Quick start guide

Configuring a SafetyBridge technology (V3) system on a Schneider controller using Unity

This user manual is valid for:

Designation	Revision	Order No.
IB IL 24 LPSDO 8 V3-PAC	00/101/100	2701625
IB IL 24 PSDI 8-PAC	00/202	2985688
IB IL 24 PSDO 8-PAC	01/201/100	2985631
IB IL 24 PSDOR 4-PAC	00/201/100	2985864
IB IL 24 PSDO 4/4-PAC	01/201/100	2916793
IB IL 24 PSDI 16		2700994

Introduction

1.1 Purpose of this manual

This quick start guide describes how to integrate SafetyBridge Technology V3 modules in a Modbus TCP system into an Schneider M340 controller.

The document does not describe the complete configuration of a system or how to create a project under Unity. It only describes what has to be observed with regard to SafetyBridge Technology V3.

For easier integration, you can refer to the chapter ‘Easy integration if SBT V3’ at the end of this quick start.

For additional information, please refer to the documents listed in Additional documentation.

This Quickstart refers to several Unity application examples. Here is how to choose the best one, according to your knowledge of the SafetyBridge technology and the type of installation you need to control :

Name of the Unity application	Unity version	Purpose
SBT_V3_Quickstart	Pro S	Not simplified and detailed application, using IOScanning
SBT_V3_Easy	**Pro S**	**Simplified version, multiple islands, multiple PLCs, using Read_Var and Write_Var instead of IOScanning**
SBT_v3_Easy_10_bks	Pro XL	Simplified version, single island, multiple BKs, using IOScanning for Diagnostic, and Read_Var and Write_Var for process data refresh
SBT_v3_Easy_10_bks_Fast	Pro XL	Simplified version, single island, using IOScanning for Diagnostic, and simultaneous Read_Var and Write_Var for faster process data refresh
SBT_v3_Easy_Premium	Pro XL	Simplified version, single island, using IOScanning for Diagnostic, and Read_Var and Write_Var for a Premium PLC

1.2 Requirements

Knowledge

Knowledge of the following is required:

– The target system (Modbus TCP)
– The configuration of Inline in Modbus TCP network
(see www.phoenixcontact.com)
– The components used in your application
– The Unity software used
– The Microsoft Windows operating system

Hardware

To start up the example system, the following hardware is required:

– M340 (or Premium), with Modbus TCP interface (see example)
– IL ETH BK DI8 DO4
– Programming device/PC
– I/O devices (Safety devices) used in the example project
(see Section "Example bus configuration" on page 3-1)

Software

To start up the example system, the following software is required:

– Unity S (or higher) or unity XL for some examples
– SafetyBridge Technology V3 example as add-on instructions
– Microsoft Windows
– SAFECONF V2.8 or later from -Phoenix Contact (software for configuring the safety logic and parameterizing the channels)
This is available on the Internet at www.phoenixcontact.net/catalog.
– Internet Browser such as Internet Explorer or Firefox to access to the IL ETH BK DI8 DO4 web pages

1.3 Additional documentation

Comprehensive information on Modbus TCP is available on the Internet at www.modbus.org.

Please refer to:

– The documentation for the Unity software
– The documentation for the components used in your application
– The documentation for the function blocks used

The documentation for the SafetyBridge Technology V3 modules used must be strictly observed.

Description	Type	Order No.
User manual: Inline module with integrated safety logic and safe digital outputs	UM EN IB IL 24 LPSDO 8 V3-PAC	2992051
User manual: Inline module with safe digital inputs	UM EN IB IL 24 PSDI 8-PAC	2910457
User manual: Inline module with safe digital outputs	UM EN IB IL 24 PSDO 8-PAC	2910538
User manual: Inline module with safe digital inputs	UM EN IB IL 24 PSDI 16-PAC	2992158
User manual: Inline module with safe digital outputs	UM EN IB IL 24 PSDO 4/4-PAC	2910554
User manual: Inline module with safe digital relay outputs	UM EN IB IL 24 PSDOR 4-PAC	2910729

The documentation for Phoenix Contact devices is available on the Internet at www.phoenixcontact.net/catalog.

1.4 Safety hotline

Should you have any technical questions, please contact our 24-hour hotline.

Phone: +49 5281 9462777

E-mail: safety-service@phoenixcontact.com

Pre-requisites

To use this quick start, a good knowledge of the software Unity and the network Modbus TCP is necessary.

2 Integration of a SafetyBridge Technology V3 system in three steps

2.1 Safety with the SafetyBridge Technology V3 system

Within a SafetyBridge Technology V3 system, safety can only be ensured by using the modules of this system (IB IL 24 LPSDO 8 V3-PAC and 1 to 16 satellites). None of the other components in the overall system are safety-related components. Errors at non-safety-related components or errors during integration of the SafetyBridge Technology V3 system are reliably detected by the SafetyBridge Technology V3 system components. These errors only reduce the system availability but not the system safety.

No safety controllers are required for the implementation of safety functions.

2.2 Integration of a SafetyBridge Technology V3 system

A SafetyBridge Technology V3 system can be integrated into an existing system in three steps.

Integration of a SafetyBridge Technology V describes the steps for integrating a SafetyBridge Technology V3 island.

Table 2- 1 Integration of a SafetyBridge Technology V3 island

Step	Process	Safety-related	See ...
1	**Configure the safety logic (-SAFECONF >= 2.8)**		
	– Configure the safety island (island number, satellites)	Yes	Page 3-6
	– Parameterize the I/O channels of a safety island		Page 3-8
	– Configure the safety functions		Page 3-11
	– Export the configuration and parameter data record		Page 3-14
2	**Integrate the SafetyBridge Technology V3 modules into the controller (M340)**		
		No	Documentation for the controllers and Unity
	– Create a project		Page 3-3
	– Import SBT add-on instructions		Page 3-18
	– Add SafetyBridge Technology V3 operation to the standard application program		Page 3-19
	– Import the configuration and parameter data record into the M340 project in Unity (*.XDB) output format		Page 3-22
3	**Install the SafetyBridge Technology V3 modules**		
	Install the SafetyBridge Technology V3 modules (hardware) (including island and satellite number settings)	No	Page 3-25 and user manuals for the modules used
	Overall safety validation	Yes	

Components and steps for integrating the SafetyBridge Technology V shows the hardware and software components used and the steps for integrating a SafetyBridge Technology V2 system.

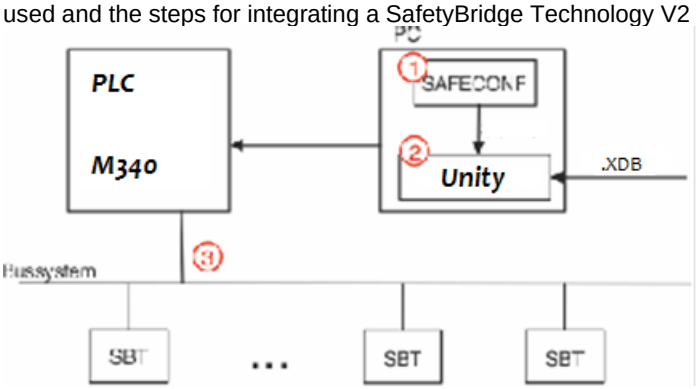


Figure 2- 1 Components and steps for integrating the SafetyBridge Technology V3 system

Key:

1	Step 1: configure the safety logic
2	Step 2: integrate the SafetyBridge modules into the controller
3	Step 3: install the SafetyBridge modules

PC	PC with -SAFECONF and Unity
SAFECONF	Software for configuring the safety logic (configuration of the safety function and parameterization of the channels)
Unity	Engineering software
.XDB	-SAFECONF add-on module (DFB) Configuration and parameter data record created with -SAFECONF; this must be imported into the Unity project as a DFB module; structured text according to IEC 61131
SBT Library	SafetyBridge add-on instruction (DFB) Add-on instruction for handling SafetyBridge Technology V3 modules from -Phoenix Contact – Download of the configuration and parameter data record from the standard control system to the IB IL 24 LPSDO 8 V3-PAC – Cyclical routing of the SafetyBridge Technology V3 data flow – Safety I/Os monitoring
Bus system	Modbus TCP
SBT	Modules of the SafetyBridge Technology V3 system

3 Example

This section describes the use of SafetyBridge Technology V3 modules in Modbus TCP. Only the safety modules are described in detail. You can use standard modules in the -Inline station, but these are not described in detail here.

3.1 Example bus configuration

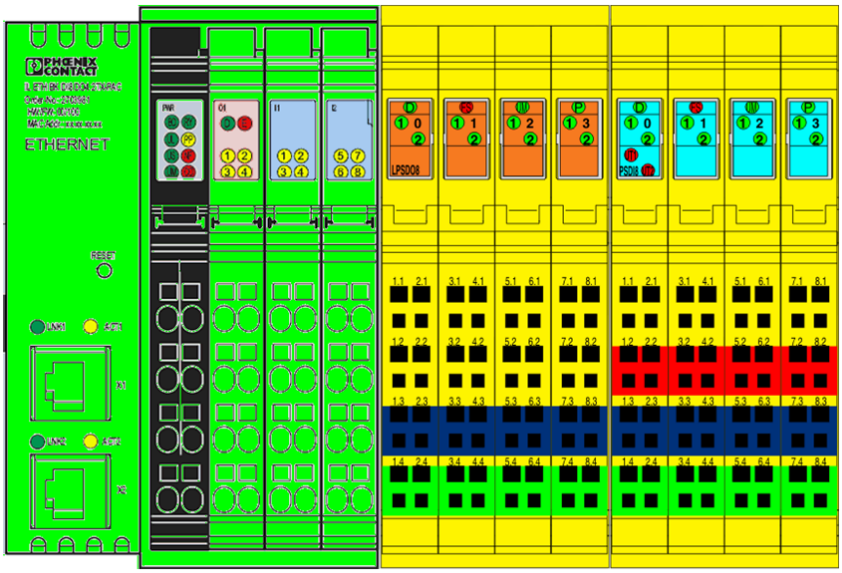


Figure 3- 1 Example bus configuration

Key:

S1	Safety switch; emergency stop (EStop/button S1)
K1 (R)	Forcibly guided N/C contact for monitoring the state of the relay (readback contact). The example does not describe this readback.

Devices used in the example bus configuration

Bus coupler

IL ETH BK DI8 DO4 2TX-PAC	Bus coupler

Safety modules

IB IL 24 LPSDO 8 V3-PAC	-Inline module with integrated safety logic and safe digital outputs
IB IL 24 PSDI 8-PAC	-Inline module with safe digital inputs

Additional I/O modules:

I/O modules

IB IL 24 PSDO 8-PAC	Inline module with safe digital outputs
IB IL 24 PSDOR 4- PAC	Inline module with safe digital relay outputs
IB IL 24 PSDO 4/4- PAC	Inline module with safe digital outputs

3.2 Step 1: configuring the safety logic (-SAFECONF)

This section only describes the steps that are essential for the SafetyBridge Technology V3 system. Therefore, you will not find all of the screen views shown here. If you have any questions about -SAFECONF, please refer to the online help or software documentation.

The first operation in Safeconf is to include the XML file dedicated to Unity, for Safeconf is able to create the DFB wich will be imported in Unity.

This file is called UNITY.XML. It is provided in the quickstart package. In the case it does not appear in the Safeconf parameters, it will be necessary to include it in the dedicated directory (see the examples below and adapt the path):

Windows7:
C:\ProgramData\Phoenix Contact\SAFECONF\2_x\OutputFormats
Windows XP:
C:\Dokumente und Einstellungen\All Users\Anwendungsdaten\Phoenix Contact\SAFECONF\2_x\OutputFormats
XP on VM :
C:\Documents and Settings\All Users\Application Data\Phoenix Contact\SAFECONF\2_x\OutputFormats

3.2.1 Creating a project

Use -SAFECONF to configure and parameterize the SafetyBridge Technology V3 system. A configuration and parameter data record is subsequently created and saved as a dfb file for work involving an M340 controller.

Table 3‑1 Output format for the configuration and parameter data record for work involving Unity

Output format	Handling the file
UNITY (*.XDB)	Import the Unity file (*.XDB) with structured text according to IEC 61131 into the Unity project as a DFB. The Unity project is the overall project. It contains both the standard user program and the imported safety logic. If a change is made to the -SAFECONF project, a new file is created in Unity output format. Once this has been done, reimport this file into Unity and generate an overall project. This means that whenever a change is made to your safety logic in SAFECONF, you must also adjust the Unity project.

- Open the -SAFECONF software (Version 2.8 or later).
- Create a new project with the Project Wizard. To do this, select "File... New Project".
- Specify the name and storage location for the project.

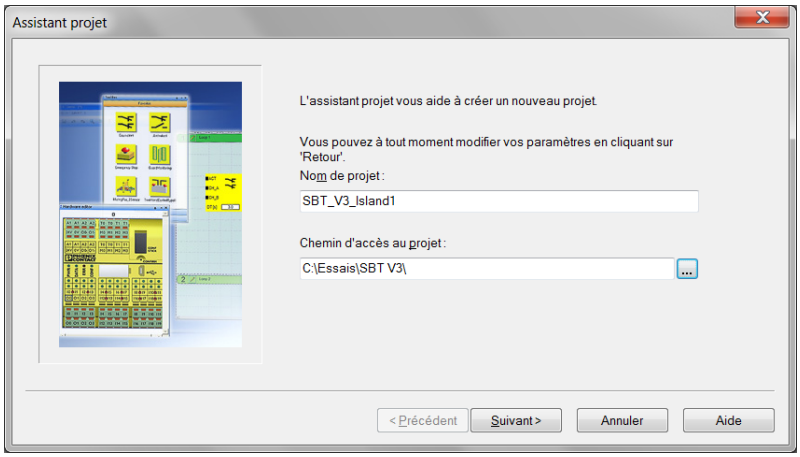


Figure 3- 2 Creating the project name and path

- Select the IB IL 24 LPSDO 8 V3-PAC master device for working in the SafetyBridge Technology V3 system.

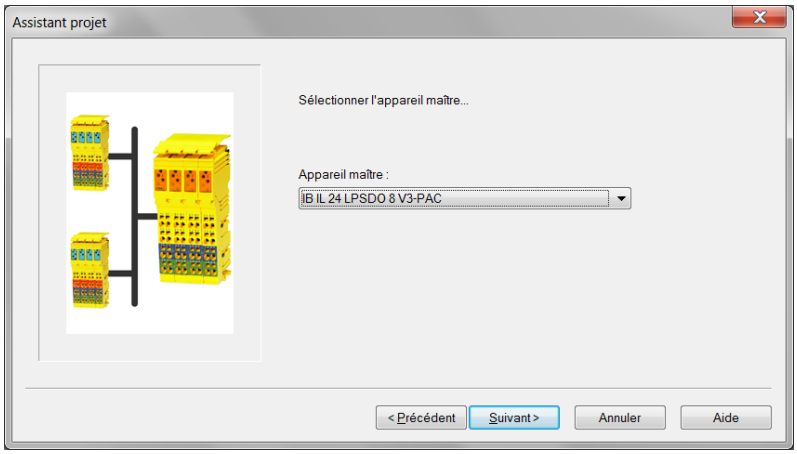


Figure 3- 3 Selecting IB IL 24 LPSDO 8 V3-PAC

- Select the file format in which the configuration and parameter data record is to be output (see Output format for the configuration and parameter data record for work in).
  If you are working with Unity, you need the Unity output format.

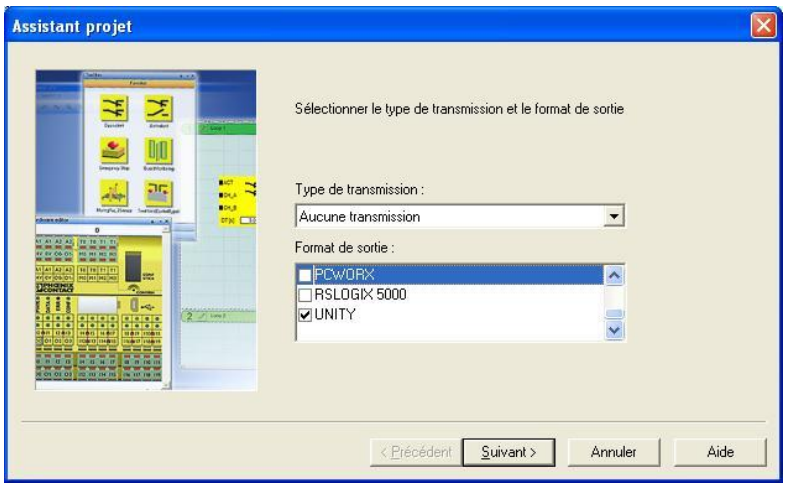


Figure 3- 4 Selecting the output format

- Enter a complete description of the project.

Table 3- 2 Describing the project

Field	Contents
Custom description	Maximum of 4 characters
Custom version	Maximum of 4 characters

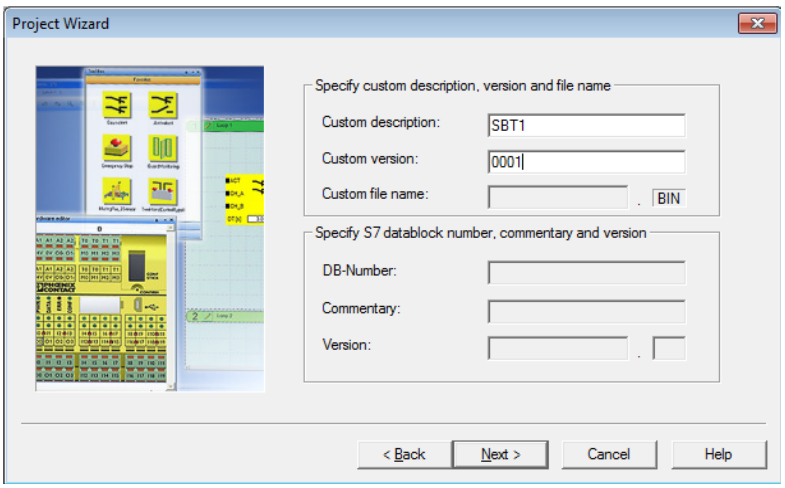


Figure 3- 5 Describing the project

- Complete the project creation process.

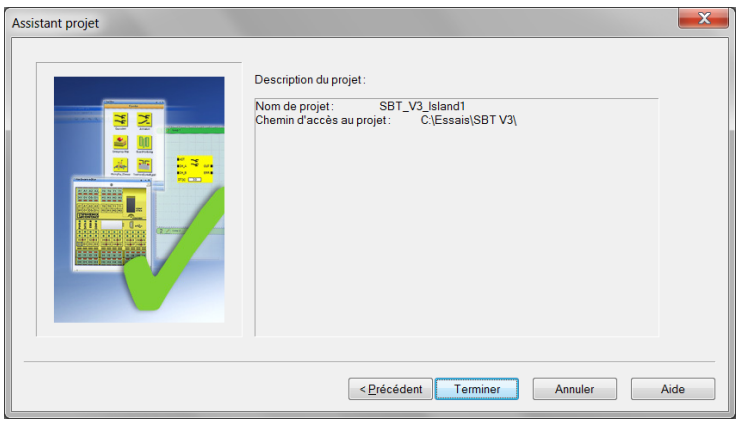


Figure 3- 6 Completing the project creation process

3.2.2 Configuring the safety island

In -SAFECONF, devices are shown in the form of a safety island view rather than as a network view. Configuration of a safety island is independent of whether it is associated with one -Inline station or distributed across several -Inline stations.

When the project is completed, a window opens prompting you to enter the island number.

