

PENKO Engineering B.V.

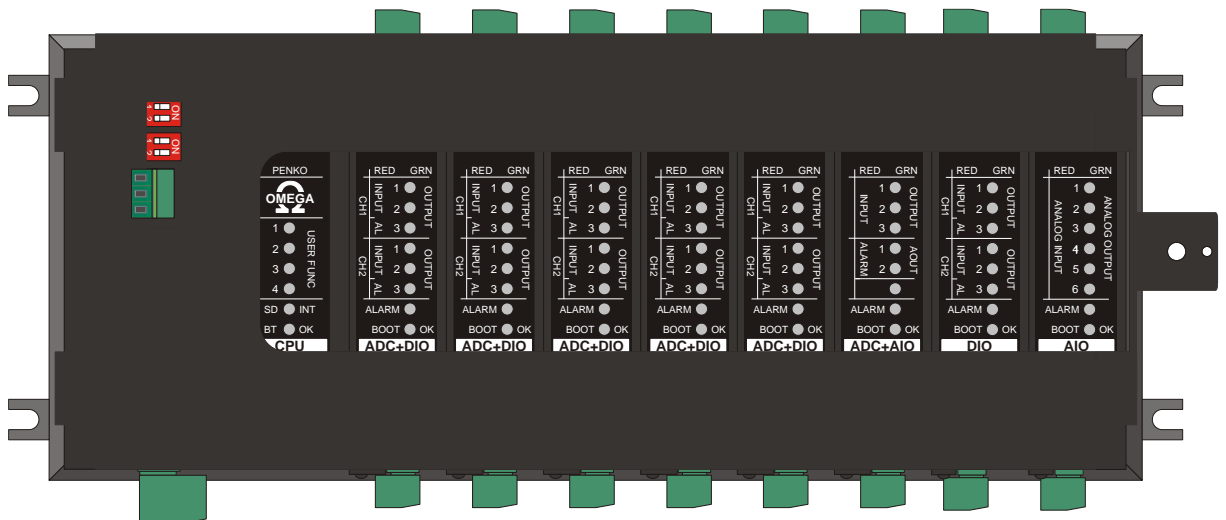
Your Partner for Fully Engineered Factory Solutions

Omega System CodeSys manual



CODESYS

&



PENKO

an ETC Company

Penko EasyWeigher manual

CONTENTS

Chapter 1. Introduction.....	4
Chapter 2. Prerequisites.....	5
Chapter 3. Setup your environment	6
Step 1: Download CodeSys files from your Omega	6
Step 2: Prepare the development environment.....	9
A. Open the environment	9
B. Install the omega device descriptions in the device repository	9
C. Install the CodeSys libraries.....	10
Chapter 4. Create your first application.....	12
Step 1: Creating a new project	12
A. Create a new project	12
B. Select the project type.....	12
C. Select the Omega device	12
Step 2: Write your first application	13
A. Add the installed library to the current project.....	15
B. Library documentation	16
C. Select PLC_PRG (PRG).....	16
D. Write a blink program: example 1	17
Step 3: Run your program.....	18
A. Select the Omega device	18
B. Download the program to the device.....	19
C. Program in running mode.....	20
Chapter 5. Write a blink program: example 2.....	22
Chapter 6. EasyWeigher: First application	24
Step 1. Prerequisites	24
Step 2. Create the project.....	24
Step 3. Example IO mapping.....	26
Step 4: Optional: Add a visualization	27
Chapter 7. EasyWeigher: Library documentation	30
7.1 View library documentation in CodeSys	30
7.2 Weigher status struct mapping	30
7.2 Weigher commands (tare, zero, etc)	31
7.3 Reading configuration.....	34
7.4 Writing configuration – Industrial Mode	36



Penko EasyWeigher manual

7.5 Writing configuration – Certified Mode	37
7.6 Calibration	38
Appendix I: Blink program example 1	40
Appendix II: Blink program example 2	41



PENKO

an ETC Company

Penko EasyWeigher manual

CHAPTER 1. INTRODUCTION

This document describes how to setup and work with CodeSys on the Penko Omega. Before you start, it is expected that you have read the Omega Manual and that you can access the Omega Web User Interface in your browser.

There are four main parts to this document:

1. Setup (Chapter 2). Here we download all necessary files and setup the development environment
2. Creating a project (Chapter 3). This chapter describes how to start a new project for your Omega device.
3. Blinky (Chapter 4). In this chapter we create simply Blinky program to blink the LEDs on the CPU module.
4. EasyWeigher (Chapter 5....). These chapters explain how to create your first weighing project with the provided EasyWeigher library.

Penko EasyWeigher manual

CHAPTER 2. PREREQUISITES

This tutorial is designed to create your first PLC application by using CodeSys and a Penko device. There is no need for previously proven software development skills to successfully complete this tutorial. All that is required is listed below.

- ✓ Installed the CodeSys IDE V3.5 or higher from store.codesys.com on your computer. The CodeSys store requires registration before you can download the software.
- ✓ Omega device with CodeSys license up and running in your office or factory network¹.

The goal is to create, upload and run a blink application in the Omega. This tutorial provides two different example applications for controlling the LED's. There are four user function LED's on the Omega CPU card. Each LED can either be off, red, green, or yellow.



¹ Having trouble with the getting the Omega up and running? Consult the omega manual at penko.com for this.

Penko EasyWeigher manual

CHAPTER 3. SETUP YOUR ENVIRONMENT

STEP 1: DOWNLOAD CODESYS FILES FROM YOUR OMEGA

This chapter will guide you through downloading and installing the necessary files and libraries, and help you setup a very simple starter project. Subsequent chapters will further expand this example with more functionality.

It is expected that you have setup your Omega, and that you can connect to your Omega's web portal from the PC you're working on. See the Omega manual ('7600M1082-<LANGUAGE>-R8 MANUAL OMEGA.pdf') on how to setup your Omega.

An archive with device descriptions and libraries can be downloaded from the Omega web portal.

Navigate to your Omega's web portal, login and click on "CODESYS Downloads". Follow the steps there to download the libraries and device descriptions as .zip file from that page, and install in CODESYS, see Figure 1.

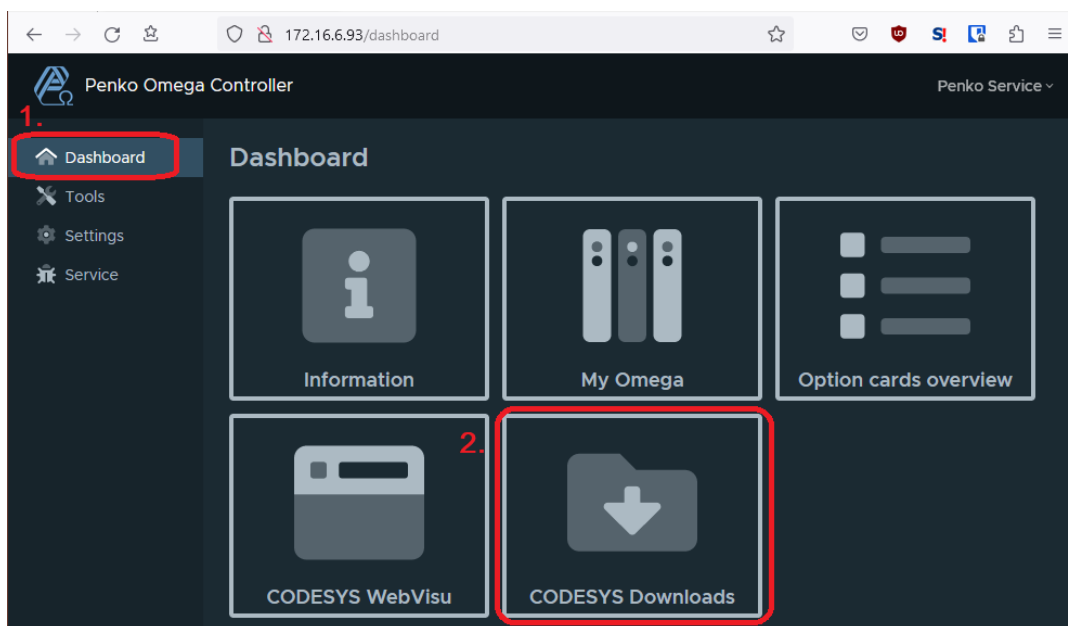


Figure 1. Navigate to "CODESYS getting started".

Click the button to 'Download CodeSys libraries & dependencies' (see Figure 2). This will download a zip file to your computer. Some computers will automatically download it to a default directory, other computers may ask you where you want to save it (Figure 3). Take note of where the files is stored!

Penko EasyWeigher manual

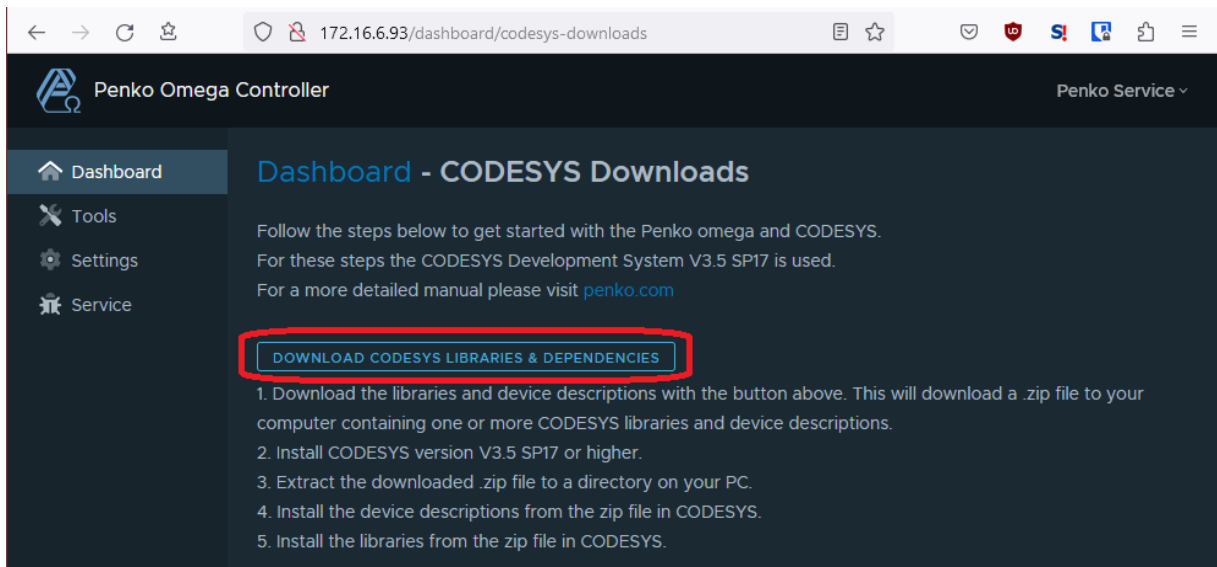


Figure 2. Download CodeSys libraries and dependencies

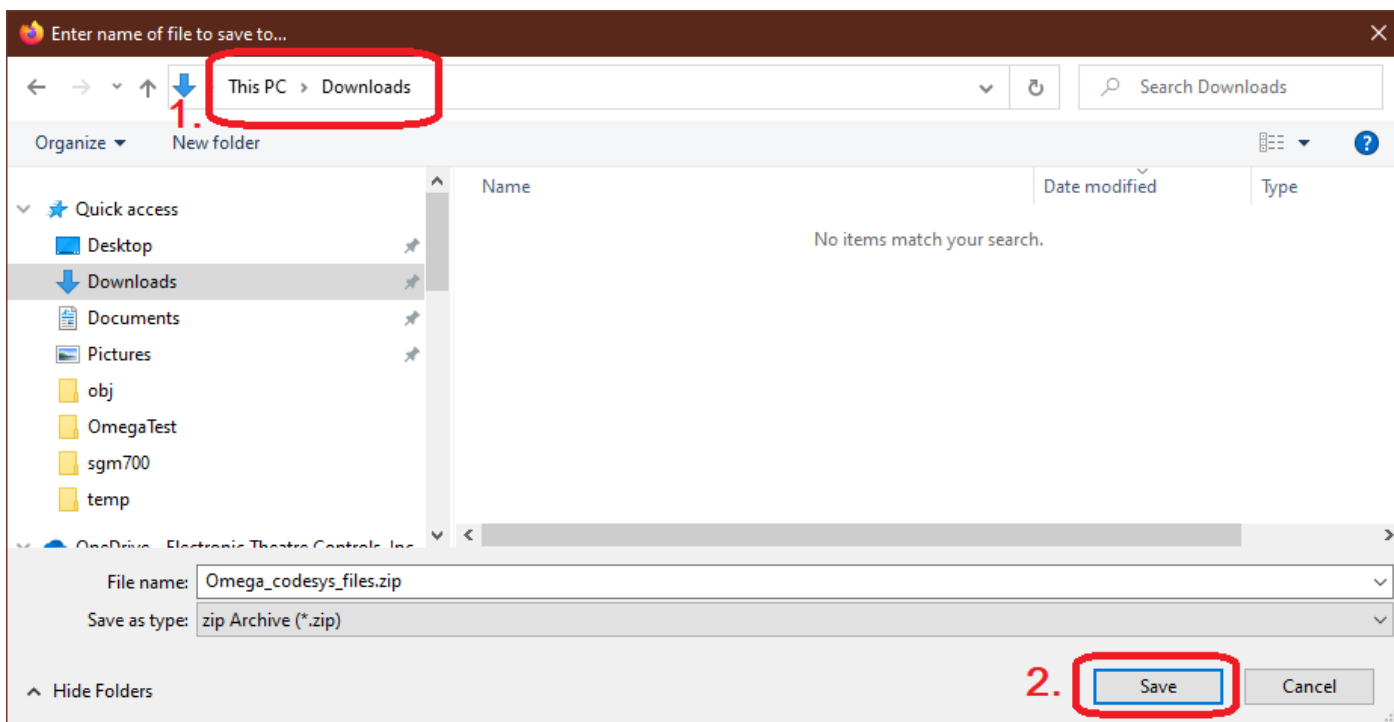


Figure 3. Save the archive with CodeSys files.

Finally, in your file browser, navigate to the location where the archive was stored and extract it (Figure 4).

Penko EasyWeigher manual

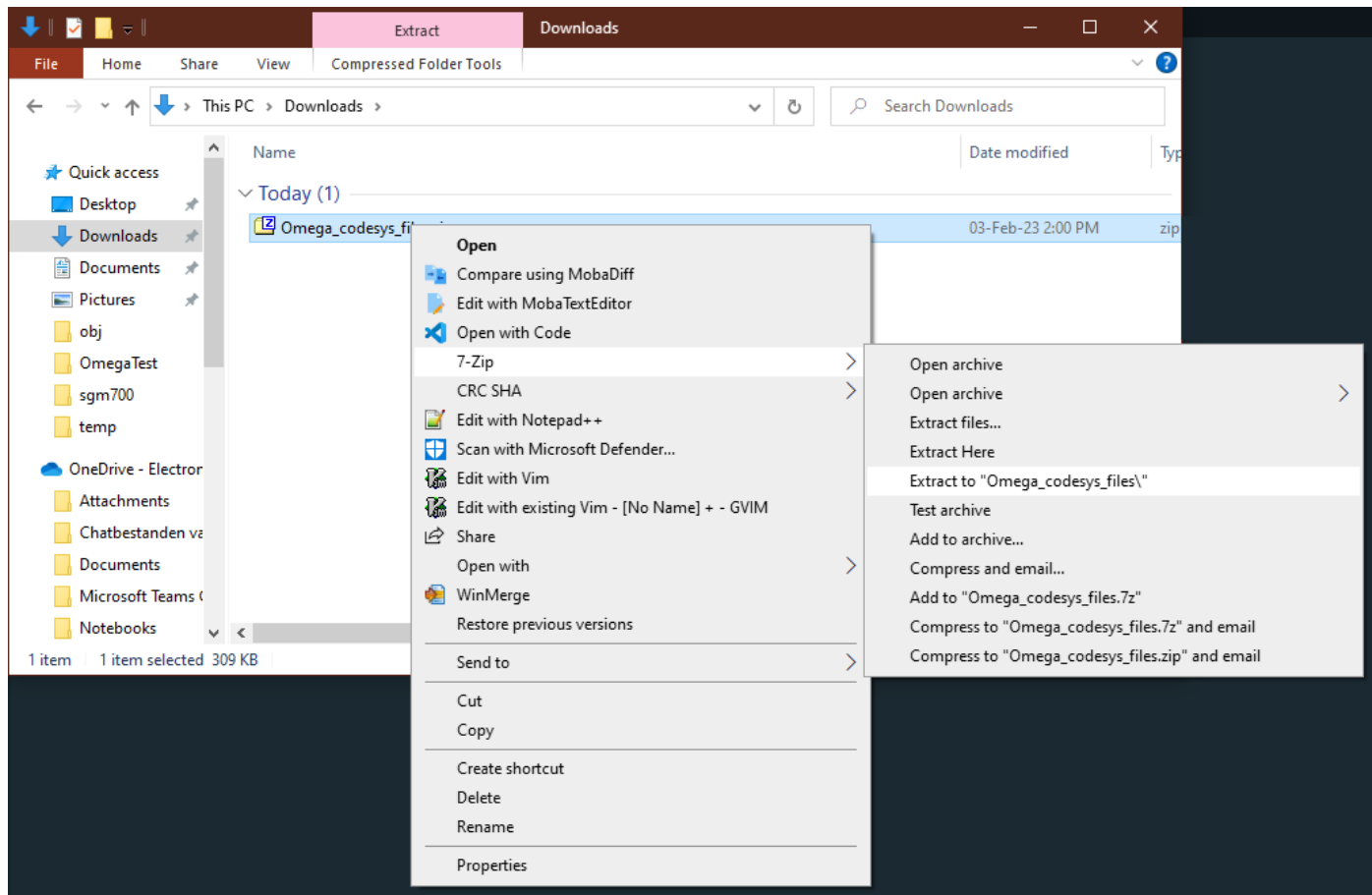


Figure 4. Navigate to the archive and extract it

Penko EasyWeigher manual

STEP 2: PREPARE THE DEVELOPMENT ENVIRONMENT

Install the CodeSys environment as downloaded from the CodeSys store onto your PC. This is needed to start writing program code for the Omega CodeSys PLC.

