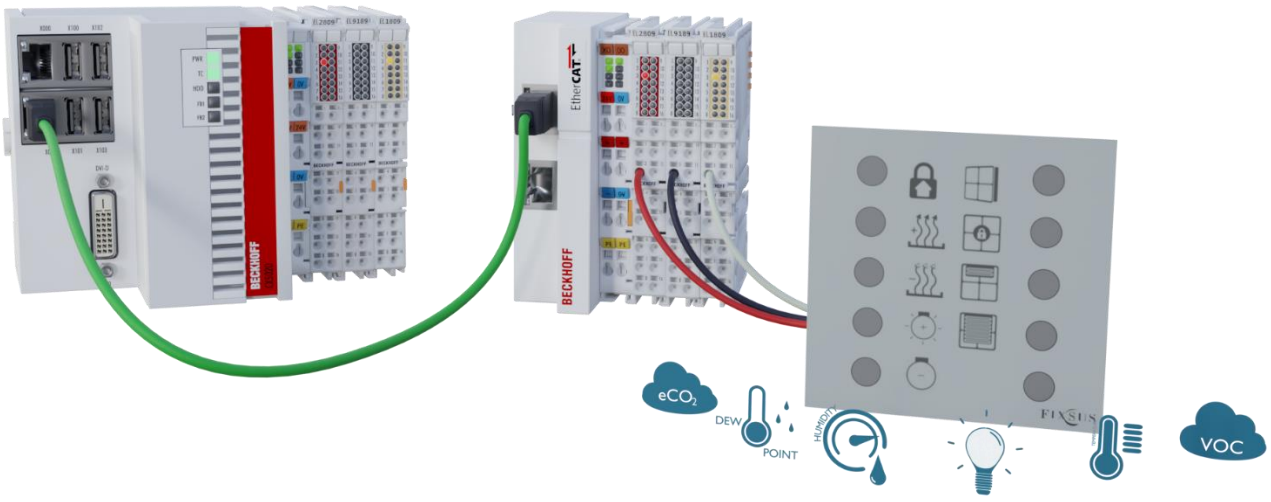
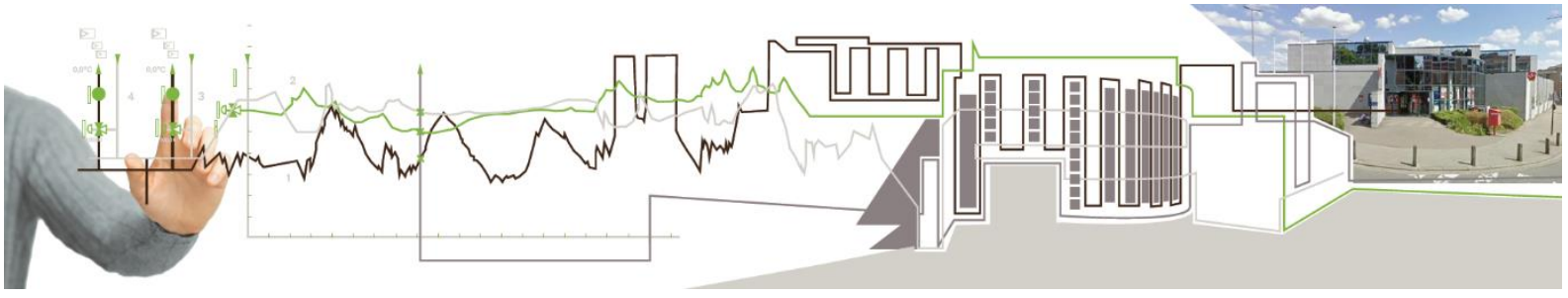
This document explains how to implement DIOC devices into a TwinCAT 3 project.

Manual TP10/DIOC





1. Contents

2. Introduction

This manual is provided to help people implement the TP10 into their own TwinCAT 3 projects. If required, you can visit our site, www.fixsus.be, or our forum, https://forum.fixsus.be, for additional info.

3. Short guide to implementing DIOC into TwinCAT 3

- Step 1: Use E-bus digital input and output terminals
- Step 2: Download the DIOC library 'DIOC_Library' and install it
  The latest version of the library can be found on the Fixsus forum, https://forum.fixsus.be
- Step 3: Implement the library into a TwinCAT PLC project
- Step 4: Change the cycle time to 12 ms
  - Method 1: change the standard cycle time to 12 ms and call the instances in MAIN
  - Method 2: make a new task with a cycle time of 12 ms and call the instances in the new task
- Step 5: Implement the new visualizations, if required
  - Implement the 'TP10' or 'TP10 mini' visualization for each TP10.
- Step 6: Change the system manager settings
  - Check if the in- and outputs of the DIOC devices are being called in the correct task
  - Enable the 'I/O at task begin for the linked PLC program

4. Detailed manual to implementing DIOC into TwinCAT 3

Step 1: Use E-bus digital input and output terminals

To implement the DIOC protocol, the E-bus must be used. To do this, E-bus digital input and output terminals must be used (e.g. EL1809, EL2809 or EL1859).

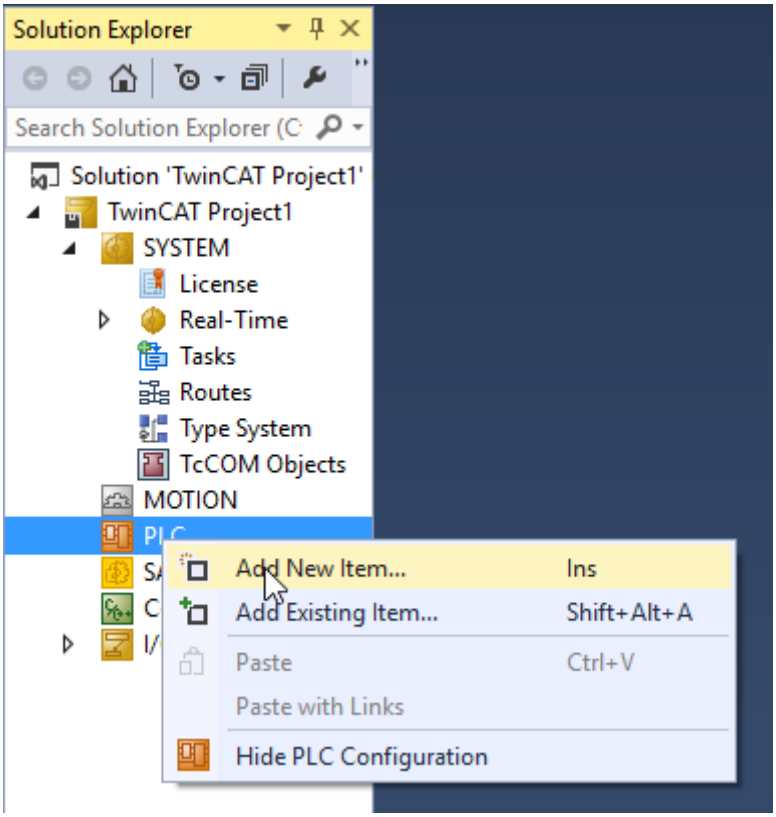
The DIOC protocol cannot be used on the K-bus. If the amount of inputs/outputs of the K-bus is too large, the I/O cycle time will get an offset causing the DIOC protocol to not function properly. Therefore, the K-bus is not officially supported.