- Enter an island number (1 in the example).

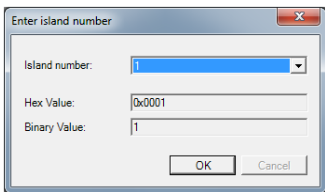


Figure 3- 7 Entering an island number

- Specify a password of at least six characters for the project (123456 in the example).

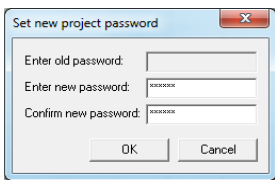


Figure 3- 8 Specifying a password

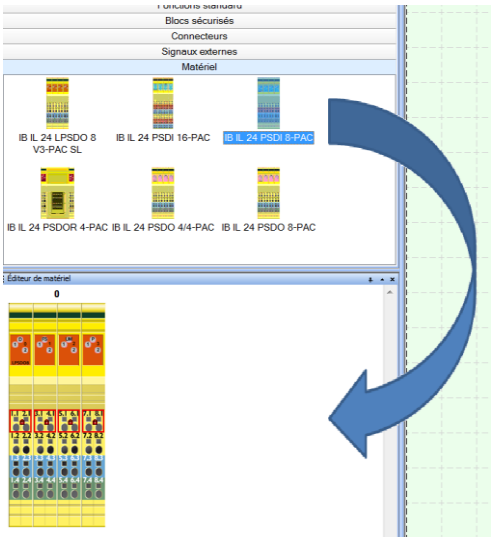
- Configure the hardware structure of the SafetyBridge Technology V3 island. To do this, use drag & drop to move the relevant modules from the "Hardware" toolbox to the hardware editor.
- 

Figure 3- 9 Drag and drop devices from the hardware catalog to the hardware configuration.

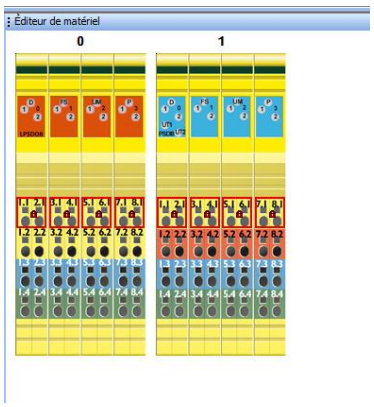


Figure 3- 10 Configuring the SafetyBridge island

3.2.3 Parameterizing the I/O channels of a safety island

NOTE:

For two-channel assignment, use the same parameterization for both channels.

Parameterize the input and output channels of the SafetyBridge Technology V3 modules. Two options are available:

1 In the hardware editor, double-click on the module. This opens the window for parameterizing the entire module.

2 In the hardware editor, double-click on a terminal point. This opens the window for parameterizing the selected terminal point.

Parameterize the modules as described in the user manual. This is available on the Internet at www.phoenixcontact.net/catalog or as online help via the module context menu (right-click on the module in the hardware editor).

- Parameterize the output channels of the IB IL 24 LPSDO 8 V3‑ PAC.

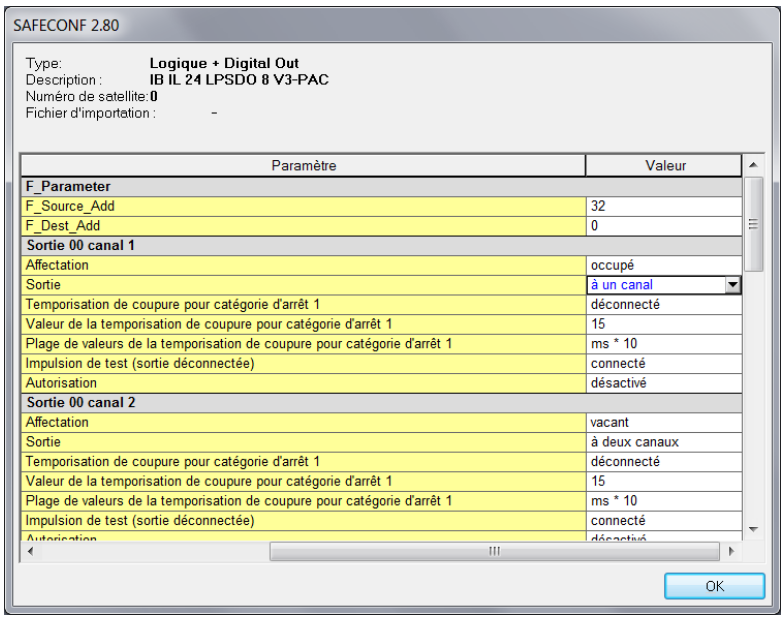


Figure 3‑ 11 Parameterization of IB IL 24 LPSDO 8 V3‑ PAC: output 00 channel 1 (Here: parameterization by double-clicking on the module)

The values F_Source_Add and F_Dest_Add are automatically entered. F_Source_Add is derived from the island number, F_Dest_Add is derived from the island and satellite number.

- Parameterize the input channels of the IB IL 24 PSDI 8- PAC.

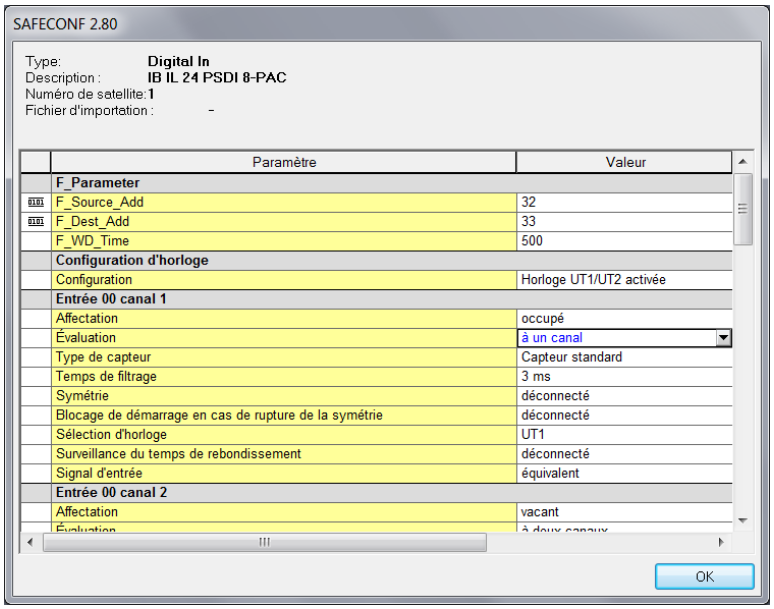


Figure 3- 12 Parameterization of IB IL 24 PSDI 8- PAC: input 00 channel 1 (Here: parameterization by double-clicking on the module)

Inputs or outputs parameterized for 2-channel operation are indicated by a lock.

Figure 3- 12

You can specify the F_WD_Time (in ms) according to your application. The default value is 150 ms. When using Modbus TCP, make a test with 500 ms, and then adjust the value to the measured transmission times of each device.

You can only set the clock configuration for the channels by clicking on the module.

Figure 3- 13 Automatically entered data and clock configuration

3.2.4 Configuring the safety function

- Configure the safety function.

Configure the safety function by using drag & drop to move the elements from the individual areas of the toolbox to the workspace.

Various sources are available for safe and standard signals. In the example, this means the following sources:

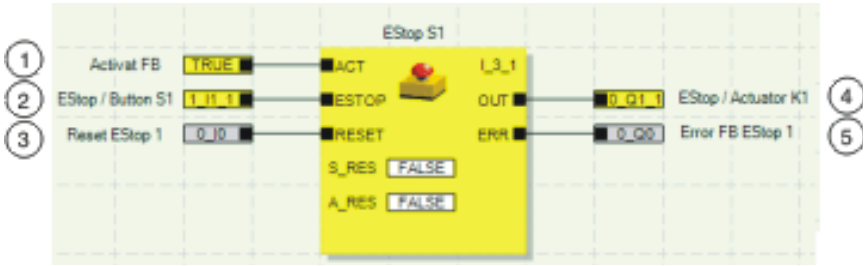


Figure 3- 14 Sources of the safe signals

1 "Safe Functions" toolbox

2 IB IL 24 PSDI 8- PAC hardware editor

3 "External signals" toolbox; standard signals from the standard control system

4 IB IL 24 LPSDO 8 V3- PAC hardware editor

5 "External signals" toolbox; standard signals to the standard control system

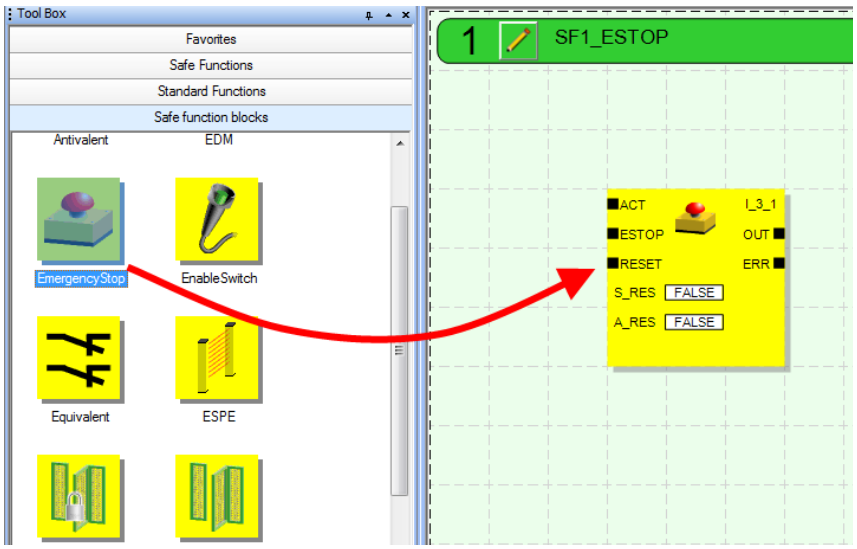


Figure 3- 15 Inserting a function block from the "Safe function blocks“ toolbox

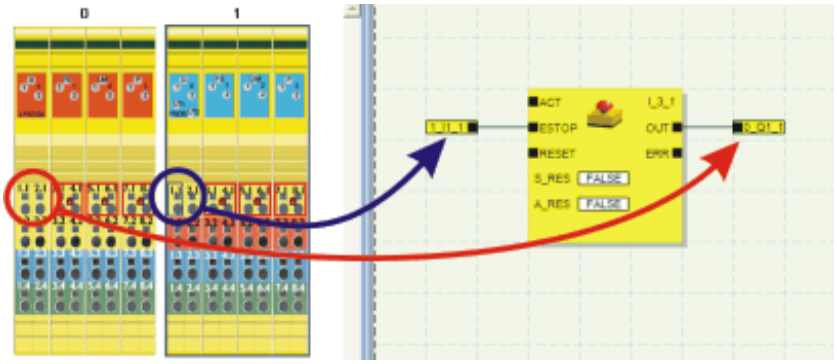


Figure 3- 16 Inserting a safe output from the hardware editor using drag & drop

When you use drag & drop to place the safety module terminal point directly onto a function block input or output (as illustrated for an output in Inserting a safe output from the hardware editor using drag & drop), the connecting line is created automatically.

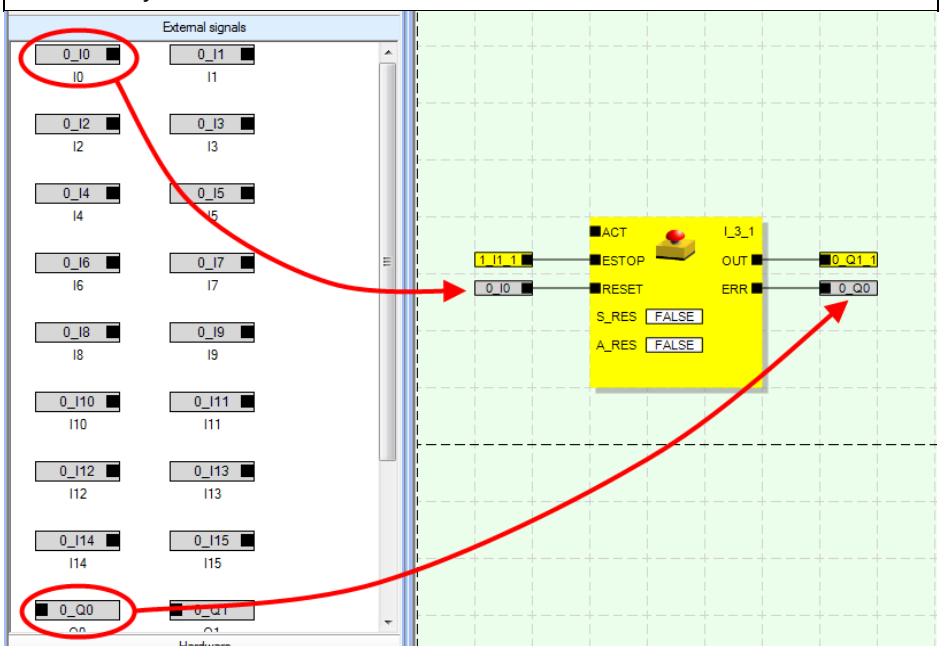


Figure 3- 17 Inserting an external signal from the "External signals" toolbox

Move your mouse over an external signal to display the corresponding tooltip.

Figure 3- 18 Tooltip

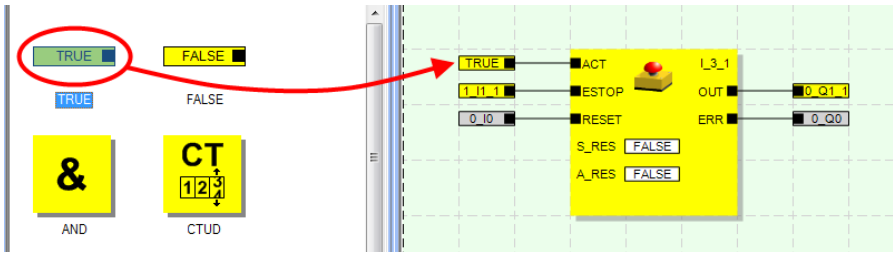


Figure 3- 19 Inserting a safe function (TRUE) from the "Safe Functions" toolbox

- You can add comments to both the function block and the signals.
  To do this, select the "Insert Comment" command in the context menu (right mouse button) for the function block or a signal.
  Then move the comment to the desired position.

You can see the entire commented safety function for this example in Configured safety function with comments

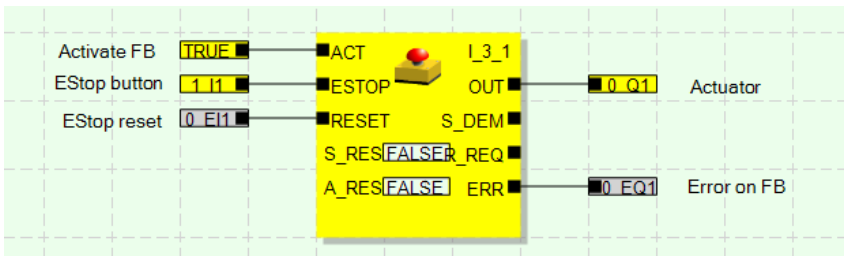


Figure 3- 20 Configured safety function with comments

3.2.5 Exporting the configuration and parameter data record

- Check the project. To do this, select the "Project... Check Project" command.

A message window opens displaying the progress of the check.

Once the check is complete, the amount of program memory used by the program is displayed.

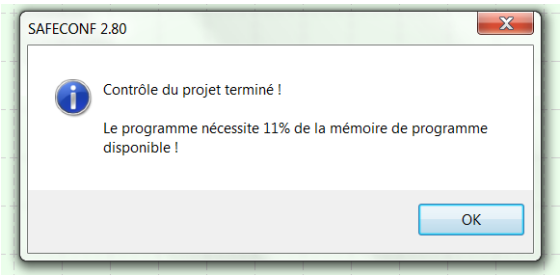


Figure 3- 21 Program memory used

If the check is completed without errors, the configuration and parameter data record is created as an *.XDB file. This is saved in the path that you have entered for the project (see Creating the project name and path) in the "FileOutput" folder.

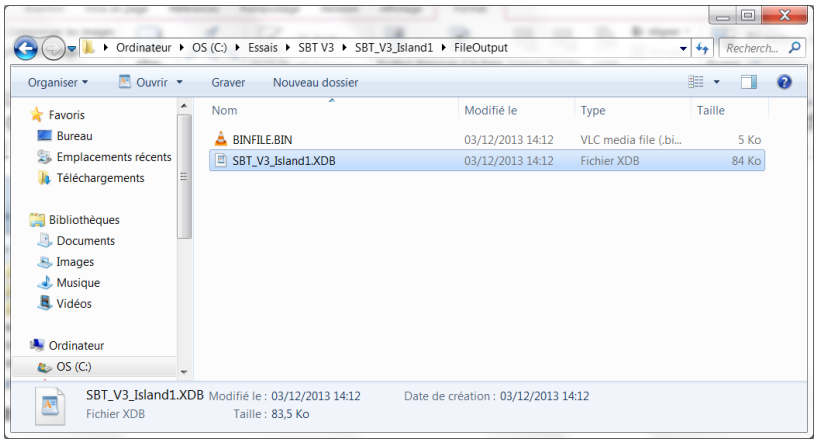


Figure 3- 22 XDB file

The XDB file is later loaded in the standard control system as a DFB.

This completes step 1 "configuring the safety logic".

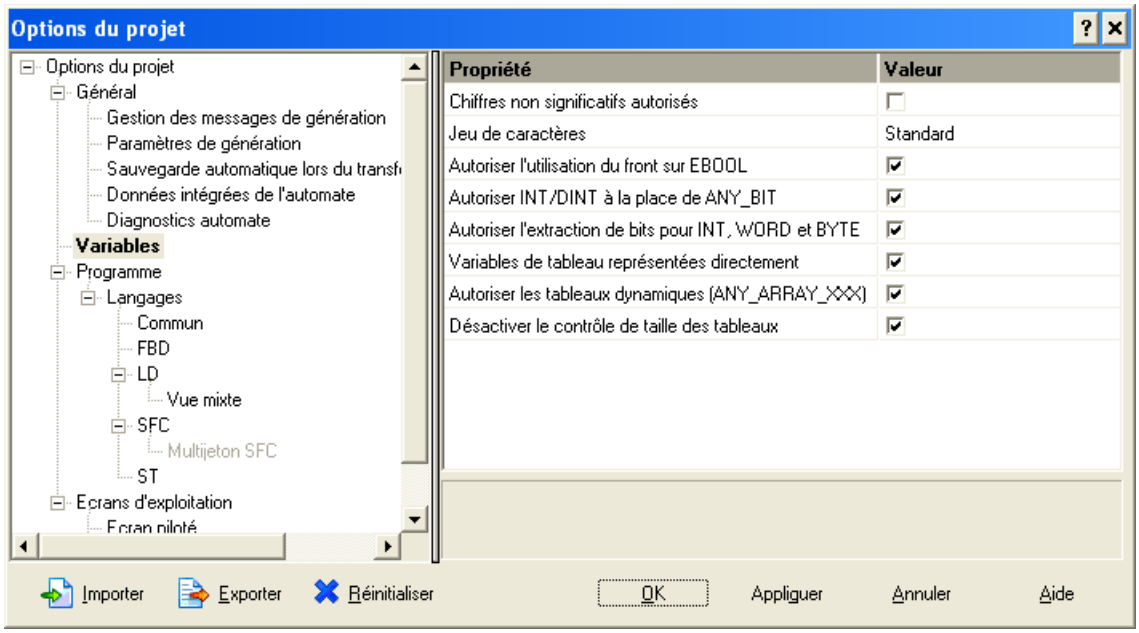
3.3 Step 2: integrating the SafetyBridge Technology V3 modules into the controller (M340)

This section only explains fundamental steps that are relevant to the SafetyBridge Technology V3 system.

