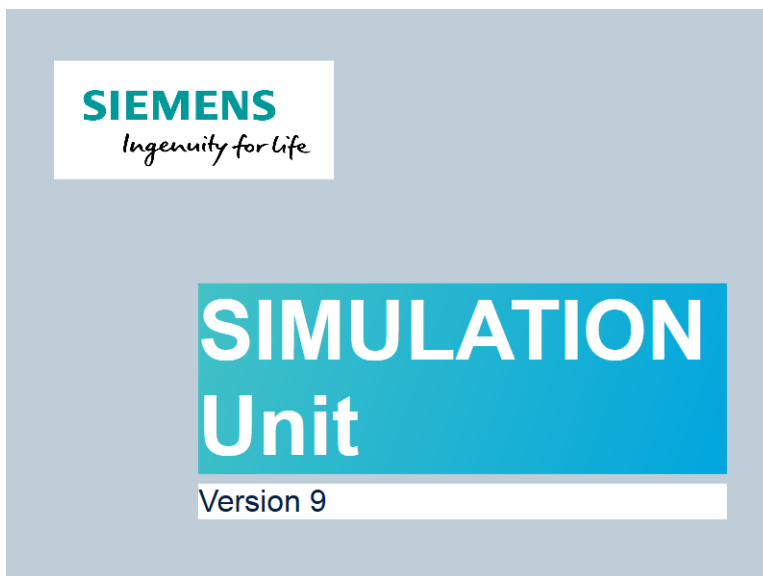

SIMULATIONUnit Manual



1. Functional Description

SIMULATIONUnit is a simulation system for simulating field bus devices using Profibus DP protocol and/or Profinet protocol. This powerful tool supports simple inputs/outputs from Profibus slaves/Profinet devices as well as dynamic simulation of user defined complex functions for end devices e.g. Motors, Valves. The behavior of such devices can be manipulated using already integrated digital control logic modules. SIMULATIONUnit has been designed for Windows. This simple and easy to use simulator provides all the tools for quick designing and testing of automation projects during factory acceptance tests (FAT).

1.1. Simulation of PROFINET IO with SIMBA PNIO / SIMULATION UNIT PN





With the SIMULATION UNIT PN module you are able to simulate up to 256 Profinet I/O devices at one PROFINET IO network (e.g. ET200S PN, CP1616 as device... etc).

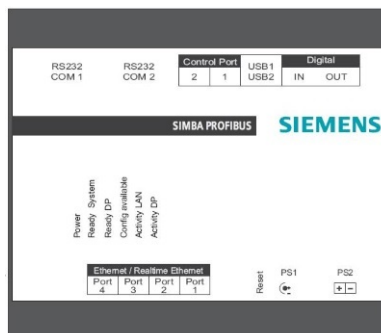
SIMULATIONUnit can simulate up to 32 Profinet channels in one project. It is also possible to simulate with SIMULATION UNIT PN and SIMULATION UNIT PB modules in the same project at the same time.

The simulation of PROFINET IO can be carried out using any PROFINET IO MASTER system. For example:

- PROFINET CPU from the SIMATIC S7-300, S7-400, S7-1200 or S7-1500 product family
- External Ethernet communication processors e.g. CP443-1 Advanced
- CP1616 as Master
- Further PROFINET IO norm master

See also [Getting started](#)

1.2. Simulation of PROFIBUS-DP with the SIMBAProfibus Box / SIMULATION UNIT PB



With the SIMULATION UNIT PB box you are able to simulate up to 2 profibus lines with 125 DP slaves at one PROFIBUS DP bus (e.g. ET200M, SIMOCODE... etc) with user selectable transmission rate. SIMULATION UNIT PB can simulate up to 250 profibus slaves.


SIMULATIONUnit can simulate up to 32 Profibus channels in one project. It is also possible to simulate with SIMULATION UNIT PN and SIMULATION UNIT PB modules in the same project at the same time.

The simulation of Profibus can be carried out using any Profibus DP MASTER system. For example:

- DP CPU from the SIMATIC S7-300 and S7-400 product family
- DP-CPU or DP-CP from the S7-1200 and S7-1500 product family
- External DP communication processors e.g. CP443-5, IM467
- CP581 with PROFIBUS DP communication interface
- Further PROFIBUS DP norm master

See also [getting started](#)

1.3. Naming convention

SIMULATIONUnit		Simulation Software
SIMULATION UNIT PB		Simulation Hardware for Profibus
SIMULATION UNIT PN		Simulation Hardware for Profinet IO
Simba		old, deprecated name of Simulation Unit
SU		Abbreviation for SIMULATIONUnit or SIMULATION UNIT

2. Getting Started

This page contains a short description for installing and setting up a simulation project. Click at the links to get more detailed descriptions.