Step 2: Download the DIOC library 'DIOC_Library' and install it

The first step of the implementation is to install the necessary libraries.
If the library is already installed continue to step 2.

Start by creating a new TwinCAT (not PLC) project.

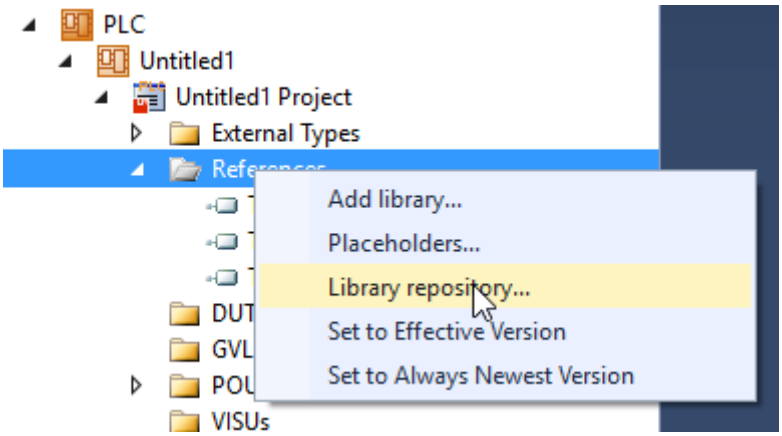
Add a TwinCAT PLC project by right clicking in the solution tree on item PLC



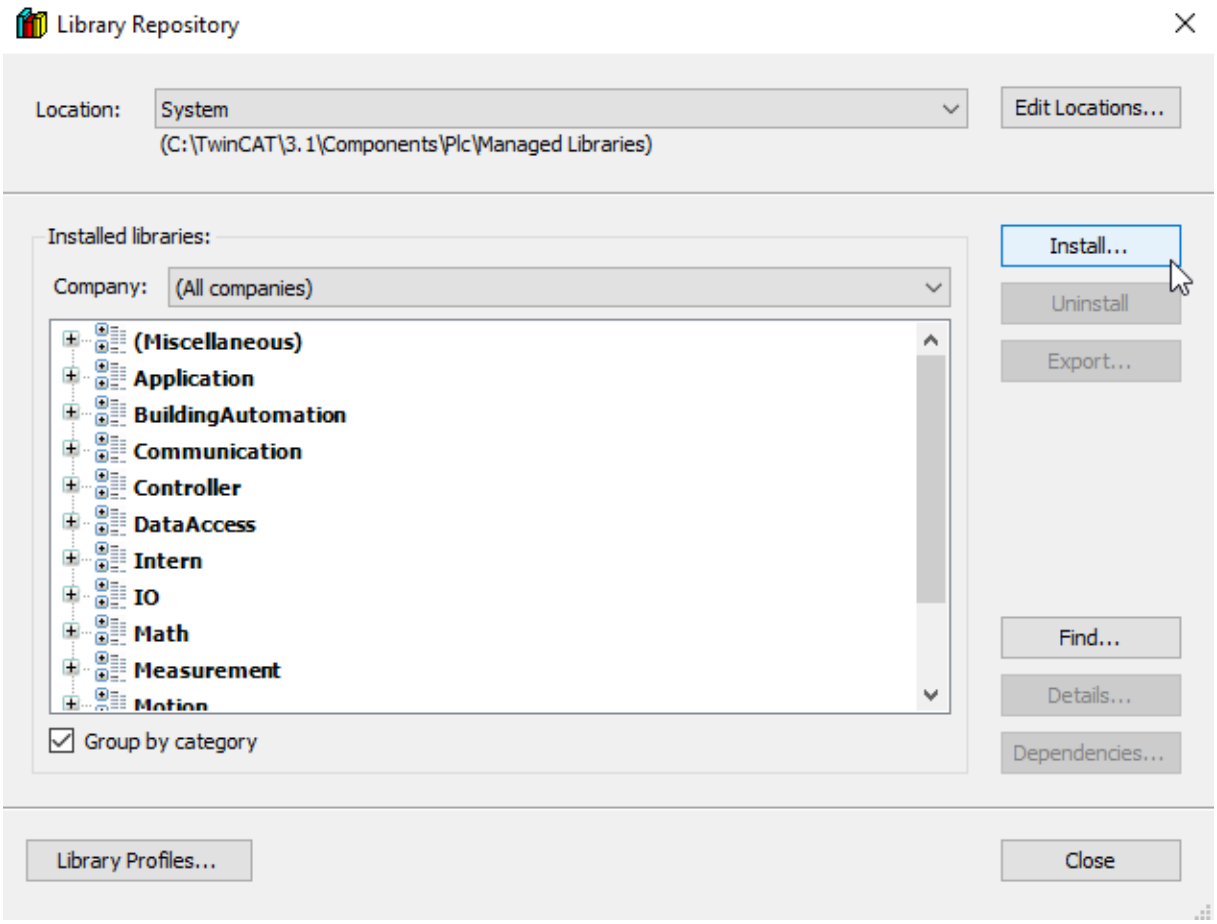
Download the latest TC3 library from our forum. (see https://forum.fixsus.be/topics/8-TP10---RA---Technician--Programming)

The library file 'TC3_fixsus_dioc_library.library' must be added to the TwinCAT library repository. This must only be done once, or every time there is a new library version.

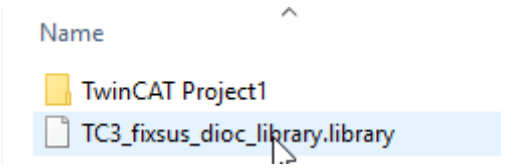
Do this by right clicking on 'references' in the plc, selecting library repository