A. OPEN THE ENVIRONMENT

Open your installed CodeSys IDE (programming environment) version.



B. INSTALL THE OMEGA DEVICE DESCRIPTIONS IN THE DEVICE REPOSITORY

In the CodeSys IDE, navigate to 'Tools' in the top bar, then 'Device Repository', and click 'Install' (Figure 5). Navigate to the location where the archive of Step 1 was extracted, and enter the directories 'Omega_codesys_files' and 'codesys_devicedescriptions'. Select all files and click 'Open' (Figure 6).

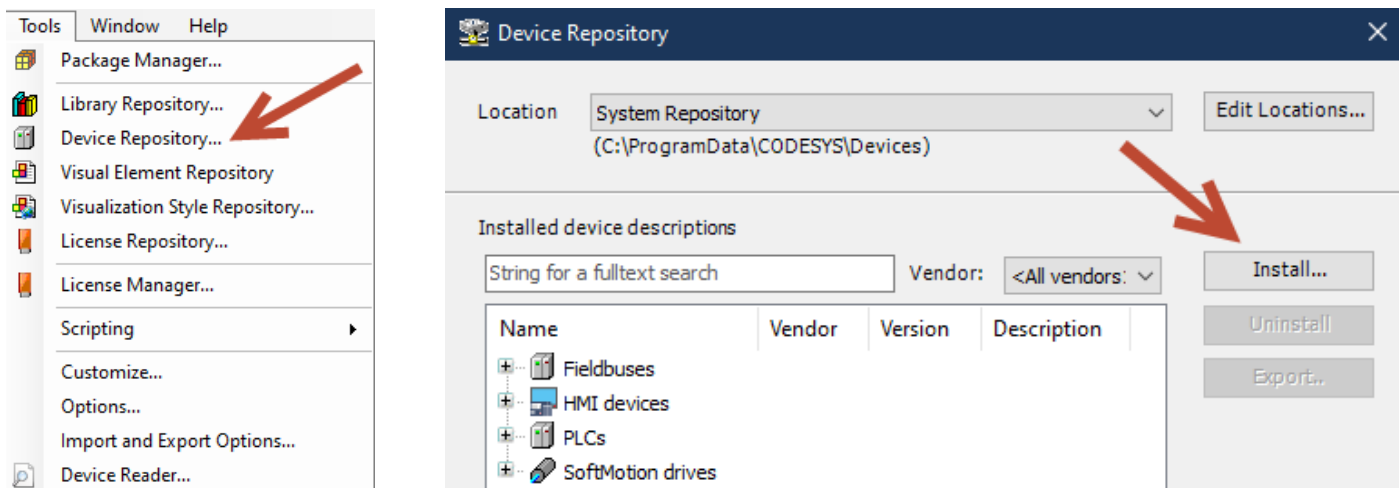


Figure 5

Penko EasyWeigher manual

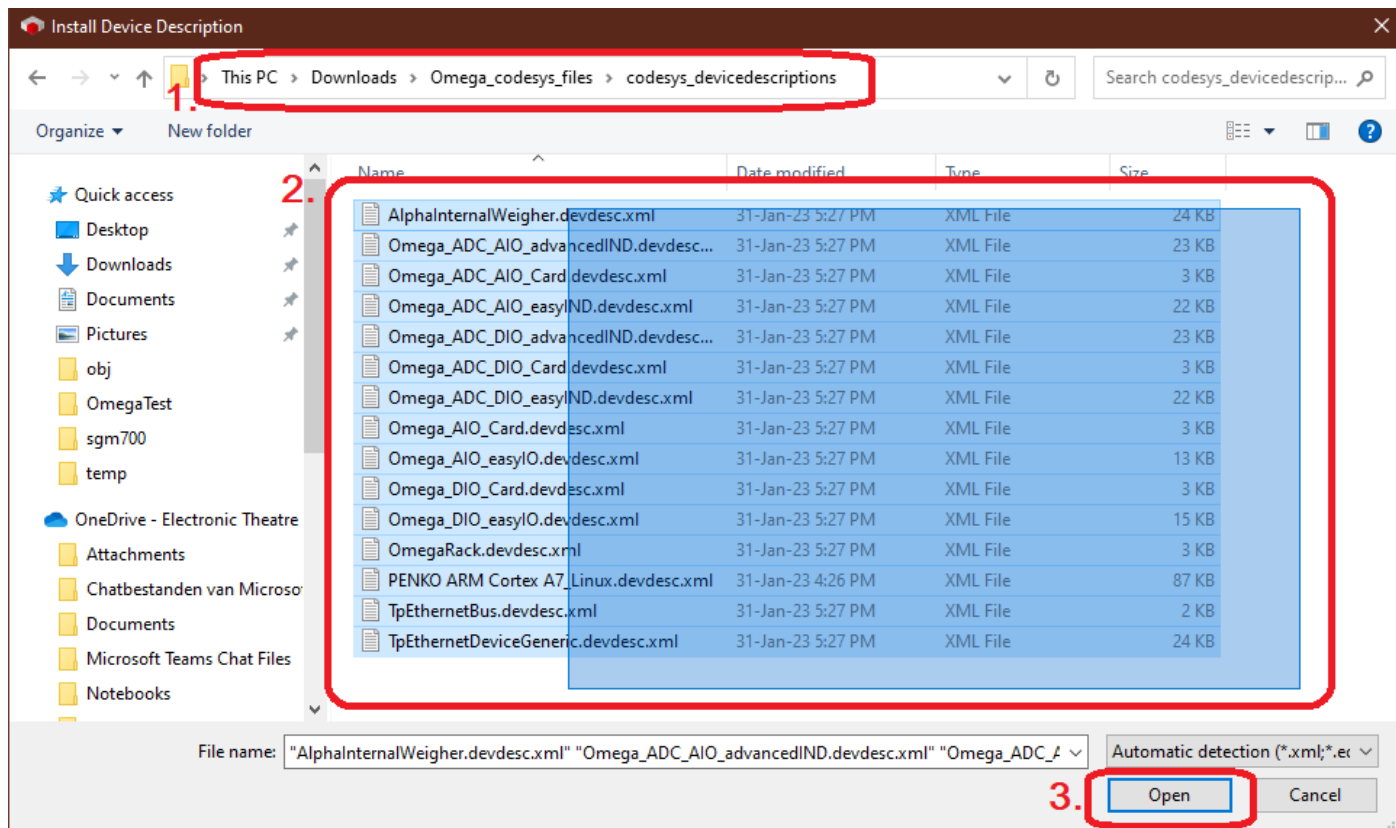


Figure 6

Now the Omega has been added as a device to the CodeSys IDE.

Close this window to go back to the main screen.

C. INSTALL THE CODESYS LIBRARIES

The CodeSys libraries for the Omega can be installed in a similar fashion. Navigate via the 'Tools' menu to the Library Repository, and click 'Install' to select the libraries to install. See Figure 7 below.

Penko EasyWeigher manual

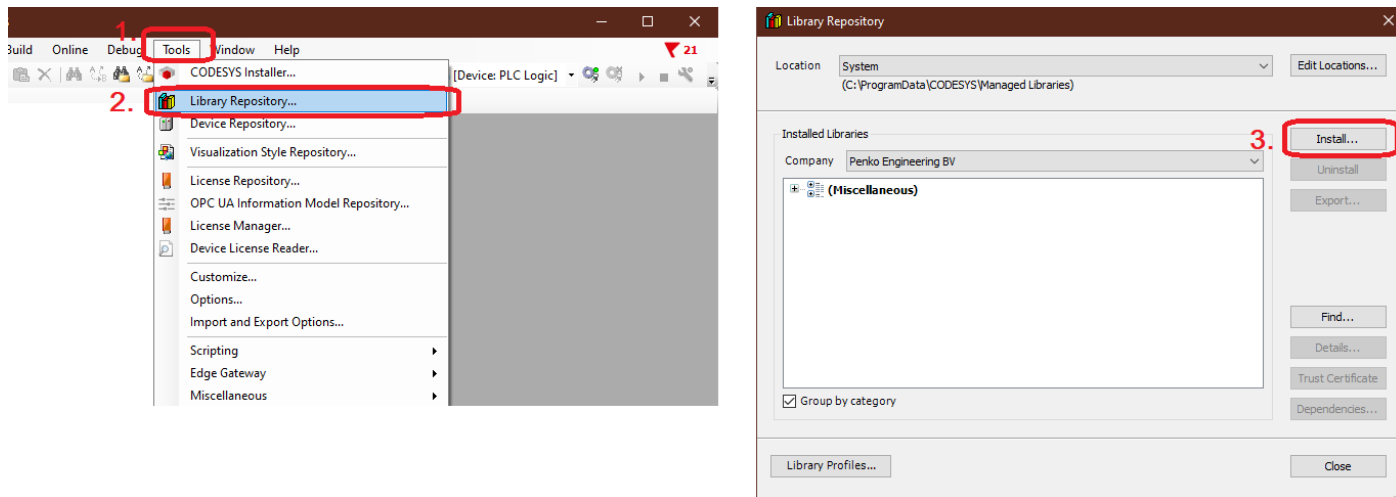


Figure 7. Open the Library Repository.

Navigate to the location of the extracted archive and enter the 'codesys_libraries' directory. Select all files in the folder and click 'Open' (Figure 8).

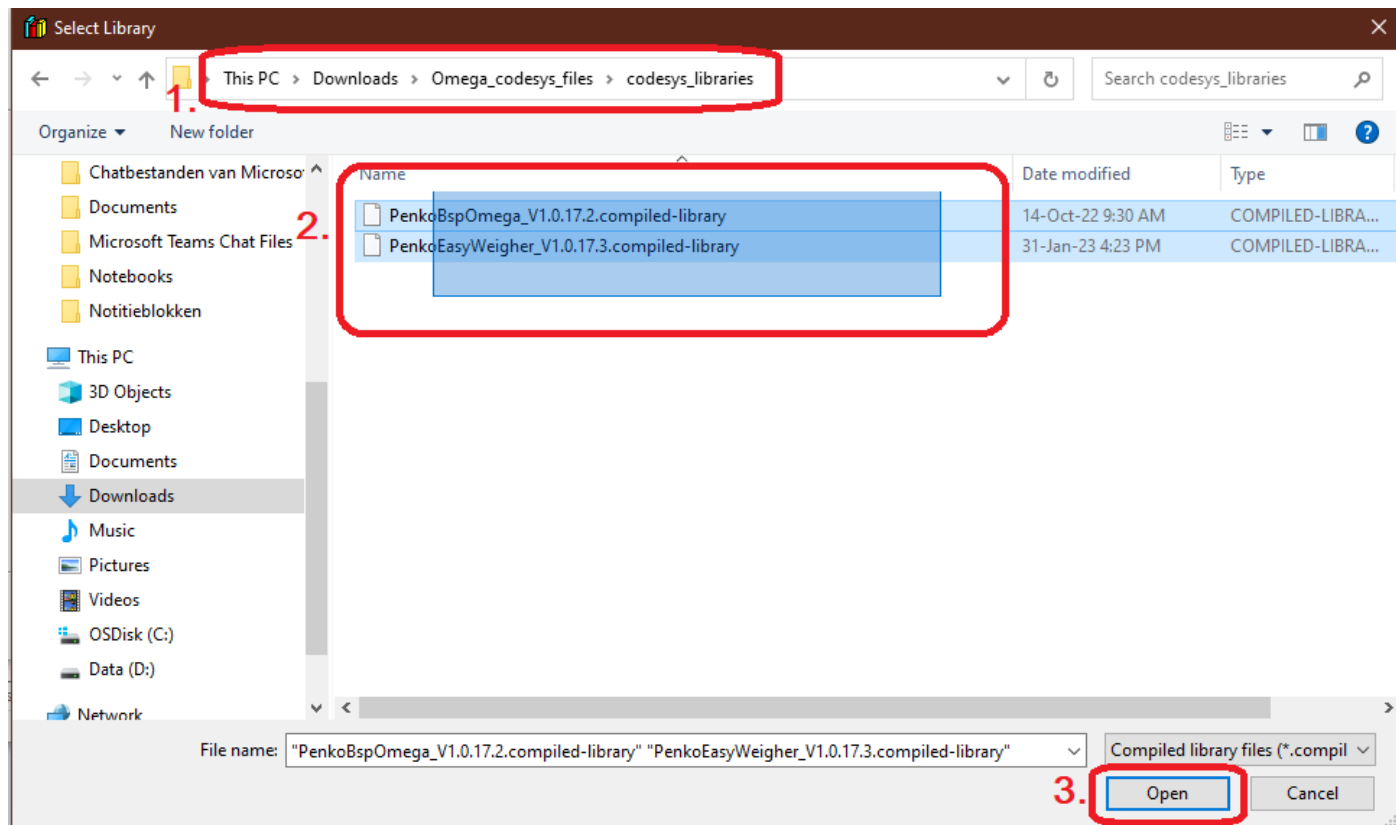


Figure 8. Install the libraries from the 'codesys_libraries' directory.


You have now installed the necessary device descriptions and libraries to start a first CodeSys project for the Omega.

Penko EasyWeigher manual



CHAPTER 4. CREATE YOUR FIRST APPLICATION

STEP 1: CREATING A NEW PROJECT

A. CREATE A NEW PROJECT

 CODESYS V3.5 SP15 Patch 2

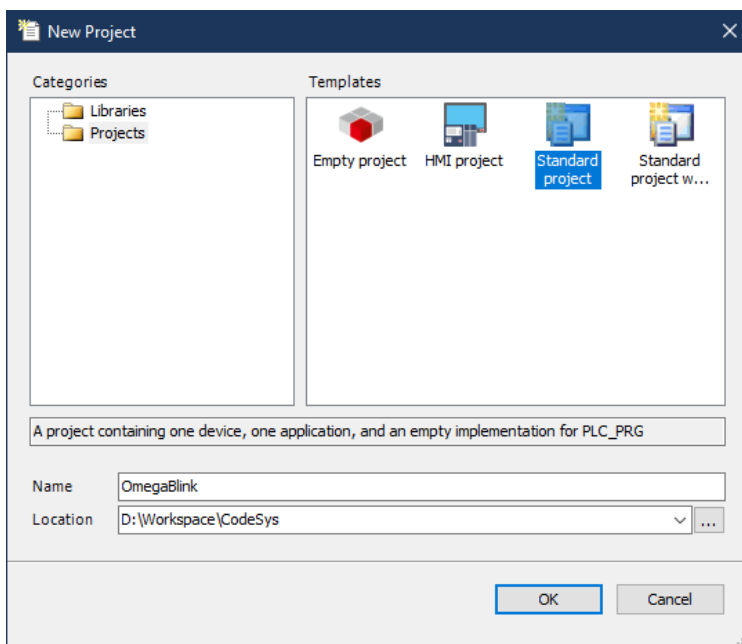
Basic operations

-  New Project... 
-  Open Project...
-  Open Project from PLC...

Recent projects

B. SELECT THE PROJECT TYPE

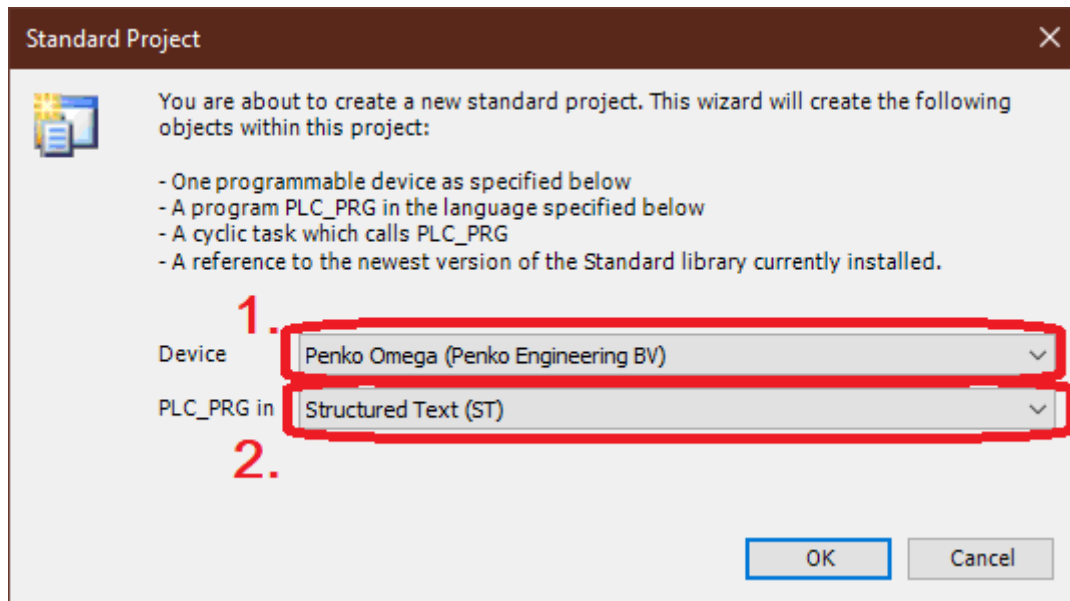
For this tutorial we use the “standard project”. Select this option and fill-in an appropriate project name and folder location.



C. SELECT THE OMEGA DEVICE

Select the Penko Omega device and keep the PLC_PRG at “Structured Text (ST)”.

