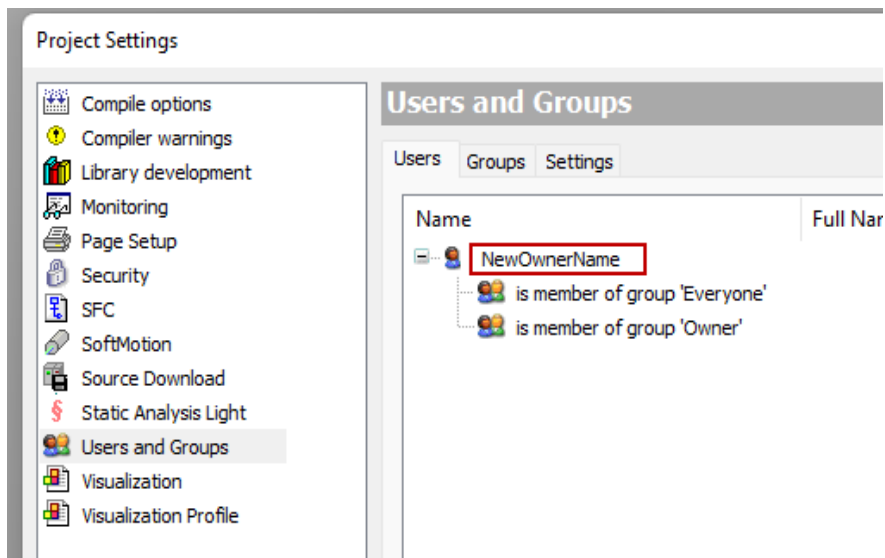
Select *Memberships* as **Owner**, click *OK*



Another popup window will appear, enter *Owner* and leave the *Password* blank as shown below,



After this, the new owner is setup and should look as follows:



Danger: If the Owner password is lost, there is no way to recover the project without it.

Create Users and Groups

Once an *Owner* is established, you can now add additional *Users* and create *Groups* of *Users*. For example, the Test Engineering group may have permissions to login, view the project and logging info but not be able to *Edit* objects.

You can either create *Users* first, then create *Groups* for those users be a member of, or you can first create the *Groups* you want and then add *Users* to these groups.

The example below first creates the Group and then adds Users to the Group.

Select **Project > Project Settings > Users and Groups**, click on the **Groups** and add. For Username/Password use the new owner Username and Password.

To add *Users*, in the previous project setting window click **Add** under the *Users* tab. Assign a Username and Password for this user and select their *Membership* (i.e. which Group they belong to).

If you are not currently logged in as the owner, you will be prompted to login to add the Group.

Access Control

Next, you can assign *Access Control* for a Group for this *Application*.

Right-click on the *Application* in the Devices view and select **Property > Access Control** tab.

Permissions can be assigned in the following areas:

- VIEW - users in the group may only view the contents of the object. No edits permitted.
- MODIFY - users may View and Modify, but may not delete the object.
- REMOVE - users may delete the object and all its dependencies.
- ADD/REMOVE - users may Add or Remove objects that are below this object, e.g. a *FunctionBlock*.

Select the permissions for each group as necessary.

Then for example, if an *Edit* is attempted on a file, you will be prompted to login as the Owner to be able to modify the file. A group without this privilege (*Deny*) will not be allowed to modify the project.

You can verify this by logging out if logged in as the owner- **Project > User Management > User Logout**, and attempting to edit the project.

You will be prompted to login as a user with *Modify* access control.

Limit User “Everyone” Access

By default when the CODESYS IDE is opened and no one is logged in, *Everyone* is the default user and thus everyone including groups with limited access which are part of the *Everyone* group (and its associated wide open access) have open access.

To resolve this we need to limit the *Everyone* user to *View* only.

This is done by selecting **Project > User Management > Permissions**.

Then under the **Project objects** and **Users, groups and permissions**, and every subheading i.e. **Add or remove children** edit **Everyone** to **Deny**

Only grant the **View** option

Now whenever an activity is attempted that has limited access, a login prompt will pop-up and require login by a user to a group that can take that action e.g. *Modify*.

Objects

Objects in CODESYS provide specific functionalities for creating your application. For example, Applications, Programs, Functions, Library Manager, Devices, and Image Pools are all Objects.. Objects are managed in tree structures in the Devices, POU's, and Modules views.

Objects within a Project can be assigned permissions for each Group just as previously done for the *Application*.

Right-click on the *Object* in the Devices view and select **Property > Access Control**.

Permissions can be assigned in the following areas:

- VIEW - users in the group may only view the contents of the object. No edits permitted.
- MODIFY - users may View and Modify, but may not delete the object.
- REMOVE - users may delete the object and all its dependencies.
- ADD/REMOVE - users may Add or Remove objects that are below this object, e.g. a *FunctionBlock*.

Actions

Along with placing restrictions on an *Object*, every *Action* in the CODESYS IDE may be restricted to certain groups.

This can be accessed by selecting **Project > User Management > Permissions**.

The items that can be selected include:

- COMMANDS - all the commands within CODESYS e.g. Login, Alarm Management, etc.
- OBJECT TYPES - permission to create an object e.g. POU, DUT, etc.
- PROJECT OBJECTS - additional way to set permissions the objects in the current project.

- USERS, GROUPS, PERMISSION - set which *Group* has permission to Add, Remove or Modify Users, Groups and Permissions.

Logging On/Off

Users may login or logoff through the **Project > User Management**

For more information on the Security of Projects, go to the CODESYS Help at [**SecureProject**](#)

43.1.3 Project File Encryption

By default, the project can be opened by anyone. The project file can be encrypted using the Security section of the *Project Settings*.

There are three levels of protection available.

To create the initial password to open a project, go to **Project > Project Settings > Security** and select the **Encryption, Password** options and define a password.

Every time thereafter the CODESYS IDE will be prompt the user to login (at this point the files are encrypted).

To use *Certificates* to encrypt your project, refer to online Help at [Certificates](#)

43.1.4 IDE to Device Connection Encryption

Communication with the device can be protected by means of encryption and user management on the device. You can change the current security preset on the Communication Settings tab of the device editor.

For more information on the Security of Communication, go to [SecureComm](#)

43.1.5 Encrypt Boot Application and Download

To encrypt boot applications, downloads, and online changes with a certificate to make sure that the application on the controller cannot be exchanged at will. Download a corresponding certificate of the type “Encrypted Application” from the controller and install it to the “Windows Certificate Store” of your computer. This certificate is required for all development environments that need to make changes to the application on the controller.

For example, if this application needs to be downloaded from another computer, then the certificate also must exist on this computer.

For the details of the procedures, see [SecureBoot](#)

43.1.6 Visualization Security

Protect the connection between the web server of the controller and the visualization client with the following measures:

- Configure an HTTPS connection (encryption with SSL/TSL) between the visualization client and the web server.
- Restrict access to the visualization and configure a visualization user management.