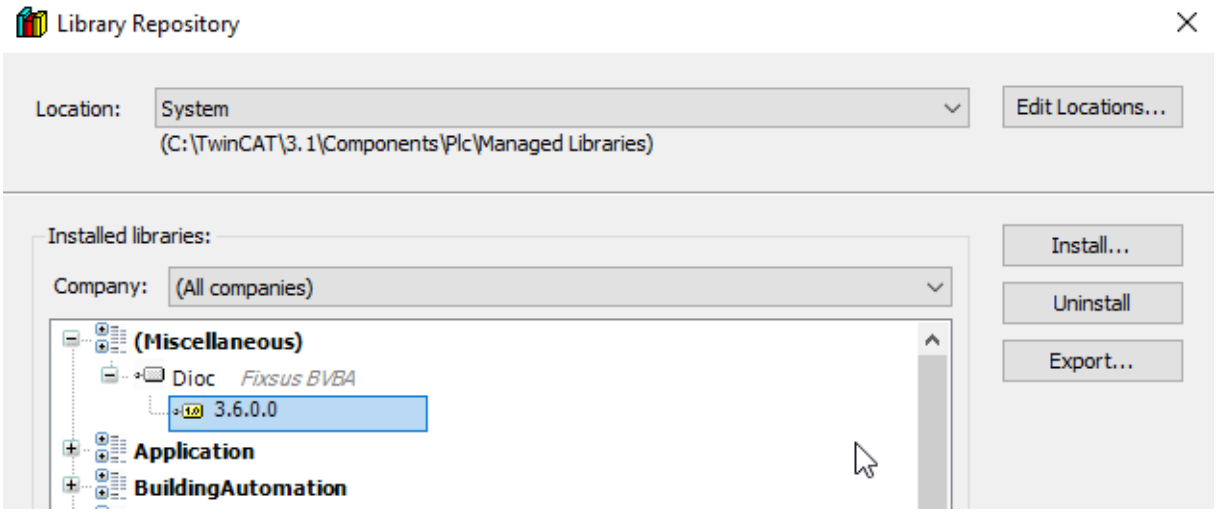
Press install



And install the downloaded library by double clicking it



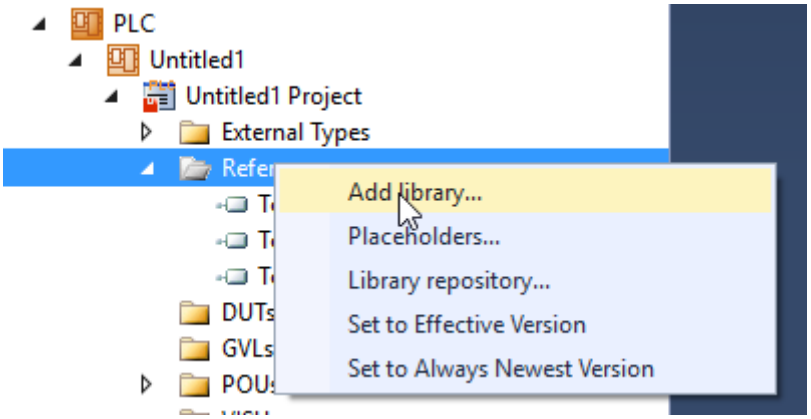
You should now find the (new) DIOC library under "Miscellaneous"



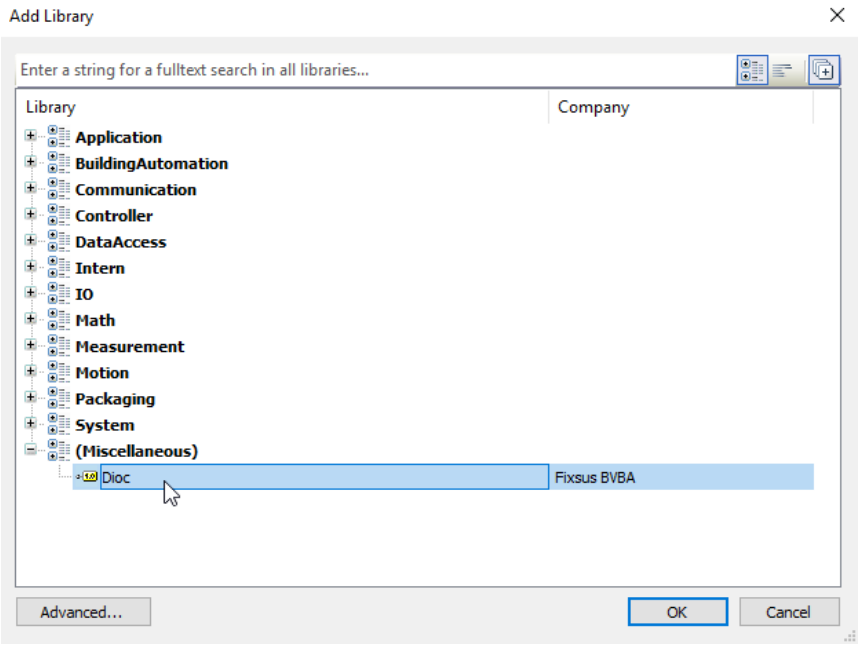
Step 3: Implement the library into a TwinCAT PLC project

If the 'DIOC_library' was installed, it needs to be implemented in every project using DIOC.

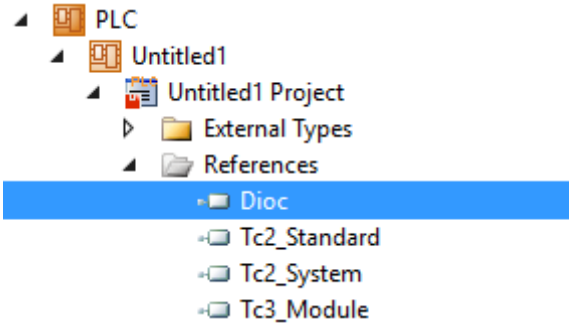
Do this by right clicking on 'references' in the plc and selecting 'add library'



Search for Dioc or select the library under "Miscellaneous" and press ok



The library will now be imported into the PLC project



Step 4: Set the cycle time

To ensure a good communication with the TP10 and other DIOC devices, instances of the TP10 and other DIOC function blocks must be called with a fixed cycle time. This cycle time is currently 12ms.

There are two methods to do this:

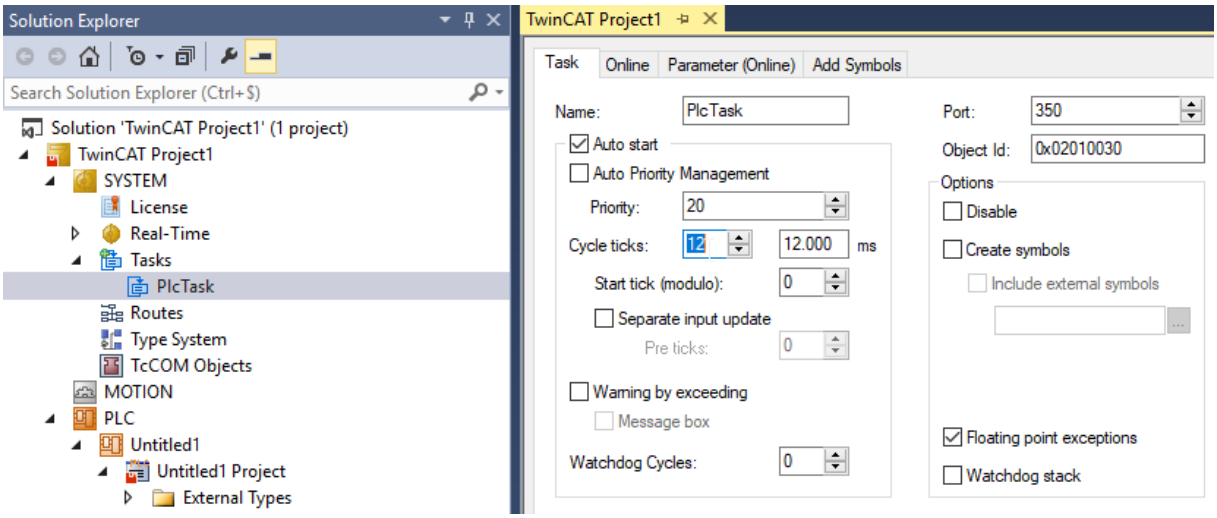
1. The standard cycle time can be set to 12ms, the DIOC instances should then be called in the standard program (the MAIN program)
2. A new task can be made with a cycle time of 12ms in which the DIOC instances can be called