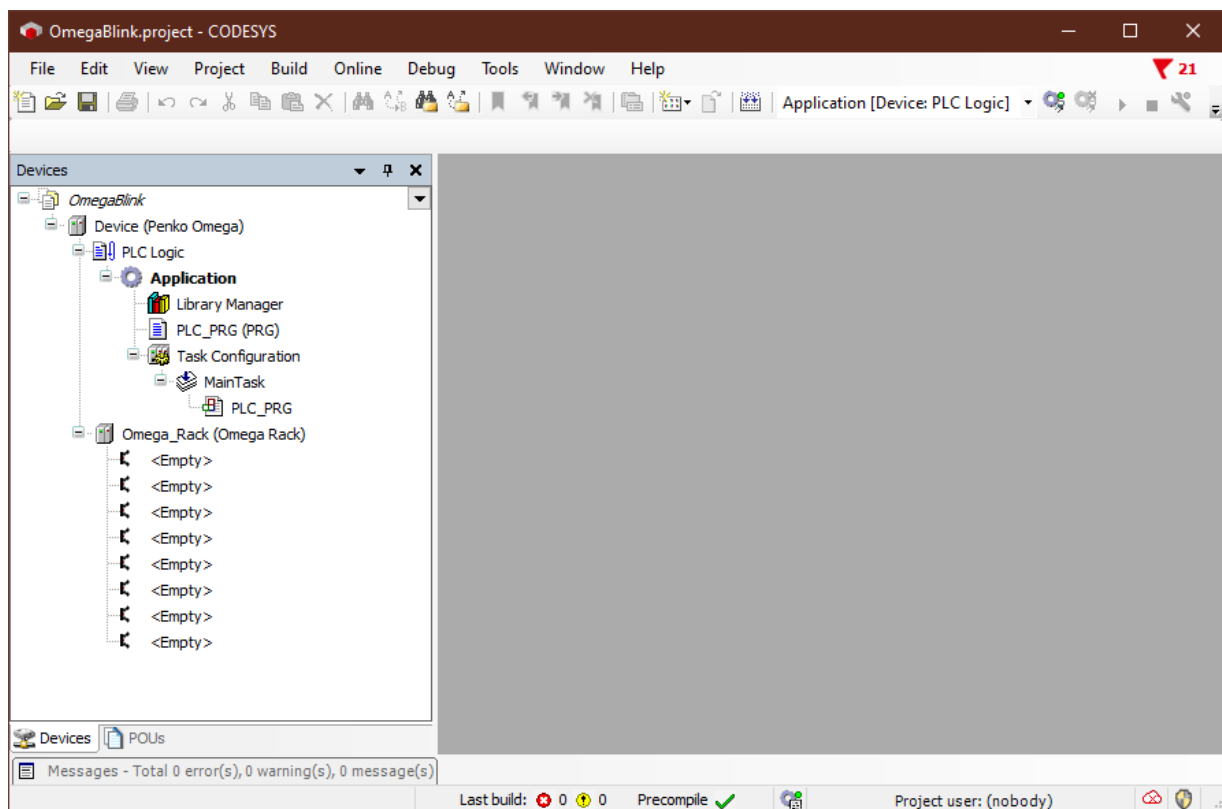
Penko EasyWeigher manual



STEP 2: WRITE YOUR FIRST APPLICATION

After preparing the environment, the project will open in an Integrated Development Environment (IDE) where the application can be created. Below the initial screen with several items added by default. The menu bar on the left gives easy access to everything needed to create your program.

'Application' contains your libraries and your code. 'Omega_Rack' allows you to access the peripheral cards of the Omega, but we will ignore those in this example. These are used in the EasyWeigher example later in this manual.



Penko EasyWeigher manual



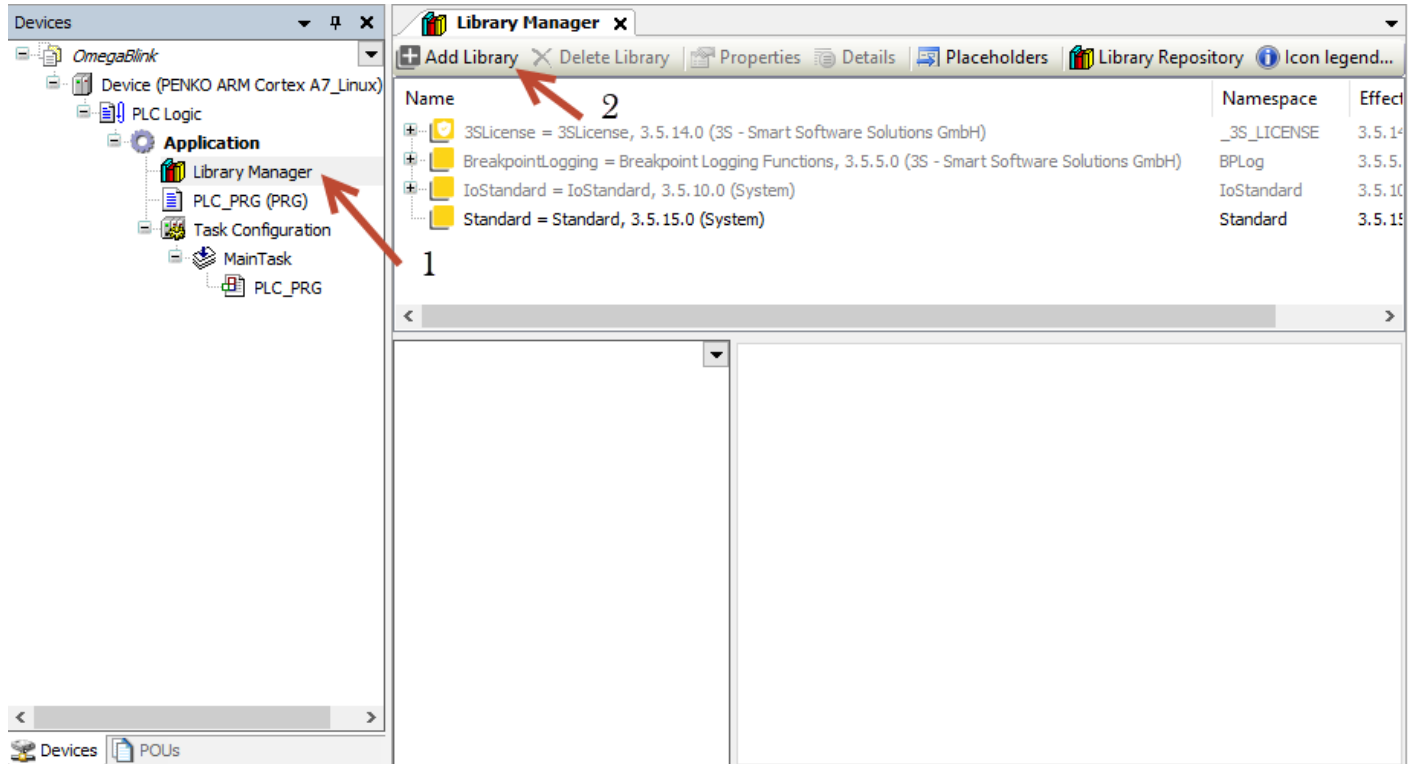
PENKO

an ETC Company

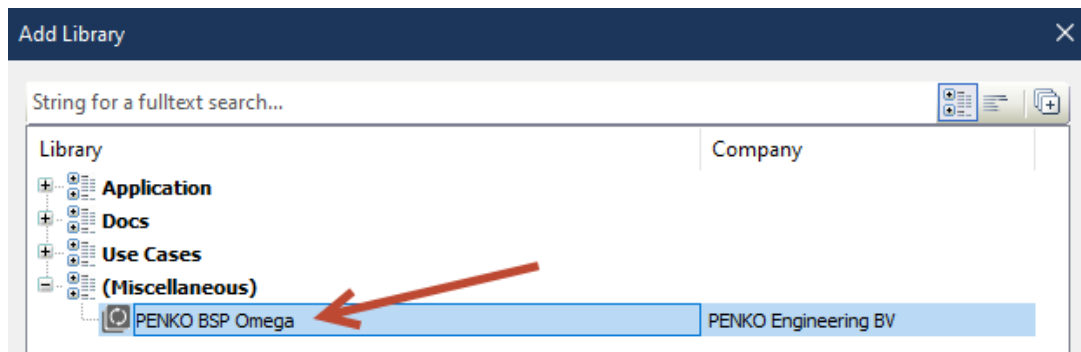
Penko EasyWeigher manual

A. ADD THE INSTALLED LIBRARY TO THE CURRENT PROJECT

In the previous step we installed the PenkoBspOmega library in CodeSys. In this step, we will add the installed library to our project. This library allows your program to access the leds on the CPU module. To add the library to the project, go to the library manager (1) and hit the “Add library” button (2).



Select under the “(Miscellaneous)” category the “Penko BSP Omega” library.

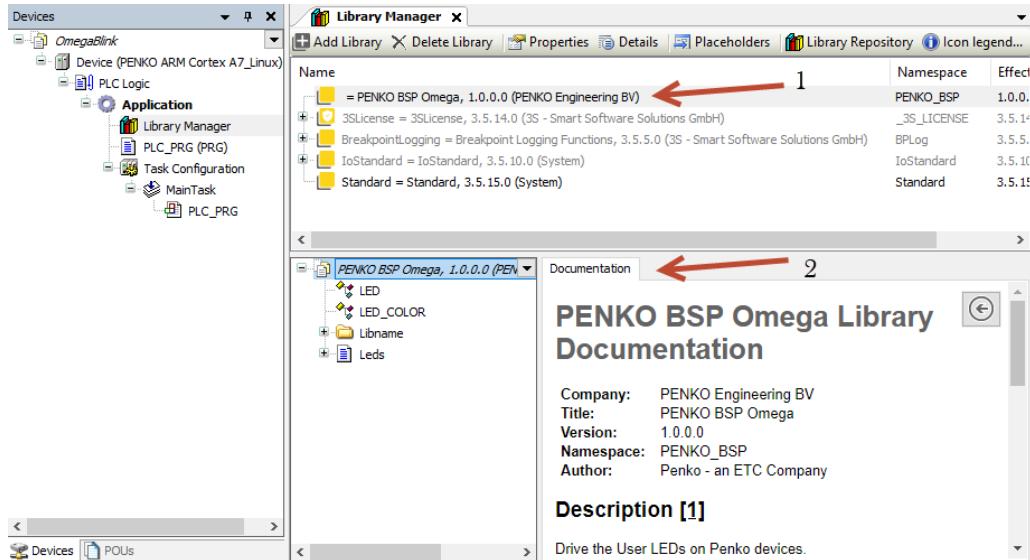


Penko EasyWeigher manual

B. LIBRARY DOCUMENTATION

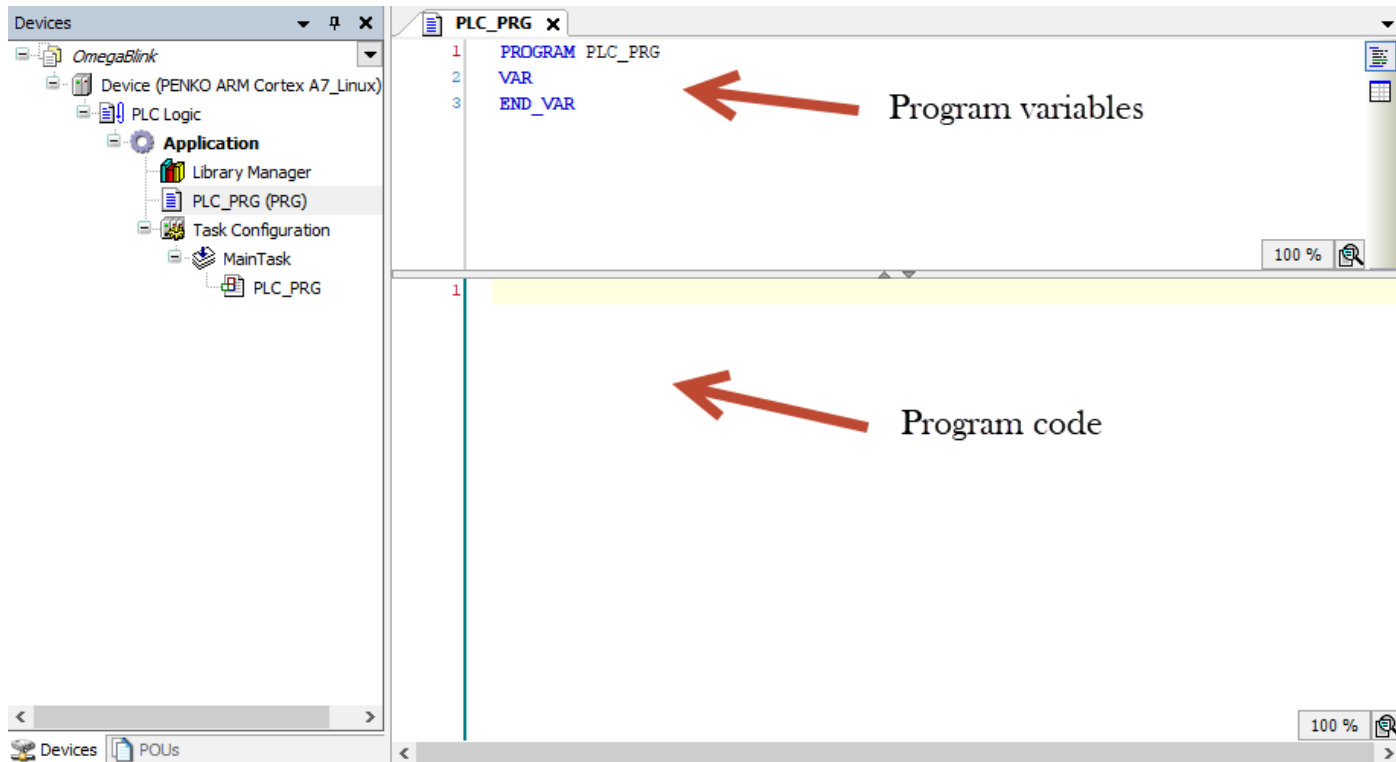
Now the Penko BSP Omega library has been added to the project (1).

To view all the library functions, see the included documentation (2).



C. SELECT PLC_PRG (PRG)

PLC_PRG is the default name for new programs and selected under the Main Task to be executed. To develop your application, there are two important areas. The bottom part where the program code is constructed. The top part is used to set the program name and to define the program's variables between VAR and END_VAR.



Penko EasyWeigher manual

D. WRITE A BLINK PROGRAM: EXAMPLE 1

All the code below can be found as text in Appendix I: Blink program example 1. The available LED control functions are described in the included library documentation as mentioned before in the library manager.

First, we start by defining some variables in the top area. These variables are used to hold the state of the LED's and by changing the value we can change the LED color, turn a LED on or off and control each available LED.

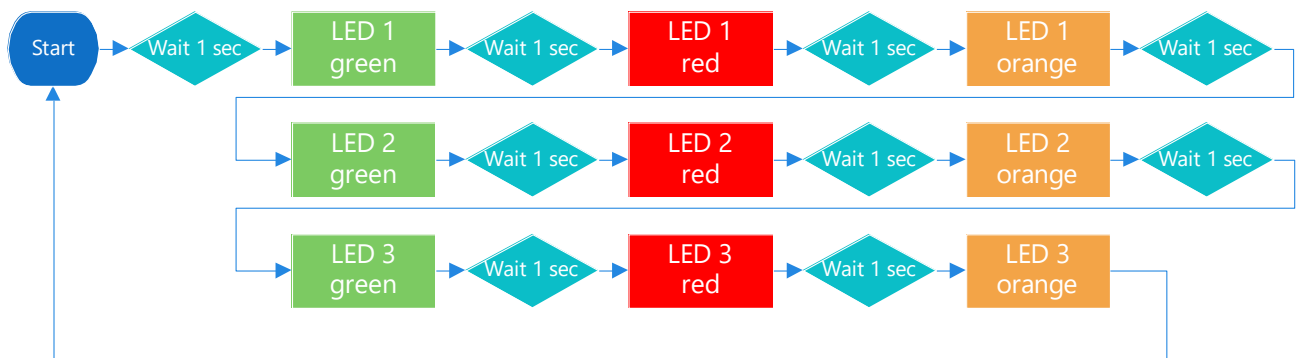
All text on a line after '//' is a comment. It is not part of the code but used to document the program.

```
Library Manager | PLC_PRG x
1 PROGRAM PLC_PRG
2 VAR
3     leds: PENKO_BSP.Leds; // Get the LED's from the library
4     Delay: TON; // Time to determine blink rate
5     color_ctr: INT := PENKO_BSP.LED_COLOR.GREEN; // Start at color green
6     led_ctr: INT; // For selecting the LED (LED1, LED2, LED3, LED4)
7 END_VAR
```

After defining the variables, we start writing some program code in the lower area to control the LED's. In the code below LED4 is always turned on in the color red. The LED's, 1, 2 and 3 change color one after the other in a fixed sequence from green to red to orange. To turn off the previous LED's the led.setToBits(0) is called.

```
1 Delay(IN:=TRUE, PT:=T#1S); // Turn on the timer and set it to one second
2 IF NOT(Delay.Q) THEN // Wait till the timer has reached one second
3     RETURN; // Timer not at one sec? Don't execute the rest of the code below
4 END_IF
5 Delay(IN:=FALSE); // Turn off the timer
6 leds.setToBits(0); // Turn off all LED's
7
8 leds.setColor(PENKO_BSP.LED.LED4, PENKO_BSP.LED_COLOR.RED); // Always turn on LED4 in red
9
10 leds.setColor(led_ctr, color_ctr); // Set the LED number and color
11 color_ctr := color_ctr + 1; // Change color
12
13 IF color_ctr = PENKO_BSP.LED_COLOR.NUM THEN
14     color_ctr := PENKO_BSP.LED_COLOR.GREEN; // Go back to green
15     led_ctr := led_ctr + 1; // go to the next LED
16     IF led_ctr = PENKO_BSP.LED.LED4 THEN
17         led_ctr := PENKO_BSP.LED.LED1; // Go back to LED1
18     END_IF
19 END_IF
```

Below the flowchart of the above program.