2.1. Installation of SIMULATIONUnit software

Install SIMULATIONUnit Software: execute SIMULATIONUnit_Vx.x.x_Setup.exe on a Windows 7 PC or newer Windows version.

2.2. Connect the simulation box

Connect to a 24V power supply.

SIMULATION UNIT PN	Connect the simulation PC to the jack labeled with CTRL . Connect the ProfinetIO controller to the jack labeled with P1 .
SimbaPNIO	Connect the simulation PC to the jack labeled with Control Port 1 . Connect the ProfinetIO controller to the jack labeled with Realtime Ethernet 1 .
SIMULATION UNIT PB	Connect the simulation PC to the jack labeled with CTRL . Use one or two Profibus cables to connect the Profibus controller to the Profibus ports of the SIMULATION UNIT
SimbaProfibus	Connect the simulation PC to the jack labeled with Control Port 1 . Use up to 8 Profibus cables to connect the Profibus controller to the Profibus ports of the SimbaProfibus. The two channels of a redundant H machine must be connected to opposite ports (e.g. 0 and 1, or 2 and 3).
Profibus-Migration gateway	Connect the simulation PC to the jack labeled with Control Port 1 . Connect a real system with Profibus controller and devices to Profibus-Port 0 and 1 . Connect Port 2 and 3 to a master to simulate the slaves. The Migration gateway listens to the I/Os at Port 0 and 1 and copies them to Port 2 and 3 .

2.3. Export a Step 7 hardware configuration

If SIMULATIONUnit and STEP7 are running on different machines, copy the file SDBView.reg from the SIMULATIONUnit directory to the STEP7 PC. Double click at SDBview.reg. The data files to be imported to SIMULATIONUnit will be created in the directory <STEP7>/S7tmp/SDBData/program/DOWN/rXXsXX if you click "Save and Compile" in a STEP7 hardware project.



Export the Hardware configuration to a .cfg file. Menu Station → Export...






2.4. Export a TIA Portal hardware configuration

SIMULATIONUnit must be installed on a PC with TIA Portal.

Select a PLC in TIA Portal and compile the hardware configuration. The folder C:\TIAExports then contains sdb*.dat or *.oms files to be imported to SIMULATIONUnit.



2.5. Creation of a Simulation Project

- create a new simulation project: click the button  in [portal view](#).
- define a project name. After "OK", SIMULATIONUnit creates a project folder (*.spf) and the necessary files for an empty project and changes into [project view](#) for further working steps.
- Select and configure a SIMULATIONUnit box: **menu "Configuration"** →  **"Import hardware"**. In the following view [Import hardware](#) you can add import files as f. i. system datablocks (.dat), OMS files (.oms) or XML files with hardware informations. At last you can press the button  and the hardware import for all detected bus systems will be started.



- After hardware import is successful the program switches to the  [hardware editor](#). Here you can change or complete the imported configuration.





- Here you can assign the imported hardware to a SIMULATION UNIT box found in the network. Enter the IP address by hand or scan the network and drag/drop a found SIMULATION UNIT Box in the list to the imported Simba.

Remarks:

the selected IP address has to be in the same subnet as the simulation computer!
 When simulating PNIO systems the IP address of the SIMULATION UNIT Box has to be in another subnet as the simulated devices!

Example:

IP address of the simulation computer:	140.80.123.1	Subnet Mask: 255.255.0.0
IP address of the SIMULATION UNIT box:	140.80.123.110	Subnet Mask: 255.255.0.0
IP addresses of the simulated devices:	140.90.x.y	Subnet Mask: 255.255.0.0

- Pressing the button  saves the configuration.
- Download project into the SIMULATION UNIT box: first press . After the SIMULATION UNIT boxes are connected, button  becomes active for downloading the configuration into the boxes.