Method 1: Change the standard cycle time

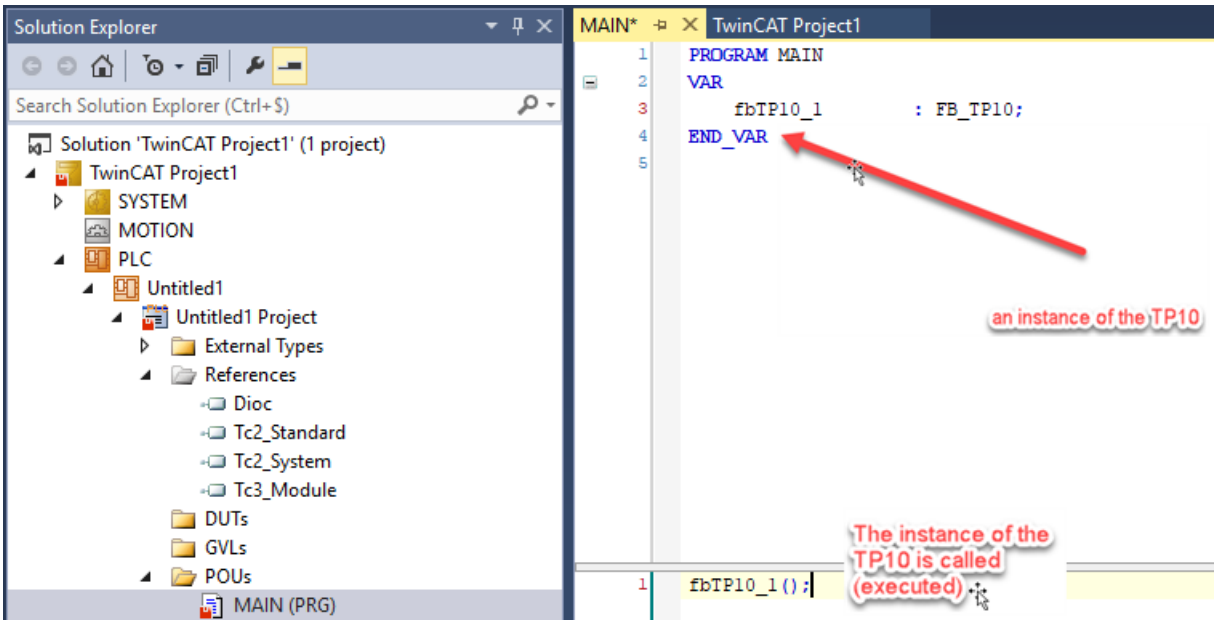
This is the least complicated method. However, when other components of the program need to run on a different cycle time or when the whole program is too large to run on a cycle time of 12ms, the second method should be used.

The cycle time can be changed under System/Tasks

Select the PLC task and change the default cycle time to 12ms by increasing the cycle ticks to 12.



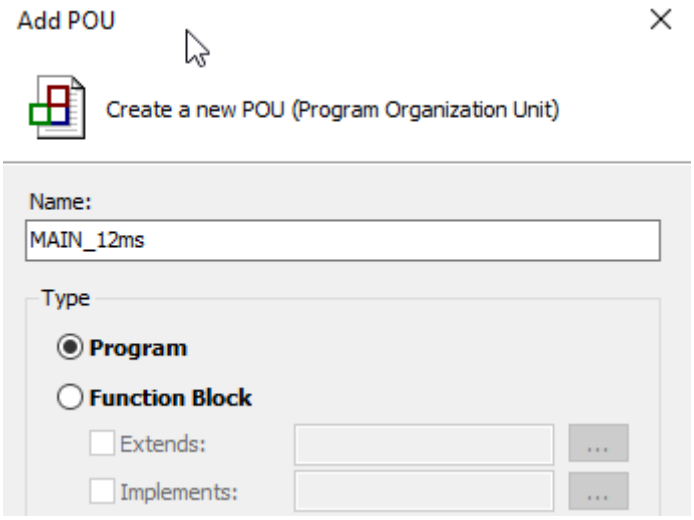
Now the DIOC instances must be called in the programs called by the PlcTask running at 12ms. For instance MAIN (PRG)



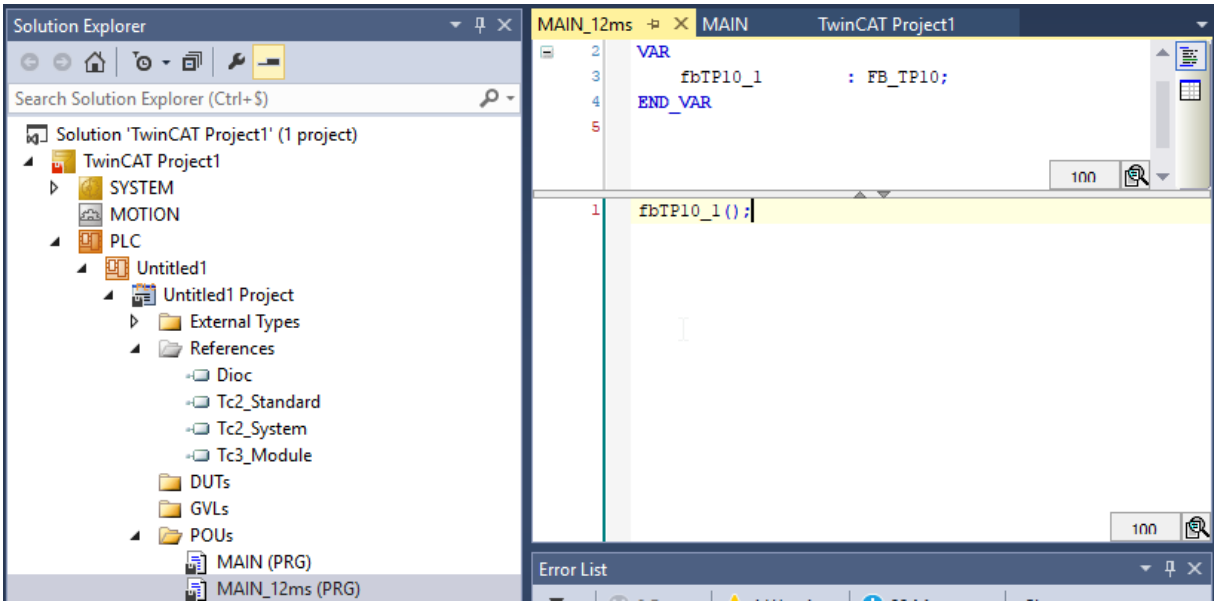
Make sure the instance is called every cycle to ensure a good communication.

Method 2: create a new task with a 12ms cycle time

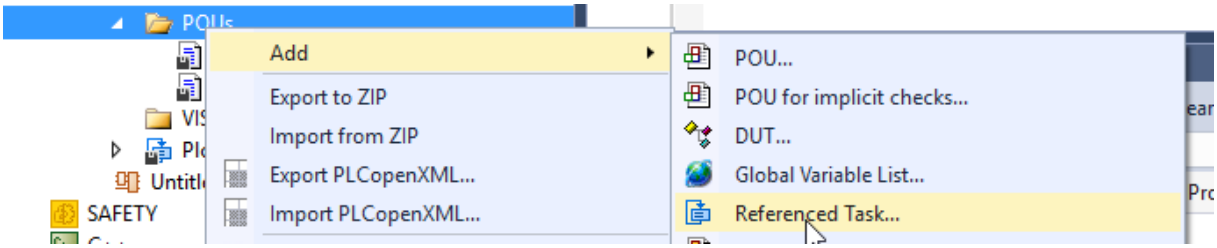
The first step is to make a new program that will be executed in the new task. Making a new program can be done by right-clicking under POU and adding a new object. The new program is named MAIN_12ms in the example.