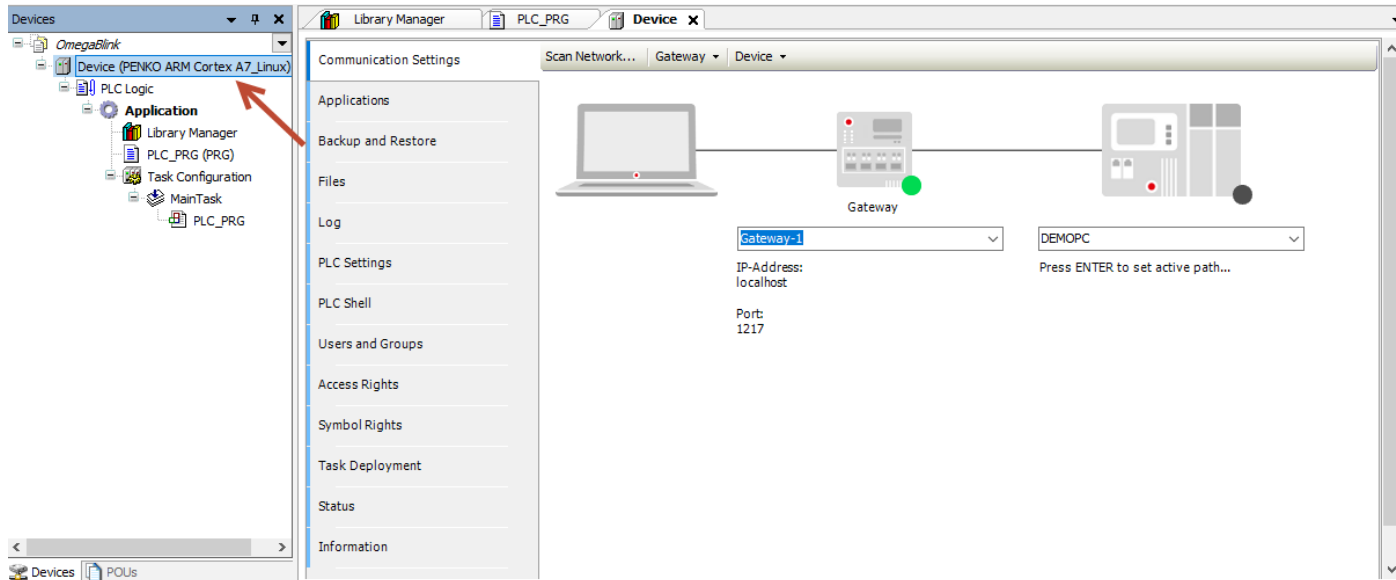


Penko EasyWeigher manual

STEP 3: RUN YOUR PROGRAM

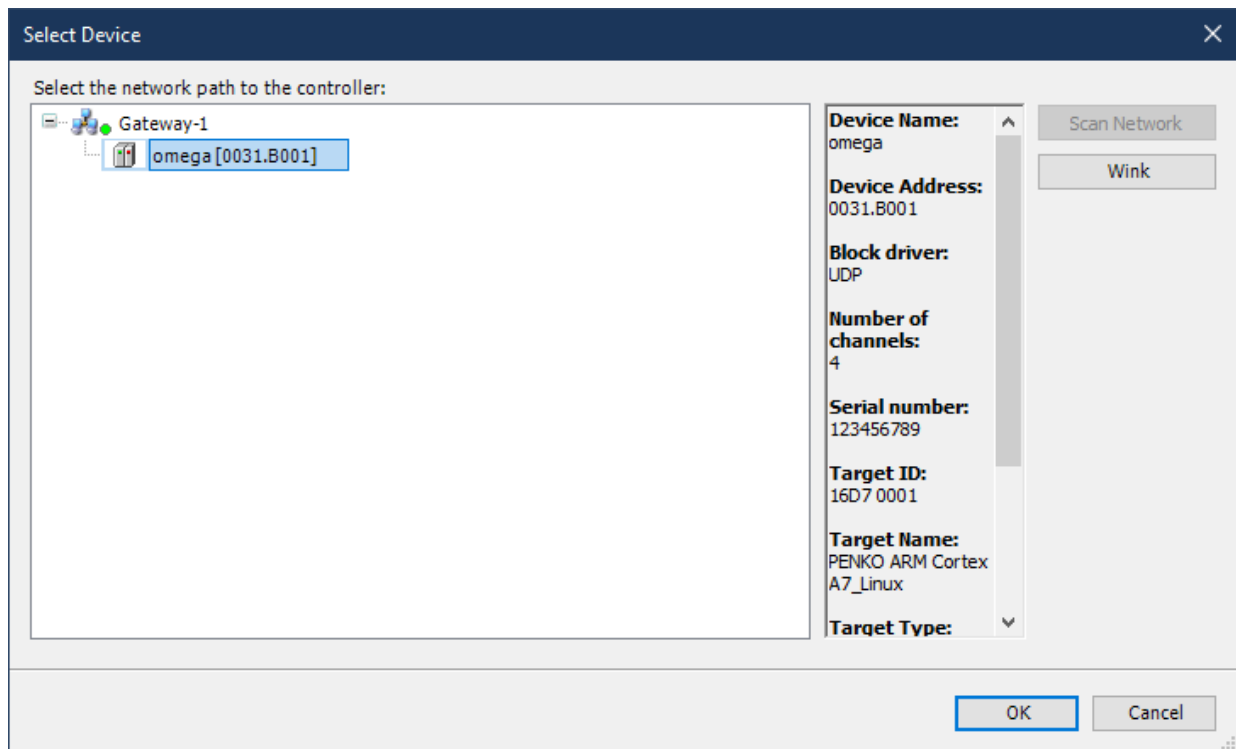
After writing the program code, go to the device item. In the next steps, the code from example 1 is used.

A. SELECT THE OMEGA DEVICE



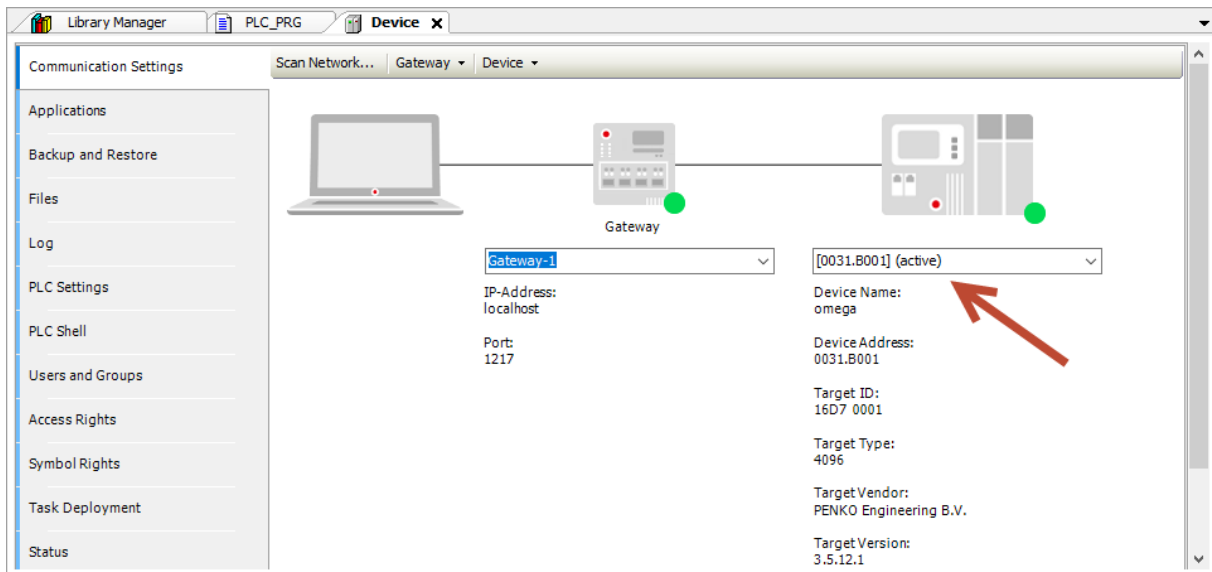
The gateway status circle is already green.

Now hit the “Scan network” button to find the Omega device in your network.



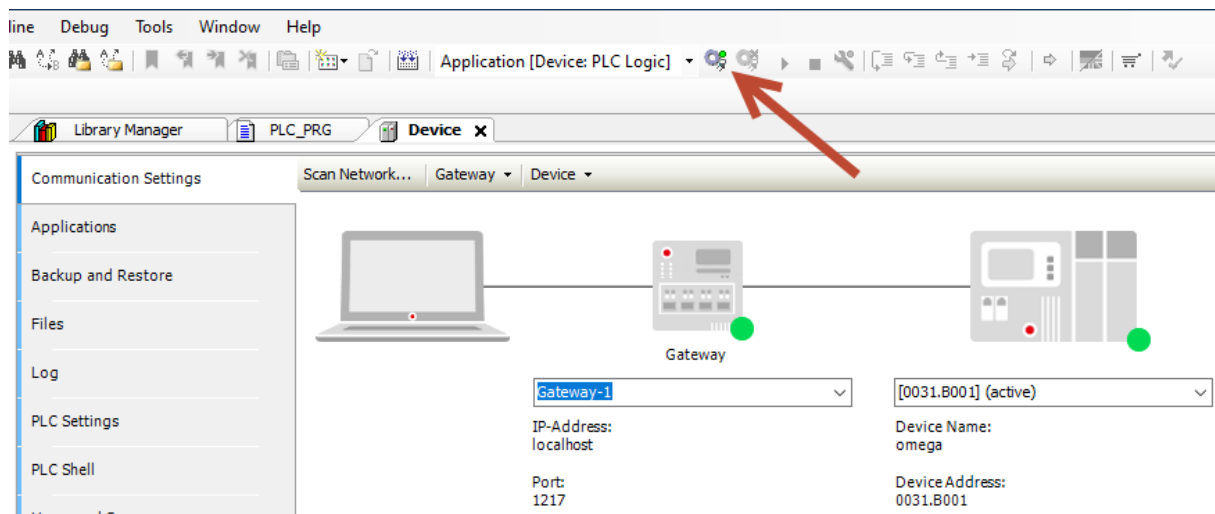
Penko EasyWeigher manual

Now the omega device is selected.

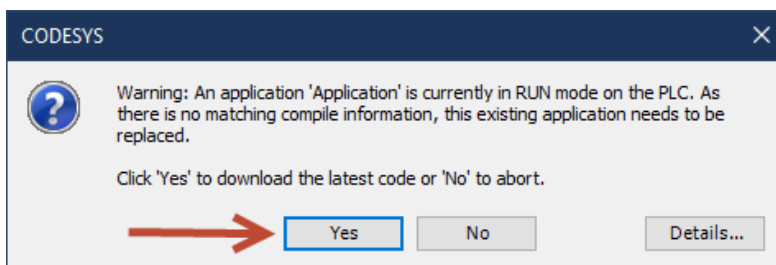


B. DOWNLOAD THE PROGRAM TO THE DEVICE

Hit the "login" button to download the program to the device.

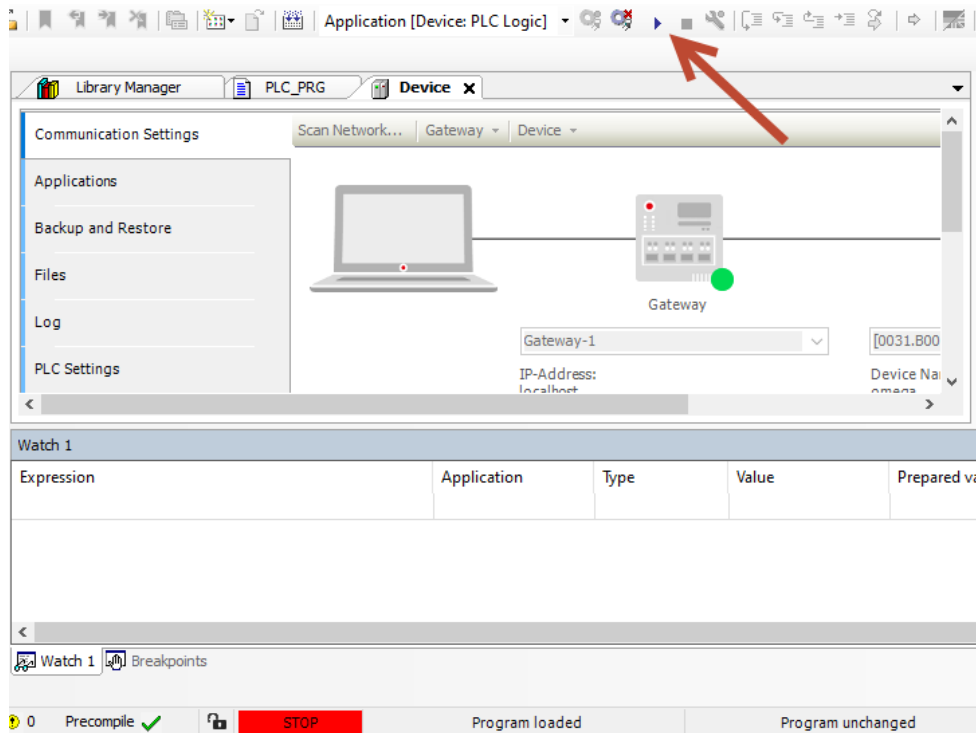


A pop-up appears, hit "Yes".



Penko EasyWeigher manual

After the program has been downloaded, start the program by clicking on the start symbol.



C. PROGRAM IN RUNNING MODE

Go back to the PLC_PRG to see if the program is running. The values behind the color_ctr and led_ctr are changing and the LED's on the Omega CPU card are changing color every second.

Expression	Type	Value	Prepared value	Address	Comment
leds	PENKO_BSP.Leds				Get the LED's from the library
Delay	TON				Time to determine blink rate
color_ctr	INT	4			Start at color green
led_ctr	INT	1			For selecting the LED ...1, LED2, LED3, LED4)

```

1  Delay(IN TRUE:=TRUE, PT T#1s :=T#1S); // Turn on the timer and set it to one second
2  IF NOT(Delay.QFALSE) THEN           // Wait till the timer has reached one second
3      RETURN;                          // Timer not at one sec? Don't execute the rest of the code below
4  END_IF
5  Delay(IN TRUE:=FALSE);              // Turn off the timer
6  leds.setToBits(0);                  // Turn off all LED's
7
8  leds.setColor(PENKO_BSP.LED.LED4, PENKO_BSP.LED_COLOR.RED); // Always turn on LED4 in red
9
10 leds.setColor(led_ctr 1, color_ctr 4); // Set the LED number and color
11 color_ctr 4 := color_ctr 4 + 1;      // Change color
12
13 IF color_ctr 4 = PENKO_BSP.LED_COLOR.NUMGREEN 5 THEN
14     color_ctr 4 := PENKO_BSP.LED_COLOR.GREEN 2; // Go back to green
15     led_ctr 1 := led_ctr 1 + 1;           // go to the next LED
16 IF led_ctr 1 = PENKO_BSP.LED.LED4 3 THEN
17     led_ctr 1 := PENKO_BSP.LED.LED1 0; // Go back to LED1
18 END_IF
19 END_IF RETURN
    
```

Penko EasyWeigher manual

Penko EasyWeigher manual

CHAPTER 5. WRITE A BLINK PROGRAM: EXAMPLE 2

All the code below can be found as text in Appendix II: Blink program example 2. The available LED control functions are described in the included library documentation as mentioned before in the library manager.

Repeat Step 2 to create a new project.

At step 3 at chapter D write the following program:

Again, we start by defining some variables for controlling the LED's. These variables are used to change the LED color, turn a LED on or off and control each available LED.

```
Library Manager | PLC_PRG x | Device
1 PROGRAM PLC_PRG
2 VAR
3     leds: PENKO_BSP.Leds; // Get the LED's from the library
4     Delay: TON; // Time to determine blink rate
5     state: UINT := 0; // Start at 0 (all LED's off)
6 END_VAR
```

In the code below, multiple LED's are turned on by using the setOnMask() function. The similar function setMaskOff can be used to turn off multiple LED's. The code loops throw all the 16 different states (0 to 15) to show the output for each state. The output is also described in the table below the code.

```
1 Delay(IN:=TRUE, PT:=T#500MS); // Turn on the timer and set it to one second
2 IF NOT(Delay.Q) THEN // Wait till the timer has reached one second
3     RETURN; // Timer not at one sec? Don't execute the rest of the code below
4 END_IF
5 Delay(IN:=FALSE); // Turn off the timer
6 leds.setToBits(0); // Turn off all LED's
7
8 leds.setOnMasked(state); // Turn one or more LED on at once
9 state := state + 1; // Go to the next state
10
11 IF state > 15 THEN // If all 15 states reached
12     state := 1; // Go back to only the first LED on
13 END_IF
```

State	LED1	LED2	LED3	LED4
0	OFF	OFF	OFF	OFF
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	ON	ON	OFF	OFF
4	OFF	OFF	ON	OFF
5	ON	OFF	ON	OFF
6	OFF	ON	ON	OFF
7	ON	ON	ON	OFF
8	OFF	OFF	OFF	ON
9	ON	OFF	OFF	ON
10	OFF	ON	OFF	ON
11	ON	ON	OFF	ON
12	OFF	OFF	ON	ON
13	ON	OFF	ON	ON
14	OFF	ON	ON	ON
15	ON	ON	ON	ON

Penko EasyWeigher manual

Finally, repeat step 4 to run the program.

Penko EasyWeigher manual

CHAPTER 6. EASYWEIGHER: FIRST APPLICATION

The Penko EasyWeigher library for CodeSys provides an easy way to interface with the peripheral modules of the Omega.

STEP 1. PREREQUISITES

Before setting up an example project, please make sure that your Omega is setup correctly. For more information on how to do this, and for all connection diagrams, refer to the Omega manual (7600M1082-<LANGUAGE>-R8 MANUAL OMEGA.pdf). Please make sure that:

1. You can navigate to the Omega web portal in your browser via its IP address.
2. The first slot in the rack is occupied by an Omega ADC DIO module. It does not matter what is plugged into the other slots.
3. A load cell or load cell simulator is attached to the channel 1 load cell connector of the Omega ADC DIO module.

STEP 2. CREATE THE PROJECT

When you have confirmed that your Omega is online and its web portal is reachable in your browser, start CODESYS and create a new 'Standard project'.