A good knowledge of the Unity software is needed to use this quickstart.

To be able to use the Unity example, Unity must be configured as described here :

Project options :



3.3.1 Creating/opening a project

- In Unity, create a new project or open an existing project. For the example shown, create the project with a PLC type M340 with Modbus TCP controller. The best solution is to start from the example and to modify and adapti it for your own application.

3.3.2 Configuring the IL ETH BK DI8 DO4 with -SafetyBridge V3 modules

For further use in the Unity project, you will require the addresses and offsets created during configuration for the individual SBT modules within a Modbus TCP station, as specified in Table 3-4.

To configure the IL Modbus TCP head station, proceed as specified in the application note.

Generic Ethernet module configuration with -SafetyBridge example:

Table 3- 3 Generic Ethernet module configuration with –SafetyBridge

Number of I/O words	IN data size	IN base address	OUT data size	OUT base address
IL ETH BK DI8 DO4	1	0 (8000)	1	384 (8029)
LPSDO 8 V3	24	1 (8001)	24	385 (8030)
PSDI	4	25 (8025)	4	409 (8054)
Diag/Command registers	4	7996	1	2006

3.3.3 Integrating a Modbus TCP BK into the Unity project

Here is an example of IOScanning contracts.

The Modbus TCP BK is not imported into the Unity software. It has just to be mapped in the IOScanning window

The 3 first contracts can be merged together. They are separated here to give more details

Contract 4 is for the Diagnostic registers (7996 to 7999)

Contract 5 is for the Command register (quit netFail, mode PP)

See the User Manual of the IL ETH BK DI8 DO4 for more details.

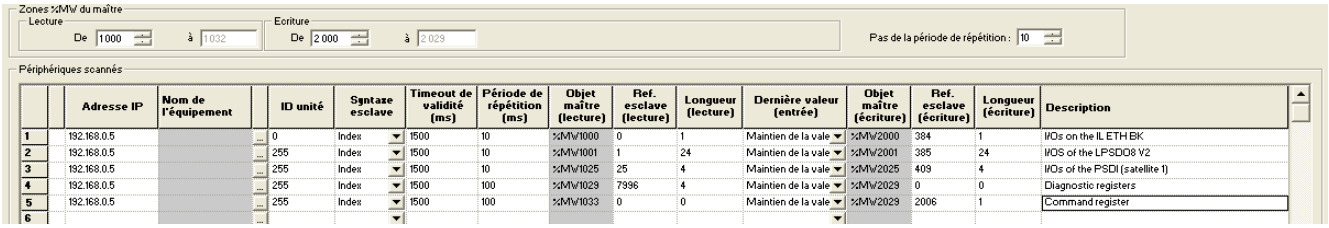


Figure 3- 23 Mapping the new module

The web server of the IL ETH BK DI8 DO4 also shows the mapping. But it only shows the dynamic addresses, and not the static addresses :

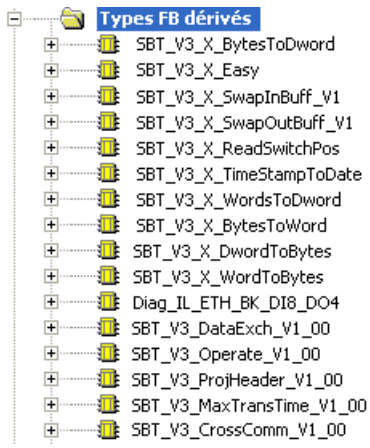
Bus Configuration

Baudrate: 500 kBaud

Number	Symbol	Description	Modbus Process Data Register Address	
			IN	OUT
0		IL ETH BK DI8 DO4		
1		Module with 4 digital outputs.	-	8029
2		Module with 8 digital inputs.	8000	-
3		Module with 384 digital inputs and outputs.	8001...8024	8030...8053
4		Module with 64 digital inputs and outputs.	8025...8028	8054...8057

3.3.4 Importing SBT DFBs

You will need the following DFBs in your program: (if you start with the example, these FBs are already integrated in the application)



The main DFB dedicated to the SBT are called SBT_V3_.....

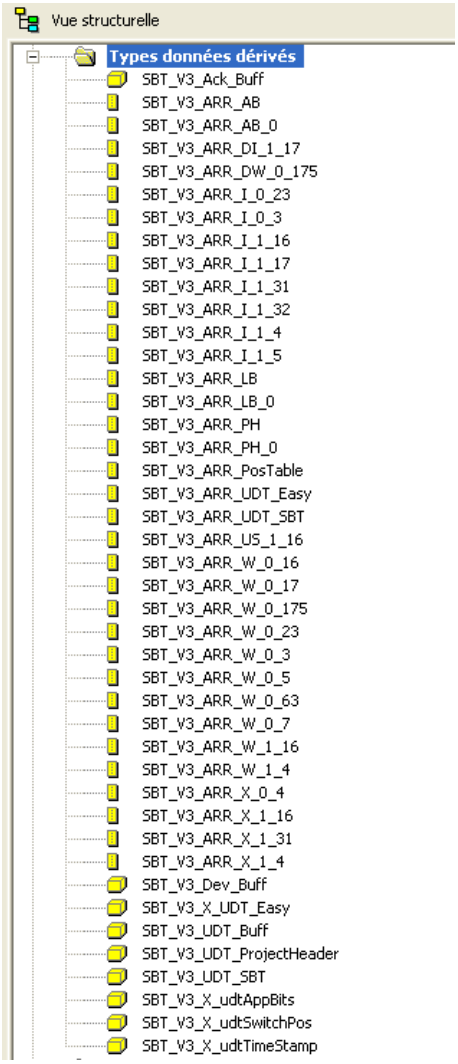
The DFB with a name SBT_V3_X_.... are auxiliary DFB.

The DFB Diag_ILETH_BK_DI8_DO4 permits to restart automatically the IL ETH BK and gets some diagnostic information about the local bus of the IL ETH BK…

The other DFB are used by the application example, and should also be used in your project

These DFB are included in the Unity project delivered as an example, They can also be imported directly into Unity.

You will also need the specific variable types listed here :



These types are directly created when a DFB is imported. They are also included in the Unity example.

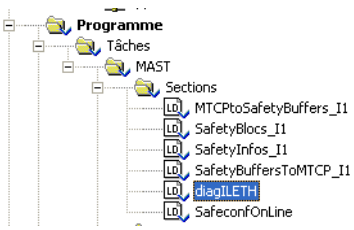
The types with a name as SBT_V3_X_.... are auxiliary types.

The types with a name as SBT_V3_... are main types used by the DFB.

The Unity example includes a section, wich shows all the necessary blocks to be used, and how to link them together :

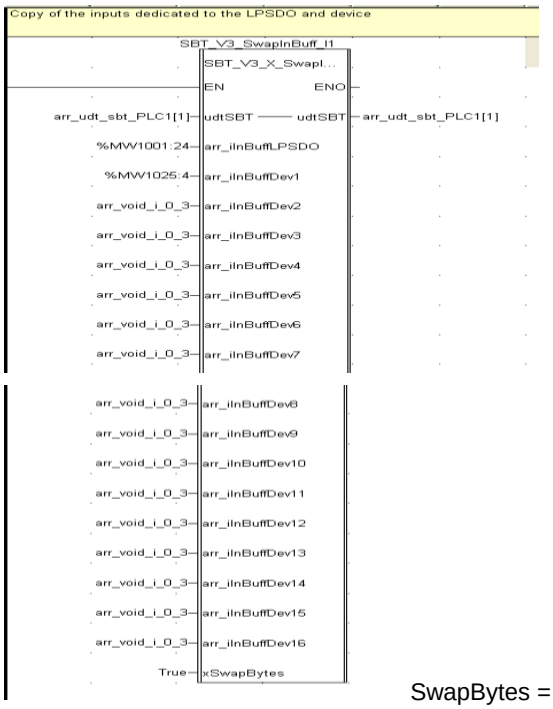
The example is made with 1 island. The application is separated into several sub-sections :

In our example, MTCPtoSafetyBuffers_I1, SafetyBlocks_I1, SafetyInfos_I1, SafetyBuffersToMTCP_I1, and DiagILETH.

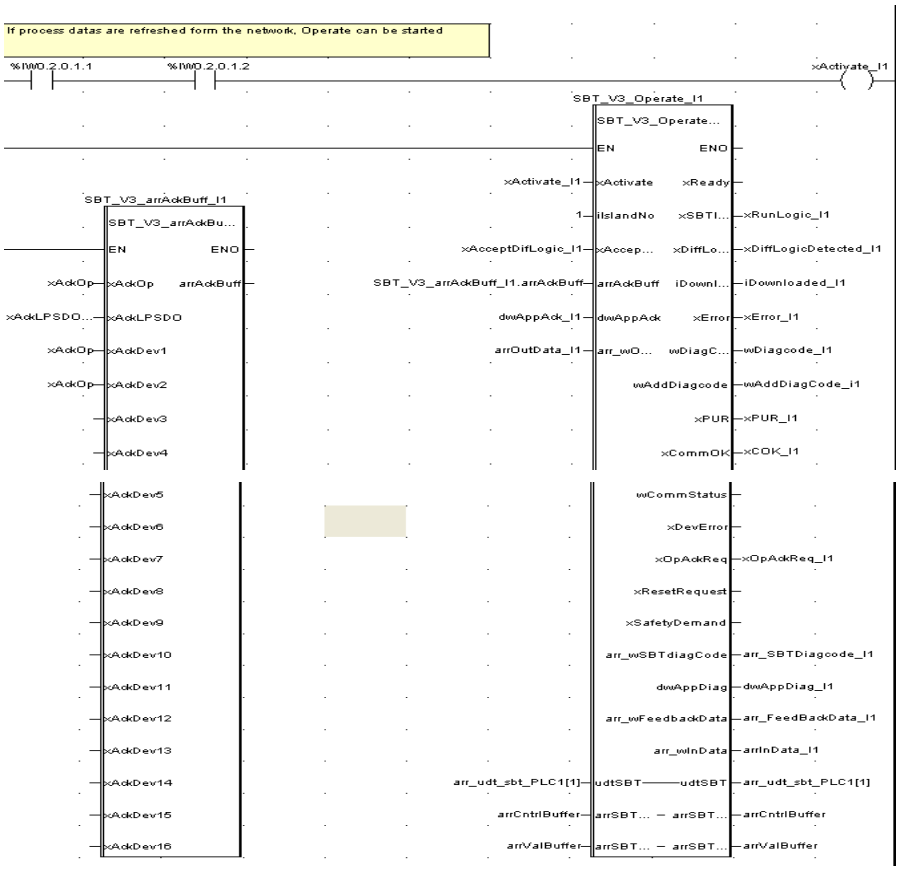


SBT_V3 makes the adaption between Modbus TCP process datas and the SBT Function blocs, and integrates the Function blocks Operate, (Max)TransTime, ProjectHeader, wich are necessary to handle the communication protocol of the safety devices.

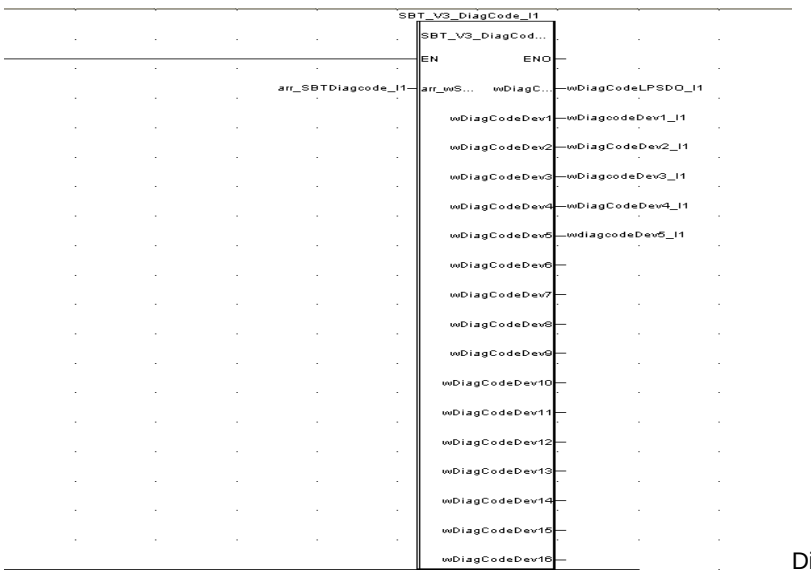
DiagILETH is optional and permits to make some diagnostic and restart on the IL ETH BK.



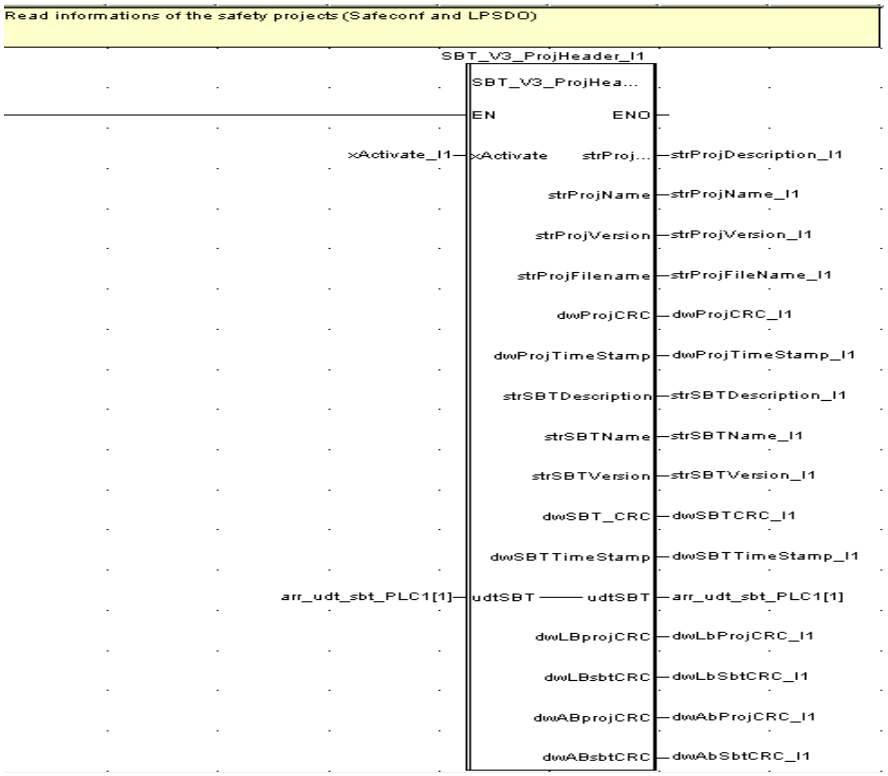
SwapBytes = True permits to swap High and Low bytes



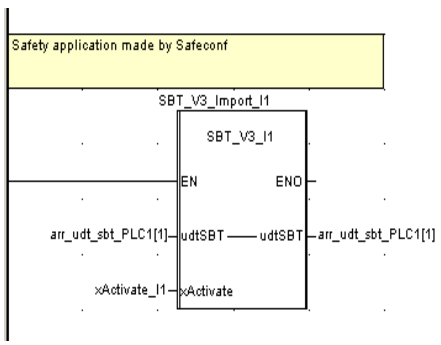
Main part of the system : Operate



Diagcode : extracts diag code of each device



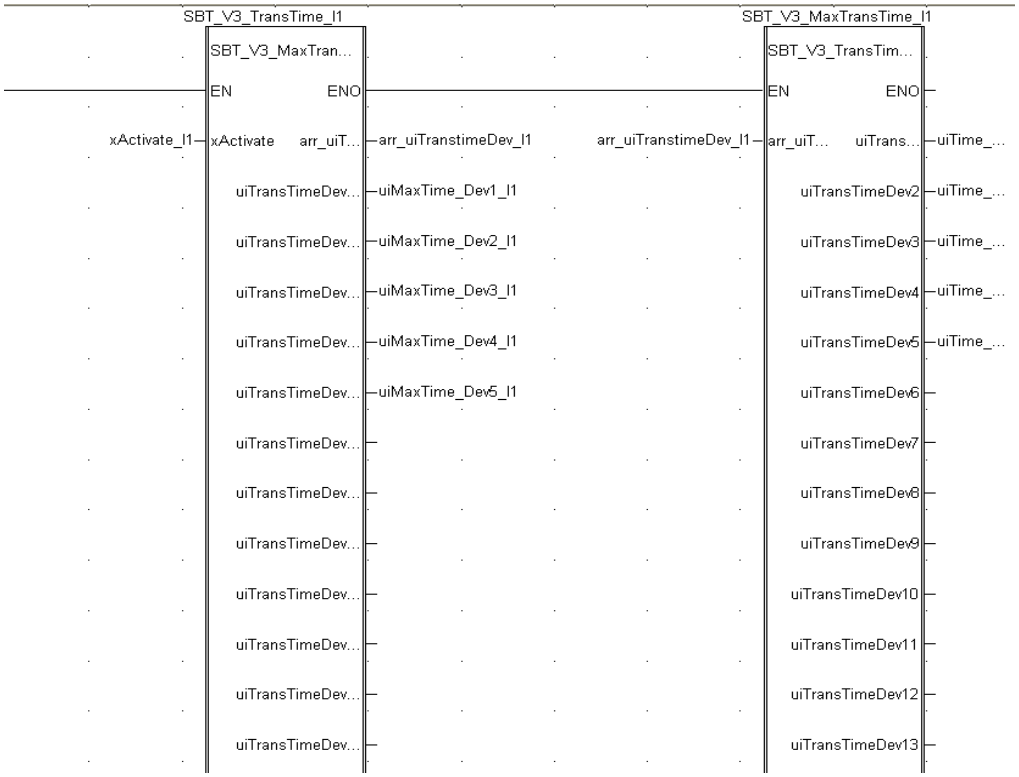
ProjHeader : gives details about the LPSDO and Safeconf Projects (CRC, timestamps)



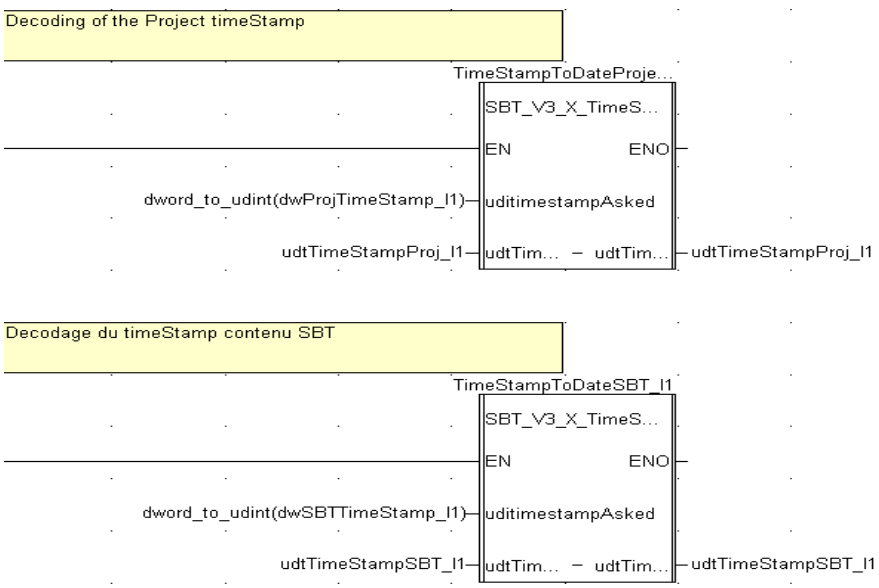
DFB created by Safeconf : safety application

The sub section SafetyInfos gives details about the Communication times for each device.