For more information on the Security of Visualizations, go to [SecureVisu1](#) also [SecureVisu2](#)

An example project is located in the *Setting up WebVisu* section of the site.

SYSTEM ERROR FLAGS DEFINED

This section provides an exhaustive list of all errors that a user can encounter developing an IEC application with the P2CDS-622.

Errors can occur during two phases of execution listed below:

- Configuration
- Scan

This page outlines severity types and their implications for the execution phases. In addition, it will describe the format for error IDs assigned to each error described in this document. Finally, it describes each execution phase and the errors that can occur each phase.

44.1 Severity Type

There two severity types listed below that a user can experience with the use of the P2CDS-622:

- Critical
- Non-critical

This section will describe the severity and their implications.

44.1.1 Critical Errors

Critical errors are errors that occur in either the configuration or scan phase that indicate a risk of the system not functioning safely or properly. The occurrence of a critical error in the configuration phase versus in the scan phase differ slightly.

Both occurrences of critical errors cause the P2CDS-622 to remain or enter Stop Mode until the error is corrected. The sections below describe the specifics of each phase when a critical error occurs.

Configuration Phase

When a critical error occurs, the P2CDS-622 returns an exception to the CODESYS IDE. This indicates to the CODESYS IDE that the PLC was unable to configure. The project download will fail and the Runtime will return an exception. The CODESYS IDE will get the exception and indicate to the user that the Runtime has generated an exception.

The CODESYS IDE will direct the user to the PLC log (example shown below) to determine the cause of the exception. Each exception generated in the Runtime has a respective log message indicating what caused the error. The user must correct the issue before they can install a new project.

Scan Phase

When a critical error occurs, the P2CDS-622 will enter Stop Mode if not already in Stop Mode. The P2CDS-622 will not enter Run Mode, until the user has corrected the error. Additionally, the P2CDS-622 will update the “Status” tab in the P2CDS-622 device window with the latest error.

44.1.2 Non- Critical Errors

Non-critical errors are errors that occur in either the configuration or scan phase that require attention, but do not indicate a risk of the system not functioning safely or properly.

The occurrence of a non-critical varies differently between the configuration and scan phase. The sections below describe the specifics of each phase when a non-critical error occurs.

Configuration Phase

When a non-critical error occurs, the P2CDS-622 will log a message. Currently, all non-critical errors during the configuration phase are only warnings of intentional actions by the user. For example, when a user configures a module with *Hot Swap* enabled, this will trigger the P2CDS-622 to log a warning. This does not necessarily require attention from the user.

Scan Phase

When a non-critical error occurs, the P2CDS-622 will log a message and update the “Status” tab in the P2CDS-622 device window.

44.2 Error List

Each error ID has an identifier describing the error. This allows customer service to easily identify the error and provide support to the customer. This section defines the error ID legend and numeric definitions for error IDs.

The Error Log can be found under the Project Tree **Device > Log**, for example:

Communication Settings	Components	0	6	0	172	60	Search in messages
Applications	Severity	Time Stamp	Description				
Backup and Restore	✖	08.03.2023 10:50:45.581	E02302: Slot 4 - Base reported this module as missing				
Files	✖	08.03.2023 10:50:45.581	E02302: Slot 3 - Base reported this module as missing				
Log	✖	08.03.2023 10:50:45.581	E02302: Slot 1 - Base reported this module as missing				
PLC Settings	✖	08.03.2023 10:50:45.457	E02205: Unable to communicate with the Base				
PLC Shell	ⓘ	08.03.2023 10:50:45.417	Application [Application] loaded via [Download]				
	✖	08.03.2023 10:49:14.181	E02205: Unable to communicate with the Base				
	ⓘ	08.03.2023 10:49:14.144	Application [Application] loaded via [Download]				
	ⓘ	08.03.2023 10:49:05.178	Webserver stopped				
	ⓘ	08.03.2023 10:48:52.939	*** Interactive Login: LoginRequest: Waiting for Run/Stop				
	✖	08.03.2023 10:48:19.916	E02205: Unable to communicate with the Base				

Note: The Prefix “IMP” in the descriptions below are for internal use only and have no CODESYS specific relevance.

44.2.1 Configuration Phase

When a user downloads a new project, the P2CDS-622 executes code to update the configuration of the entire system. Configuration refers to parsing the P2CDS-622 device description and configuring the system to all items in the device description. This can include, but is not limited to the following:

- Latest supported component versions
- Latest supported parameters
- Latest supported PLC configurations (4, 7, 11 and 15 slot)

During configuration, a variety of errors can occur and can be a result of bugs introduced by the user, P2CDS-622 design engineers, or both. This section will list each error, the responsible party, the severity of the issue, how the CODESYS IDE notifies the user, and any possible fixes.

1). Module Configuration Failure

- ID: *E02101*
- DESCRIPTION: The Runtime failed to configure the module because of an invalid configuration.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Exception
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the module device description to contain the expected parameters.

2). IMP 100 Message Failure

- ID: *E02201*
- DESCRIPTION: The Runtime failed to send a 0x100 IMP message to the base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: Verify base has power. If problem persists, cycle system power.

3). IMP 129 Message Failure

- ID: *E02202*
- DESCRIPTION: The Runtime failed to send a 0x129 IMP message to the base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: Verify base has power. If problem persists, cycle system power.

4). IMP 138 Message Failure

- ID: *E02203*
- DESCRIPTION: The Runtime failed to send a 0x138 IMP message to the base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: Verify base has power. If problem persists, cycle system power.

5). IMP 13F Message Failure

- ID: *E02204*
- DESCRIPTION: The Runtime failed to send a 0x129 IMP message to the base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: Verify base has power. If problem persists, cycle system power.

6). BASE H/W Info Failure

- ID: *E02205*
- DESCRIPTION: The Runtime failed to query the Base information from the Base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: User and/or Tech Support must investigate hardware to determine the cause of failure to communicate with Base.

7). Power Supply H/W Info Failure

- ID: *E02206*
- DESCRIPTION: The Runtime failed to query the power supply information from the Base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: User and/or Tech Support must investigate hardware to determine the cause of failure to communicate with Base.

8). IMP 621 (RJ-12) Message Failure

- ID: *E02207*
- DESCRIPTION: The Runtime failed to send a 0x621 IMP message to the Base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: Verify base has power. If problem persists, cycle system power.

9). IMP 621 (TBLK) Message Failure

- ID: *E02208*
- DESCRIPTION: The Runtime failed to send a 0x621 IMP message to the Base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: User or AutomationDirect.com
- POSSIBLE FIX: Verify base has power. If problem persists, cycle system power.

10). Module ID Mismatch Failure

- ID: *E02301*
- DESCRIPTION: The Runtime detected the project module and the physical module do not match.
- SEVERITY: Non-Critical
- NOTIFICATION METHOD: Log
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: Verify base has power. User must ensure the module in the project and the module in the base match.