After clicking "OK", CODESYS asks you to select the device this project is for, and the programming language you'll be working in. Follow Figure 9 to select the "Penko Omega (Penko Engineering BV)" device. Then, select the programming language. In your own future projects you can use what you are most comfortable with, but this example project will use the Function Block Diagram language (Figure 10)

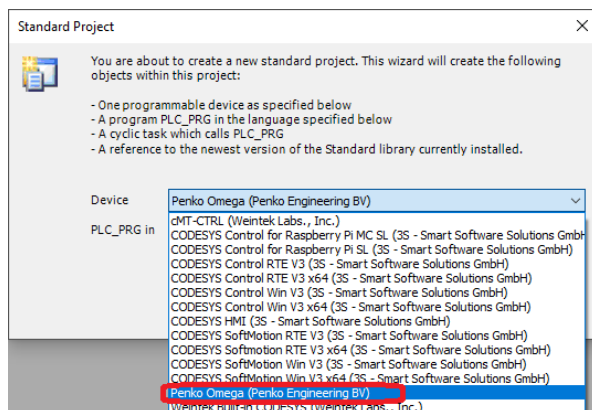


Figure 9. Select Penko Omega as device

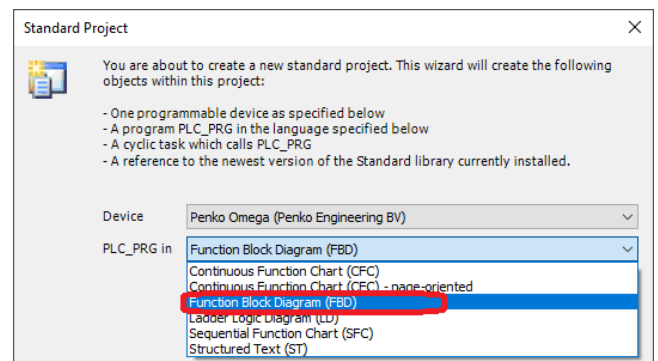


Figure 10. Select FBD as programming language

When the device and language are configured, click OK. You should now see an empty project for a Penko Omega device, with an "Omega_Rack" as attached device as in Figure 11.

Penko EasyWeigher manual

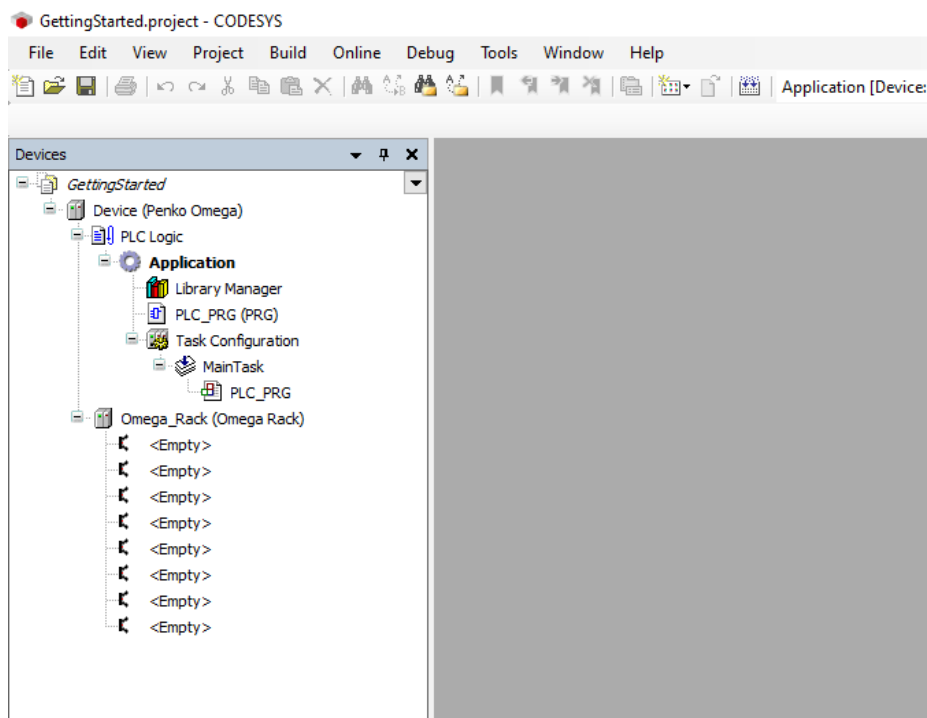


Figure 11. An Omega starter project.

Add an Omega ADC DIO module to the Omega Rack

Let CODESYS know about your Omega ADC DIO module by right-clicking the first slot in the Omega Rack, and select “Plug Device” (Figure 12). You should see four options in the popup window (Figure 13), one of which is the Omega ADC DIO Card. Select this and click “Plug Device” in the right-bottom corner. After CODESYS has added the card, click “Close”.

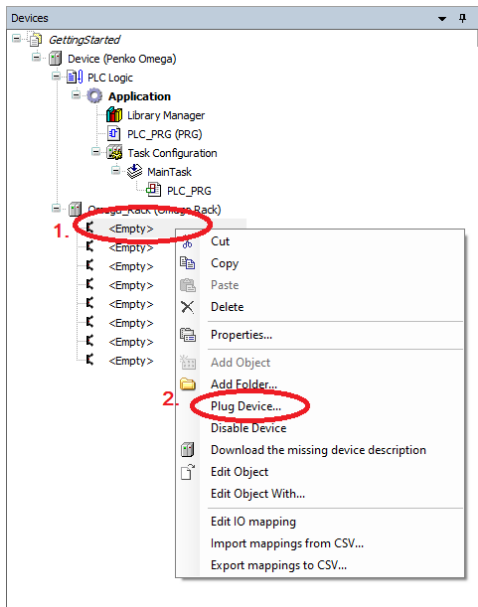


Figure 12. Plug a device in the first slot.

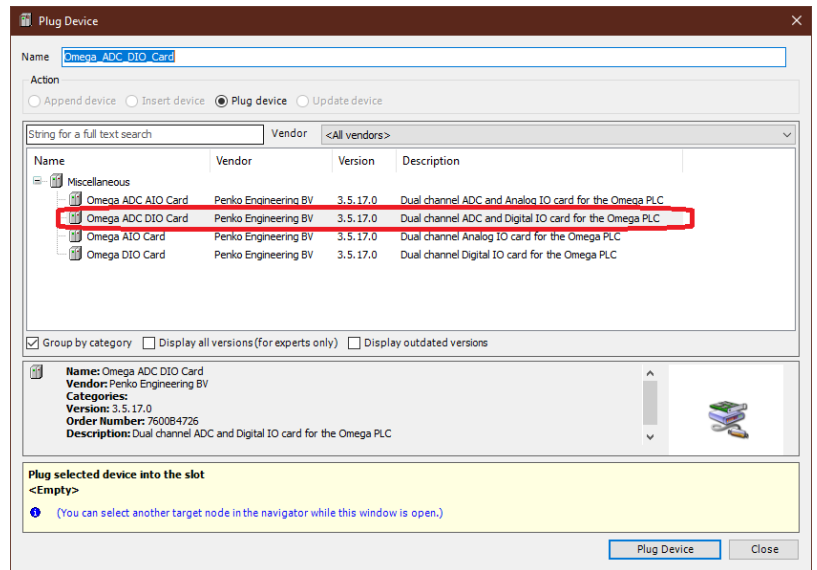


Figure 13. Select 'Omega ADC DIO Card'

Two new devices have been added of type "Omega ADC DIO easyIND" below the plugged device, one for each channel in the card.

STEP 3. EXAMPLE IO MAPPING

Create a new variable in PLC_PRG called "rNetWeight" of type "std:REAL":

```
PROGRAM PLC_PRG
VAR
    rNetWeight : REAL;
END_VAR
```

Then map this variable to the "Net" parameter of the first channel. To do this, double click the first channel, navigate to the "Mapping" tab and open the "Weights" folder. Click the three dots to open a window where you can select the variable to map (Figure 14). Then follow Figure 15 to add a mapping to "rNetWeight".

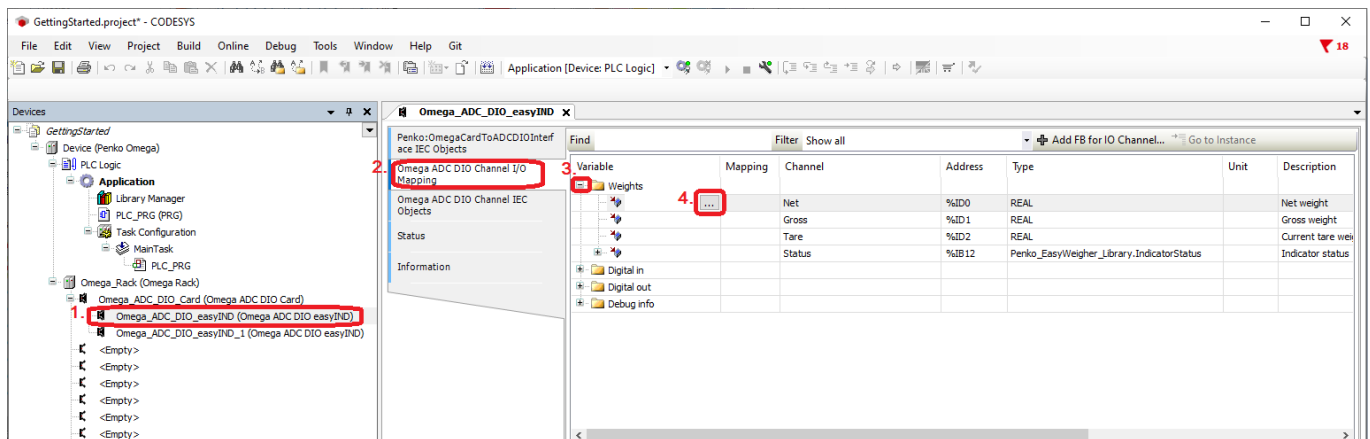


Figure 14. Navigate to IO Mapping.

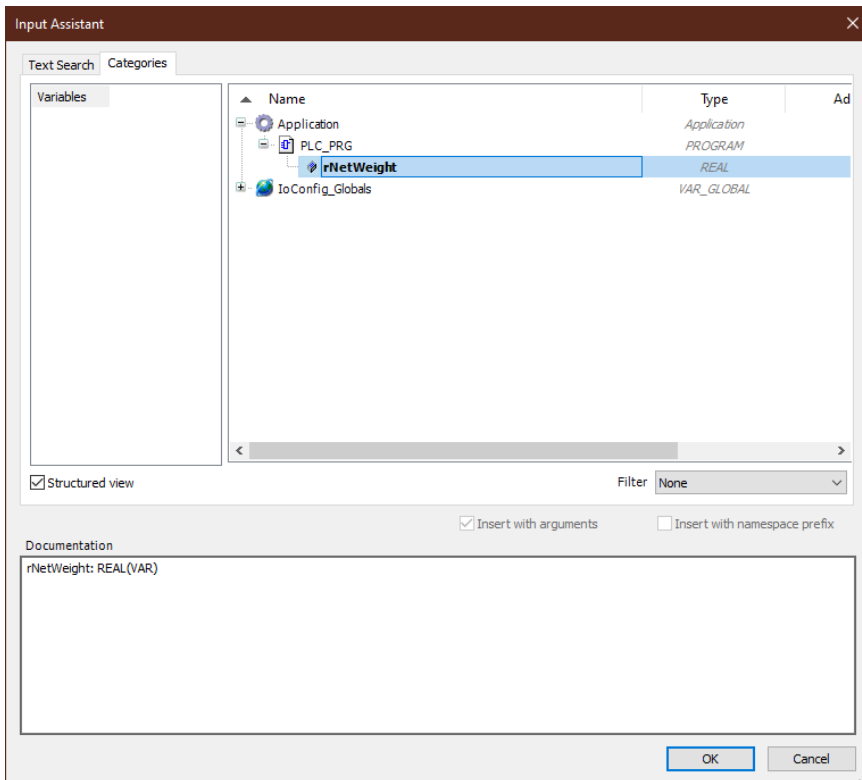


Figure 15. Map rNetWeight to the IO Channel.

The “rNetWeight” variable is now mapped to the “Net” IO channel of Channel 1 of the Omega ADC DIO Card. There is one more thing to do though:
 If we would upload this, CODESYS will notice that “rNetWeight” is not used in the code, and optimize its access away. To prevent this, force CODESYS to always update all mapped variables by selecting “Enabled 1” for “Always update variables”. See Figure 16 for how to do this.

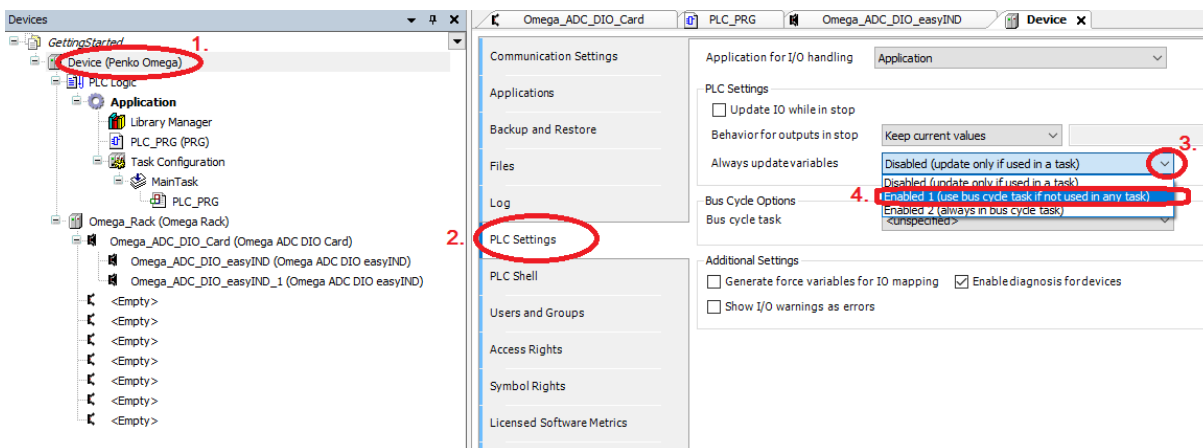


Figure 16. Tell CODESYS to always update all variables.

Now we’re ready to login and upload. Refer to “Step 3: Run your program” in Chapter 4 on how to upload your program and view variables while debugging. Changing the weight settings on the load cell or load cell simulator should be visible in the rNetWeight variable.

STEP 4: OPTIONAL: ADD A VISUALIZATION

The next step is to add a basic visualization, mainly as a preparation for the next chapter. If you are familiar with Codesys visualizations, you can skip this chapter.



To start, right-click on 'Application', select 'Add Object' and then 'Visualization'. A new window pops up. Make sure to activate 'VisuSymbols', and click 'Add' (Figure 17).

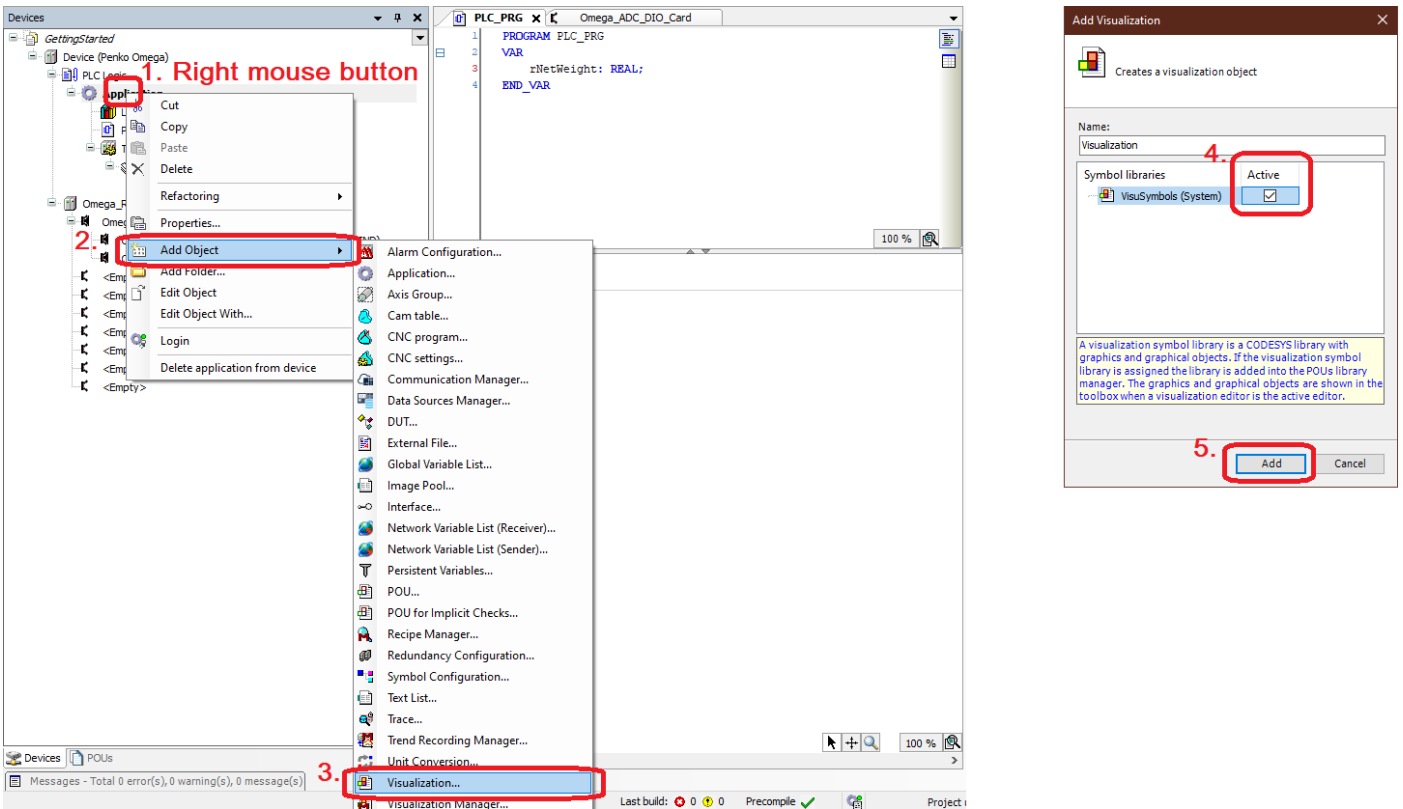


Figure 17. Add a visualization

CodeSys may hang for a bit while it adds all visualization objects. After some time, the information tree at the left side in the IDE will show that a VISU_TASK has been added below *Task Configuration*, and a Visualization Manager and Visualization have been added to *Application*. Add a Textfield to the visualization by opening the Visualization on the left of the screen (double click), opening the Visualization Toolbox in the bottom right, selecting the Common Controls visualization elements, and dragging the Textfield to the visualization window as shown in Figure 18 below.