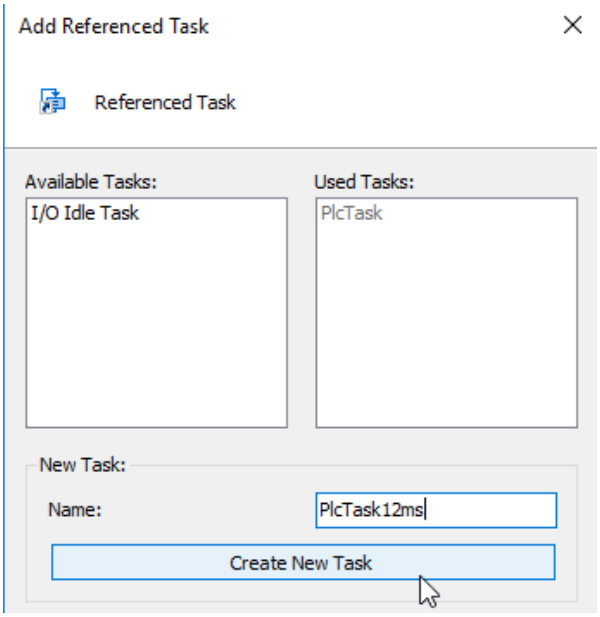
In this new program DIOC instances must be called, this means they will be executed.



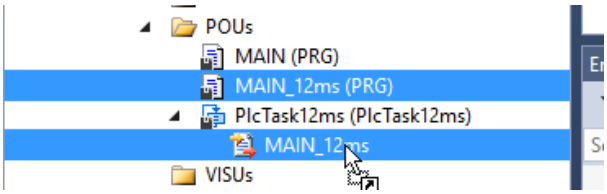
After the block is created we must create a referenced task in the plc.



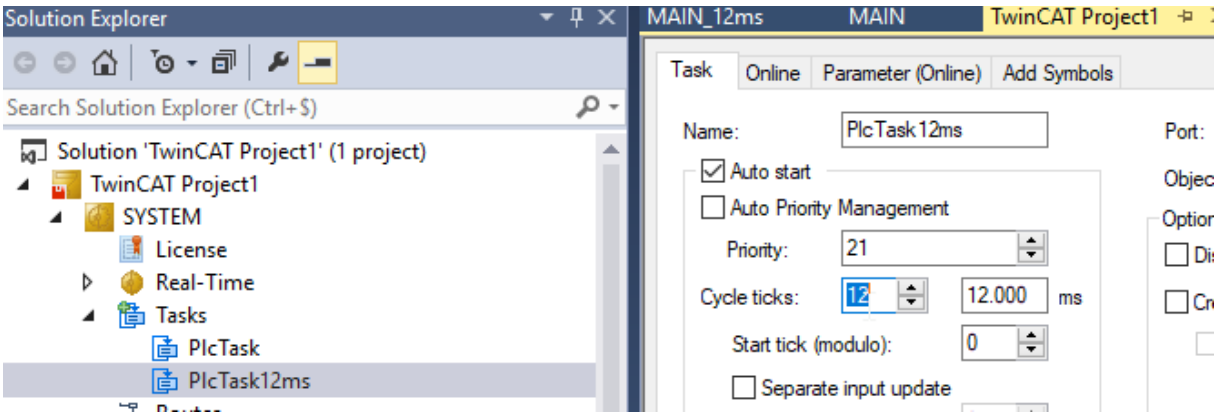
Give the name for the new task, press create new task and press open



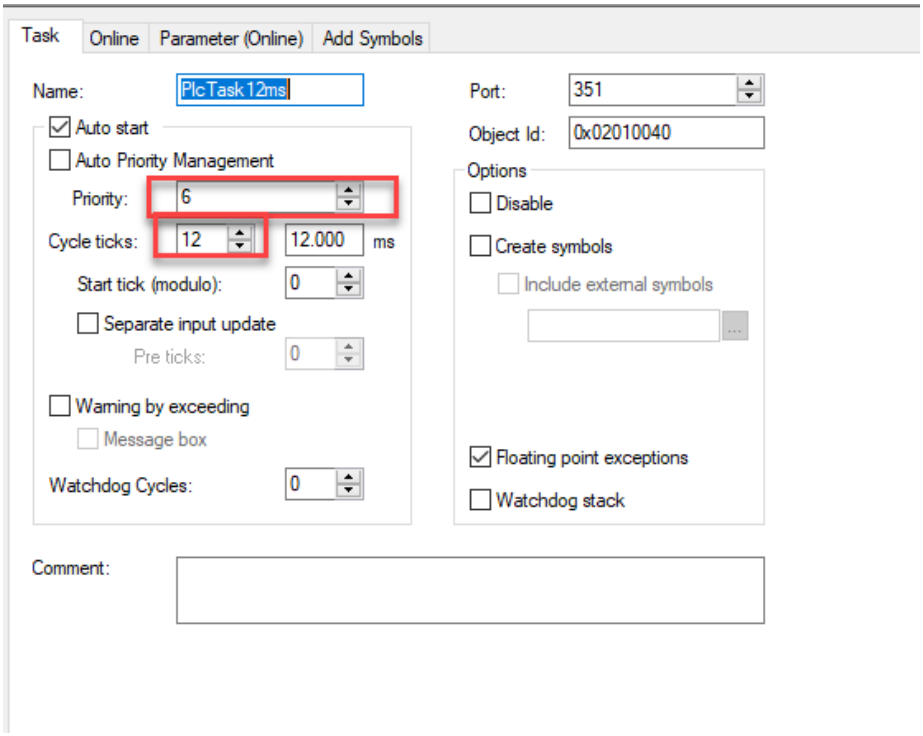
Once the task has been created drag and drop the program created before to the task

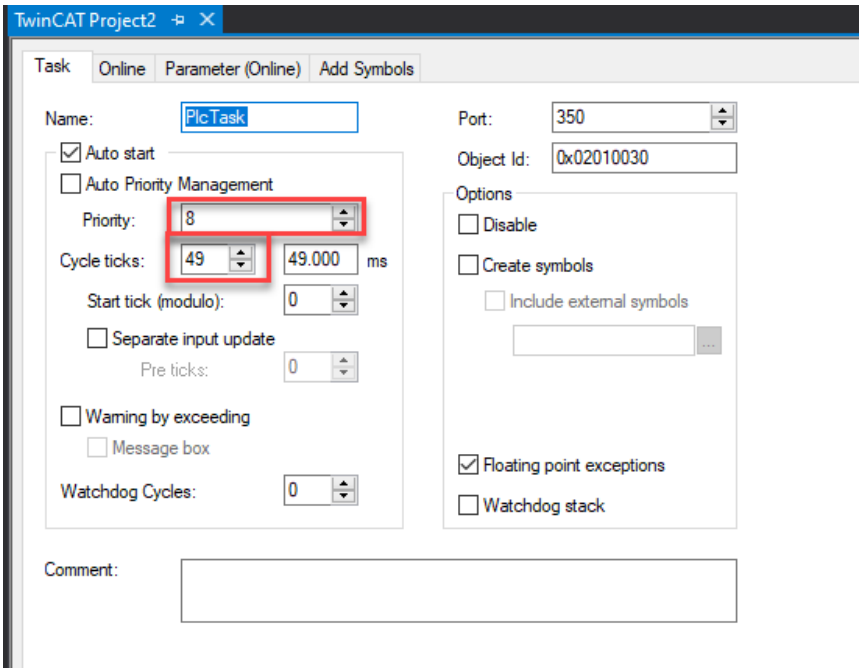


After linking set the cycle time of the new task to 12ms



The priorities of the tasks should also be set in order. The task with the lowest cycle time should always get the lowest priority number (lowest priority number means highest priority).