11). Module ID Parameter Failure

- ID: *E06101*
- DESCRIPTION: The Runtime failed to parse expected module ID from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

12). Module Data Size Parameter Failure

- ID: *E06102*
- DESCRIPTION: The Runtime failed to parse expected module data size from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

13). Configuration Parameter Failure

- ID: *E06103*
- DESCRIPTION: The Runtime failed to parse expected module configuration from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

14). I/O Parameter Failure

- ID: *E06104*
- DESCRIPTION: The Runtime failed to parse expected I/O Channel parameters from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

15). Module Optional Parameter Failure

- ID: *E06105*
- DESCRIPTION: The Runtime failed to parse expected optional parameter (indicates if module is hot swappable) from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

16). Ethernet Settings Parameter Failure

- ID: *E06106*
- DESCRIPTION: The Runtime failed to parse expected Ethernet settings parameters from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

17). Status Parameter Failure

- ID: *E06107*
- DESCRIPTION: The Runtime failed to parse the expected PLC parameter from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

18). Serial (Terminal Block) Parameter Failure

- ID: *E06108*
- DESCRIPTION: The Runtime failed to parse expected serial terminal block connector parameter from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

19). Serial (RJ-12) Parameter Failure

- ID: *E06109*
- DESCRIPTION: The Runtime failed to parse expected serial RJ12 connector parameter from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

20). Info Parameter Failure

- ID: *E06110*
- DESCRIPTION: The Runtime failed to parse expected info parameters (e.g. power supply info, base info, etc) from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

21). SROM Parameter Failure

- ID: *E06111*
- DESCRIPTION: The Runtime failed to parse expected SROM parameters from the device description XML.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Expectation
- RESPONSIBLE PARTY: AutomationDirect.com
- POSSIBLE FIX: AutomationDirect.com must update the P2CDS-622 device description to contain the expected parameters.

21). Module is Optional Warning

- ID: *E06301*
- DESCRIPTION: The Runtime detected that a user has configured a module to hot swappable.
- SEVERITY: Non-Critical
- NOTIFICATION METHOD: Log
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: n/a

44.2.2 Scan Phase

During scan, a variety of error can occur that will require the user's attention. In the CODESYS IDE, the P2CDS-622 device maintains a "Status Parameters" tab to indicate to users the status of the P2CDS-622. The *Status Parameters* tab is located under Device > Status Parameters. In addition to indicating status, the Runtime will log all error events.

1). System Restart

- ID: *E01005*
- DESCRIPTION: The PLC restarted due to an unrecoverable internal error.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log, Stop Mode, and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: Switch to STOP then RUN. If problem persists, power cycle CPU.

2). SD Card Open Error

- ID: *E01607*
- DESCRIPTION: The PLC failed to mount the SD card.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log, Stop Mode, and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: Switch to STOP then RUN. If problem persists, power cycle CPU.

3). System SRAM Error

- ID: *E01802*
- DESCRIPTION: There was an error restoring retentive data.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log, Stop Mode, and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: Switch to STOP then RUN. If problem persists, power cycle CPU.

4). Base Communication Error

- ID: *E02205*
- DESCRIPTION: The PLC is unable to communicate with the backplane.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log, Stop Mode, and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: Switch to STOP then RUN. If problem persists, power cycle CPU.

5). Module Missing (Critical)

- ID: *E02302*
- DESCRIPTION: The Base has indicated that a module is missing from the base.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log, Stop Mode, and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: User must place the correct module into the Base.

6). Module Missing (Non-Critical)

- ID: *E02303*
- DESCRIPTION: The Base has indicated that a “Hot Swap” enabled module is missing from the base.
- SEVERITY: Non-Critical
- NOTIFICATION METHOD: Log and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: User must place the correct module into the Base.

7). Battery Voltage Low

- ID: *E05101*
- DESCRIPTION: The Runtime detected that the battery backed SRAM voltage is too low.
- SEVERITY: Non-Critical
- NOTIFICATION METHOD: Log and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: User must replace the battery.

8). Base ID Invalid

- ID: *E05102*
- DESCRIPTION: The Base ID does not match expected ID.
- SEVERITY: Critical
- NOTIFICATION METHOD: Log and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: User must replace the Base.

9). RS232 5V Overload

- ID: *E05123*
- DESCRIPTION: The 5V pin on the RS232 port is overloaded or has a short-circuit.
- SEVERITY: Non-Critical
- NOTIFICATION METHOD: Log and Status Updated
- RESPONSIBLE PARTY: User
- POSSIBLE FIX: User must inspect/reduce the load or CPU could have damaged RS-232 port.

SYSTEM ERRORS ACCESSED

This section describes how to read the error codes in your project. These codes are defined in the *Error Codes* section of this site.

The variables are- *sErrorMessageCritical*, *sErrorMessageNonCritical*.

45.1 Variable Access

To access these variable, install the device *P2CDS622_SystemTags*, by right-clicking on **Device** and **Add Device > Miscellaneous**.

Then add the device and connect to the P2CDS-622.

The variable in the IO Mapping space is of an *Array of Bytes*. This ARRAY can be converted to a STRING by creating a variable in your project of type STRING` and assigning this variable in the IO Mapping. It automatically converts the array to a string.

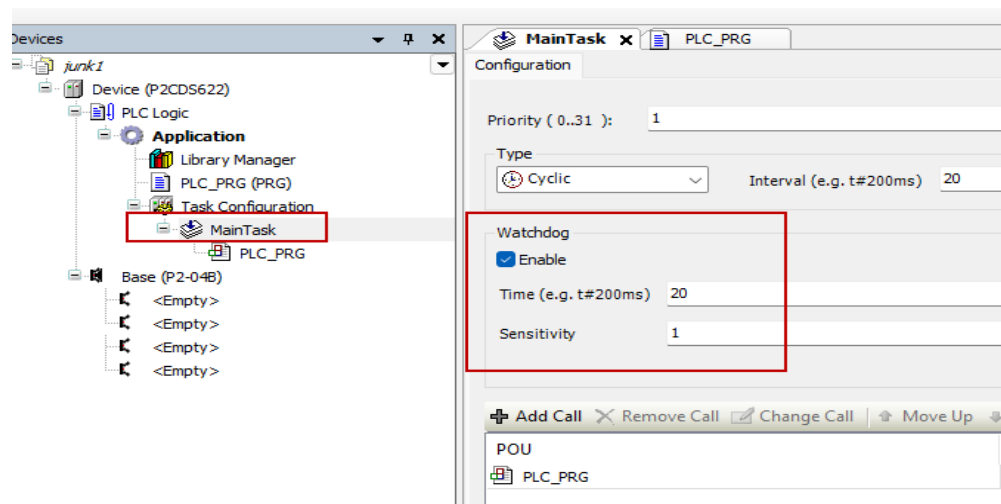
An example value for this string is shown below (“No Errors Detected” or “E023202: Module Missing”).