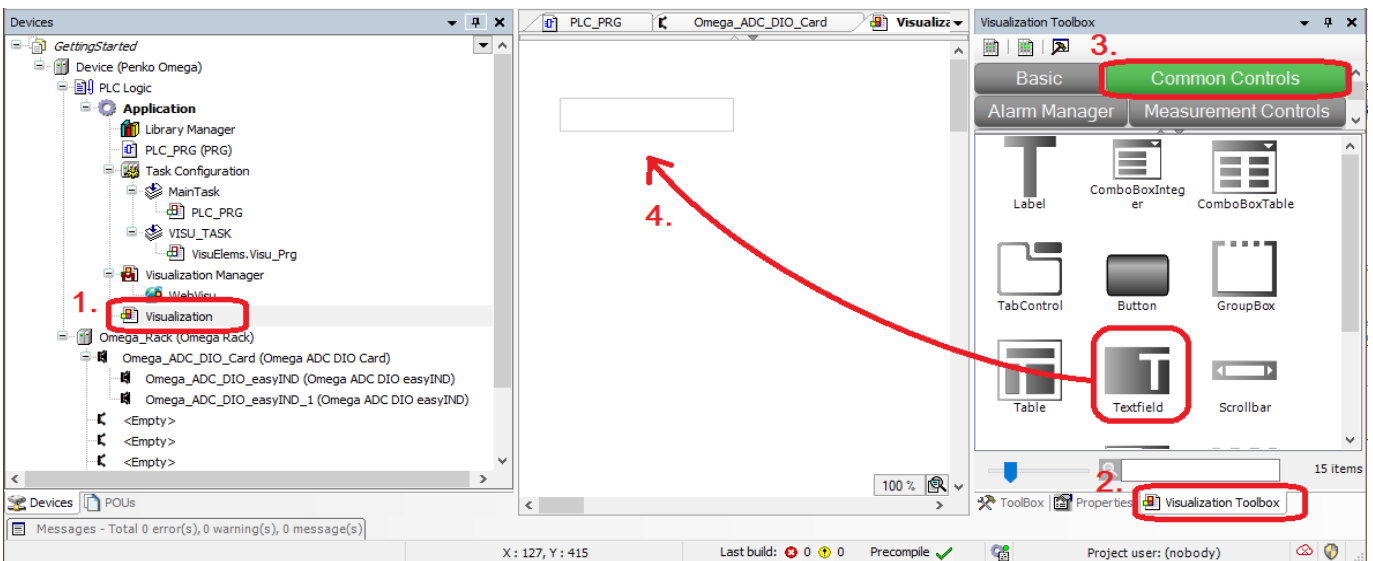


Figure 18. Add a 'Textfield' to your visualization.

Next, we'll add a text and text variable to the Textfield. Codesys has automatically selected the Textfield that we just added to the Visualisation, and has opened the 'Properties' tab (tab list in the right bottom). Open the 'Texts' group in the Properties window, and add the following text to the 'Text' property (Figure 19):

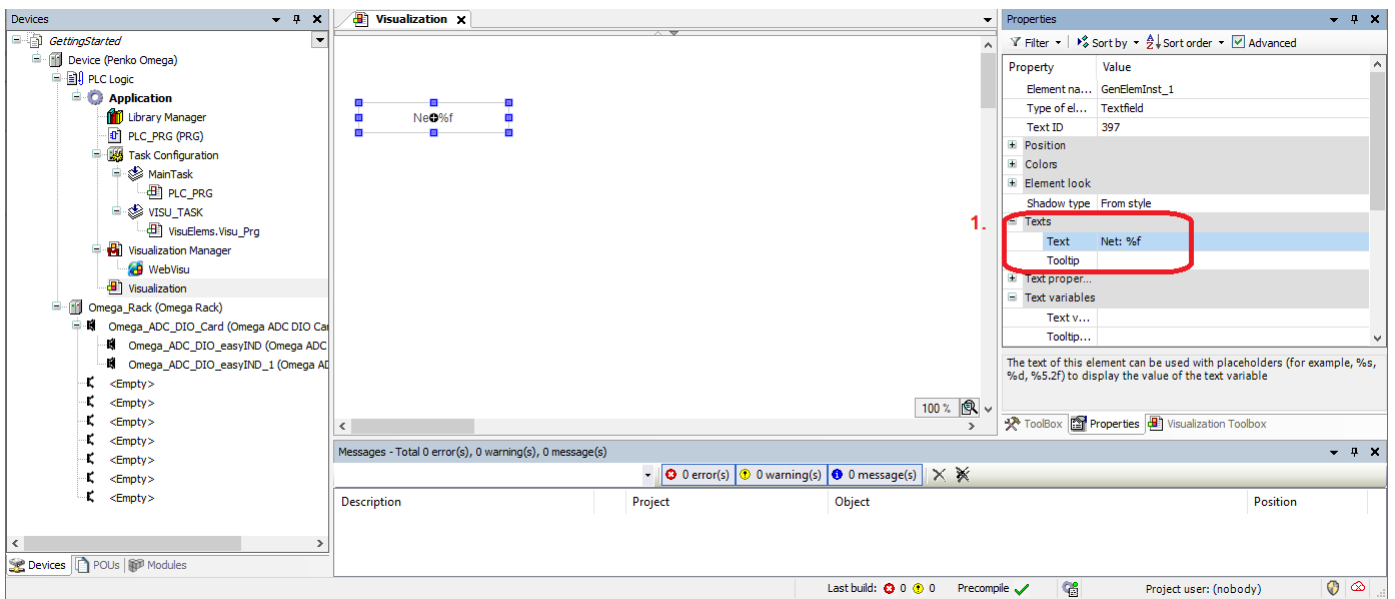


Figure 19. Add a text to the Textfield

The '%f' tells CodeSys to fill in the value of the 'Text variable' property (under the 'Text variables' group). We still need to tell CodeSys *which* variable it has to substitute. Navigate in the 'Properties' tab to the 'Text variables' group and click the square button with three dots. This opens a window to select a variable. Find the 'rNetWeight' variable under 'Application/PLC_PRG', select it and click 'OK'. See .

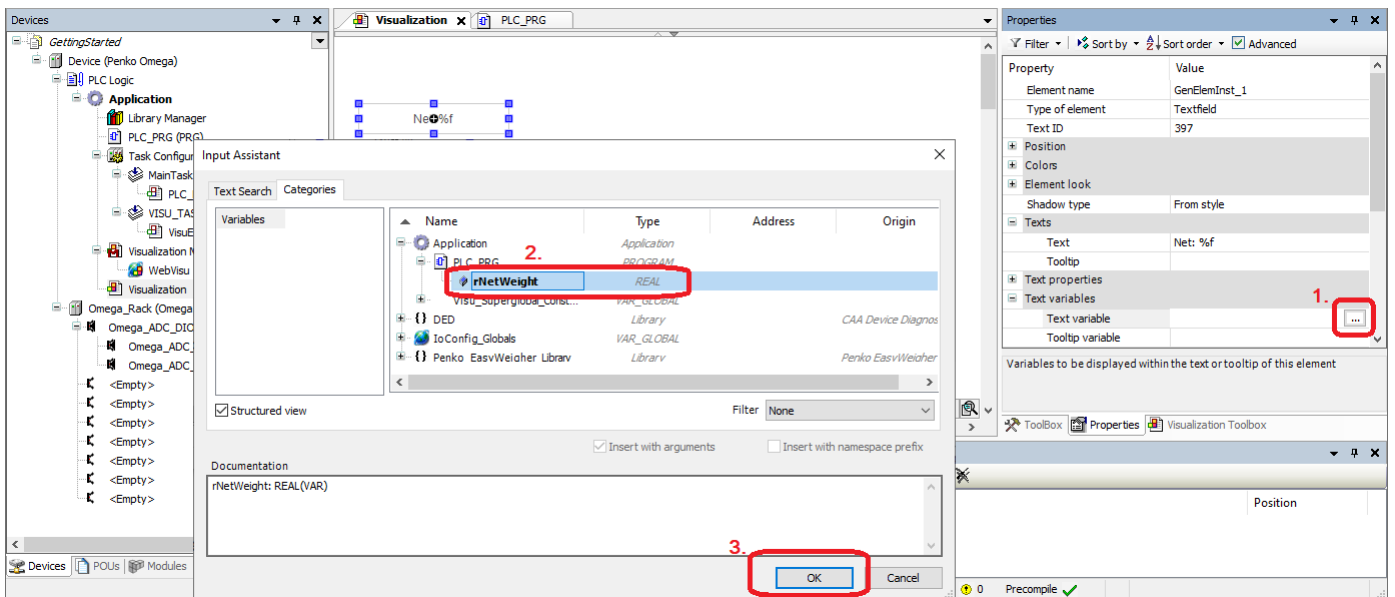


Figure 20. Add a text variable to the TextField

Finally, upload the program to you Omega, and use your web browser to go to the Omega webportal and open the 'Codesys WebVisu' from the dashboard. The visualization should show 'Net: ' followed by the current weight.

CHAPTER 7. EASYWEIGHER: LIBRARY DOCUMENTATION

The previous chapter showed how to use basic IO mapping with the Penko EasyWeigher library. This chapter will explain the rest of the available functionality in the library and how to use it.

All documentation below is only for the option cards that can do weighing: the 'Omega ADC DIO Card' and the 'Omega ADC AIO Card'. The two IO-cards ('Omega AIO Card' and 'Omega DIO Card') do not have weighing functionality, and only provide the IO-mapping functionality from Chapter 6.

7.1 VIEW LIBRARY DOCUMENTATION IN CODESYS

All function blocks and data types are explained in the EasyWeigher library documentation from within CodeSys. This is the place to search for answers if you are ever unsure about what inputs or outputs a function block accepts, what the meaning is of some constants, or what options are available.

Access the library documentation as follows (Figure 21): double click on the Library manager in your project and select the Penko EasyWeigher library. The window below the library manager will show all Function Blocks and Enums in the library, a graphical visualization of the object (tab 'Graphical'), and extensive documentation below the tab 'Documentation'.

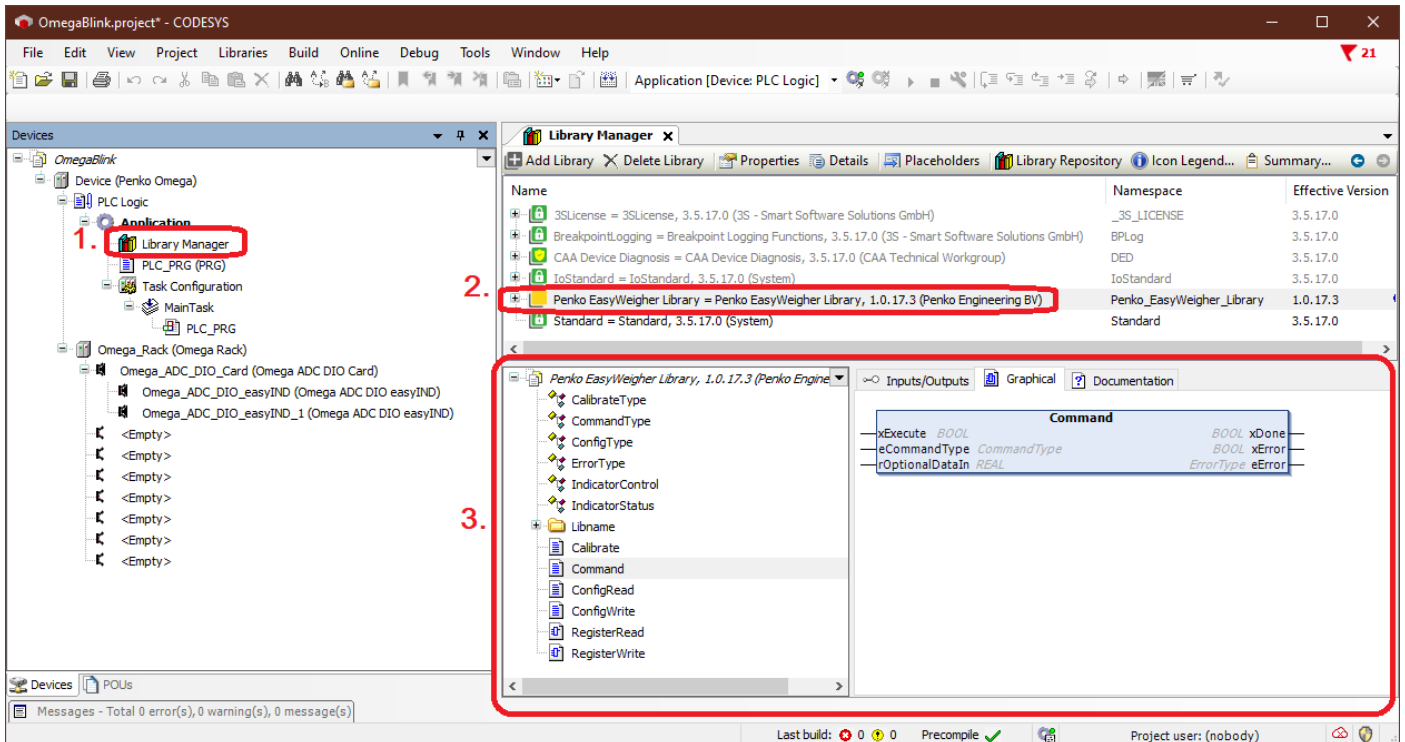


Figure 21. Access the EasyWeigher documentation.

7.2 WEIGHER STATUS STRUCT MAPPING

The weigher status is a struct consisting of a number of bits that describe the current status of the weigher. For example, it tells you whether or not the weight is stable, if the weight is higher than what the hardware can manage, or if (preset) tare is active. The mapping is found in the 'Weights' folder of the IO mapping, with the name 'Status'.

To use this status in your program, create a new variable with the correct struct type in your program:

```
PROGRAM PLC_PRG
```



```

VAR
    rNetWeight : REAL;
    stStatus : Penko_EasyWeigher_Library.IndicatorStatus;
END_VAR

```

The CODESYS autocompletion is your friend in entering the struct type. Start typing 'Penko_', press "CTRL+Space", and CODESYS will automatically fill the rest of the EasyWeigher name. After the '.', CODESYS will show 'IndicatorStatus' as one of the options. Map stStatus to the Status channel in the IO mapping, similar to how rNet was mapped in the previous section. After this, your IO mapping should look something like Figure 22 below.

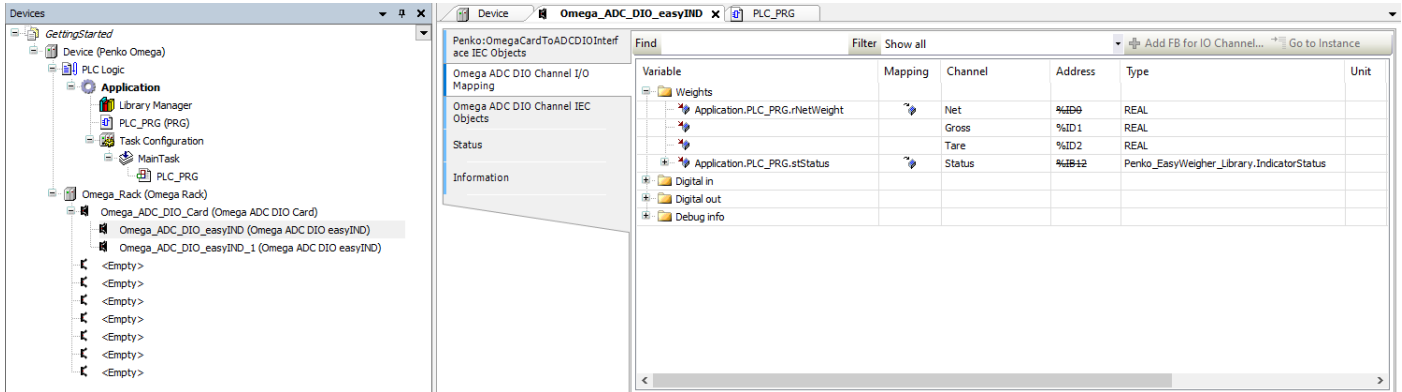


Figure 22. Map the stStatus struct to the Status Channel

Upload your program to the Omega, and inspect the changing of the status bits when the weight changes.

7.2 WEIGHER COMMANDS (TARE, ZERO, ETC)

Each channel in the rack automatically generates four objects that can be found in the "IEC Objects" tab of the respective channel. The name of the object is prefixed with the name of the channel it belongs to. In the example that is "Omega_ADC_DIO_easyIND", then an underscore, and then the type of object.

If the name of the channel changes, then the prefix changes as well.

This section will focus on the Command object. We will extend the example program with visualization that shows the current weight, and a button to perform a tare action.

Insert the Command function block for the first channel.

Insert an empty Box in the first network of the PLC_PRG (Figure 23). It should create an empty square, as indicated on the right side of Figure 23.