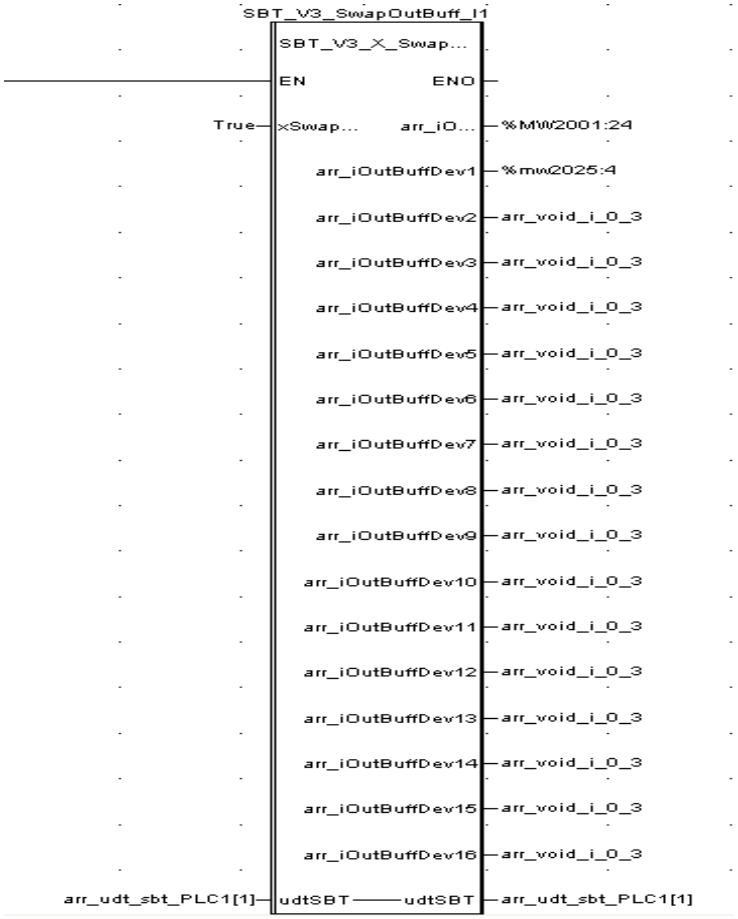
You should adapt the PSDx timeout of each device accortding to these measured values in Safeconf.



The timestamps are decoded by the DFB TimestampToDate, and all informations are available in the structure udtTimeStamp.



The sub-section SafetyBuffersToMTCP copies the datas created by the Operate DFB to the devices LPSDO.



3.3.5 Adding SafetyBridge operation to the standard application program

Add SafetyBridge Technology V3 operation to your standard application program using the Unity example.

The most simple is to start from the Unity example. Nevertheless, if you prefer integrate the SBT into your existing application, you can import the task file (*.XPG).

Right click on Task, Import, select the Mast.XPG file delivered in the example, and then “import”.

3.3.5.1 "SBT_Operate" DFB and others

The DFB performs the following functions:

– Download of the configuration and parameter data record from the standard control system to the IB IL 24 LPSDO 8 V3- PAC

– Cyclical routing of the SafetyBridge Technology V3 data flow

Where there are several SafetyBridge Technology V3 islands, an "SBT_Operate" DFB is required for each island.

- Insert the "SBT_Operate" DFB.
- The udtSBT structure variable is used for data exchange between the DFBs of the program.
- Connect the inputs and outputs of the "SBT_Operate" DFB as shown below.

The Operate DFB is used in conjunction with SwapInBuff and SwapOutBuf. The “Swap” DFBs permit to invert low and high bytes of the Modbus TCP Process datas, for they are compatible with the Operate Function block. This swap is made if xSwapBytes is True.

The DFB ProjHeader permits to read the useful information about the Safety application.

Variables with …SBT… display the content of the LPSDO.

Variables with …Proj… display the content of the Safeconf Project (the application made by Safeconf and exported into Unity XDB format)

The DFB MaxTransTime displays the maximum transmission time measured with each device. The DFB Transtime displays the dynamic cycle time.

These values must be taken in account to adjust the Watchdog of each safety device.

Importing the safety bridge application (coming from Safeconf)

This step enables you to save and manage the safety logic created in -SAFECONF as a DFB to be imported.

In step 1, the *.XDB (e.g., xxx.XDB) was created and saved under FileOutput in the project path (see XDB file).

- Open your project in Unity.
- Make sure that you are offline.
- Select "Types FB dérivés", right clic, import

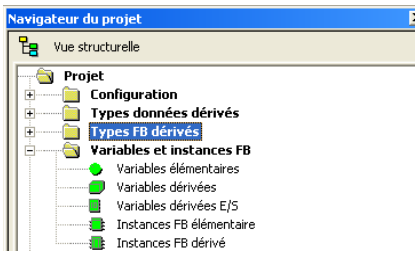


Figure 3- 33 "type FB, Import"

- Select the XDB file and confirm your selection with "Import".

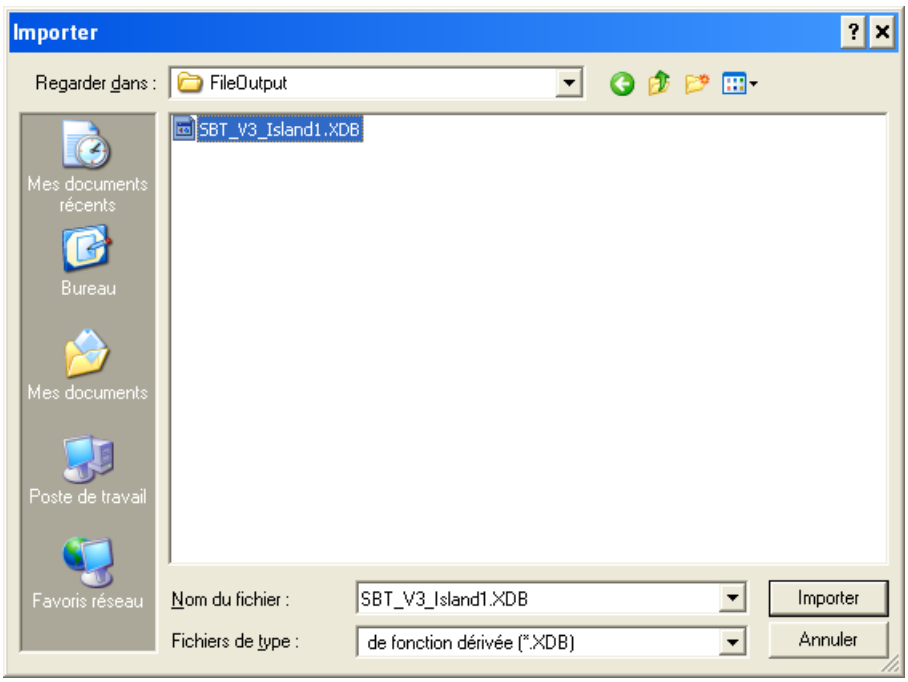
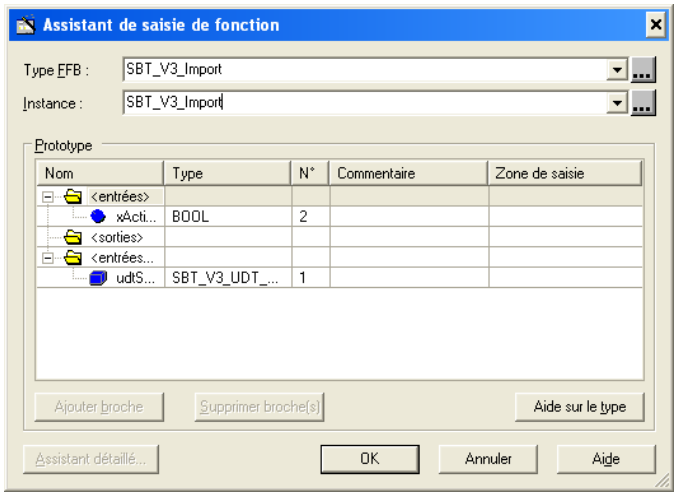


Figure 3- 34 Selecting the XDB file

It is now necessary to include the imported DFB into the application.

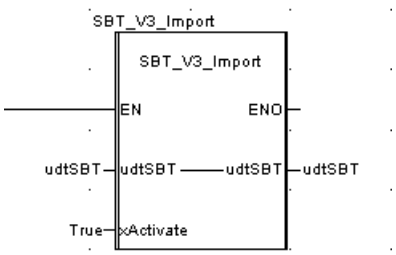
So, you have to go after the DFB ProjHeader and TransTime and add this new DFB in the program :



The name of the DFB is the one wich was imported, but the name of the instance has to be defined by yourself.

The link with the other DFBs of the SBT is made via the variable udtSBT, wich is used by every DFB dedicated to the SBT.

Safeconf application (inside the imported DFB) :



3.3.7 Example program

Once all the necessary DFBs and Sections have been imported and used, the example program is able to communicate with the LPSDO V3 and its devices.

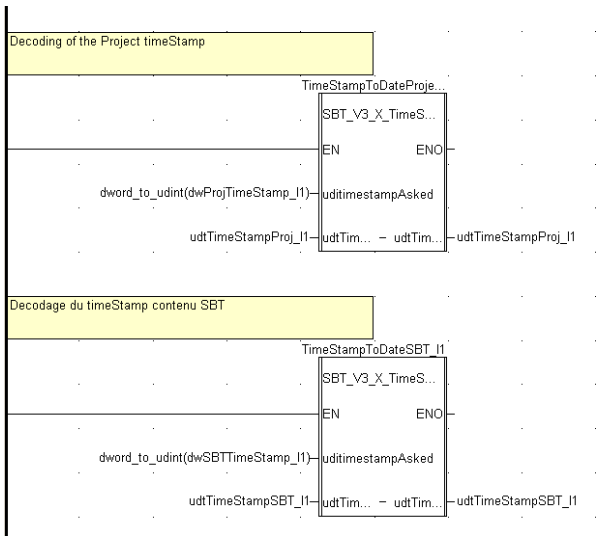
The bit xActivate is set if the IOScanning polling contract(s) dedicated to the SBT devices are active. Adapt the conditions according to your network settings. The DFB “Operate” contains many IN and OUT parameters.

In our example, as contracts 2 and 3 (bits 1 and 2) are dedicated to the LPSDO and the PDSI, the xActivate should be the sum of the IOScanning contracts bits :

The list of parameters of the SBT_V3_... function blocs can be found in detail in the document attached to the PcWOrx library for SBT V3. (SBT_V3_V1_00_001.pdf) This file is also part of the SBT V3/Unity/M340 package. For more details, please refer to this document.

Optional DFB : TimeStamp decoding :

The timestamp of the safeconf applications are given in a Dint format, wich is not easily understandable. This DFB permits to convert a dint timestamp into a comprehensive date and time



This decoding is made for both timestamps given by the ProjHeader DFB.

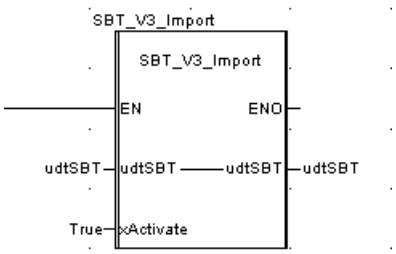
TimeStampToDate_Project is for the Safeconf application

TimeStampToDate_SBT is for the application already in the LPSDO.

The result is available in a data structure using either a complete Date and Time string, or separated elements (hour, minut, second, day, month, year) :

SBT_V3_X_udtTimeStamp	<Struct>	Auxiliairy Data type
iYear	INT	
iMonth	INT	
iDay	INT	
iHour	INT	
iMinut	INT	
iSecond	INT	
strDateAndTime	string[24]	

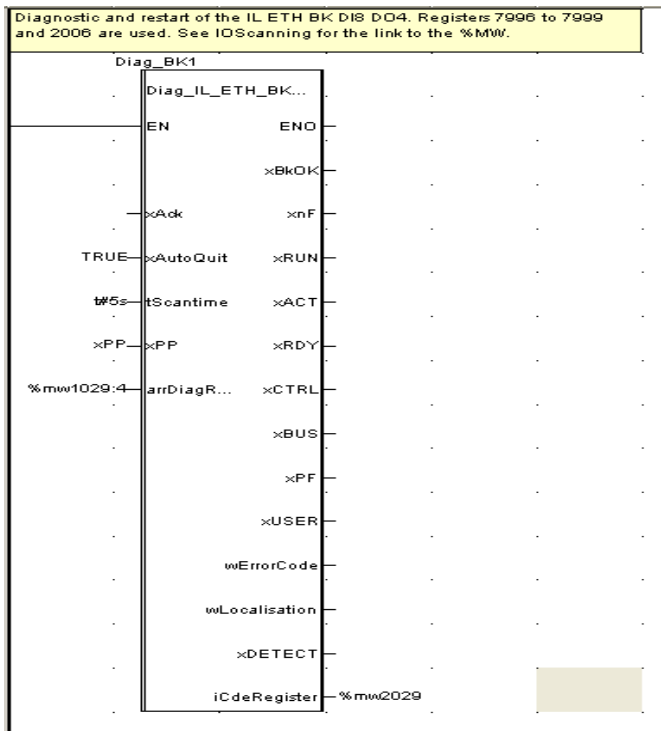
Safeconf application (inside the imported DFB)



Optional but usefull Section diagILETH:

This section permits to handle the diagnostic and the control of the IL ETH BK itself.

It gives information about the status, and permits to (re)set the PP (Plug and Play) mode, and make an acknowledgement of the netFail. If several IL ETH BKs are used in the project, this FB will have to be duplicated as many times as the number of BKs. Of course, the IOScanning list will also have to be adapted.



Input parameters :

arrDiagRegisters (array of 4 words), located on registers 7996 to 7999

xPP : Plug and play mode

xAck : Acknowledgment of a netFail or PF

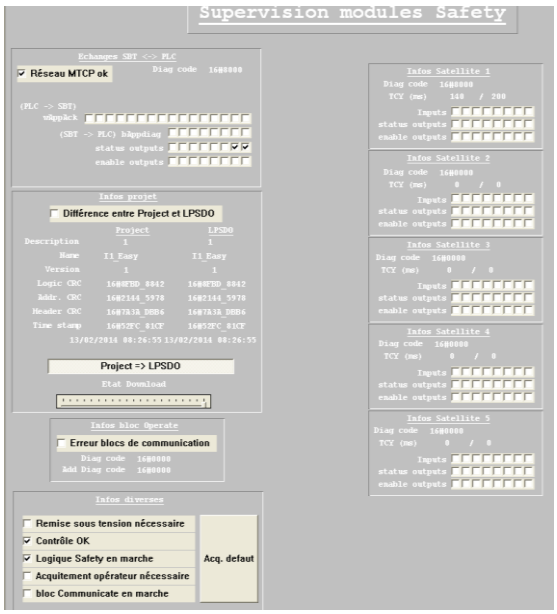
xAutoQuit : Continuous acknowledgment of a netFail or PF

Output parameters are explained in the datasheet of the IL ETH BK DI8 DO4. They give the status of the BK.

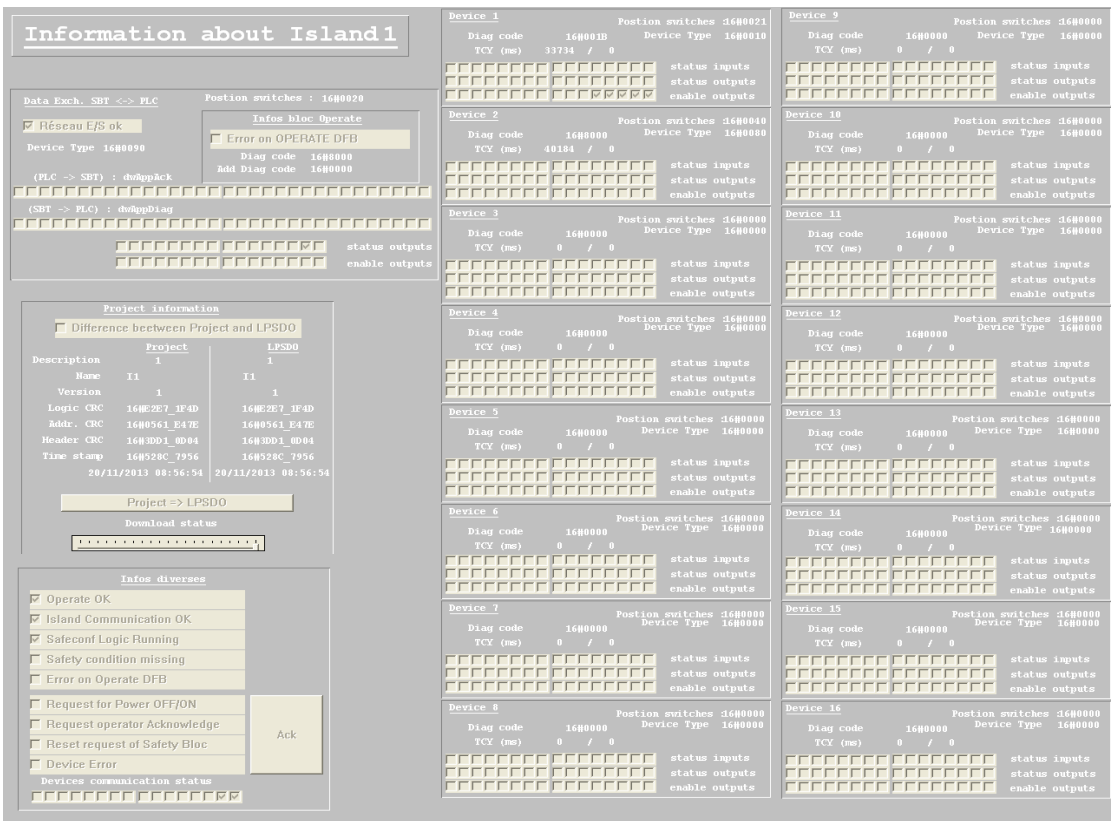
Help screen for debug :

An exploitation screen is given as an example wich summups useful information for the SBT:

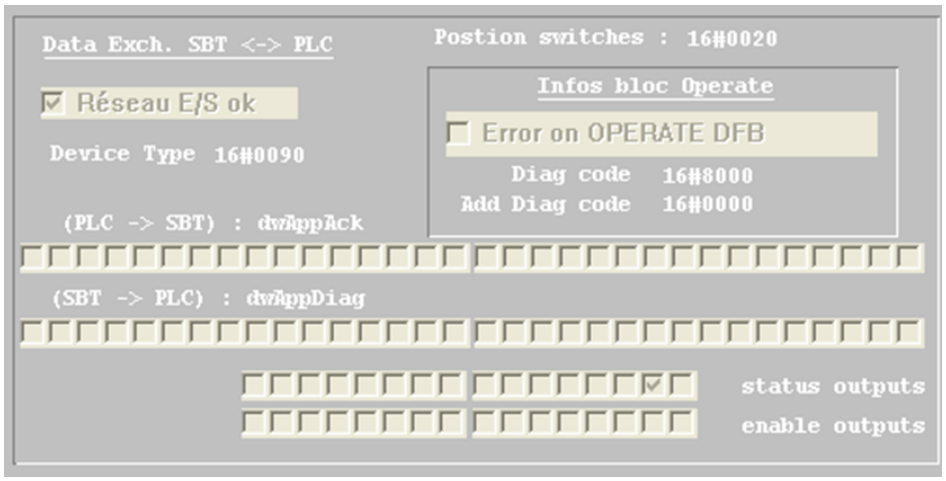
This screen is not mandatory in your application, but it shows many interesting information :



An other more complete page is available in the “Easy” quickstart. You can find it in the SBT_V3_Easy application, instead of the SBT_V3_quickstart.