Variable	Mapping	Channel	Ac
byBatteryIsVoltageTooLow		byBatteryIsVoltageTooLow	%I
uiBatteryCurrentVoltage		uiBatteryCurrentVoltage	%I
byErrorIsCritical		byErrorIsCritical	%I
Application.PLC_ORG.myCritical		sErrorMessageCritical	%I
byErrorIsNonCritical		byErrorIsNonCritical	%I
Application.PLC_ORG.myNonCritical		sErrorMessageNonCritical	%I
sFirmwareVersion		sFirmwareVersion	%I

Variable	Mapping	Channel	Ac
errorNonCritical			
myCritical			
myNonCritical			

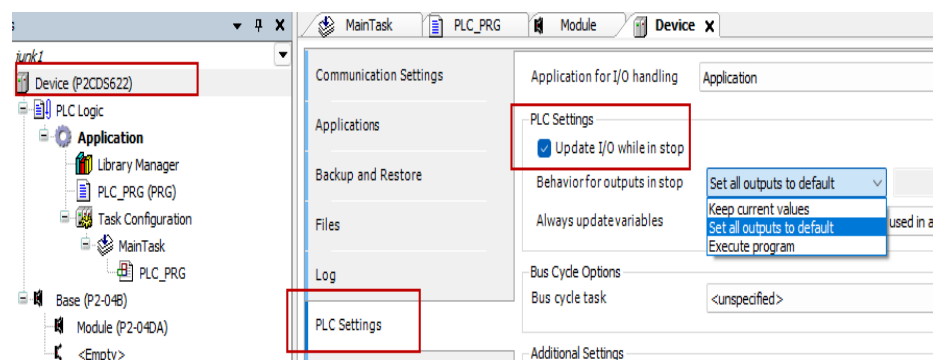
WATCHDOG/OUTPUTS

The P2CDS-622 utilizes the Watchdog feature that is in the CODESYS IDE under the **Task Configuration** section as shown below:



The state the Outputs driven out during a Watchdog error event is also selected in the IDE on a per module basis.

First, one selects the mode e.g. *Default* to drive the outputs of that module. Note- module goes into **STOP** mode after a watchdog event.



Then within the IO Module *Mapping* configuration, the default mode value i.e. **Default Value** in the options is defined by the user as shown below.

If a Watchdog occurs, the outputs will be driven to these levels.

Devices

junk1

Device (P2CDS622)

PLC Logic

Application

Library Manager

PLC_PRG (PRG)

Task Configuration

MainTask

PLC_PRG

Base (P2-0-4B)

Module (P2-0-4DA)

Module

P2_04DA Parameters

P2_04DA I/O Mapping

Status

Information

Find

Filter

Show all

Variable	Mapping	Channel	Address	Type	Default Value
myOutSigs_1		Ch1	%QD0	DINT	65536
myOutSigs_2		Ch2	%QD1	DINT	65536
myOutSigs_3		Ch3	%QD2	DINT	0
myOutSigs_4		Ch4	%QD3	DINT	0
		Status	%IB0		

RECIPE MANAGER

The recipe manager provides functions for maintaining user-defined variable lists, known as recipe definitions. The recipe definitions can be stored in recipe files on the CPU and are created and edited in the Recipe Manager object and saved to a file.

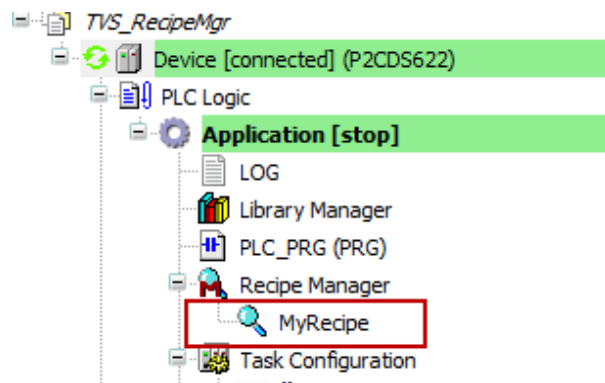
For more information on the Recipe Manager, go to [Recipe Mgr](#)

To clarify on how to specify the storage location of the Recipe file in the CPU, see the screen shots below.

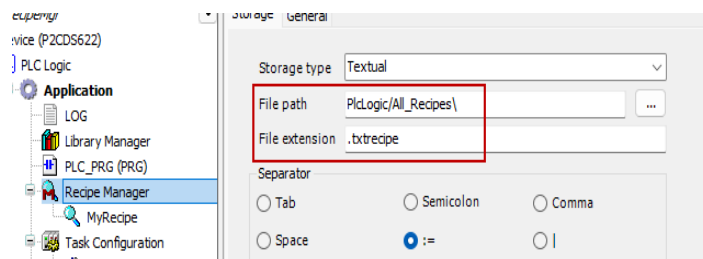
MyRecipe is the name of the recipe I am setting up.

PlcLogic/All_Recipes is the directory on the CPU to where the file is stored. A backslash “\” is required at the end of directory name”.

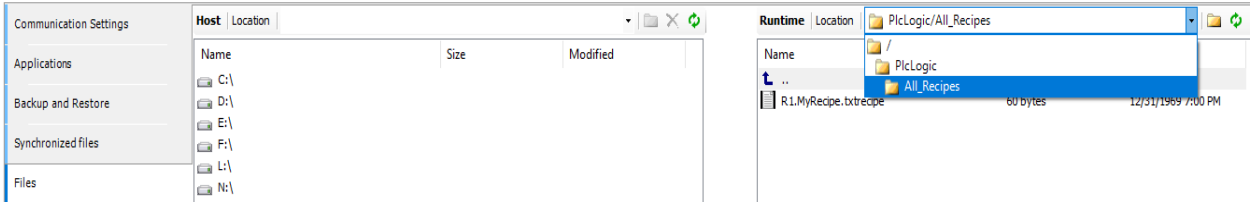
Name of the Recipe:



Setting up the directory:



Looking at the file generated on the PLC (*RI.MyRecipe.txtrecipe*)



47.1 Alternate Approach - CSV File

The Recipe file that is generated with CODESYS is a binary file that is not readable or editable. An alternative solution to produce a readable/editable recipe file is to use the Controlsphere *Central Configuration Service (CSS)* Library.