See also: [simulation of Profinet components](#) or: [simulation of Profibus components](#)

3. SIMULATIONUnit Features

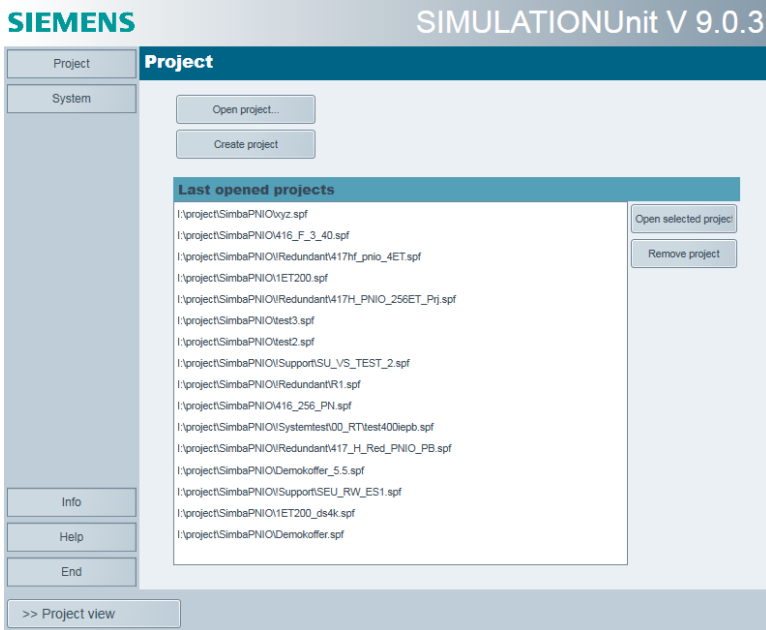
SIMULATIONUnit uses two different views (Portal view and Project view) for arranging control functions into logical groups.

3.1. The portal view

The portal view contains all necessary control elements for the administration of simulation projects, for global system adjustments and the help system.

3.1.1. Projects

Open, create or delete simulation projects.



Create project: Select a folder and a project name.

SU creates a folder with the ending .spf (SIMULATIONUnit project folder). The project must not be created inside a folder with the ending .spf.

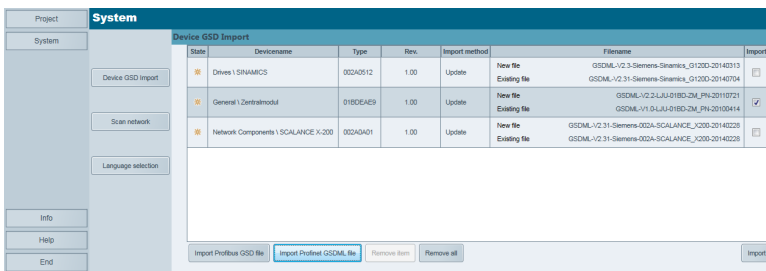
Open project: double click onto a project in the list of last opened projects or click onto "Open project..." and select a Project.xml file inside a .spf folder.

Remove project: If you answer the question "Delete project from disc ?" with Yes, the project will be deleted from the list and from the disc.

3.1.2. System

In the view "System" you can find the controls for different system adjustments.

3.1.2.1. Device GSD import



You can import device descriptions for new devices from GSD files (for Profibus) or GSDML file (for Profinet) into the SIMULATIONUnit hardware database. The new devices are disposable if you create a new project or import new hardware data after that.

Select one ore more files to import. SIMULATIONUnit will check if the device is already in the library and if the selected file is newer. If you click onto "Import", all files with the checkbox selected in the column Import will be imported.

Note:

The imported files are stored in the user directory (C:\Users\User\AppData\Roaming\SIMULATIONUnit\hwlib) After a software update they are still available.

3.1.2.2. Scan network

In this view you can find or configure SIMULATION UNIT boxes available in the network.

SIEMENS SIMULATION Unit V 9.0.0

Project: **System**

System

Device Import

Scan network

Language selection

Info

Help

End

>> Project view

Type	Name	IP-Address	Netmask	Gateway
SimbaPNI0	Simba_Schneidk	140.80.61.140	255.255.0.0	0.0.0.0
SimbaPNI0	Simba NFS4A3	140.80.123.161	255.255.0.0	140.80.123.109
SimbaPNI0	SU Bergmann f:	140.80.123.181	255.255.0.0	0.0.0.0
SimbaProfibus	SimbaProfibus_J	140.80.123.119	255.255.0.0	0.0.0.0
SimbaProfibus	SimbaProfibus_J	140.80.61.86	255.255.0.0	0.0.0.0
SimbaPNI0	SIMBA_PNI0_Jv	140.80.99.99	255.255.0.0	0.0.0.0