Information about the LPSDO :



Réseau E/S OK : Value of xActivate (according to the I/O network status (contracts, for example, with IOScanning)

Position switches : Position of the DIP switches on the LPSDO

Device Type : Type of the device (h90 = LPSDO V3)

dwAppAck : Datas corresponding to the same information in Safeconf (bit 0 on the right)

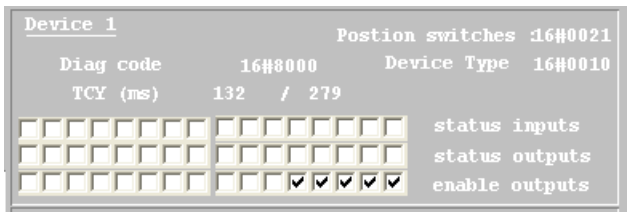
dwAppDiag : Datas corresponding to the same information in Safeconf (bit 0 on the right)

Status outputs : Status of each output of the LPSDO (bit 0 on the right)

Enable outputs : data wich permit to ENABLE the dedicated output (bit 0 on the right)

Error on OPERATE DFB : See the DiagCode and AddDiagCode and refer to the document SBT_V3_V1_00_001.pdf.

Information about the Devices



Position switches : Position of the DIP switches on the device

Diag code : Diag code given by the device. Refer to the document SBT_V3_V1_00_001.pdf.

Device type : Type of the device (h10 = PSDI 8)

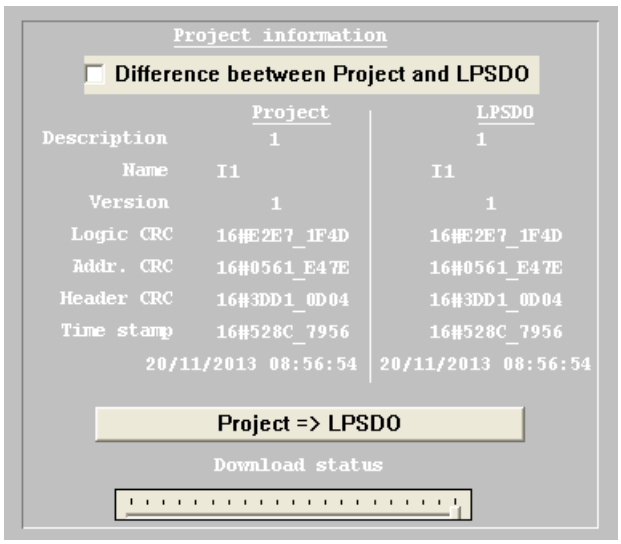
Status input : status of each input. (bit 0 on the right) *

Status output : status of each output. (bit 0 on the right) **

Enable output : Enable of each output. (bit 0 on the right) **

Notice : * : only significant if the device is an Input device

**: only significant if the device is an Output device



Information about the projects.

Difference beetween Project and LPSDO : status of the xDiffLogicDetected.

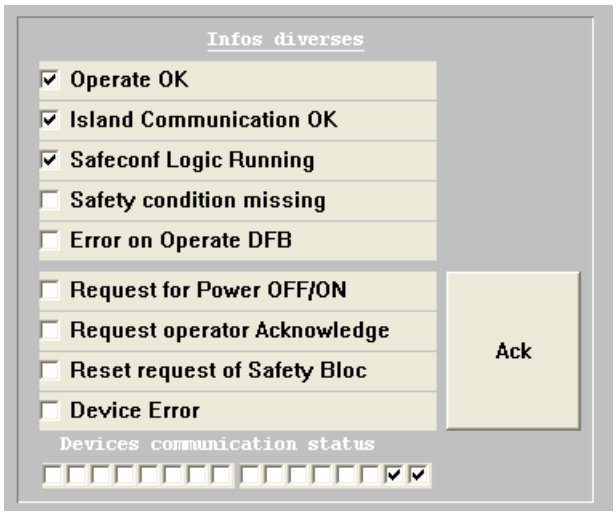
Column Project : Detail of the project contained in the imported DFB

Column LPSDO : Detail of the application in the LPSDO

Project -> LPSDO : Enable transfer of the safety application from the PLC (the DFB) to the LPSDO

Download status : Bargraph moving according to the download step.

Information about the safety application



Operate OK : Function block is successfully initialized and operating without errors

Island Communication OK : communication status of the island is OK

Safeconf Logic running : Safety logic (SAFECONF logic) is running on the LPSDO

Safety condition missing : Reset-Request signal of one or more safety function blocks is true

Error on OPERATE DFB : Function block error

Request for power OFF/ON : Non-acknowledgeable failure state and power-up is requested

Request operator acknowledge : Operator acknowledge requested

Reser request of safety bloc : Reset-Request signal of one or more safety function blocks is true

Device error : Indicates error in one or more SBT-devices

Devices communication status : Communication status of each module. Each bit represents the status of a module. For example: Bit 0 corresponds to module 1.

3.4 Step 3: installing the SafetyBridge Technology V3 modules

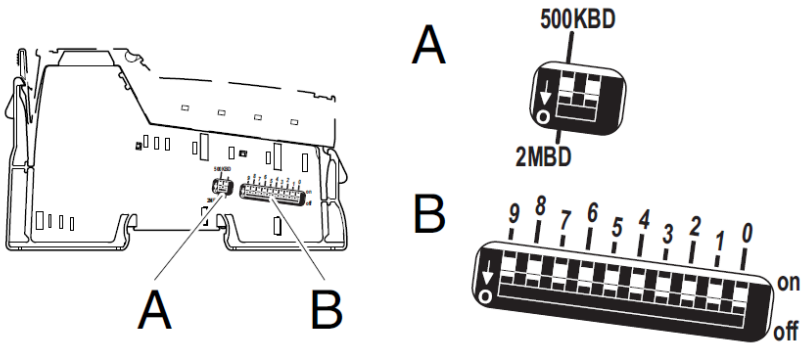
Install the SafetyBridge Technology V3 modules. To do this, proceed as described in the user manuals for the modules used and the -Inline installation manual (see "Additional documentation" on page 1-2).

Please note the following in particular:

Set the DIP switches **before** assembling the module in the Inline station. The switches cannot be accessed when the safety terminal is installed in the Inline station.

The switch numbers correspond to the labeling on the housing and not the numbering on the switch itself.

The DIP switches are located on the left-hand side of the safety module.



SafetyBridge V3									
Sélecteur d'adresse									
Numéro d'îlot					Réservé				
9	8	7	6	5	4	3	2	1	0
					off	off	off	off	off
1_{dec} à 31_{dec}					0_{dec}				

Figure 3- 37 DIP switches on the IB IL 24 LPSDO 8 V3- PAC

A Switch for setting the transmission speed and the operating mode

B Switch for setting the address

For more detailed information on the DIP switches, please refer to the documentation for the IB IL 24 LPSDO 8 V3- PAC and the IB IL 24 PSDI 8- PAC.

Switch positions of the modules for the example

Table 3- 6 Switch positions in the example

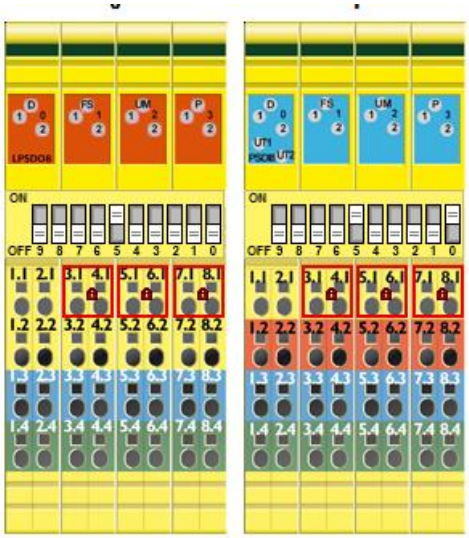


Figure 3‑ 38 Switch position of the IB IL 24 LPSDO 8 V3‑ PAC

Only use devices with a uniform transmission speed within an -Inline station (a local bus). It is not possible to operate a mixture of devices with different transmission speeds.

Since the SafetyBridge Technology V3 modules of an island can be located in different -Inline stations, it may be the case that different transmission speeds (500 kbaud/2 Mbaud) are set for the modules.

This completes step 3 "installing the SafetyBridge modules". You have now integrated a SafetyBridge Technology V3 system into an existing system in three steps.

3.5 Overall safety validation

Perform an overall safety validation before you start up your system.

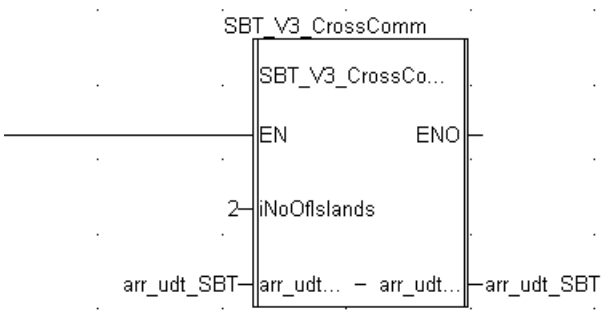
Cross Communication beetwen 2 or more islands

In the new SBT version V3, the islands can communicate with each other. The cross communication is a master-slave model. One or more islands react as a slave for another master island. Each island has the data-structure ***udtSBT***. The function block SBT_V3_CrossComm works with an array of this data-structure as an in-/output variable (***arr_udtSBT***), where each ***udtSBT*** of an island is a part of this array; and that is how the cross communication happens.

In case the island is in another PLC, another function block is needed (SBT_V3_DataExch_V1_00). The function block is responsible for data exchange between the master and the slave island

For more details about the cross communication, please refer to the LPSDO 8 V3 User manual, chapter A 2.4.

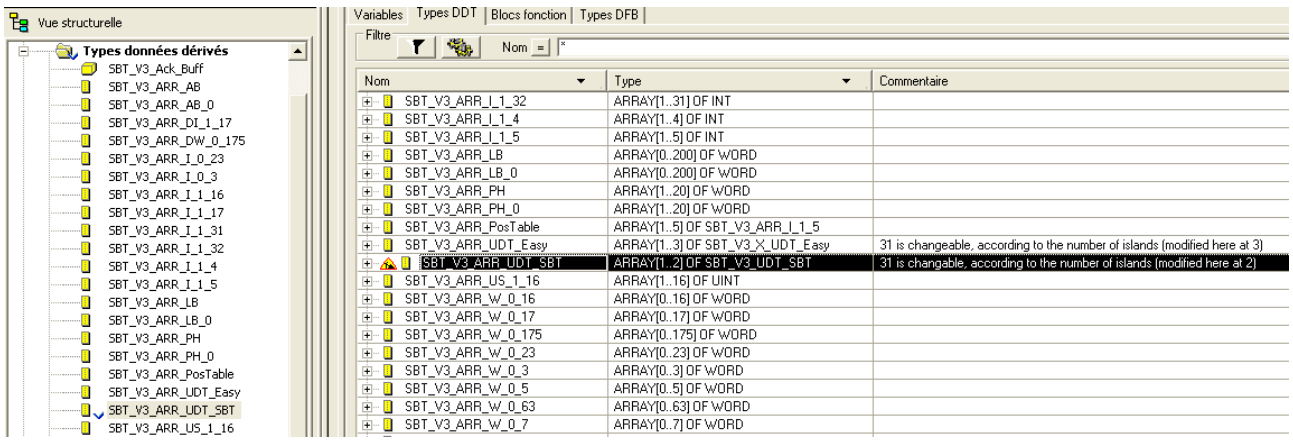
The CrossComm DFB is as follow :



Arr_udt_SBT is an array of udt_SBT (the structure used to link every DFB of the same island).

In the above example, only 2 islands are communicating together (Island 1 and Island 2).

So, the length of the arr_udt_sbt must be adjusted to at least '2' :

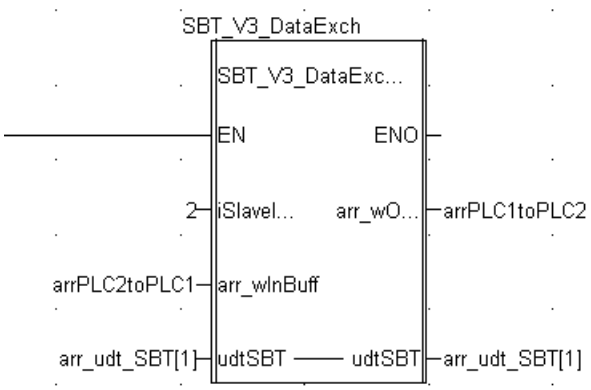


<u>This DFB must be inserted only once in the PLC controlling the LPSDO V3.</u>

If the cross communication must be handled between 2 islands wich are not on the same PLC, the DFB SBT_V3_DataExchange must be used and configured on each PLC.

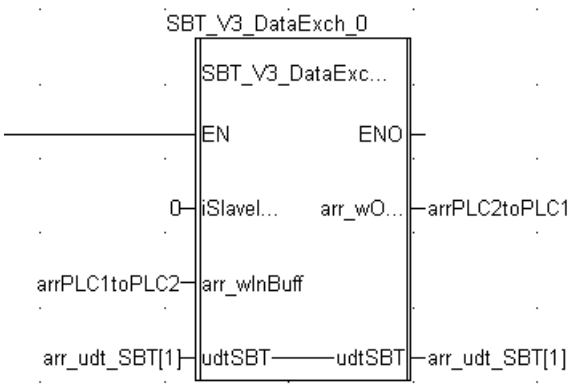
If Island1on PLC1 is the master of Island 2 on PLC2, the application in PLC1 is as follow :

iSlaveNr : 2 : Number of the slave island (destination)



And the application in PLC2 is as follow :

iSlaveNr : 0 : The exchanged buffers are handled by the master (on the other PLC)



3 Using the “Easy” quickstart

An “easy to handle” example has also been inserted in the Quickstart package. This easy example includes complex structure and sections. The goal is to simplify the understanding and the typing of an LPSDO V3 application in an M340 application. Nevertheless, it uses more memory space than the simple SBT_V3_quickstart.

This example is easier to use because nearly no typing is necessary. Only adapt and delete some elements, regarding your own configuration.

This example uses 2 main structures. These structures will be indexed by an island N°.

These structures are :

SBT_V3_X_UDT_Easy and SBT_V3_UDT_SBT.

These structures will not be used directly, but they will be included in an array of elements :

SBT_V3_ARR_UDT_Easy : ARRAY[1..x] OF SBT_V3_X_UDT_Easy

SBT_V3_ARR_UDT_SBT : ARRAY[1..x] OF SBT_V3_UDT_SBT

‘x’ will have to be modified to create the adapted number of elements necessary to the application :

1 : 1 island, 2 : 2 islands, 16 : 16 islands

The name of the variables are :

Arr_udt_easy and arr_udt_sbt.

Each of these variables will be used in the application with an index :

Arr_udt_easy[1] and arr_udt_sbt[1] for island 1

Arr_udt_easy[2] and arr_udt_sbt[2] for island 2, etc……

The most important structure, wich will be used in the whole application, will be “arr_udt_easy” :

Name	Type	Usage
xActivate	BOOL	IOs are correctly refreshed by the (MTCP, for example) Master
xAckOp	BOOL	Operator Acknowledgement
xAckLPSDO	BOOL	Error acknowledgment in case of LPSDO error
arrAckDev	ARRAY[1..16] OF BOOL	Error acknowledgment in case of Device error
dwAppAck	DWORD	xAppAck,X0 à xAppAck.X31 in safeconf : Un-Safe data PLC -> SBT
arrWoutput	SBT_V3_ARR_W_0_16	Enable output Enable for each xLPSDO
xReady	BOOL	Operate FB : Ready
xSBTlogicRunning	BOOL	Operate FB : Logic running in the SBT
iDownloaded	INT	Operate FB : percent of the Safeconf application downloaded in the LPSDO
xError	BOOL	Operate FB : Error
wDiagCode	WORD	Operate FB : Diag code
wAddDiagCode	WORD	Operate FB : Additive diag code

xPUR	BOOL	Operate FB : Power Up request : the LPSDO must be powered OFF/ON
xCommOK	BOOL	Operate FB : Communication with the LPSDO is correct
wCommStatus	WORD	Operate FB : each bit of this word shows if the communication with the corresponding device works fine (bit 1 = Device 1)
xDevError	BOOL	Operate FB : at least one device is in error. See Diag code for each device
xOpAckReq	BOOL	Operate FB : Operator acknowledgment is requested
xResetRequest	BOOL	Operate FB : reset of the PLC is requested
xSafetyDemand	BOOL	Operate FB : Safety FB are used in the LPSDO
dwAppDiag	DWORD	Operate FB : xAppDiag.X0 à xAppdiag.X31 in safeconf : UnSafe data SBT-> PLC
arr_wFeedBackData	SBT_V3_ARR_W_0_16	Operate FB : Status of each output of the xPSDOx devices
arr_wInData	SBT_V3_ARR_W_0_16	Operate FB : Status of each input of the PSDI devices
arrSBTOnlCntrlBuf	SBT_V3_ARR_DW_0_175	Variable for ONLINE mode with SAFECONF. The Variable must be named exactly as it is (i.e. arrSBTOnlCntrlBuf). The variable must be VAR_GLOBAL with PDD option checked (see help-document)
arrSBTOnlValBuf	SBT_V3_ARR_W_0_175	Variable for ONLINE mode with SAFECONF. The Variable must be named exactly as it is (i.e. arrSBTOnlValBuf). The variable must be VAR_GLOBAL with PDD option checked (see help document)
wDiagCodeLPSDO	WORD	DiagCode FB : DiagCode of the LPSDO
arr_wDiagCodeDev	ARRAY[1..16] OF WORD	DiagCode FB : diag code for each device
udtProjHead	SBT_V3_UDT_ProjectHeader	Project Header of the Safeconf project (coming from SafeConf Export)
udtSBTHead	SBT_V3_UDT_ProjectHeader	Project Header in the LPSDO
arr_uiTransTimeDev	ARRAY[1..16] OF UINT	Time transmission for each device
arr_uiMaxTransTimeDev	ARRAY[1..16] OF UINT	Max Transmission time for each device
udt_Buff	SBT_V3_UDT_Buff	Buffers of the LPSDO and devices (process datas)
iIslandNo	INT	Island N° of the LPSDO
xAcceptDiffLogic	BOOL	Acceptance of the different logic detected beetween Project and LPSDO : enable down-load to the LPSDO