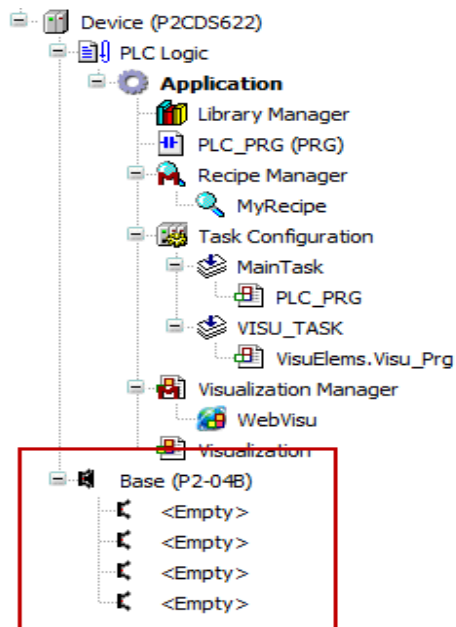
Go to- [CCS RecipeManager Solution](#)

SCAN BASE FOR MODULES

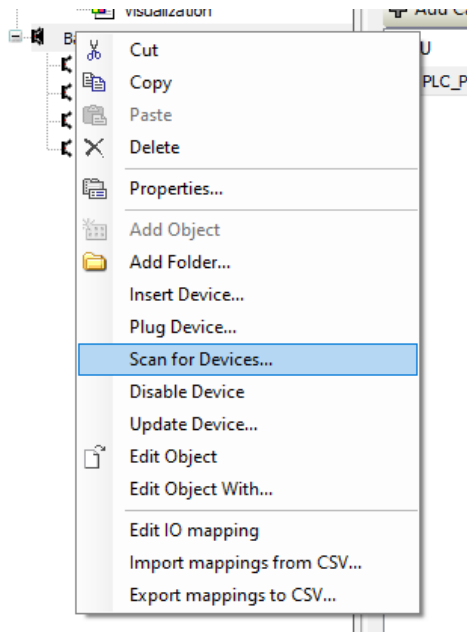
An alternative to manually configuring IO Modules into the Base in the CODESYS IDE, there is an option that allows a user to scan the Base for IO Modules present and then populate these into the Base Device in the Project Tree.

48.1 Operation

The image below shows a system with IO Modules plugged into the Bases slots, but not yet populated in the Base device in the project tree as below:

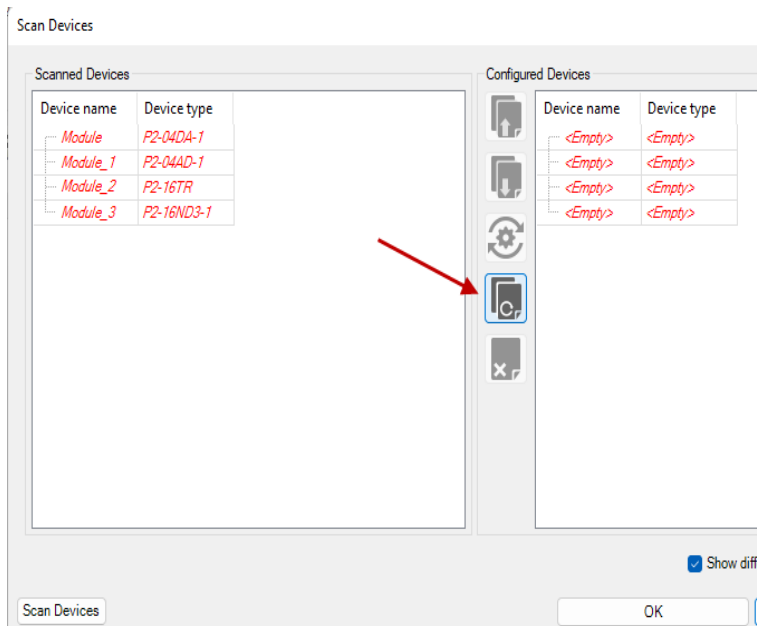


Right click on **Base** and select **Scan for Devices..** .



Select the **Show differences in project** option in lower right corner.

Click on the **Copy All** icon. Click ok and all modules will be populated.



MEMORY UTILIZATION

The amount of memory available on the P2CDS-622 is as follows:

- Program Memory: 50MB
- Battery Backed SRAM: 1MB (*RETAIN* Data Type - 800KB, *RETAIN-PERSISTENT* Data Type - 200KB)

- (1). Connect to the P2CDS-622 and Login (download Project into CPU)
- (2). To determine the amount of memory that has been utilized by the PLC program, click on the *Device* (*P2CDS622*) in the device tree.

Then select the **License Software Metrics** and note the **Size of User Code**.

The image below illustrates this:

The screenshot shows the 'Device' window for P2CDS622. The left sidebar lists various settings, with 'Licensed Software Metrics' selected and highlighted by a red box. The main area displays a table of metrics. The 'Size of User Code' metric is highlighted with a red box, showing a value of 21,280 Byte. Other metrics include 'Number of OPC UA tags' (0), 'Variables used in visualizations' (58 Variables), 'IO Channels' (0 Channel(s)), and 'SoftMotion' (0 Axes).

Metric	Value in Project
Size of User Code	21,280 Byte
CSVReaderExample	n/a
CSVWriterCFCEExample	21,280 Byte
CSVWriterSTExample	n/a
Communication	
Number of OPC UA tags	0
CSVWriterCFCEExample	0
CSVWriterSTExample	0
CSVReaderExample	0
Variables used in visualizations	58 Variables
CSVReaderExample	n/a
CSVWriterCFCEExample	58 Variables
CSVWriterSTExample	n/a
Fieldbus	
IO Channels	0 Channel(s)
CSVWriterCFCEExample	0 Channel(s)
SoftMotion	
Number of axes	0 Axes

Note: The column entitled *Value on Device* and *Max. allowed* do not apply and will not have any values.

RETENTIVE MEMORY

Variables can be assigned the **Data Type** `retain` or `retain-persistent` and their values will be maintained if power is removed.

See the following for help- [Retain Memory](#)

Both variable types are stored in the Battery-backed SRAM memory.

- Battery Backed SRAM: 1MB (*RETAIN* Data Type - 800KB, *RETAIN-PERSISTENT* Data Type - 200KB)

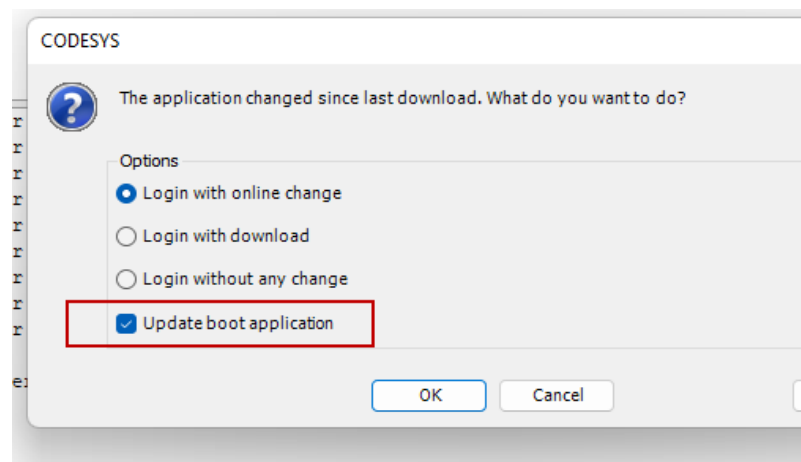
Caution: The `retain-persistent` variable values are NOT copied into Non-volatile Flash memory on a system power down and will be lost, if there is no battery or if the battery is in a low voltage state for an extended time period.