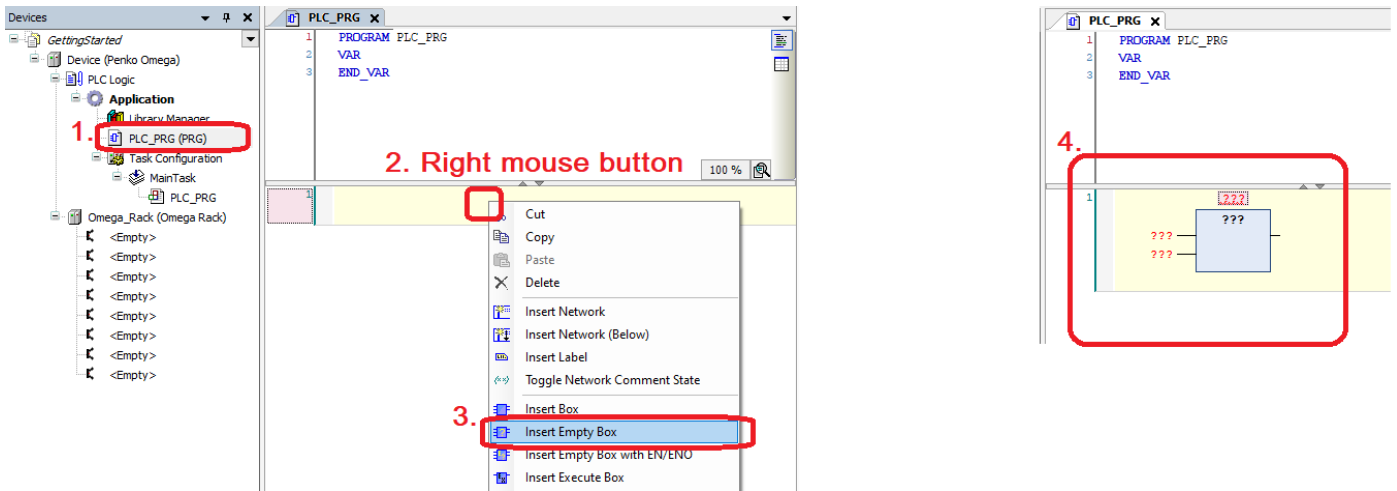


Figure 23. Add an 'Empty Box' to your PLC_PRG.

Make this Empty Block a “*_Command” function block by clicking the red question marks above the empty block, and then clicking the blue square with three dots that appears next to it.

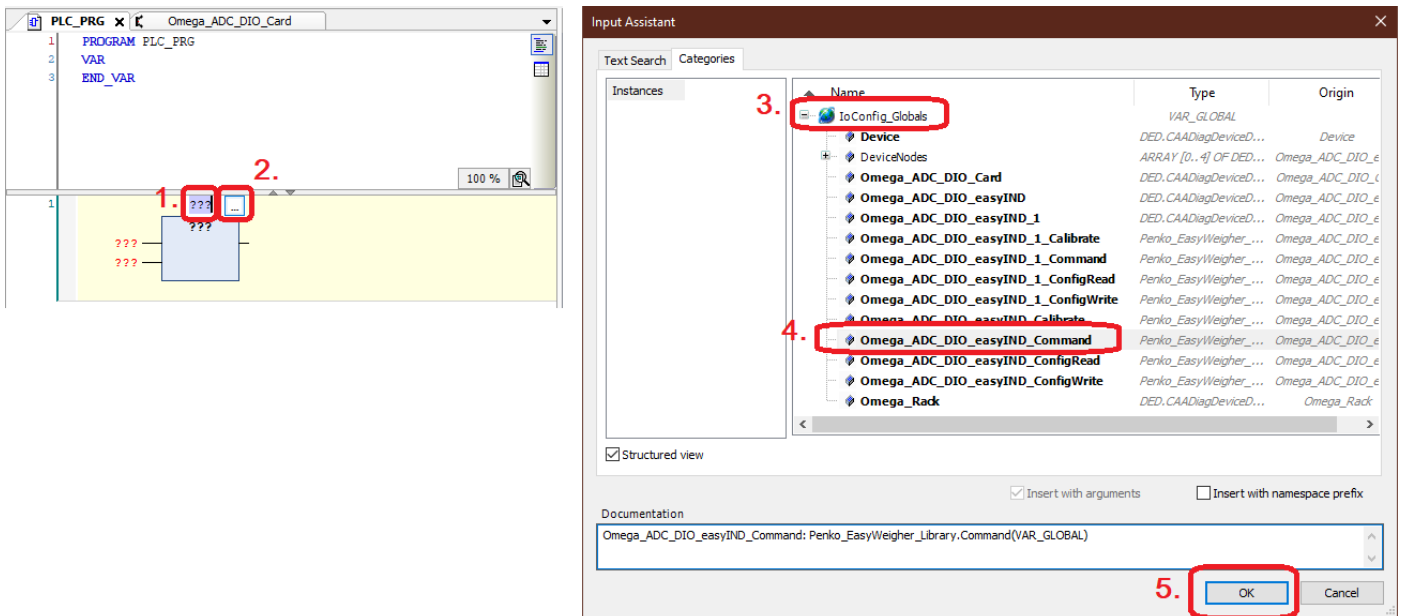


Figure 24. Select 'Omega_ADC_DIO_easyIND_Command' as object.

Make this block the “*_Command” object that starts with the same name as your first channel. If you did not rename the first channel, the object will be called “Omega_ADC_DIO_easyIND_Command”. See Figure 24. If all went well, the first network now contains the Omega_ADC_DIO_easyIND_Command object as shown in Figure 25.

In Figure 24, there is a second object ending with ‘_Command’. Note that that command block starts with the name of your second channel, and thus operates on the second channel.

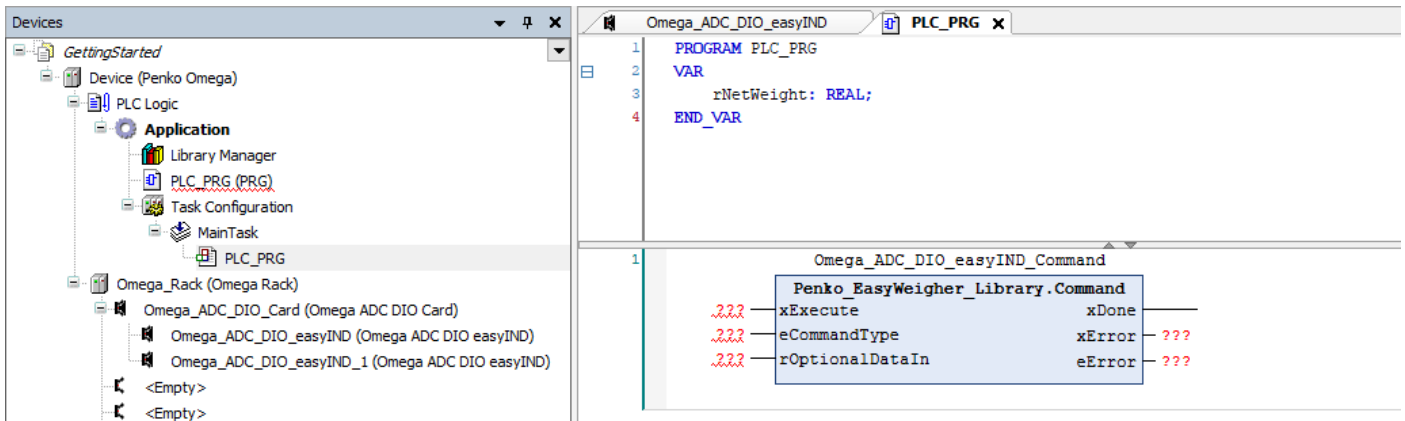


Figure 25. The Omega_ADC_DIO_easyIND_Command function block has been added.

The Command object has three inputs:

1. xExecute: a Boolean that indicates whether the function block should be doing its command.
2. eCommandType: an Enum of type CommandType. The Enum is defined in the EasyWeigher library, and defines for example COMMAND_TARE_SET. See the documentation inside the EasyWeigher library for more commands (follow Chapter 7.1).
3. rOptionalDataIn: Some commands require an input argument, such as COMMAND_PTARE_SET to set the preset tare to a predefined weight. Other commands such as COMMAND_TARE_SET do not require an input argument. In those cases, this field can be left empty.

The Command also has three outputs:

1. xDone: a Boolean that is TRUE when the function block is done, and FALSE when the function block is not executing (xExecute is FALSE) or not yet done executing and thus needs another execution cycle.
2. xError: a Boolean that is TRUE when something went wrong during execution.
3. eError: an enum of type ErrorType, which indicates what went wrong in case of an error. When there is no error, this will be ERROR_SUCCESS. All values of this enum are defined and documented in the EasyWeigher library.

Use the Command function block for the first channel.

We will set and reset a tare as a test. First, add two booleans to your PLC_PRG that indicates whether the Command functions block will be run, one to set the tare, and one to reset the tare:

```
PROGRAM PLC_PRG
VAR
    rNetWeight : REAL;
    xExecuteTareSet : BOOL;
    xExecuteTareReset : BOOL;
END_VAR
```

Then, tie xExecuteTareSet to the xExecute input of the Command function block, and set the eCommandType input to COMMAND_TARE_SET. The easiest way to do this is start typing "COMMAND_" in the eCommandType field, and press CTRL+SPACE to let CODESYS do the autocompletion. This will give a list of COMMAND_ values that can be filled in. COMMAND_TARE_SET can be selected from the list by double clicking the entry. CODESYS will automatically add the namespace, which is 'Penko_EasyWeigher_Library.CommandType' in this example.

For testing it would be nice to automatically set xExecuteTareSet back to FALSE when the command block is done: to achieve this, xExecuteTareSet is set to FALSE when xDone is TRUE, and is unchanged while xDone is FALSE. See Figure 26.

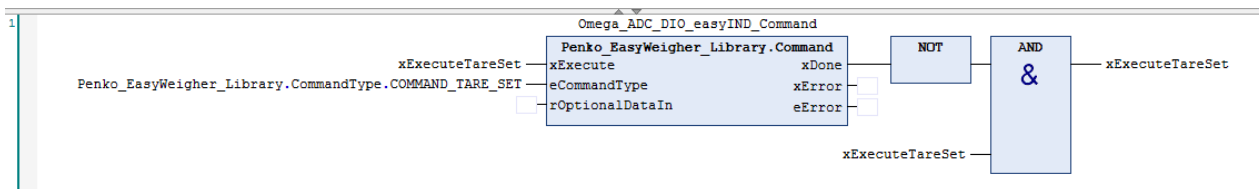


Figure 26. A simple network to activate a tare.

In a similar way, we can create a network to reset the tare below that. Note that we added two new Boolean values: `xExecuteTareReset` and `xDoneTareReset`, and that the `eCommandType` input has changed to `COMMAND_TARE_RESET`.

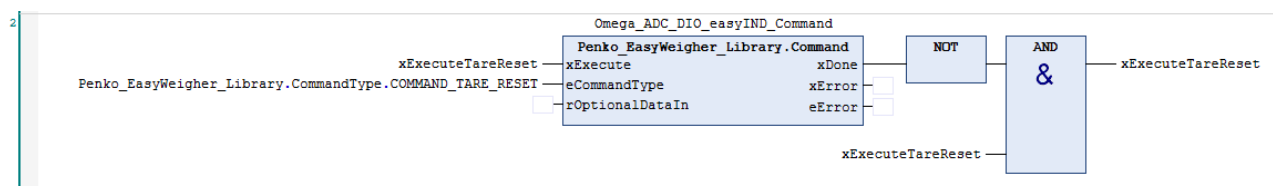


Figure 27. A simple network to deactivate the tare.

Set the `xExecuteTareSet` and `xExecuteTareReset` variables via the debugging functionality, and check that setting a tare sets the net weight to zero, and resetting the tare sets the weight back to what it was.

As an exercise, set and reset the tare with buttons in the visualization. Open the visualization toolbox, and drag a `DipSwitch` (under 'Lamps/Switches/Bitmaps') to the visualization. Tie its 'Variable' property to the `xExecuteTareSet` variable, and upload your program. The button should set a tare, and automatically be toggled back because of the NOT-AND circuit that we added after the `_Command` block.

Add a second button to reset the tare.

7.3 READING CONFIGURATION

The function block to read configuration is named `$(deviceName)_ConfigRead`. If the first channel of the first Omega card was not renamed, the function block named `Omega_ADC_DIO_easyIND_ConfigRead` can be used to read configurations of that channel.

Insert the `ConfigRead` function block for the first channel.

To start, add a third network, insert an empty box in that network and make it the `_ConfigRead` function block, similar to how the `_Command` function block was inserted in **Error! Reference source not found.** Your network should look like Figure 28 below:

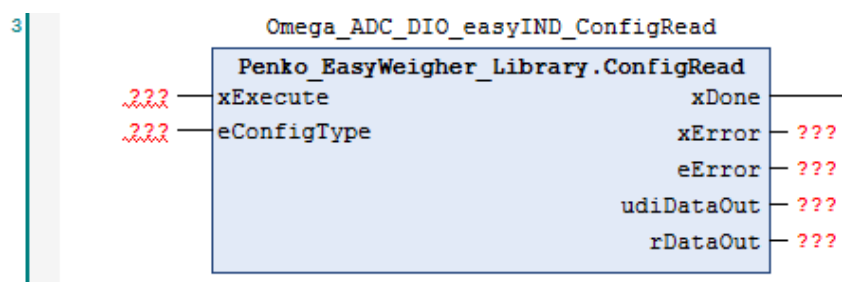


Figure 28. The `Omega_ADC_DIO_easyIND_ConfigRead` block is added to a network.

All inputs and outputs of this ConfigRead block are explained in detail in the EasyWeigher library documentation within CODESYS, (see Chapter 7.1 on how to get there. The Config block works following the same principles as the Command block: it executes while input xExecute is high, and input eConfigType defines what configuration parameter is being read. It is important to note that there are two data outputs: udiDataOut and rDataOut. Some configuration parameters are weights. As before, all weights are of type REAL, and these parameters thus need to be read from rDataOut.

For example, when eConfigType is set to CONFIG_MAXLOAD, the block will read the MaxLoad parameter from the weigher configuration, and this is a weight. Therefore, the correct output appears at rDataOut. Similarly, when eConfigType is set to CONFIG_DECIMALPOINT, the block will read out the number of decimals the weigher uses internally. This is inherently an integer number. Therefore, the correct output appears at udiDataOut. All possible enum values for eConfigType can be found in the EasyWeigher library documentation within CODESYS (see Chapter 7.1), and each value documents which output must be used.

Use the ConfigRead function block for the first channel.

Let’s use this ConfigRead block to read out the decimal precision that is used in the weight. First, add some variables to the PLC_PROG, one to execute the block and one to store its result:

```
PROGRAM PLC_PRG
VAR
    rNetWeight : REAL;
    xExecuteTareSet : BOOL;
    xExecuteTareReset : BOOL;
    xExecuteConfigReadDecimals : BOOL;
    udiDecimalsRead : UDINT;
END_VAR
```

The number of decimals that the weigher uses internally is read by setting input eConfigType to PENKO_EASYWEIGHER.ConfigType.CONFIG_DECIMALPOINT, adding the udiDecimalsRead variable to the udiDataOut output, and tying the xExecuteConfigReadDecimals variable to the xExecute input like we did before. For easier testing, the NOT-AND network is added after xDone to automatically reset xExecuteConfigReadDecimals. See Figure 29 below.

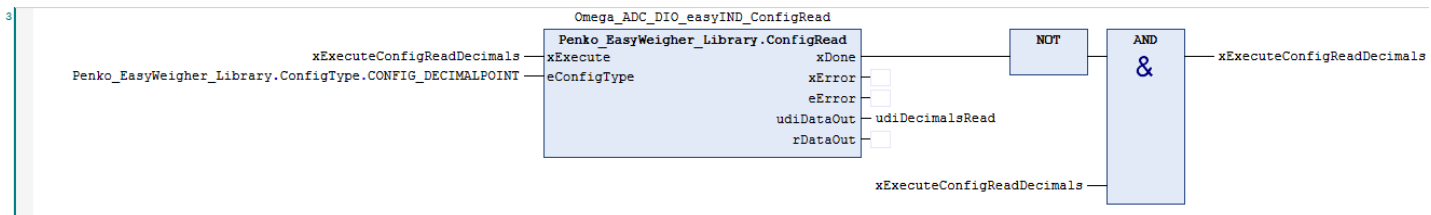


Figure 29. Variables are tied to the ConfigRead block.

Use the CODESYS debugging functionality to set xExecuteConfigReadDecimals to TRUE, and observe that the number of decimals that are used in the weighing card are put into udiDecimalsRead. In the example below, the weight used three decimals precision (Figure 30).

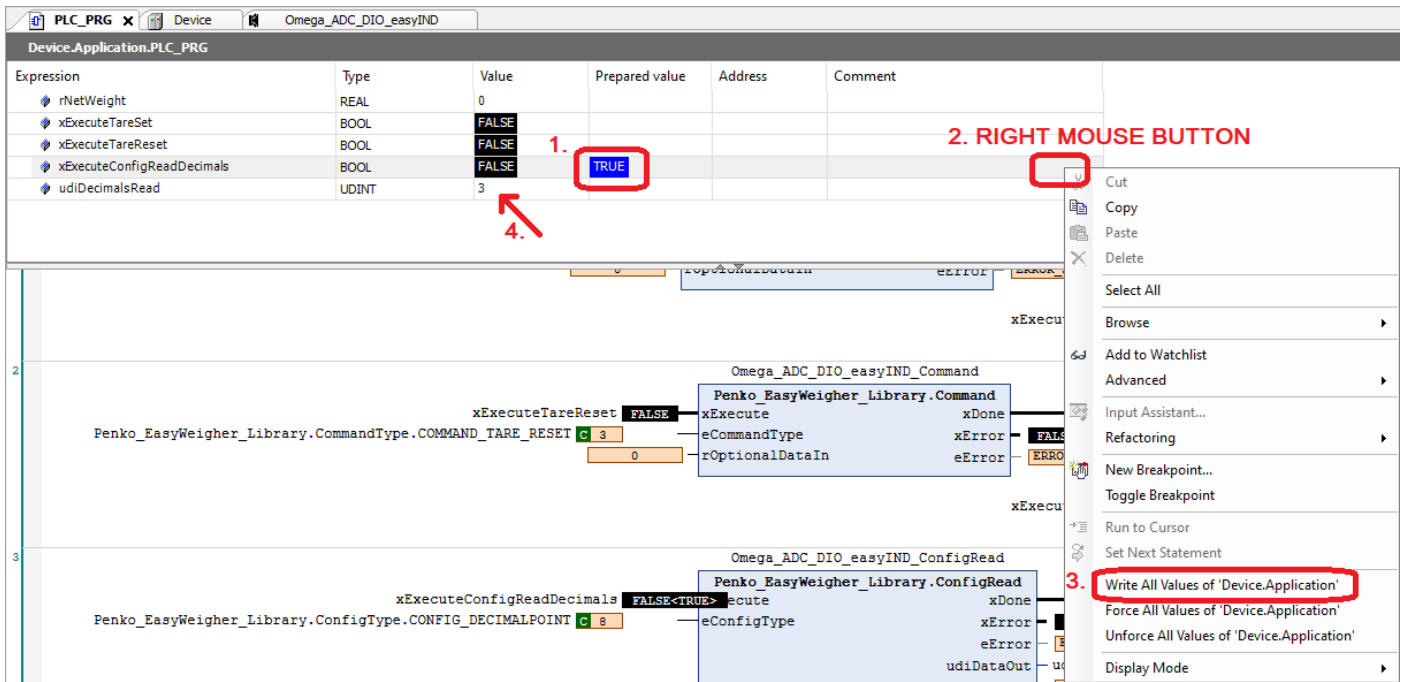


Figure 30. Use the CODESYS debugger to toggle reading the number of decimals.