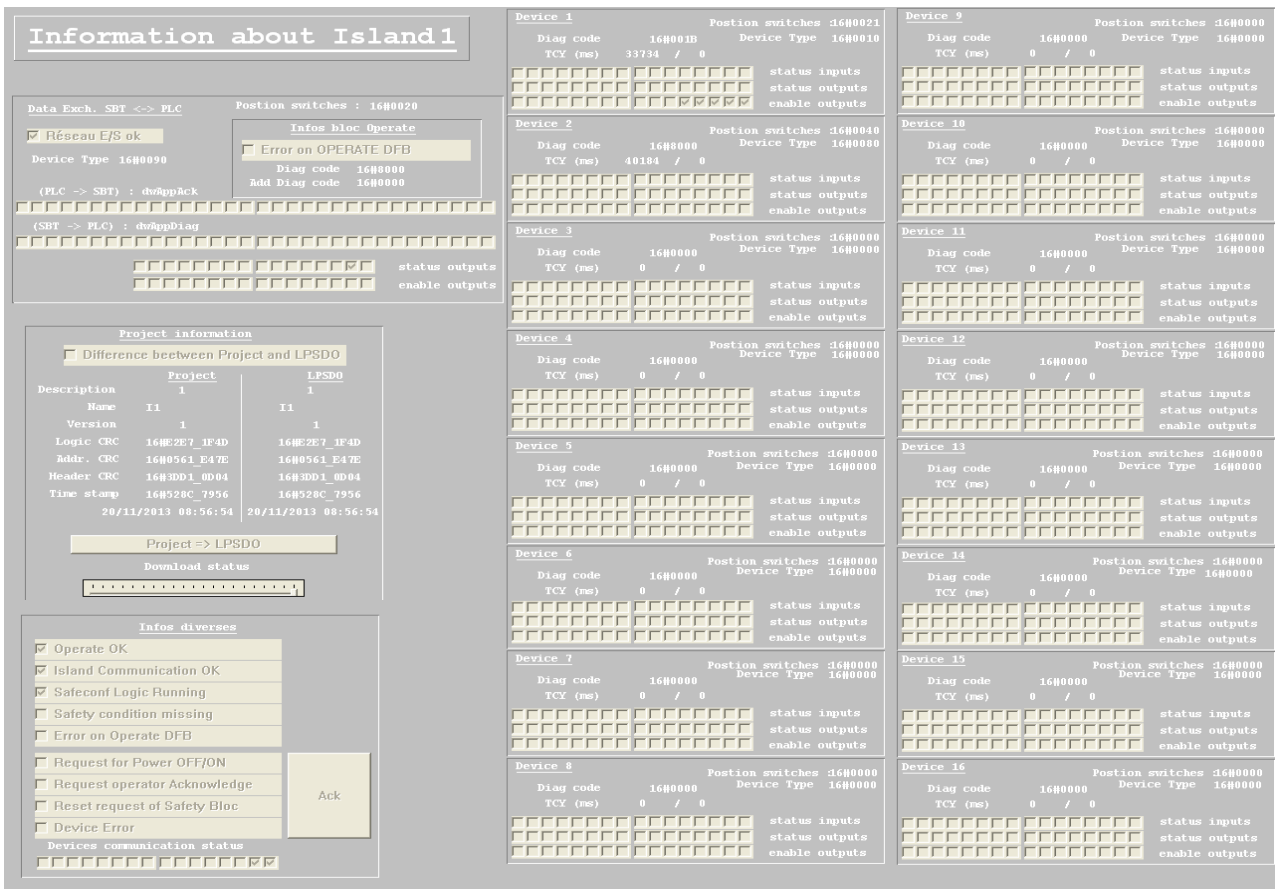
xDiffLogicDetected	BOOL	difefrence beetween the aplication in the LPSDO and the one exported by Safeconf
strProjFileName	STRING	Name of the BIN file created by Safeconf
udtProjTimeStamp	SBT_V3_X_udtTimeStamp	Time and Date of the application in the project (FB created by Safeconf)
udtSBTTimeStamp	SBT_V3_X_udtTimeStamp	Time and Date of the application in the LPSDO
udtAppAck	SBT_V3_X_udtAppBits	dwAppAck separated in 2 words for bits display
udtAppDiag	SBT_V3_X_udtAppBits	dwAppDiag separated in 2 words for bits display
udtSwitchPos	SBT_V3_X_udtSwitchPos	Position of the Dip Switches
xAckGeneral	BOOL	General acknowledge (for test purpose only)
wTypDev	SBT_V3_ARR_W_0_17	Type of each device

This structure will be used for each Island, associated to the udt_SBT_V3 structure.

The udt_SBT_V3 is the link between all the DFBs controlling the LPSDO.

The udt_Easy structure is the data wich is used to display all necessary information on the Screen dedicated to the LPSDO.

In this page, the variables are accessed via their name.



When a screen is exported, an ascii file named “screenname.XCR” is created. If you open this ascii file with a simple text editor, you can see, for example :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SCRExchangeFile>
    <fileHeader company="Schneider Automation" product="Unity Pro XL V7.0 - 120823C" dateTime="date_and_time#2013-12-6-9:40:49" content="Fichier source écrans d'exploitation" DTDVersion="41"></fileHeader>
    <contentHeader name="Projet" version="0.0.115" dateTime="date_and_time#2013-12-6-8:58:59"></contentHeader>
    <IOScreen version="V1.0">
        <screen name="Visu Ilot 1 PLC1" screenX="1280" screenY="1024" BKColor="12632256" valScreen="0" location="" creationDate="28/11/2011 - 09:37:12" modificationDate="03/12/2013 - 12:02:53" customInfos="" isPattern="0" valPattern="0">
            <object objectID="2" description="(703,17,1006,377),3"></object>
            <object objectID="11" description="(703,17,1006,377),(0,1,8421504),(10,0,8421504)"></object>
            <object objectID="10" description="(704,376,1006,376),(0,1,16777215),4"></object>
            <object objectID="10" description="(1005,18,1005,376),(0,1,16777215),3"></object>
            <object objectID="2" description="(102,8,341,502),3"></object>
            <object objectID="11" description="(102,8,341,502),(0,1,8421504),(10,0,8421504)"></object>
            <object objectID="10" description="(103,501,341,501),(0,1,16777215),4"></object>
            <object objectID="10" description="(340,10,340,501),(0,1,16777215),3"></object>
            <object objectID="18" description="(223,479,239,494),(-1,0,0,0),||">
                <varPilot name="arr_Udt_Easy[1].udtappack.wword0.0" typeName="BOOL" description="Pilot:|0|1|"></varPilot>
            </object>
            <object objectID="18" description="(223,464,239,479),(-1,0,0,0),||">
                <varPilot name="arr_Udt_Easy[1].udtappack.wword0.1" typeName="BOOL" description="Pilot:|0|1|"></varPilot>
            </object>
            <object objectID="18" description="(223,449,239,464),(-1,0,0,0),||">
                <varPilot name="arr_Udt_Easy[1].udtappack.wword0.2" typeName="BOOL" description="Pilot:|0|1|"></varPilot>
            </object>
```

In this file, we can see the only variable wich is used in this screen is arr_udt_Easy[x].

Is several island must be controlled by the same PLC, and if several screens must be also created, it will just be necessary to replace the index number [x] and the correct information about the defined island will be displayed.

Of course, after the index has been modified by the text editor, it will be necessary to import the screen back to the Unity application.

For the correct use of this structure, some parameters have to be adapted according to the application.

The quickstart example includes a section, wich is called SBT_Easy.

This section is divided in several parts :

Part 1 :

```
arr_Udt_Easy[1].xAckOp := arr_Udt_Easy[1].xAckGeneral;
arr_Udt_Easy[1].xAckLPSDO := arr_Udt_Easy[1].xAckGeneral;
arr_Udt_Easy[1].arrAckDev[1]:= arr_Udt_Easy[1].xAckGeneral;
arr_Udt_Easy[2].xAckOp := arr_Udt_Easy[2].xAckGeneral;
arr_Udt_Easy[2].xAckLPSDO := arr_Udt_Easy[2].xAckGeneral;
```

This part must be adapted to define the information of Acknowledgment for each reason and device of each island.

Part 2 :

```
(* Island 1 handling *)
arr_Udt_Easy_PLC1[1].xActivate := Diag_BK1.xBkOK and Diag_BK1.xRUN;(* Communication with BK1 OK, and local bus OK *)
arr_Udt_Easy[1].udt_Buff.xSwapBytes := True; (* Swap High and Low bytes in the process datas *)
arr_Udt_Easy[1].iIslandNo := 1; (* Island N° *)
(* IN buffers *)
arr_Udt_Easy[1].udt_Buff.arr_I_In_LPSDO := %mw1001:24;
arr_Udt_Easy[1].udt_Buff.arr_I_In_Dev[1] := %mw1025:4;
(* Main part *)
SBT_V3_Easy_I1 (udt_SBT_Easy := arr_Udt_Easy[1],
            udt_SBT_V3 := arr_udt_sbt[1]);
(* OUT buffers *)
%mw2001:24 := arr_Udt_Easy[1].udt_Buff.arr_I_Out_LPSDO;
%mw2025:4 := arr_Udt_Easy[1].udt_Buff.arr_I_Out_Dev[1] ;
(* Call of the safeconf application *)
SBT_V3_I1 (udtSBT := arr_udt_sbt[1],
        xActivate := True);
```

This part includes :

- xActivate : conditions wich indicates that all process datas of the island (LPSDO + Devices) are correctly refreshed. When using IOScanning only, it is just necessary to control that each Modbus TCP contract is active. The words %IW0.2.0.1 to %IW0.2.0.4 indicate in each of their bits that the dedicated contact (for example 1 to 4) is active and runs fine (%IW0.2.0.1.1 : true : Means that the first line (contract) of the IOScanning is OK).
- xSwapByte : for Modbus TCP/M340, bytes must be swapped
- iIslandNo : the island N° must be the same as the one defined on the dip switches
- InBuffers : Add here all the Process datas areas of every device (as Input)
- Main part : Call of the SBT_V3 standard DFBs. (no adaption is necessary)
- OUT buffers : Add here all the process datas areas of every device (as output)
- Call of the safeconf application : Here the file created by safeconf must be inserted

Part 3 :

```
(* Creation of the CrossComm beetween islands controlled by the same PLC. Only one is necessary in the PLC*)
SBT_V3_CrossComm_PLC1 (iNoOfIslands := 2,
                       arr_udtSBT := arr_udt_sbt);
(* Data exchange with an LPSDO wich is not controlled by the same PLC. One DataExch DFB will be necessary in each PLC for each island communicating with an other one *)
(* In this example, Island 3 is the slave of Island 2 *)
SBT_V3_DataExch_PLC1_I23 (iSlaveIsland := 3 (* 3 means the island N° of the (remote) slave Island *),
                          arr_wInBuff := arr_DataExch_I3toI2 (* Buffer created by the remote PLC by the DataExch DFB as the arr_wOutBuff*),
                          udtSBT := arr_udt_sbt[2], (* DataStructure of the Master Island *)
                          arr_wOutBuff => arr_DataExch_I2toI3 (* Buffer wich will be used by the remote PLC, as the arr_wInBuff *));
```

This part is not necessary if no communication between several island is involved. You just have to delete it.

If cross communication between islands is necessary, adapt the island N° and the variables used here, and refer to the SBT_V3_V1_00_001.pdf file for more details about the function blocs SBT_V3_CrossCom and SBT_V3_DataExch.

If several islands are controlled by the same PLC, with or without cross communication, part 2 must be repeated for each island.

Of course, many information can be sent or retrieved with the LPSDO.

These information will be accessed via the structure dedicated to the island.

For example, as mentioned in the example Part 1, the acknowledgments bits can be accessed with the syntax :

arr_Udt_Easy[1].xAckOp, etc.…

The enable Outputs : *arr_Udt_Easy[1].arrWoutput[0]* (for the LPSDO)

The outputs status : *arr_Udt_Easy[0].arr_wFeedBackData[0]* (for the LPSDO)

The inputs status : *arr_Udt_Easy[1].arr_wInData* (for the device 1)

The diagnostic information : *arr_Udt_Easy[1].arr_wDiagCodeDev[1]* (of the device 1)

Etc…..

The goal of the ‘Easy’ concept is to make the use of the SBT devices easy, and the information useful for the PLC application is found easyly. For example, 3 variables will have an interaction between the non safety application (controlled by the PLC), and the safety application (controlled by the LPSDO). These variables are dwAppdiag, dwAppAck, arr_wFeedBackData, arr_wInData and arrWoutput.

These variables are part of the ‘easy’ structure and they will be available in the PLC application with the syntax :

(example) : arr_udt_easy[1].dwAppDiag, arr_udt_easy[1].dwAppAck, arr_udt_easy[1]. arr_wFeedBackData [x], arr_udt_easy[1]. arr_wInData[x], arr_udt_easy[1].arrWoutput[x].

(Where [x] is the safety device number).

In the same way, many other variables are available, such as :

- the Device Acknowledgement : arr_udt_easy[1].arrAckDev[x])
- the Device diag code : arr_udt_easy[1]. arr_wDiagCodeDev[x])
- the maximum cycle time of the SBT island : arr_udt_easy[1]. arr_uiMaxTransTimeDev[x]
- the actual cycle time of the SBT island : arr_udt_easy[1] .arr_uiTransTimeDev[x]

Details for the Cross Comm between 2 Islands in 2 separated PLC.

Island 3 is the slave of Island 2.

Island 3 is on a different PLC.

Island 2 is controlled by PLC1

Island 3 is controlled by PLC2.

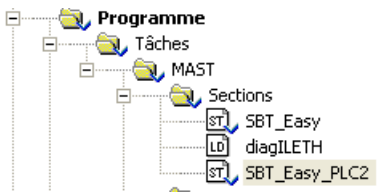
Code in PLC1 :

```
(* Creation of the CrossComm beetween islands controlled by the same PLC. Only one is necessary in the PLC*)
SBT_V3_CrossComm_PLC1 (iNoOfIslands := 2,
                       arr_udtSBT := arr_udt_sbt);

(* Data exchange with an LPSDO wich is not controlled by the same PLC. One DataExch DFB will be necessary in each PLC for each island communicating with an other one *)
(* In this example, Island 3 is the slave of Island 2 *)
SBT_V3_DataExch_PLC1_I23 (iSlaveIsland := 3 (* 3 means the island N° of the (remote) slave Island *),
                          arr_wInBuff := arr_DataExch_I3toI2 (* Buffer created by the remote PLC by the DataExch DFB as the arr_wOutBuff*),
                          udtSBT := arr_udt_sbt[2], (* DataStructure of the Master Island *)
                          arr_wOutBuff => arr_DataExch_I2toI3 (* Buffer wich will be used by the remote PLC, as the arr_wInBuff *));
```

Code in PLC2 :

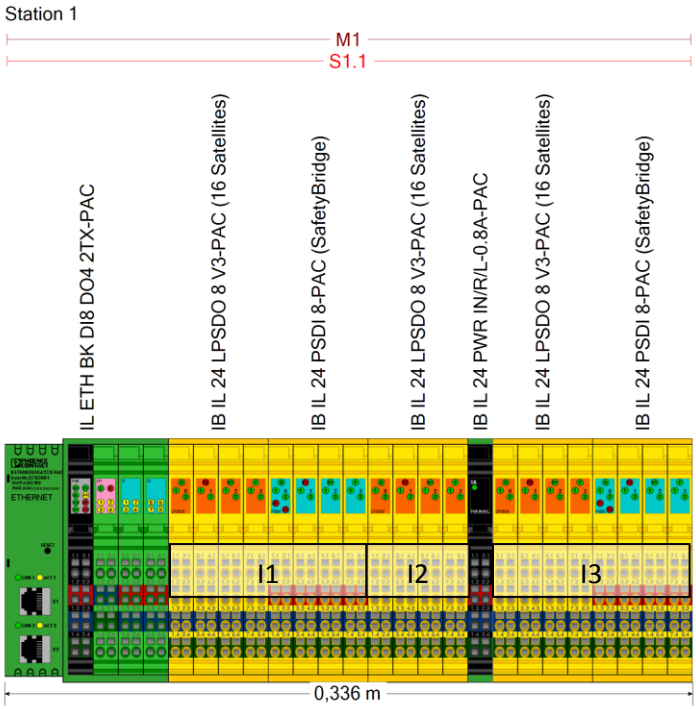
```
(* Creation of the CrossComm beetween islands controlled by the same PLC. Only one is necessary in the PLC*)
SBT_V3_CrossComm_PLC2 (iNoOfIslands := 3,
                       arr_udtSBT := arr_udt_sbt_PLC2);