Step 5: Implementation of the visualisation

The DIOC_Library does NOT contain visualizations.

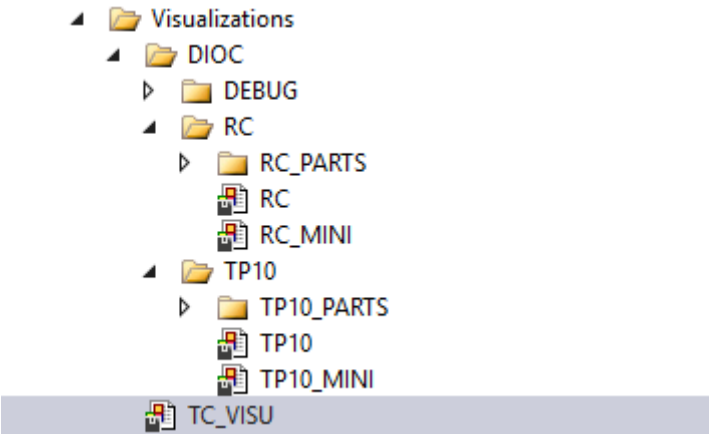
An example of visualization elements can be found in the sample that can be downloaded on the Fixsus forum. (see https://forum.fixsus.be/topics/8-TP10---RA---Technician--Programming).

Open a new instance of visual studio and choose 'open project'.

Select the archive and extract in the folder of your choice.

In the extracted project one can find 2 visualizations for each component:

- TP10 and TP10_MINI
- RC and RC_MINI



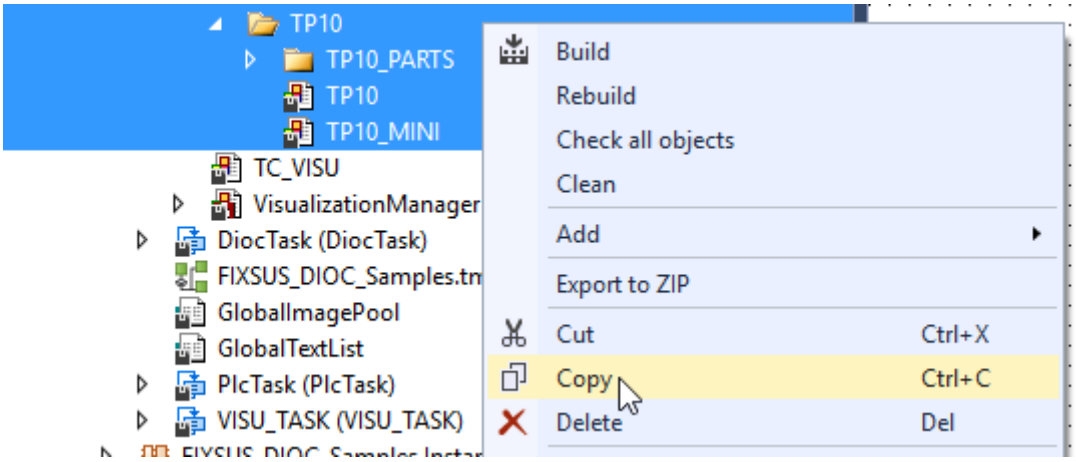
These can be used in the user program, by simple export/import or even copy/paste from the sample project to the new project.

The project also contains a sample implementation of a visualization with 1 TP10 and 1 RC called TC_VISU.

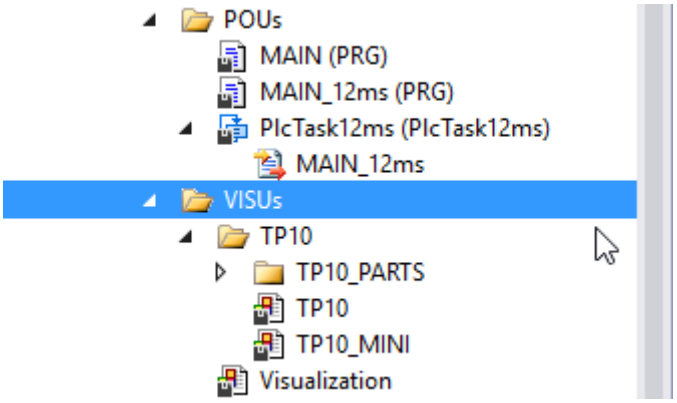
TP10 visualisation

Every TP10 can get its own visualisation where the status of the buttons and the measurements can be read. For a complete explanation of the possibilities, see chapter 'Variables of the TP10'.

In the sample project, select all components under folder TP10 and copy them



Now paste them anywhere you want in your own program

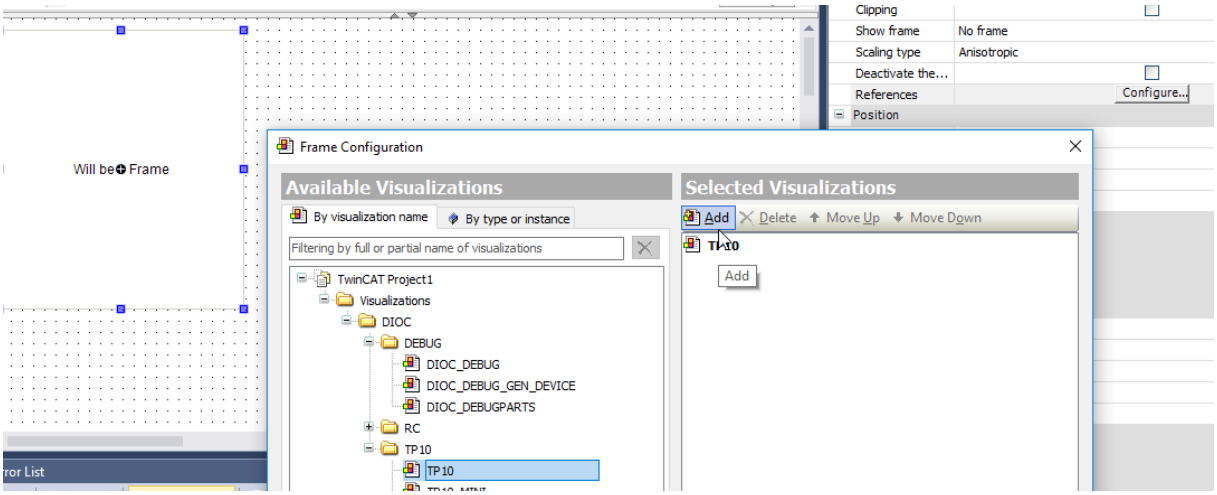


There are two possible visualisations that can be used. The 'TP10' visualisation shows the whole TP10, while the 'TP10_MINI' is a small button with which the full visualisation can be opened.