LOGIN-ONLINE CHANGE

Program changes can be made to the CPU between process cycles (scans) without upsetting the program e.g. variable resets of non-changed items. However, these changes are temporary and will not persist if the CPU is power cycled or rebooted.

To make program changes persist through a power cycle or reboot of the CPU, the changes must be written into the nonvolatile memory of the CPU.

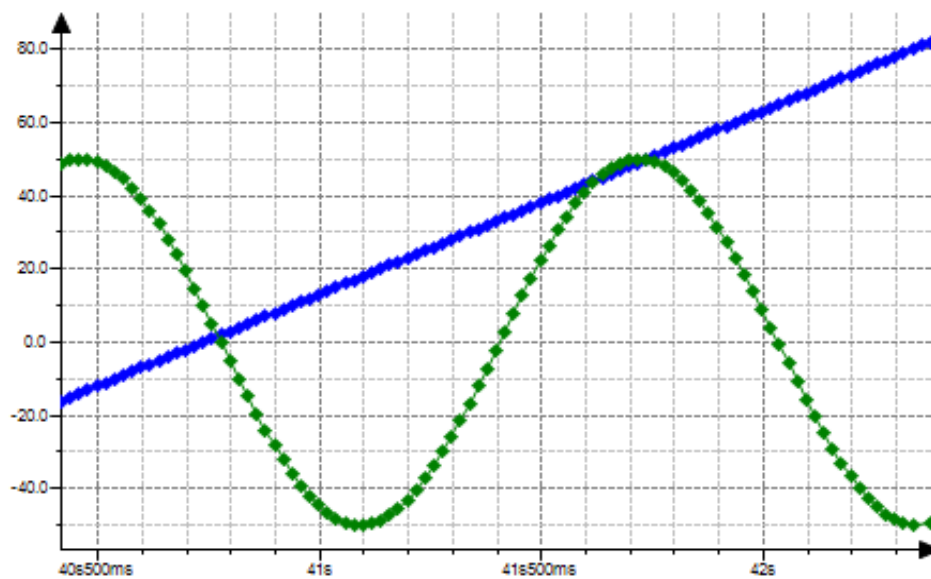
When prompted to login to the CPU, check the **Update boot application** box as shown below.



See the following for additional help- [Login Online Change](#)

TRACE

A trace can be used to follow the value curve of variables in the CPU.



52.1 The Trace Object

An object of type *Trace* is used to configure and display application-specific trace data in one or more charts. At application runtime, value curves of trace variables, which you can monitor in the trace editor in CODESYS during this time, are recorded on the controller. To monitor an object in the trace editor during runtime, a trace configuration must be configured and transferred to the controller, and the trace recording must be started.

At application runtime with trace, all statements are executed first within the task cycle. Then, data sampling starts with value storage including time stamps. These time stamps are relative to the start time of the data sampling. The data yields a discrete time signal and CODESYS displays its course in the trace editor. With this element, you can integrate a trace graph in the visualization that monitors and displays variable values permanently.

You configure the displayed trace graph in the element properties. In addition, you can add controls that control the trace functionality. This is done manually or by means of the Insert Elements for Controlling Trace command.

The sampled data is transmitted to the development system and displayed in charts according to the configuration.

You can navigate through the data when tracing.

52.1.1 Detail Usage and Example

For additional help and an example project refer to [Trace](#)

JITTER ASPECTS

Performance in real-time applications is highly dependent on system-level behavior, and requires every element to perform reliably and predictably within a specific time window to avoid failure.

For applications concerned about “real-time” performance aspects, CODESYS provides a mechanism to view key timing parameters that are involved.

There are three metrics associated with the PLC’s timing performance (cycle time)- Jitter, Latency and Release Jitter.

A PLC *Bus Cycle Process Loop* is measured in cycle time of the particular task.

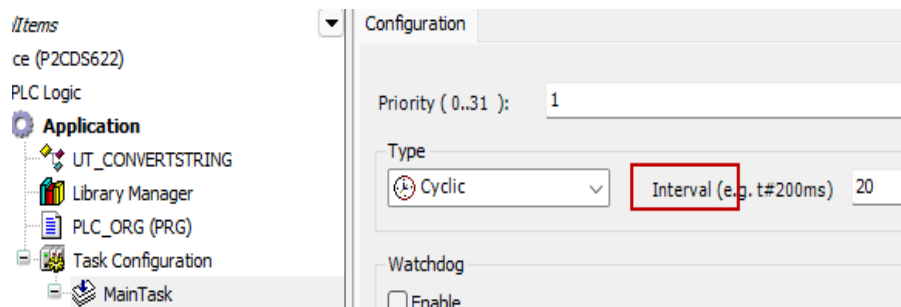
This loop starts from Reading the PLC Inputs, then Running the User Programs and finally Writing the Outputs. The time it takes to complete one(1) cycle is the PLC’s *Task Cycle Time*.

In many applications such as motion, the time variance from cycle-cycle must be kept as small as possible.

The deviation of the task cycle time from the desired set task cycle time (Tset) is referred to as **Periodic Jitter** (Jper).

$$J_{per} = T_{per} - T_{set}$$

The desired (ideal) cycle time Tset is specified in the Configuration of the Task as *Interval*.



Latency (L) is the delay between the *invocation* of a task and the actual *start* of its release.

The **Release Jitter** (Jr) is the difference between the maximum and the minimum Latency that has ever occurred.

$$J_r = L_{max} - L_{min}$$

Note: If the sum of all negative Jper values and the sum of all positive Jper values does not equal zero(0), then a **drift** results.

The associated CODESYS Help section can be found here- [Jitter](#)

53.1 Display of Metrics

The current value, as well as the maximum and minimum values of the periodic jitter, can be viewed on the *Monitor* tab of the Task Configuration section for that specific task.

The Task Configuration include:

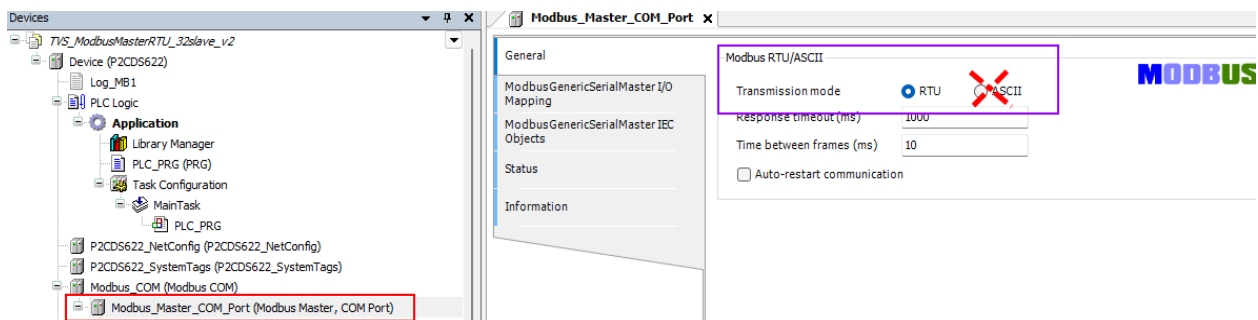
- Task - name defined in the task configuration.
- Status - current operation of the Task.
- IEC-Cycle Count - (not supported)
- Cycle Count - Number of executed cycles since logging in to the controller, even if the task is in STOP mode.
- Configured Cycle Period - user defined value in configuration.
- Last Cycle Time (us) - last measured cycle time.
- Average Cycle Time (us) - average cycle time over all cycles.
- Max. Cycle Time (us) -
- Min. Cycle Time (us) -
- Jitter (us) - current value of the Periodic Jitter.
- Min. Jitter (us) - minimum measured periodic jitter.
- Max. Jitter (us) - maximum measured periodic jitter.