(* Data exchange with an LPSDO wich is not controlled by the same PLC. One DataExch DFB will be necessary in each PLC for each island communicating with an other one *)
(* In this example, Island 3 is the slave of Island 2 *)
SBT_V3_DataExch_PLC2_I32 (iSlaveIsland := 0 (* 0 means that this DFB supports the slave device *),
                          arr_wInBuff := arr_DataExch_I2toI3 (* Buffer created by the remote (Master) PLC by the DataExch DFB as the arr_wOutBuff *),
                          udtSBT := arr_udt_sbt_PLC2[2], (* DataStructure dedicated to the Master Island *)
                          arr_wOutBuff => arr_DataExch_I3toI2 (* Buffer wich will be used by the remote (Master) PLC, as the arr_wInBuff  *));
```

In the quickstart example, the section SBT_Easy is the main section. The section SBT_Easy_PLC2 is an additive section, wich simulates a second PLC controlling another LPSDO V3. Is a safety island is not controlled by another PLC, this section must be deleted . It is present only as an example to illustrate the cross communication between PLCs.



Hardware example used with this quickstart

The IL ETH BK is configured as follow:



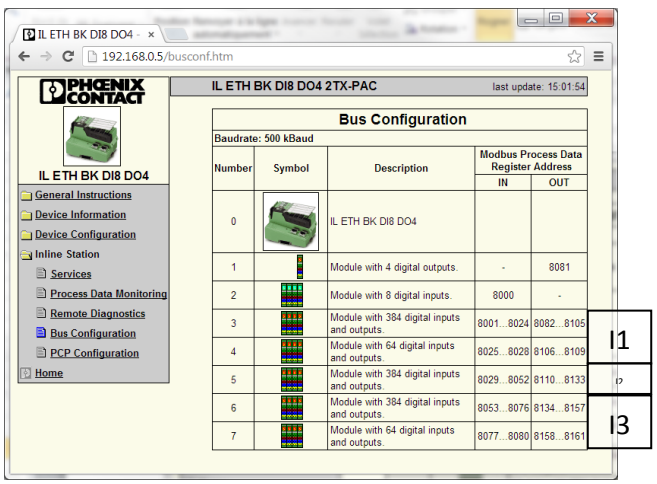
The first LPSDO and the first PSDI8 are for Island 1

The second LPSDO is for the Island 2

The third LPSDO and the second PSDI are for the isalnd 3.

This is just a constellation to illustrate the quickstart exemple.

The process datas of the IL ETH BK are mapped as follow :



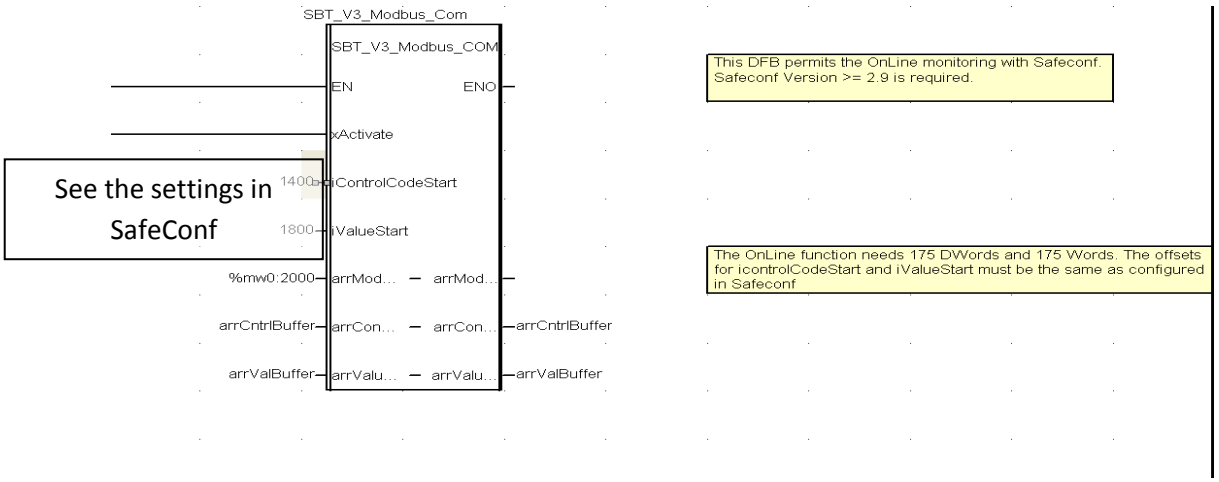
OnLine monitoring with Safeconf >= 2.9

From the version 2.9 of safeconf, it is possible to make OnLine monitoring. That means it is possible to view status of each variable in Safeconf, while the SBT application is running. For this, the whole PLC network used by the SBT must be running properly.

Some settings must be adapted in Safeconf and the PLC application :

- Settings in Unity.

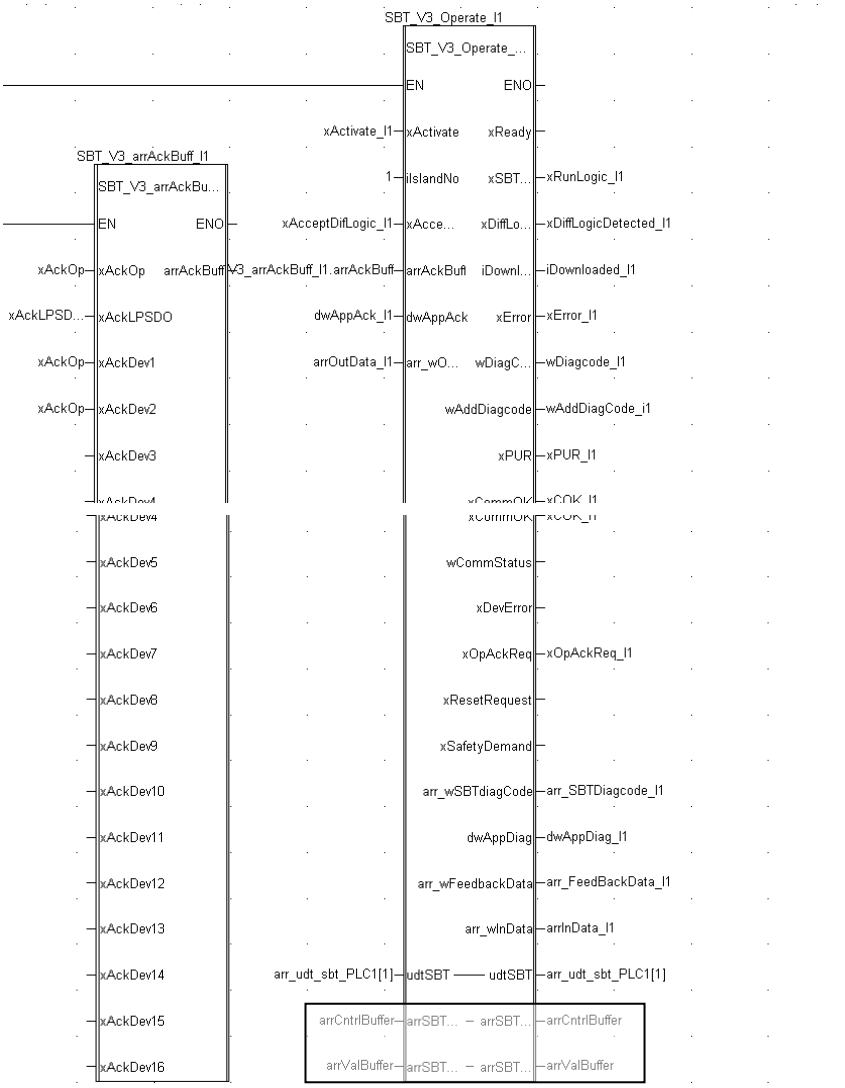
  The Unity example includes a section called SafeconfOnLine.



For this DFB, the settings are quiet simple :

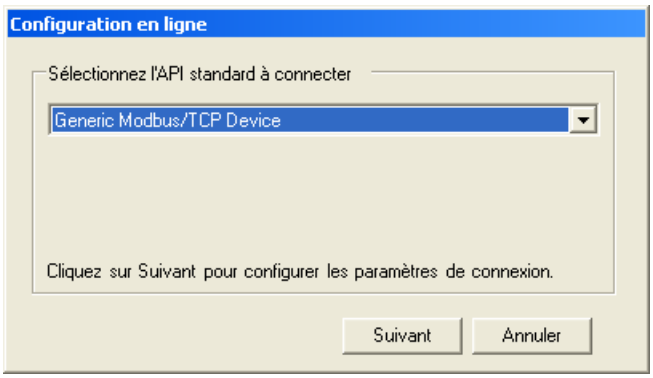
Adjust the value of iControlCodeStart and iValueStart to the same settings made in Safeconf (1400 and 1800)

Use 2 variables (arrCntrlBuffer and arrValBuffer), on the SBt_V3_Modbus_Com DFB and in the SBT_V3_Operate DFB :



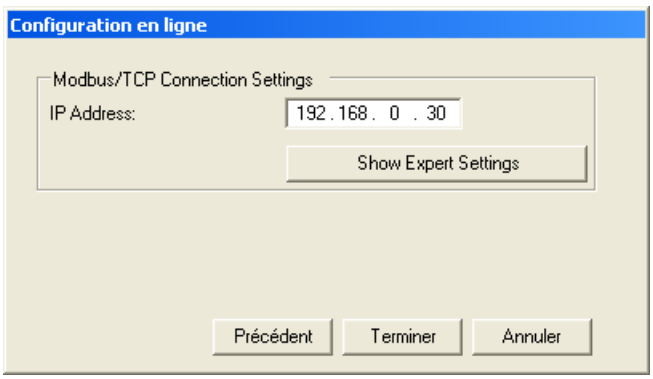
- Settings in Safeconf.

Right clic on the LPSDO V3 in the hardware list of the safeconf application, and select “On Line configuration”

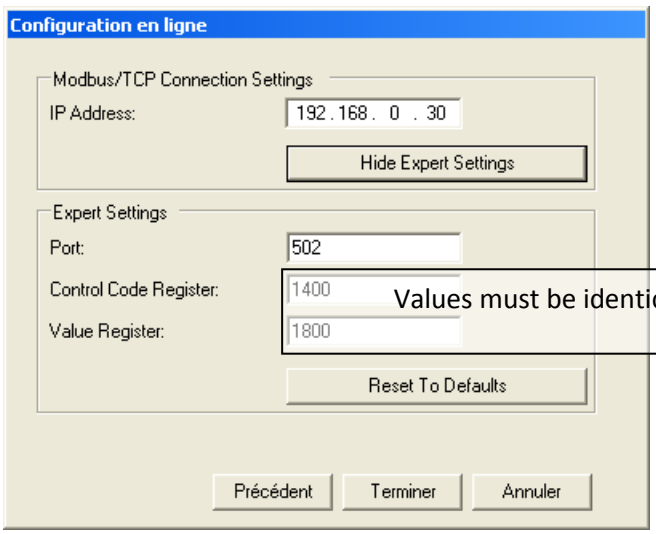


Select Generic Modbus/TCP Device.

Clic on “Next”



Type in the PLC adress (192.168.0.30), and clic “Show expert settings”.



Values must be identical to the SBT_V3_Modbus_COM settings

Now, it is necessary to adjust and/or memorize the control Register offset, and the Value Register.

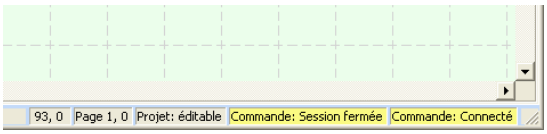
In this example, they are respectively set to 1400 and 1800.

The port N° has to be defined with 502, because the Schneider PLC (M340) Modbus TCP server is using this one, wich is not modifiable

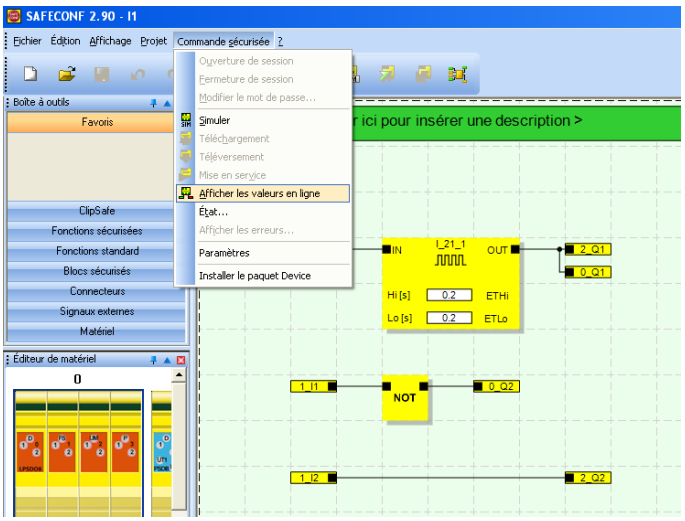
Clic “Terminate”.

The OnLine Configuration is finished in SafeConf.

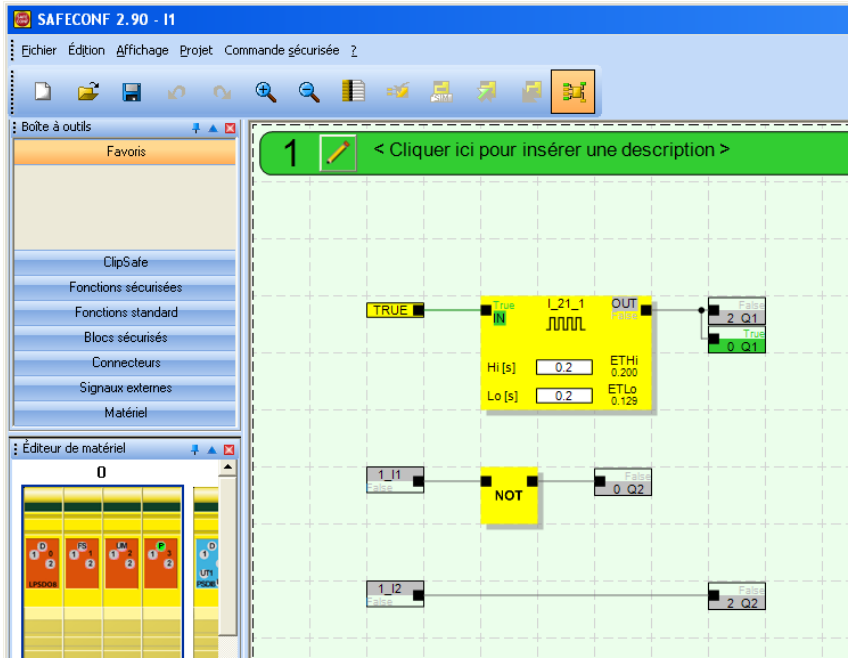
If the communication beetween Safeconf and the SBT is correct, you will see an information lighted at the right bottom of the screen :



Then, you can launch the OnLine visualisation, by the icon or the menu :

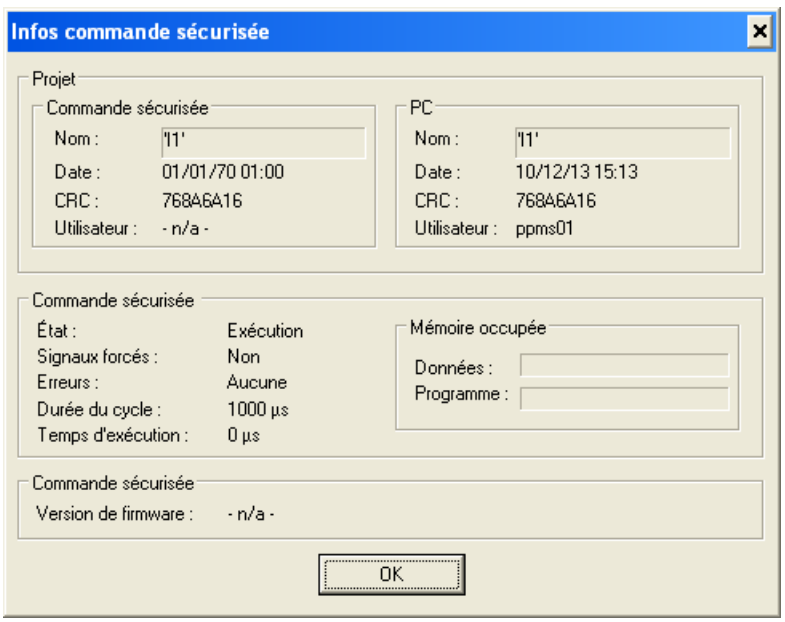


After some few seconds, the values will be displayed :



It is also possible to get some information about the SBT project:

With the menu “Command – status”, you will get this window :



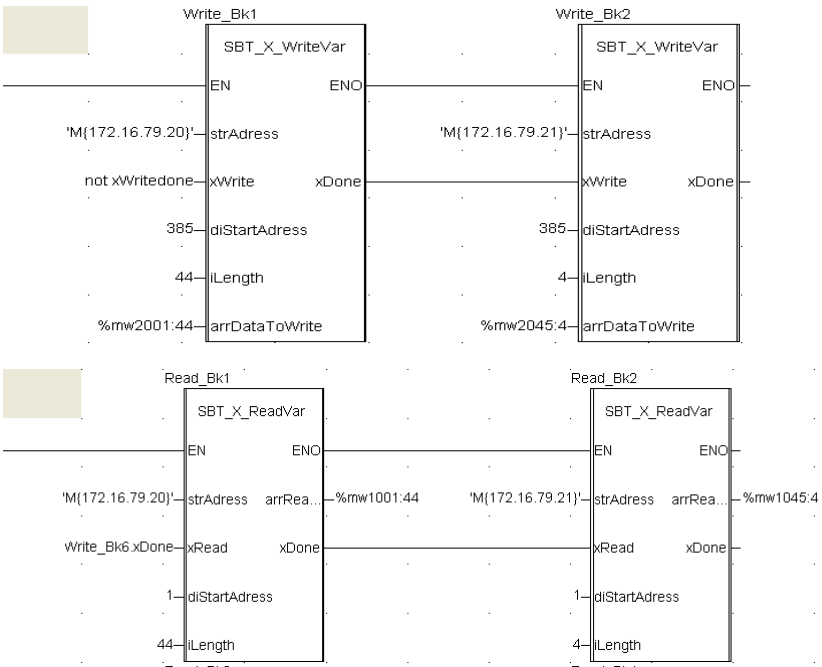
Problems with IOScanning.

In some cases (according to the CPU type of the PLC, or the type of PLC, it is impossible to synchronize the IOScanning and the application program. In such cases, the data constitency of the xPSDx devices can be perturbated, and so, the safety communication fails.

To be able to use the SBT technology if such problem occurs, it is necessary to use an other method to access to the devices process datas.

- For M340 and Premium, we need to use the Read_Var and Write_Var FBs. These FB permit to control each Modbus request. By this way, it is now easy to refresh all outputs, all inputs and then to run the safety application FB (operate, etc…)
- For the Quantum, Read_Var and Write_Var are not useable. Some other FBs are available for this PLC.

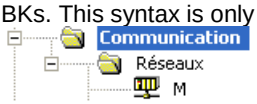
Example for M340 and Premium :



In this example, some special FBs have been designed to read and write process datas.

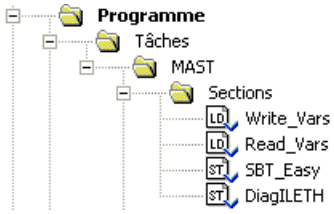
SBT_X_WriteVar is a FB wich permits to write the process datas of the BKs. This FB allows the writing of the SBT process datas, and also the standard process datas.

- strAdress : In this example, ‘M’ is the name of the network, and ‘172.16.79.20’ is the IP adress of one of the BKs. This syntax is only useable with M340 (for Premium, an other syntax is necessary).



- xWrite, xRead and xDone must be used to cascade the FBs, the one after the other. The FBs can either be launched the one after the other, or several at the same time. This depends of the CPU capacities. (for example, you can launch 5 FBs at the same time on a quantum, maybe more, depending on the CPU load)
- diStartAdress indicates the first adress of the registers of the BK we want to read or write
- iLength indicates the number of registers to read or write on the BK
- arrDataToWrite is the array of registers used in the PLC (Out)
- arrReadData is the array of registers used in the PLC (In)
- xRead or xWrite lanuches thes FB
- xDone indicates that the Read or Write was successful.

Architecture of the application :



In the Sections, we can find 3 parts :

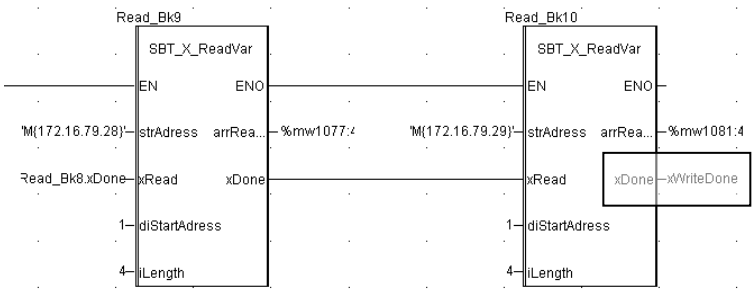
- Write_Vars uses all the FBs to write process datas to the BKs
- Read_Vars uses all the FBs to read process datas from the BKs
- SBT_Easy controls the safety island communications
- DiagILETH includes the FB to read Diagnostic of each BK, and restart it in case of NetFail

To synchronise R/W and the Operate FB, a bit must be generated by the Read and Write FBs. This bit indicates that all the Read and Write frames have been exchanged beetween the PLC and the BKs, so that the process datas can be used by the operate FB.

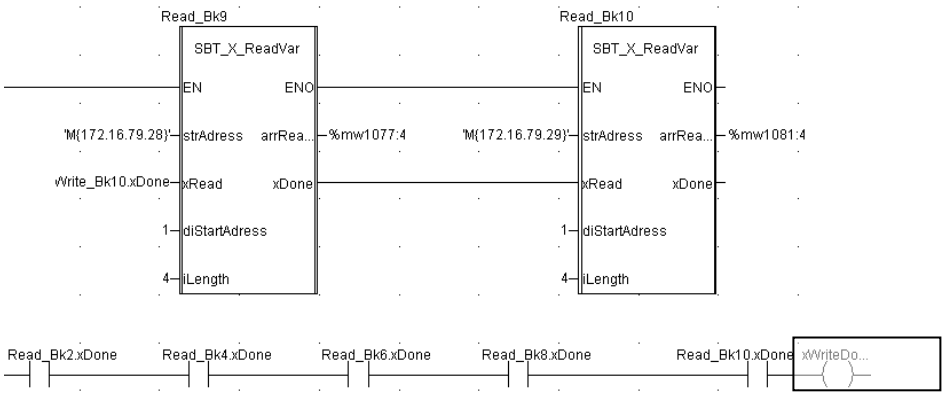
This bit is generated by the last ‘Write’ FB, or the sum of all the Read and Write FBs (if several FBs are launched together) :

There are 2 ways to sned the requests :

If FBs are <u>cascaded</u> : (low load charge of the network, but slow refresh of the I/Os)



If some FBs are <u>launched together</u> : (higher load of the network, but faster refresh of the I/Os) The number of simultaneous possible requests depends of the Ethernet coppler in the Schneider PLC. Plese refer to the dedicated documentation for more details)



The generated bit xWriteDone will be used in the ‘SBT_Easy’ part of the application.

If xWriteDone then

(For each island. Must be adapted according to the application needs *)*

(Here for the demo, all acknowledmenents are common *)*

arr_Udt_Easy_PLC1[1].xAckOp := arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].xAckLPSDO := arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[1]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[2]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[3]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[4]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[5]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[6]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[7]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[8]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[9]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[10]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[11]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

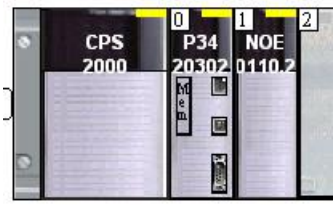
arr_Udt_Easy_PLC1[1].arrAckDev[12]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[13]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

arr_Udt_Easy_PLC1[1].arrAckDev[14]:= arr_Udt_Easy_PLC1[1].xAckGeneral;