7.4 WRITING CONFIGURATION – INDUSTRIAL MODE

Writing a configuration in industrial mode is very similar to reading a configuration. The only difference is that a different Function Block must be used, which has inputs udiDataIn and rDataIn, instead of two outputs udiDataOut and rDataOut.

Insert the ConfigWrite function block for the first channel.

Add a fourth network, insert an empty box in that network and make it the _ConfigWrite function block, similar to how the _Command function block was inserted in **Error! Reference source not found.**

Your network should look like Figure 19 below.

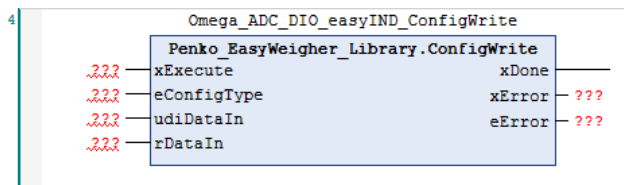


Figure 31. The Omega_ADC_DIO_easyIND_ConfigWrite block is added to a network.

All inputs and outputs are documented extensively in the EasyWeigher library documentation within CODESYS, and work very similar to the earlier Command and ConfigRead blocks.

This ConfigWrite block has two data inputs, udiDataIn and rDataIn, to allow two types of data to be written to the indicator configuration. rDataIn is used to write weights of type REAL such as the MaxLoad parameter, udiDataIn is used to write integer values such as the number of decimals that is used internally.

IMPORTANT: Whenever one input is used, the other input MUST BE ZERO.

Use the ConfigRead function block for the first channel.

Add two new variables `xExecuteConfigWriteDecimals` and `udiDecimalsWrite` to your `PLC_PRG`, and complete the network similar to the `ConfigRead` network from the previous section. In the end, your network should look like Figure 32 below:

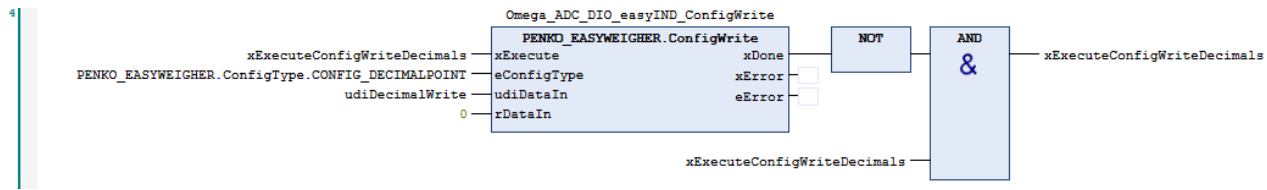


Figure 32. Variables are tied to the `ConfigWrite` block.

Note that, because `udiDataIn` must be used and not `rDataIn`, `rDataIn` has been set to zero.

Use the CODESYS debugging functionality to prepare a value for `udiDecimalsWrite` and `xExecuteConfigWriteDecimals`, similar to the previous section and Figure 30. If the number of decimals was e.g. 3, set `udiDecimalsWrite` to 2. After writing the new `udiDecimalsWrite` value (don't forget to set `xExecuteConfigWriteDecimals` high to do the write action), the weight in `rNetWeight` should have changed accordingly (e.g. by changing a factor 10).

7.5 WRITING CONFIGURATION – CERTIFIED MODE

When the Omega weighing module is configured to be in certified mode, the configuration is protected by a TAC value. The TAC value is used to prevent the configuration from being changed accidentally. Trying to write a config value without unlocking the configuration will result in an `eError=ERRORTYPE_ACCESSDENIED`.

Managing the TAC value

Unlocking the configuration is not difficult: first read the current required TAC value, and then write that same TAC value back. This will unlock the configuration for approximately 100 seconds.

In a certifiable system, the CODESYS programmer must create a bit of visualization for the end user, who has to enter the current TAC manually to write back his configuration. This way, writing configuration values becomes a conscious effort for the end user which reduces chances of accidentally changing the configuration and thereby breaking the certification.

While it is *possible* for the CODESYS programmer to programmatically read out the TAC and write it back automatically, it takes away the conscious effort for the end user, and thus takes away the safety mechanism. If your product needs to be certified, this is very probably not allowed.

The most user friendly way to write the configuration from for example a visualization is to let your user fill in the configuration they want, and then write it to the Omega weighing module all at once. That way, your user does not have to enter the TAC code each time they write a configuration value, but only the one time that all configuration values are written.

Reading the TAC value

The TAC value is part of the configuration block, and can be read just like any configuration using the `_ConfigRead` block of your module, with input `eConfigType` set to `PENKO_EASYWEIGHER.ConfigType.CONFIG_TAC`. See Figure 33 below. If you want to have `xExecuteConfigReadTAC` automatically reset when the block is done, use the same NOT+AND boxes after `xDone` as we did in the previous examples.

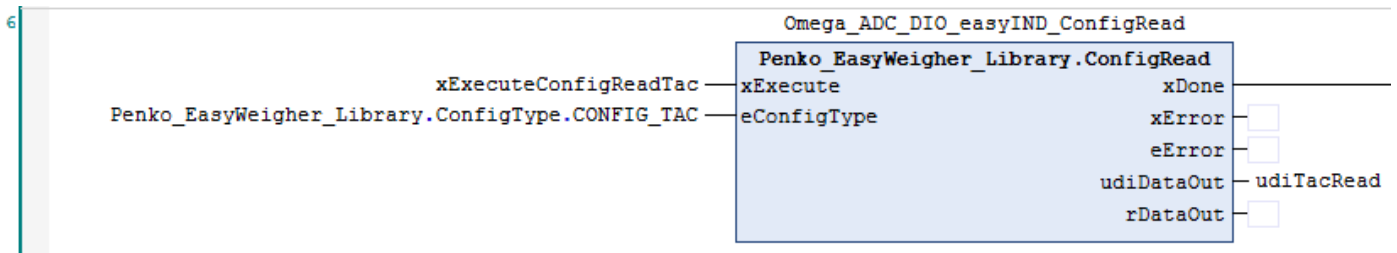


Figure 33. Read the TAC value using the ConfigRead block.

Writing the TAC value

The TAC is written in the same way as any configuration value, using the _ConfigWrite block. Again, the eConfigType must be set to PENKO_EASYWEIGHER.ConfigType.CONFIG_TAC, as in Figure 34.

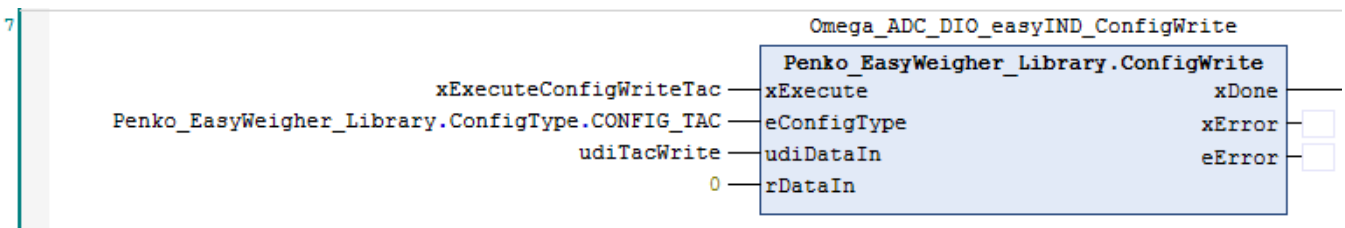


Figure 34. Write the TAC value using the ConfigWrite block.

7.6 CALIBRATION

All calibration actions are done using the \$(deviceName)_Calibrate function block, as shown in Figure 35.

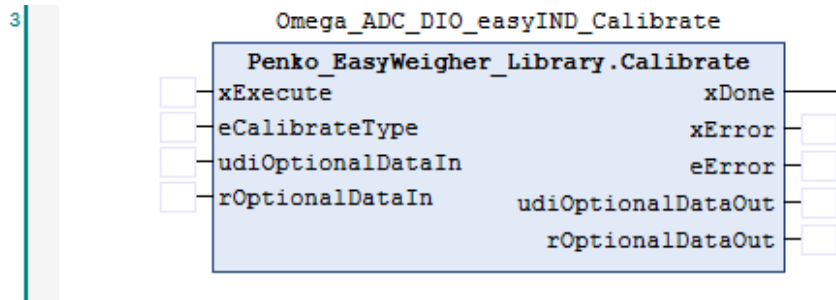


Figure 35. The _Calibrate block.

The _Calibrate block operates in the same way as the _Command, _ConfigRead and _ConfigWrite blocks. Depending on the calibration action you choose for input eCalibrateType you may have to provide extra input data to udiOptionalDataIn or rOptionalDataIn. Also depending on the calibration action is whether it function block generates an output in udiOptionalDataOut or rOptionalDataOut. Which actions there are, and what inputs they need and what outputs they generate is documented in the EasyWeigher library documentation within CODESYS.

Calibration is very similar to writing configurations in certified mode. The same way that the TAC code acts as a protection against accidental changes in the configuration, the calibration is protected by a CAL code. The CAL code needs to be managed in the same way as the TAC code in certified mode.

Managing the CAL code

The CAL code must be managed in the same way as the TAC code, see Section 4.6 above. In short, when a calibration action returns `ERRORTYPE_ACCESSDENIED`, you will have to let your user enter the CAL code to unlock the calibration.

Reading and writing the CAL code

To start, add the following variable definitions to your `PLC_PRG`:

```
PROGRAM PLC_PRG
VAR
// all the previous definitions
// ...
    xExecuteCalibrateReadCal : BOOL;
    xExecuteCalibrateWriteCal : BOOL;
    udiCalRead : UDINT;
    udiCalWrite : UDINT;
    eErrorCalibrateReadCal : Penko_EasyWeigher_Library.ErrorType;
    eErrorCalibrateWriteCal : Penko_EasyWeigher_Library.ErrorType;
END_VAR
```

Reading and writing the CAL code is done in a similar way to reading and writing the TAC code. However, this time the `_Calibrate` block is used, instead of the `_ConfigRead` or `_ConfigWrite` block. See Figure 36. Reading and writing the CAL code..

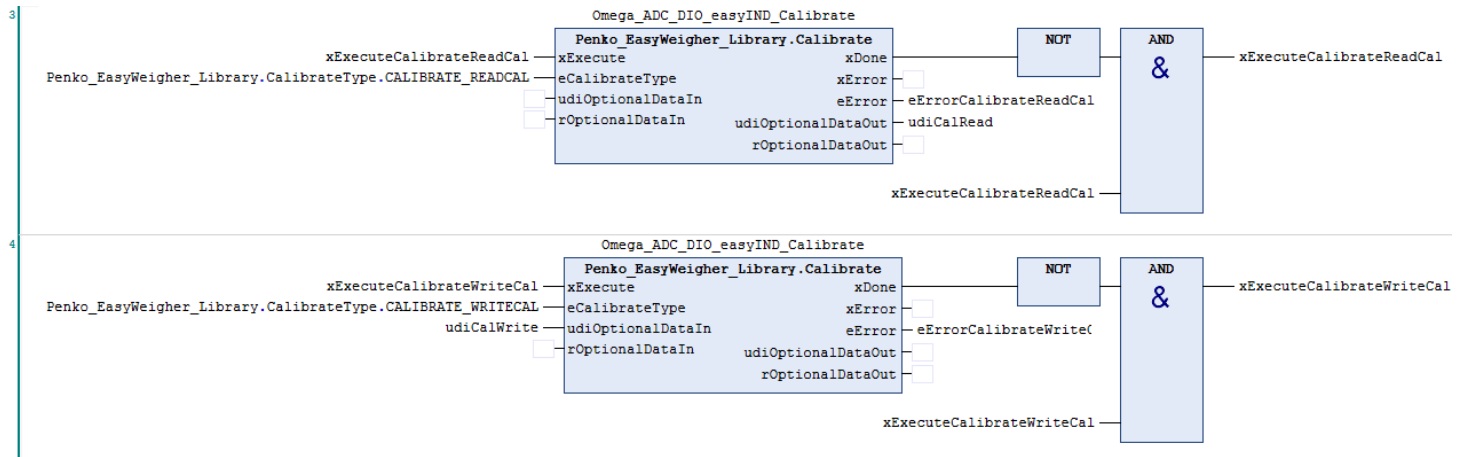


Figure 36. Reading and writing the CAL code.

APPENDIX I: BLINK PROGRAM EXAMPLE 1

Program variables:

```
PROGRAM PLC_PRG
VAR
    leds: PENKO_BSP.Leds; // Get the LED's from the library
    Delay: TON;           // Time to determine blink rate
    color_ctr: INT := PENKO_BSP.LED_COLOR.GREEN; // Start at color green
    led_ctr: INT;         // For selecting the LED (LED1, LED2, LED3, LED4)
END_VAR
```

Program code:

```
Delay(IN:=TRUE, PT:=T#1S); // Turn on the timer and set it to one second
IF NOT(Delay.Q) THEN // Wait till the timer has reached one second
    RETURN; // Timer not at one sec? Don't execute the rest of the code below
END_IF
Delay(IN:=FALSE); // Turn off the timer
leds.setToBits(0); // Turn off all LED's

leds.setColor(PENKO_BSP.LED.LED4, PENKO_BSP.LED_COLOR.RED); // Always turn on LED4 in red

leds.setColor(led_ctr, color_ctr); // Set the LED number and color
color_ctr := color_ctr + 1; // Change color

IF color_ctr = PENKO_BSP.LED_COLOR.NUM THEN
    color_ctr := PENKO_BSP.LED_COLOR.GREEN; // Go back to green
    led_ctr := led_ctr + 1; // go to the next LED
    IF led_ctr = PENKO_BSP.LED.LED4 THEN
        led_ctr := PENKO_BSP.LED.LED1; // Go back to LED1
    END_IF
END_IF
```


APPENDIX II: BLINK PROGRAM EXAMPLE 2

Program variables:

```
PROGRAM PLC_PRG
VAR
    leds: PENKO_BSP.Leds; // Get the LED's from the library
    Delay: TON;           // Time to determine blink rate
    state: UINT := 0;     // Start at 0 (all LED's off)
END_VAR
```

Program code:

```
Delay(IN:=TRUE, PT:=T#500MS); // Turn on the timer and set it to one second
IF NOT(Delay.Q) THEN          // Wait till the timer has reached one second
    RETURN;                   // Timer not at one sec? Don't execute the rest of the code below
END_IF
Delay(IN:=FALSE);            // Turn off the timer
leds.setToBits(0);           // Turn off all LED's

leds.setOnMasked(state);     // Turn one or more LED on at once
state := state + 1;          // Go to the next state

IF state > 15 THEN           // If all 15 states reached
    state := 1;              // Go back to only the first LED on
END_IF
```



About PENKO

At PENKO Engineering we specialize in weighing. Weighing is inherently chemically correct, independent of consistency, type or temperature of the raw material. This means that weighing any kind of material guarantees consistency and thus, it is essential to sustainable revenue generation in any industry. As a well-established and proven solution provider, we strive for the ultimate satisfaction of custom design and/or standard applications, increasing your efficiencies and saving you time, saving you money.

Whether we are weighing raw materials, components in batching, ingredients for mixing or dosing processes, - or weighing of static containers and silos, or - in-motion weighing of railway wagons or trucks, by whatever means required during a process, we are essentially forming vital linkages between processes and businesses, anywhere at any time. We design, develop and manufacture state of the art technologically advanced systems in accordance with your strategy and vision. From the initial design brief, we take a fresh approach and a holistic view of every project, managing, supporting and/or implementing your system every step of the way. Curious to know how we do it? www.penko.com

Certifications

PENKO sets high standards for its products and product performance which are tested, certified and approved by independent expert and government organizations to ensure they meet – and even – exceed metrology industry guidelines. A library of testing certificates is available for reference on:

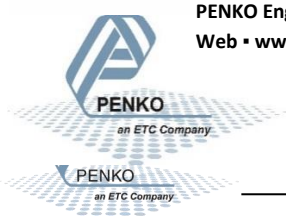
www.penko.com/nl/publications_certificates.html

PENKO Professional Services

PENKO is committed to ensuring every system is installed, tested, programmed, commissioned and operational to client specifications. Our engineers, at our weighing center in Ede, Netherlands, as well as our distributors around the world, strive to solve most weighing-system issues within the same day. On a monthly basis PENKO offers free training classes to anyone interested in exploring modern, high-speed weighing instruments and solutions. Training sessions on request: www.penko.com/training

PENKO Distributor

A complete overview you will find on: www.penko.com/Find-A-Dealer



PENKO Engineering B.V. • Schutterweg 35, NL 6718C Ede • Tel +31 (0) 318525630 • info@penko.com

Web • www.penko.com • Copyright © 2014 ETC All rights reserved. 7600M1083-EN-R3 OMEGA SYSTEM CODESYS MANUAL.DOCX