Network scan

Blink

Firmware update

Save changes

Column	Meaning	changeable
Type	Type of the simulation hardware. Simba will be displayed instead of SIMULATION UNIT.	
Name	arbitrary name of the SIMULATION UNIT	✓
IP Address, Netmask, Gateway	IPv4 settings	✓
MAC Address	unique device address	
DHCP	Automatically obtain an IP address from a DHCP server in the network.	✓

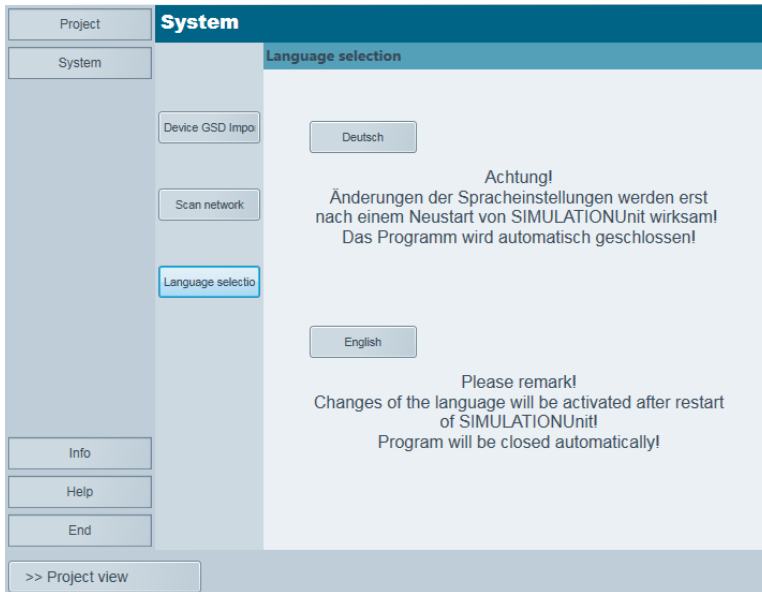
Click into a field to change the value. Press the "Save changes" Button to apply the changes.

Firmware Update

Select a SIMULATION UNIT in the list and click onto "Firmware update"

The current firmware version will be shown. Select a .upd file and follow the instructions on the screen.

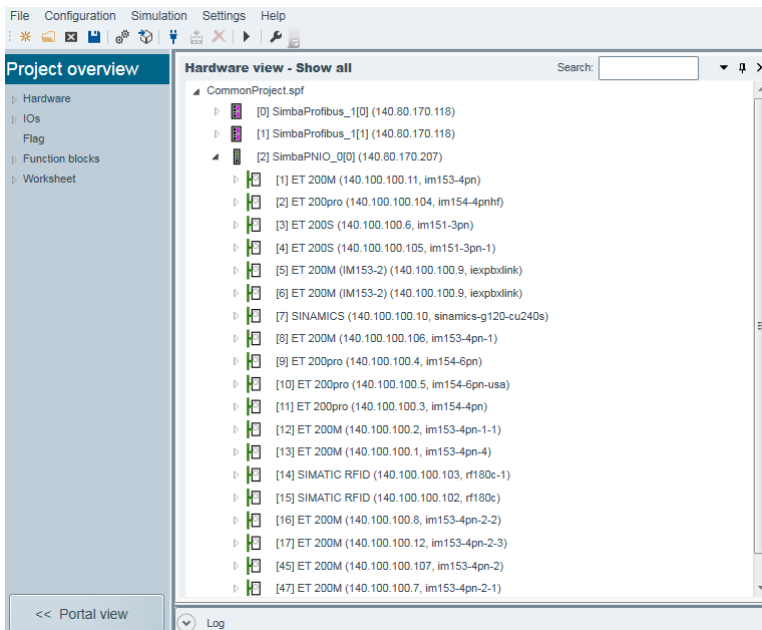
3.1.2.3. Language settings



In that view you can configure the system language of the SIMULATIONUnit software. Implemented are "German" and "English" this time. Changing the language will restart SIMULATIONUnit.

3.2. The Project view

The Project view is the "working view" of SIMULATIONUnit. The user can create, configure or run simulation projects here.



3.2.1. Navigation within the project

At the left side of the project view you can find a navigation tree that shows all elements of the opened simulation project. The view to control one element can be opened by double click on the element.