```
arr_Udt_Easy_PLC1[1].arrAckDev[15]:= arr_Udt_Easy_PLC1[1].xAckGeneral;


(* Island 1 handling *)
arr_Udt_Easy_PLC1[1].xActivate :=   (%IW0.1.0.1.0 and %IW0.1.0.1.1 and (%mw1501 = 16#00e0) and
                                    %IW0.1.0.1.2 and %IW0.1.0.1.3 and (%mw1505 = 16#00e0) and
                                    %IW0.1.0.1.4 and %IW0.1.0.1.5 and (%mw1509 = 16#00e0) and
                                    %IW0.1.0.1.6 and %IW0.1.0.1.7 and (%mw1513 = 16#00e0) and
                                    %IW0.1.0.1.8 and %IW0.1.0.1.9 and (%mw1517 = 16#00e0) and
                                    %IW0.1.0.1.10 and %IW0.1.0.1.11 and (%mw1521 = 16#00e0) and
                                    %IW0.1.0.1.12 and %IW0.1.0.1.13 and (%mw1525 = 16#00e0) and
                                    %IW0.1.0.1.14 and %IW0.1.0.1.15 and (%mw1529 = 16#00e0) and
                                    %IW0.1.0.2.0 and %IW0.1.0.2.1 and (%mw1533 = 16#00e0) and
                                    %IW0.1.0.2.2 and %IW0.1.0.2.3 and (%mw1537 = 16#00e0));
arr_Udt_Easy_PLC1[1].udt_Buff.xSwapBytes := True; (* Swap High and Low bytes in the process datas *)
arr_Udt_Easy_PLC1[1].iIslandNo := 1; (* Island N° *)
```

(* %iw0.1.0.x.y : 0 = Rack, 1 = Module, 0 = channel : **if IOScanning is used**)
(* in this example, the card NOE 0110.2 *)



(IN buffers *)* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_LPSDO := %mw1001:24;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[1] := %mw1025:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[2] := %mw1029:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[3] := %mw1033:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[4] := %mw1037:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[5] := %mw1041:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[6] := %mw1045:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[7] := %mw1049:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[8] := %mw1053:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[9] := %mw1057:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[10] := %mw1061:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[11] := %mw1065:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[12] := %mw1069:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[13] := %mw1073:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[14] := %mw1077:4;* *arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_In_Dev[15] := %mw1081:4;*	*(* Main part *)* *SBT_V3_Easy_I1 (udt_SBT_Easy := arr_Udt_Easy_PLC1[1],* *udt_SBT_V3 := arr_udt_sbt_PLC1[1],* *arrSBTOnlCntrlBuf := arrSBTOnlCntrlBuf ,* *arrSBTOnlValBuf := arrSBTOnlValBuf);*	*(* OUT buffers *)* *%mw2001:24 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_LPSDO;* *%mw2025:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[1] ;* *%mw2029:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[2] ;* *%mw2033:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[3] ;* *%mw2037:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[4] ;* *%mw2041:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[5] ;* *%mw2045:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[6] ;* *%mw2049:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[7] ;* *%mw2053:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[8] ;* *%mw2057:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[9] ;* *%mw2061:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[10] ;* *%mw2065:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[11] ;* *%mw2069:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[12] ;* *%mw2073:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[13] ;* *%mw2077:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[14] ;* *%mw2081:4 := arr_Udt_Easy_PLC1[1].udt_Buff.arr_I_Out_Dev[15] ;*

(Call of the safeconf application *)*

SBTV3_Modbus (udtSBT := arr_udt_sbt_PLC1[1],

xActivate := True);

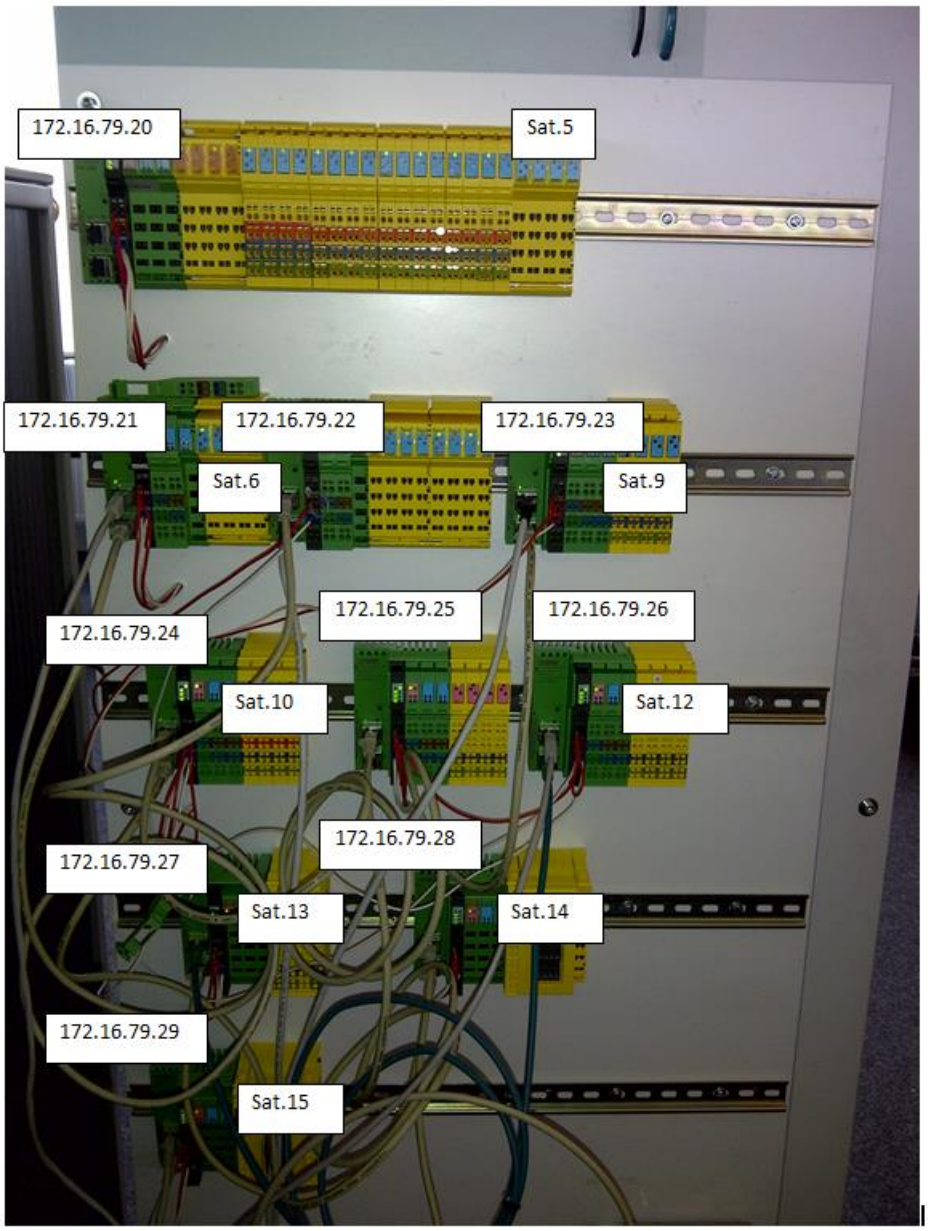
end_if;

The (* IN buffers *) part makes the link beetween the process datas read by the SBT_X_ReadVar and the variables used by the Operate FB.

The (* OUT buffers *) part makes the link beetwen the variables used by the Operate FB and the process datas written by the SBT_X_WriteVar.

The (*Island 1 handling *) part controls the activation of the mainFB ‘Operate’. This FB will have to be run only is the whole constellation of Modbus TCP BKs is active and runs OK. This is the reason why each status bit of the IOScanning contracts is controlled (to be sure that each BK is available and active), and if the local bus of the BKs is running correctly (Value E0 means RDY + ACT + RUN).

In the Quickstart package, you will find 2 applications using this constellation :

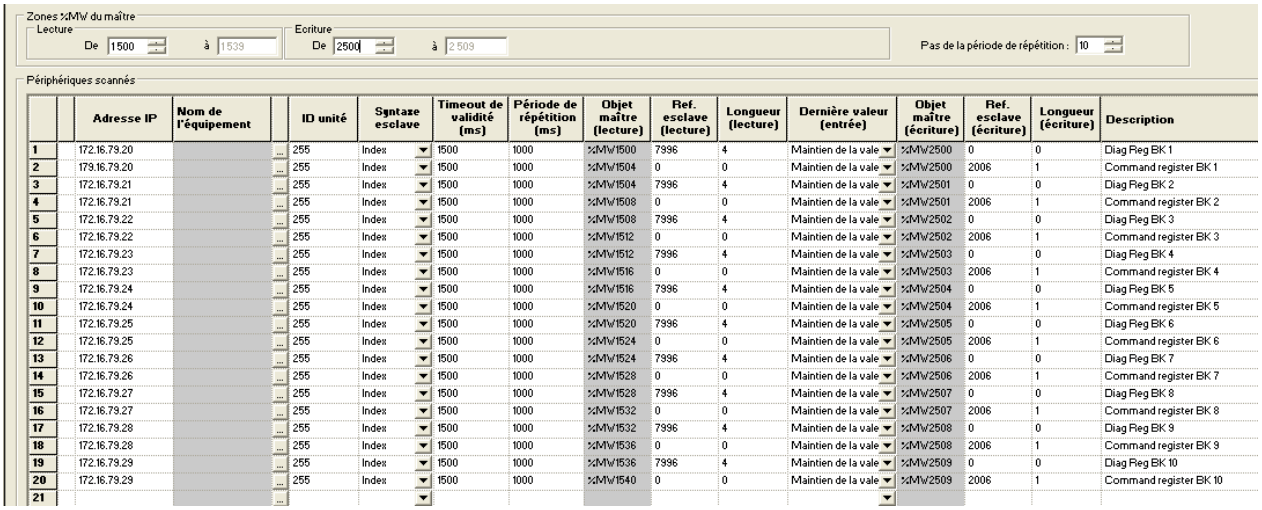


In this constellation, we can see 10 BKs, using 15 SBT devices. The LPSDO V3 is located just behind the BK 172.16.79.20.

- Sbt v3 easy 10 bks M340 (Unity XL) : each BK is read and writen the one after the other. This makes the cpu load very low, but the SBT cycle time is about 800 ms.
- Sbt v3 easy 10 bks Fast M340 (Unity XL) : a set of 5 BKs is read and writen at the same time. This makes the cpu load a little bit higher, but the SBT cycle time is about 280 ms.

For the control and the diagnostic of each BK, the FB called Diag_ILETH_BK_DI8_DO4 must still be used.

This FB needs to access to the registers 7996:4 and 2006. These registers can be accessed via the IOScanning with no problem (with 2 separated contracts), or with Read_Var and Write_Var.



Zones %MW du maître

Lecture De 1500 à 1539 — Ecriture De 2500 à 2509 — Pas de la période de répétition : 10

Périphériques scannés

	Adresse IP	Nom de l'équipement	ID unité	Syntaxe esclave	Timeout de validité (ms)	Période de répétition (ms)	Objet maître (lecture)	Ref. esclave (lecture)	Longueur (lecture)	Dernière valeur (entrée)	Objet maître (écriture)	Ref. esclave (écriture)	Longueur (écriture)	Description
1	172.16.79.20		255	Index	1500	1000	%MW1500	7996	4	Maintien de la vale	%MW2500	0	0	Diag Reg BK 1
2	179.16.79.20		255	Index	1500	1000	%MW1504	0	0	Maintien de la vale	%MW2500	2006	1	Command register BK 1
3	172.16.79.21		255	Index	1500	1000	%MW1504	7996	4	Maintien de la vale	%MW2501	0	0	Diag Reg BK 2
4	172.16.79.21		255	Index	1500	1000	%MW1508	0	0	Maintien de la vale	%MW2501	2006	1	Command register BK 2
5	172.16.79.22		255	Index	1500	1000	%MW1508	7996	4	Maintien de la vale	%MW2502	0	0	Diag Reg BK 3
6	172.16.79.22		255	Index	1500	1000	%MW1512	0	0	Maintien de la vale	%MW2502	2006	1	Command register BK 3
7	172.16.79.23		255	Index	1500	1000	%MW1512	7996	4	Maintien de la vale	%MW2503	0	0	Diag Reg BK 4
8	172.16.79.23		255	Index	1500	1000	%MW1516	0	0	Maintien de la vale	%MW2503	2006	1	Command register BK 4
9	172.16.79.24		255	Index	1500	1000	%MW1516	7996	4	Maintien de la vale	%MW2504	0	0	Diag Reg BK 5
10	172.16.79.24		255	Index	1500	1000	%MW1520	0	0	Maintien de la vale	%MW2504	2006	1	Command register BK 5
11	172.16.79.25		255	Index	1500	1000	%MW1520	7996	4	Maintien de la vale	%MW2505	0	0	Diag Reg BK 6
12	172.16.79.25		255	Index	1500	1000	%MW1524	0	0	Maintien de la vale	%MW2505	2006	1	Command register BK 6
13	172.16.79.26		255	Index	1500	1000	%MW1524	7996	4	Maintien de la vale	%MW2506	0	0	Diag Reg BK 7
14	172.16.79.26		255	Index	1500	1000	%MW1528	0	0	Maintien de la vale	%MW2506	2006	1	Command register BK 7
15	172.16.79.27		255	Index	1500	1000	%MW1528	7996	4	Maintien de la vale	%MW2507	0	0	Diag Reg BK 8
16	172.16.79.27		255	Index	1500	1000	%MW1532	0	0	Maintien de la vale	%MW2507	2006	1	Command register BK 8
17	172.16.79.28		255	Index	1500	1000	%MW1532	7996	4	Maintien de la vale	%MW2508	0	0	Diag Reg BK 9
18	172.16.79.28		255	Index	1500	1000	%MW1536	0	0	Maintien de la vale	%MW2508	2006	1	Command register BK 9
19	172.16.79.29		255	Index	1500	1000	%MW1536	7996	4	Maintien de la vale	%MW2509	0	0	Diag Reg BK 10
20	172.16.79.29		255	Index	1500	1000	%MW1540	0	0	Maintien de la vale	%MW2509	2006	1	Command register BK 10
21														

It is also necessary to run the 'Operate' FB only if the whole constellation of devices is correctly accessed via Modbus TCP. To insure a such behaviour, the 'Operate' FB will have to be run only if the IOScanning (if used) contracts are OK (%IW0.2.0.1.x) and if the BKs are running correctly.

The code to get diagnostic and control each BK in case of Netfail, for example, is given as an example in ST language :

```
Diag_IL_ETH_BK_DI8_DO4_BK1 (xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1500:4,
                iCdeRegister => %mw2500);
Diag_IL_ETH_BK_DI8_DO4_BK2(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1504:4,
                iCdeRegister => %mw2501);
Diag_IL_ETH_BK_DI8_DO4_BK3(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1508:4,
                iCdeRegister => %mw2502);
Diag_IL_ETH_BK_DI8_DO4_BK4(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1512:4,
                iCdeRegister => %mw2503);
Diag_IL_ETH_BK_DI8_DO4_BK5(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1516:4,
                iCdeRegister => %mw2504);
```

```
Diag_IL_ETH_BK_DI8_DO4_BK6(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1520:4,
                iCdeRegister => %mw2505);
Diag_IL_ETH_BK_DI8_DO4_BK7(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1524:4,
                iCdeRegister => %mw2506);
Diag_IL_ETH_BK_DI8_DO4_BK8(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1528:4,
                iCdeRegister => %mw2507);
Diag_IL_ETH_BK_DI8_DO4_BK9(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1532:4,
                iCdeRegister => %mw2508);
Diag_IL_ETH_BK_DI8_DO4_BK10(xAutoQuit := True,
                tScantime := t#5s,
                xPP := False,
                arrDiagRegisters := %mw1536:4,
                iCdeRegister => %mw2509);
```

Difference between M340 and Premium.

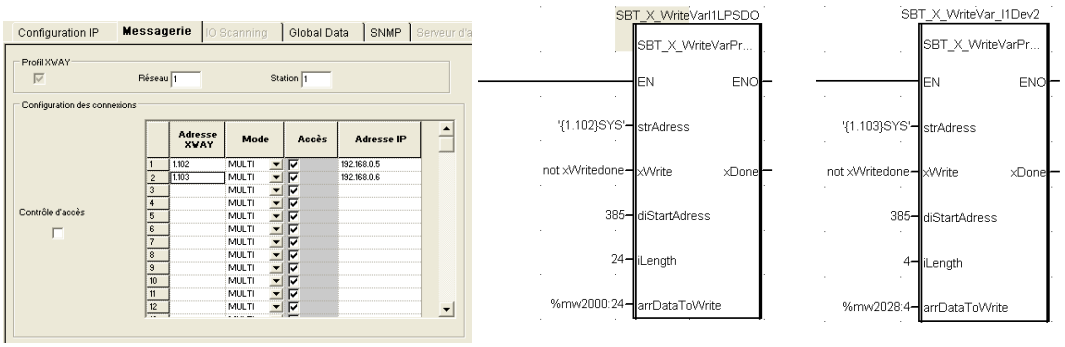
The FB SBT_X_ReadVar and SBT_X_Write_Var use a ‘standard’ FB available in Unity, wich is called ‘ADDM’. This FB creates the address to use by READ_Var and Write_Var. For the M340, the input parameter of this FB is quiet easy to create and understand.

‘M{172.16.79.20}’ means : use the communication card called M and the Modbus TCP server is @ 172.16.79.20

For a Premium, the ADDM FB is not available. The Premium needs the FB called ADDR. The syntax of the adress used by ADDR is a little bit different from the one used by ADDM. If you want to use the SBT V3 technology with a Premium using Modbus TCP (with IL ETH BK…), you will need the example dedicated to the Premium. In this example, 2 FBs are modified and specially designed for the premium (SBT_X_ReadVarPremium and SBT_X_WriteVarPremium).

- You will need to configure the “messagerie” parameters to access to the IL ETH BK with an xWay adress.

  In the example below, you can see that 2 stations are configured, with IP adresses 192.168.0.5 and 192.168.0.6. The XWay adresses 1.102 and 1.103 will be used in the application to access to the process datas of the SBT devices.



  In this example, you can see that the XWay adresses configured in the network parameters, are used also as the strAdress parameter in the SBt_X_WriteVarPremium and SBT_X_ReadVarPremium.

  The application example for Premium uses a simpler constellation (BK1 with LPSDO + PSDOR, and BK2 with PSDI8). The name of this application is SBT_V3_Premium.

The example SBT_V3_Easy_Premium is dedicated to the Premium. You can also use it to understand the way the Modbus TCP BKs are refreshed with Read_Var and Write_Var.