Tip: Values can be reset by right clicking in the context area and selecting **Reset**.

MODBUS - ASCII MODE

54.1 Description

When setting up a Modbus Master, there is an option to use “ASCII” mode.



This mode is **not** supported in P2CDS-662 due to CODESYS legacy and phasing out.

Standard **non-Modbus** Serial ASCII is available using the serial comms library functions.

The following example may be a help *Serial Comm*

54.2 Workaround

Use either Modbus-RTU or Serial ASCII if RS-232 or RS-485 required.

54.3 Future Resolution

This issue will not be resolved.

ETHERNETIP - ACD

“ACD” stands for Address Conflict Detection and is used to detect when multiple slaves on a network are using the same IP Address and a possible bus conflict could occur.

55.1 Description

The P2CDS-622 Ethernet/IP Scanner (Master) or Adapter (Slave) will not be able to detect and record the presence of duplicate IP addresses. If this scenario were to occur, the devices connected would show in the device tree in the error state.

Data integrity will be compromised if there are duplicate addresses.

55.2 Workaround

The user needs to ensure connected devices do not have the same/duplicate IP Address.

55.3 Future Resolution

This issue is being investigated for resolution.

“RETAIN-PERSISTENT” NV STORAGE

56.1 Description

The P2CDS-622 holds the “Retain-Persistent” data type values in Battery-Backed SRAM, but does NOT write these variables into the Flash memory on a system power down.

If the Battery dies, all “Retain-Persistent” data will be lost.

56.2 Workaround

The user needs to ensure that the battery voltage is never allowed to go below its operating threshold.

There is a low battery warning in the CODESYS IDE that can be used to monitor the batter voltage. If removed, there is a **minimum of 8 hours of “Super-Cap” emergency energy storage** to hold the “Retain-Persistent” data.

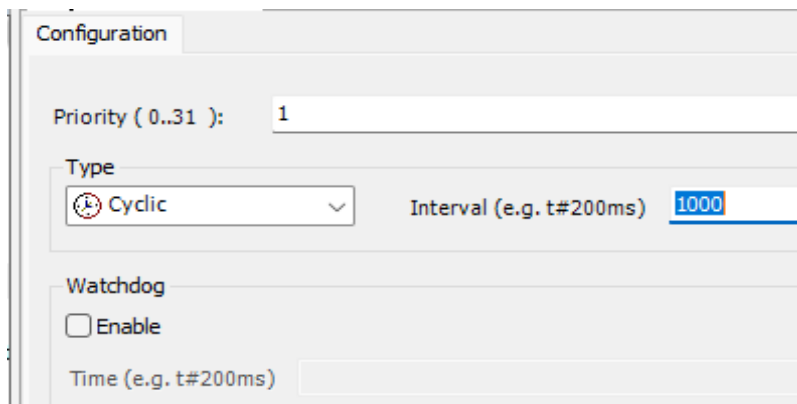
56.3 Future Resolution

This issue will not be resolved.

TASK TIMES > 1 SECOND

57.1 Description

Caution: Tasks with cycle intervals **greater than 1 second** may result in Watchdog timeouts.



The screenshot shows a configuration window with the following fields:

- Configuration** (tab)
- Priority (0..31):** 1
- Type:** Cyclic (dropdown menu)
- Interval (e.g. t#200ms):** 1000
- Watchdog:**
 - ☐ Enable
 - Time (e.g. t#200ms):** (empty text field)

57.2 Workaround

If required, set Task cycle time a maximum of **750ms** to insure margin.

57.3 Future Resolution

This issue will not be resolved.

“SYMBOL CONFIGURATION”

58.1 Description

The object “Symbol Configuration” upon building will generate build errors. This feature is used for OPC-UA and the P2CDS-622 does not support this fieldbus.

There is an associated XML file generated though in the project and this could be used to export project tags/variables if desired.

58.2 Workaround

There is no workaround for this item.

58.3 Future Resolution

This issue is not planned to be resolved.

RAPID SWITCHING OF RUN/STOP

59.1 Description

A rapid continuous toggling of the Run/Stop switch will result in the P2CDS-622 CPU going into Watchdog error mode.

A power cycle is required to recover.

59.2 Workaround

Avoid rapid successive toggling of the Run/Stop switch.

59.3 Future Resolution

This issue will not be resolved.

IMPLICIT CHECKS “CHECKBOUNDS”

60.1 Description

There are some built-in POU's in the CODESYS software to do checks on your project. One is called **Check Bounds** discussed here [Check Bounds POU](#) .

Utilizing the built-in CODESYS POU “Check Bounds” function and not doing the below modification will cause the CODESYS IDE system to hang during a project download into the P2CDS-622 CPU.

If this occurs, you need to cancel the download Task and incorporate the following workaround.

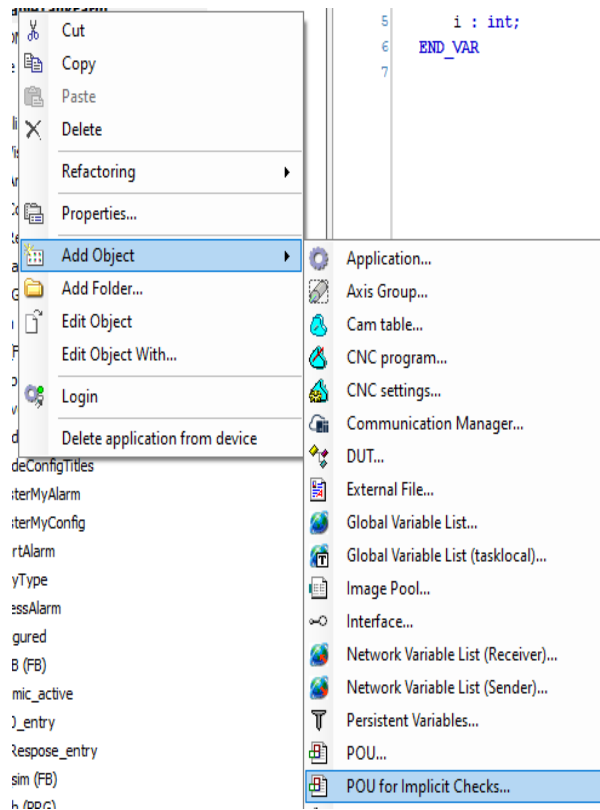
Note: Incorporating this modification will not alter the intended functionality for the “Check Bounds” Function.