Both visualisations can be added in the same way. As an example, a ‘TP10’ visualisation is added.

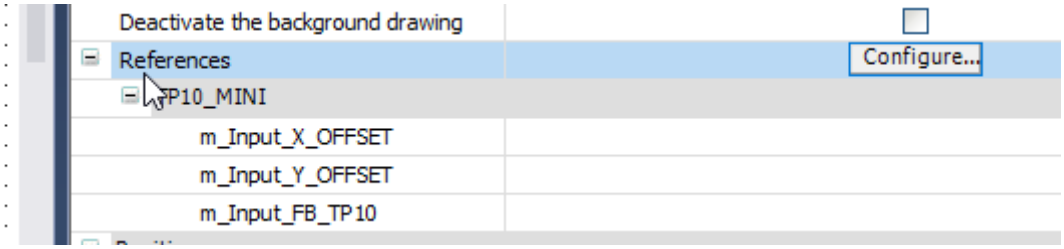
Open the visualisation screen in which the TP10 visualisation will be used.

- Add a frame
- Select TP10 or TP10_MINI and add it to the frame
- Press ok



To make sure the TP10 visualisation works as expected, the correct links must be set. This can be done in the menu of the visualisation. Select 'References' in the properties of the frame. In this menu the next configurations can be done:

- m_Input_FB_TP10 : a reference to the instance of the TP10 in the program
- m_Input_X_OFFSET and m_Input_Y_OFFSET: only used in the TP10_MINI. With these placeholders the TP10 can be moved relative to the button to open the TP10 visualisation.



Step 6: the program is to be executed after an output update

There are two VERY IMPORTANT things that have to be set to implement the TP10 and other DIOC devices

I/O at task begin

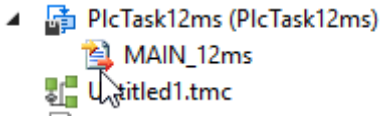
By means of a pragma we define whether a program is to be executed after an output update. This attribute replaces the TwinCAT 2 functionality of the option IO at Task begin.

The pragma must be placed in front of the PROGRAM calling the dioc blocks. In our example MAIN_12ms.

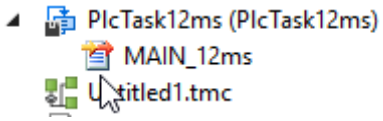
```
{attribute 'TcCallAfterOutputUpdate'}
PROGRAM MAIN_12ms
VAR
END_VAR
```

Once compiled this can be easily checked by looking at the two arrows on the task in the solution tree.

Before:



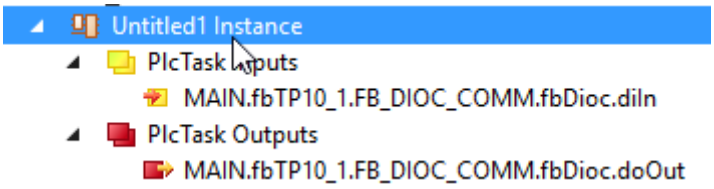
After:



Depending on the TwinCAT 3 version, it might be possible to disconnect and reconnect the program from the referenced task for the arrows to change.

Calling I/O in the correct task

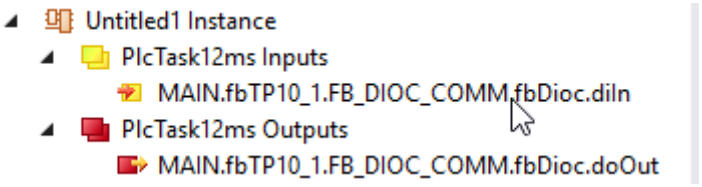
Depending on the size of the program and the situation, it can happen that the IO of the DIOC devices are not in the correct task IO. Like below, the IO is called in the task IO of MAIN and not MAIN_12ms as it should.



For this purpose as the pragma 'TcContextName' above the program where the instantiation of the dioc devices is done.

```
{attribute 'TcContextName':='PlcTask12ms'}
PROGRAM MAIN_12ms
VAR
    fbTP10_1        : FB_TP10;
END_VAR
```

After compilation it should look like



Sync unit assignment

For bigger projects, it might be a good idea to assign sync units to your I/Os. Without sync units the TP10's might not work if another I/O is missing or malfunctioning. Typically, a different sync unit should be assigned to every EtherCAT Coupler in your project. For more information on the sync units visit the Beckhoff information site.
https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_io_intro/1468206859.html&id=6053821954081018594

5. Inputs and outputs of the TP10 block

Description usage of the inputs and outputs of the TP10

The TP10 block has a lot of inputs that can change the behaviour of the TP10.

As an example below the RGB leds of the TP10 are set to red. To do this, predefined colors can be used.

```
fbTP10_1.dwRgb := fixsus_dioc.Global_DIOC_Constants.RGB_RED;
```

Other colour constants available in the DIOC library are listed below in the description of the input.

The other variables of the TP10 can also be addressed this way. The table below shows a list of all the inputs, outputs and configuration variables the TP10 has.

The inputs and outputs of the TP10 are pretty straight forward. But for an even better understanding of a full implementation of a TP10, a very simple example is implemented in the sample project.

The code of the sample project is listed at the end of the document. The code is well documented so it should be comprehensible.

If you experience troubles while implementing, please post a note or a question in the forum.

Inputs

Name	Type	Description
bRoomAnalyser	BOOL	This boolean must be true of the connected device is a Room Analyser. If this boolean is true, all buttons are disabled, except button 10. Button 10 still be used to make the Room Analyser flash green. This can be used to test the DIOC communication with the PLC.
arr_bLeds	ARRAY [0..10] OF BOOL	Every button of the TP10 has it's own led. These can be controlled by changing the values in this array. True will make the led go on, false will make the led go off. arr_bLeds [1] = led 1, arr_bLeds [10] = led 10
iIntensityLeds	INT	Value between 0 and 100 that changes the intensity of the buttonleds.
arr_bMasks	ARRAY [0..10] OF BOOL	Every button of the TP10 can be turned off, this can be done by changing the values in this array. False means the button is enabled, true means the button is disabled. arr_bMasks [1] = button 1, arr_bMasks [10] = button 10, bRoomAnalyser overrules these.
arr_sButtonComments	ARRAY [0..10] OF STRING(8)	Every button of the TP10 has a short description (maximum 8 characters) that will be displayed on the visualisation. arr_sButtonComments [1] = comment button 1, arr_sButtonComments [10] = comment button 10
bReset	BOOL	When the TP10 has to be reset, this boolean should be set to true briefly. Once bReset is false again, the reset time will count to restart the TP10.