60.2 Workaround

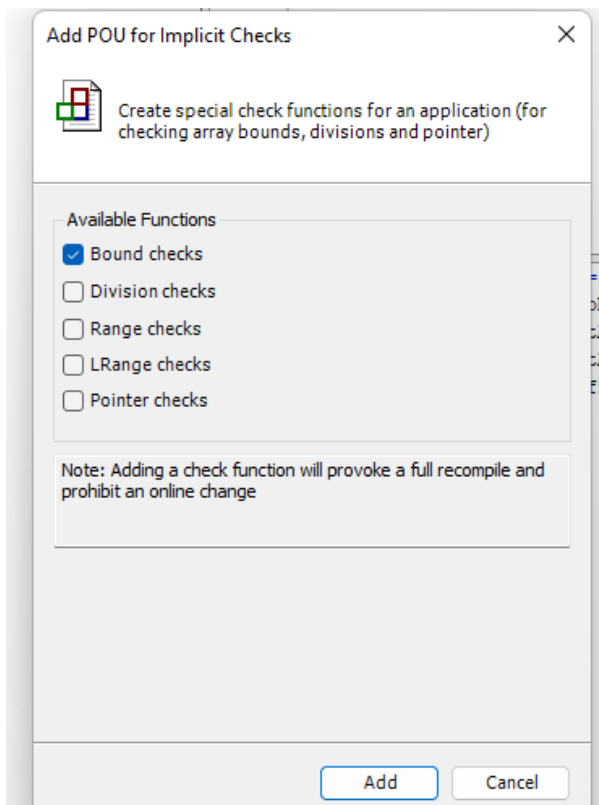
The following illustrates the steps to implement this function and the associated edits to be made to the CODESYS generated code.

60.2.1 Add the POU and Function

Add the Object into the project from drop-down menu **Project** >...



Select the **Bounds Checks**



Note: The **Check Bounds** Function is the **only** function affected by this errata. The others, e.g. **Range Checks** require no edits.

60.2.2 Inspect Generated Code

The following shows the CheckBounds Function code generated and the areas that will be modified.

```

1 // Implicitly generated code: Only an implementation suggestion
2 IF index < lower THEN
3     CheckBounds := lower;
4 ELSIF index > upper THEN
5     CheckBounds := upper;
6 ELSE
7     CheckBounds := index;
8 END_IF
9
10 (*It is also possible to set a breakpoint, log messages or e.g. to halt on an exception:
11 Add CmpApp.library, SysExcept.library and SysTypes2_Itf as newest.
12 Declaration:
13 VAR
14     _pApp : POINTER TO CmpApp.APPLICATION;
15     _result : SysTypes.RTS_IEC_RESULT;
16 END_VAR
17
18 Implementation:
19 _pApp := AppGetCurrent(pResult:=_result);
20 IF index < lower THEN
21     CheckBounds := lower;
22     IF _pApp <> 0 THEN
23         AppGenerateException(pApp:=_pApp, ulException:=RtsExceptions.RTSEXCEPT_ARRAYBOUNDS);
24     END_IF
25 ELSIF index > upper THEN
26     CheckBounds := upper;
27     IF _pApp <> 0 THEN
28         AppGenerateException(pApp:=_pApp, ulException:=RtsExceptions.RTSEXCEPT_ARRAYBOUNDS);
29     END_IF
30 ELSE
31     CheckBounds := index;
32 END_IF
33 *)

```

60.2.3 Modify Generated Code

Make the following edits to the function.

- 1). Move the VAR declarations into the **Declaration** section.
- 2). Move the _pApp code line under the **Conditionals**- IF and ELSIF.

```

1  // Implicitly generated code : DO NOT EDIT
2  FUNCTION CheckBounds : DINT
3  VAR_INPUT
4      index, lower, upper: DINT;
5  END_VAR
6  VAR
7      _pApp : POINTER TO CmpApp.APPLICATION;
8      _result : SysTypes.RTS_IEC_RESULT;
9  END_VAR

1 // Implicitly generated code: Only an implementation suggestion
2 IF index < lower THEN
3     CheckBounds := lower;
4 ELSIF index > upper THEN
5     CheckBounds := upper;
6 ELSE
7     CheckBounds := index;
8 END_IF
9
10 IF index < lower THEN
11     _pApp := AppGetCurrent(pResult:=_result); //moved to under "IF"
12     CheckBounds := lower;
13     IF _pApp <> 0 THEN
14         AppGenerateException(pApp:=_pApp, ulException:=RtsExceptions.RTSEXCPT_ARRAYBOUNDS);
15     END_IF
16 ELSIF index > upper THEN
17     _pApp := AppGetCurrent(pResult:=_result); //moved to under "ELSIF"
18     CheckBounds := upper;
19     IF _pApp <> 0 THEN
20         AppGenerateException(pApp:=_pApp, ulException:=RtsExceptions.RTSEXCPT_ARRAYBOUNDS);
21     END_IF
22 ELSE
23     CheckBounds := index;
24 END IF

```

60.3 Future Resolution

This issue is not planned to be addressed further.

INTRODUCTION

The P2CDS-622 CPU is based around on AutomationDirect.com P2000 system comprised of Bases, Power Supplies and various IO Modules.

As such, the “Installation and Wiring” chapter in the P2000 Hardware Users Manual is relevant to the P2CDS-622 based PLC.

Note that not all the IO Modules listed in the manual are supported by the P2CDS-622. Also, the CPU referenced in the manual does not apply.

This section is located here at [Install and Wiring](#) .

P2CDS-622 SYSTEM HARDWARE CENTRIC

This section is for project items that involve hardware related issues such as configuring IO Modules, serial ports not functioning as expected, etc.

For CODESYS IDE related items, please refer to the *CODESYS Centric* support links.

62.1 P2CDS-622 Support Forum

The AutomationDirect.com support forum will be monitored daily to assist the community in resolving any issues.

See [Support Forum](#) .

CODESYS CENTRIC

Below are the hyperlinks to the CODESYS IDE support areas that do **not** involve P2CDS-622 and associated IO Modules, Power Supply or Base hardware.

Also listed are links for application and coding examples.

63.1 CODESYS Textbook

- [Textbook: “The Book of CODESYS”](#)

63.2 CODESYS Online Help

- [CODESYS Online Help](#)

63.3 CODESYS Support Forum (Forge)

- [CODESYS Forge](#)

63.4 CODESYS Official YouTube

- [CODESYS YouTube](#)

63.5 CODESYS Users YouTube

- [Tohid A.](#)
- [Brian Hobby](#)

63.6 CODESYS Trainings/Classes

- [CODESYS Training](#)

63.7 CODESYS Store

- [CODESYS Store](#)