iIntervalCO	INT	Interval for the CO measurement in seconds. This determines how fast the measurements of the CO sensor must be checked. This is standard 7 (seconds). This variable must be changed before the start of the program. This value will not be sent to the TP10 once the TP10 program is running. After a restart or reset, this value will be sent again.
iIntervalCO2	INT	Interval for the CO2 measurements in seconds. This determines how fast the measurement of the CO2 sensor must be checked. This is standard 8 (seconds). The same conditions apply as iIntervalCO.
iIntervalIllumination	INT	Interval for the illumination measurement in seconds. This determines how fast the measurement of the illumination sensor must be checked. This is standard 13 (seconds). The same conditions apply as iIntervalCO.
iIntervalRoomHumidity	INT	Interval for the humidity measurement in seconds. This determines how fast the measurement of the humidity sensor must be checked. This is standard 11 (seconds). The same conditions apply as iIntervalCO.
iIntervalRoomtemp	INT	Interval for the roomtemperature measurement in seconds. This determines how fast the measurement of the roomtemperature sensor must be checked. This is standard 3 (seconds). The same conditions apply as iIntervalCO.
iIntervalVOC	INT	Interval for the VOC measurement in seconds. This determines how fast the measurement of the VOC sensor must be checked. This is standard 5 (seconds) The same conditions apply as iIntervalCO.
dwRgb	DWORD	The TP10 has a few RGB leds that can be used to light up the TP10. This value

		determines the intensity of each led. Predefined colors can be used for this input: RGB_BLACK , RGB_NAVY , RGB BLUE , RGB_GREEN , RGB_TEAL , RGB_LIME , RGB_AQUA , RGB_MAROON , RGB_PURPLE , RGB_OLIVE , RGB_GREY , RGB_ORANGE , RGB_FUCHSIA , RGB_YELLOW , RGB_WHITE You may also create your own color. To do this a DWORD has to be made. (eg. 16#1E8FE03F) In the example 1E is a hexadecimal value for the intensity, 8F is the red value, E0 is the green value and 3F is the blue value.
bEn	BOOL	Enable bit.
bLocate	BOOL	IF TRUE: makes the TP10 flash green 3 times to know which one you are currently using.
bWallSurface	BOOL	Boolean that lowers the sensitivity of the buttons. Enable this boolean for a surface mount TP10.

Outputs

Name	Type	Description
qarr_bButtons	ARRAY [0..12] OF BOOL	Every button of the TP10 can be read. This can be done by reading the values from this array. False means the button is operated, true means the button is unoperated. qarr_bButtons [1] = button 1, qarr_bButtons [10] = button 10.
qfCO2	REAL	Value of the CO2 sensor in PPM (parts per million).
qfHumidity	REAL	Value of the humidity in percent.
qfLux	REAL	Value of the illumination sensor in lux.
qfRoomTemperature	REAL	Value of the temperature measurement in °C.
qfVOC	REAL	Value of the VOC sensor in PPB (parts per billion)

Systeminfo

Name	Type	Description
qbDeviceActive	BOOL	Boolean that indicates if the TP10 is active. True = TP10 active False = TP10 not active
qdtVersionHw	DATE	Date of the hardware version of the TP10.
qdtVersionSw	DATE	Date of the software version of the TP10.
qdtVersionReg	DATE	Date of the register version of the TP10.
qrVoltageLevelA	REAL	Voltage level of the A line in Volt.
qrVoltageLevelB	REAL	Voltage level of the B line in Volt.
qsUniqueId	STRING	Unique ID of the TP10

Sample program listing TP10

```
PROGRAM P_Samples_TP10
VAR
	(* TP10 Parameters *)
	bEn			: BOOL := TRUE;
	bReset		: BOOL;
	iIntensityLeds : INT := 50; (* Change the led intensity *)

	(* Internal vars *)
	PFButton1	: R_TRIG;	(* a rising edge trigger *)

END_VAR


(* ------- CONTROL SAMPLES *)
(* Activate the device *)
fb_TestTP10_1.bEn			:= bEn;		(* OPTIONAL Device is standard enabled
internally *)

(* Reset the device *)
fb_TestTP10_1.bReset		:= bReset;

(* Toggle led when button 1 is pressed*)
PFButton1(CLK:= fb_TestTP10_1.qarr_bButtons[1]);
IF PFButton1.q THEN
	fb_TestTP10_1.arr_bLeds[1]:= NOT fb_TestTP10_1.arr_bLeds[1];
END_IF

(* No color when button 2 is pressed*)
IF fb_TestTP10_1.qarr_bButtons[2] THEN
	fb_TestTP10_1.dwRgb		:= RGB_BLACK;
END_IF

(* Disable button 3,4 and 5
	IF arr_bMasks[1] .. arr_bMasks[10] == TRUE then these buttons are disabled *)
fb_TestTP10_1.arr_bMasks[3]	:= TRUE;
fb_TestTP10_1.arr_bMasks[4]	:= TRUE;
fb_TestTP10_1.arr_bMasks[5]	:= TRUE;

(* Red color when button 6 is pressed*)
IF fb_TestTP10_1.qarr_bButtons[6] THEN
	fb_TestTP10_1.dwRgb		:= RGB_RED;
END_IF

(* (light) Red color when button 7 is pressed*)
IF fb_TestTP10_1.qarr_bButtons[7] THEN
	(* Non default colour with max (100%) intensity *)
	fb_TestTP10_1.dwRgb		:= 16#64_64_0A_0A;
END_IF

(* (light) Red color when button 8 is pressed*)
IF fb_TestTP10_1.qarr_bButtons[8] THEN
	(* Non default colour with standard (30%) intensity *)
	fb_TestTP10_1.dwRgb		:= 16#1E_64_19_19;
END_IF
```

```
(* (light) Blue color when button 9 is pressed*)
IF fb_TestTP10_1.qarr_bButtons[9] THEN
	(* Non default colour with medium (65%) intensity *)
	fb_TestTP10_1.dwRgb	:= 16#41_19_19_64;
END_IF

(* Blue color when button 10 is pressed*)
IF fb_TestTP10_1.qarr_bButtons[10] THEN
	fb_TestTP10_1.dwRgb	:= RGB_BLUE;
END_IF

(* Change the led intensity *)
fb_TestTP10_1.iIntensityLeds	:= iIntensityLeds;

(* Interval update times for the sensors *)
fb_TestTP10_1.iIntervalRoomtemp;		(* Default := 3 *)
fb_TestTP10_1.iIntervalRoomHumidity;		(* Default := 11 *)
fb_TestTP10_1.iIntervalCO2;			(* Default := 8 *)
fb_TestTP10_1.iIntervalCO;			(* Default := 7 *)
fb_TestTP10_1.iIntervalVOC;			(* Default := 5 *)
fb_TestTP10_1.iIntervalIllumination;		(* Default := 13 *)
fb_TestTP10_1.iIntervalIR;			(* Default := 17 *)

(* ------- FEEDBACK SAMPLES,
	* these are the measured values a TP10 can return
	* be aware that it might take along time before the data is available after starting
	*)
fb_TestTP10_1.qfRoomTemperature;
fb_TestTP10_1.qfVOC;
fb_TestTP10_1.qfCO2;
fb_TestTP10_1.qfHumidity;
fb_TestTP10_1.qfLux;
fb_TestTP10_1.qfInfrared;

(*Analyse data*)
fb_TestTP10_1.qdtVersionHw;
fb_TestTP10_1.qdtVersionSw;
fb_TestTP10_1.qdtVersionReg;
fb_TestTP10_1.qsUniqueId;
fb_TestTP10_1.qrVoltageLevelA;
fb_TestTP10_1.qrVoltageLevelB;
fb_TestTP10_1.qbDeviceActive;
```