3.2.1.1. Elements

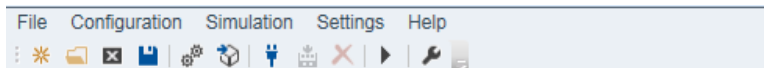
Type	Comment
Hardware	Hardware tree of the configured SIMULATION UNIT boxes or networks.
IOs	IO addresses of the imported or configured devices.
Flag	A list of all variables used in the function instances.
Function blocks	User defined function blocks and configured block instances.
Worksheet	for a better overview while simulation you can drag and drop single IO values or function blocks on a work sheet.

Remark:

The observation or control of the IO values is possible in all views at the same time.

3.2.2. The menu bar

The menu bar contains text menus and also buttons for quick access.



The function of the buttons (left to right):

Menu File

- * "Create project..." - create a new, empty .spf project folder
- 📁 "Open project..."
- 🗑️ "Close project"
- 💾 "Save project as..."
 - "Export I/O mapping" - save all I/O values to a .csv file
 - "Import I/O mapping" - load all I/O values from a .csv file
 - "Exit" - Close project and exit SIMULATIONUnit

Menu Configuration

- ⚙️ Edit hardware
- 🔗 "Import hardware" Profibus Profinet
- 📄 Device GSD Import

Menu Simulation

- 🔌 "Connect all"
- 🔌 "Disconnect all"
- 📄 "Download all"
- ✖️ "Delete all"
- ▶ Start run time system
- Stop run time system
 - new function block
 - copy function

Menu Settings

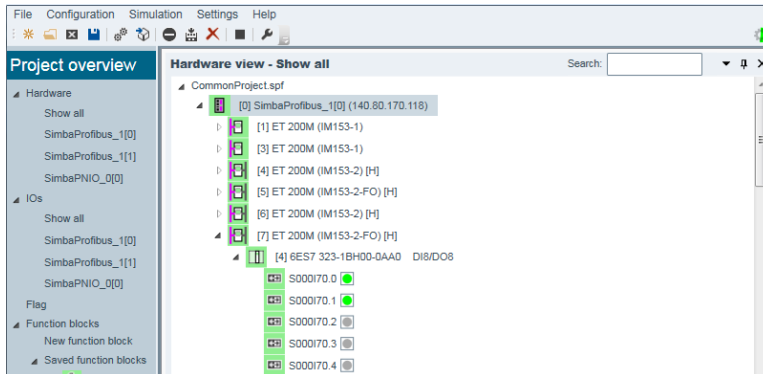
- 🔧 Options

Menu Help

- ℹ️ "Info" - Version information and copyright
- 📖 "Online manual" - this manual

At the right side in the toolbar there is an icon ⚙️ which displays the usage of the selected cycle time (see Options)

3.2.3. The hardware view



The hardware view shows the SIMULATION UNIT boxes and their configuration in a tree structure. Depending on the node type is a direct control of the IO's or different actions with the nodes via context menus possible. (f. i. device fault, alarms, module fault ...).

On the right side you can find a small search field for searching any node in the tree.

3.2.3.1. Meaning of the symbols in the hardware view

- SIMULATION UNIT PN
- SIMULATION UNIT PB
- Profibus slave
- redundant Profibus slave
- Profinet Device
- Profinet Device, active at Port P2
- redundant Profinet device, active at Port P1
- redundant Profinet device, active at Port P2
- Module or Submodule
- I/O
- I/O Bit with value 0 (false)
- I/O Bit with value 1 (true)

The background color indicates the state

- no color - not connected or not present
- green - Simulation running, I/O data exchange present
- light blue - loaded, but no connection to a controller
- red - failure
- gray - listening mode: I/Os are controlled by another program
- yellow - passive redundant Profibus Slave
- yellow-green - active failsafe module

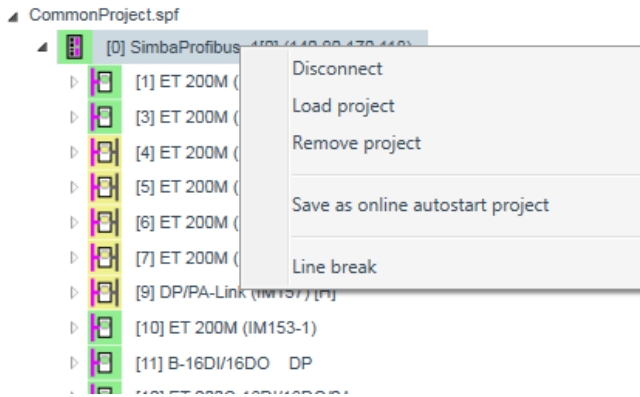
3.2.3.2. Listening mode

If more than one instance of a simulation program is connected to the same channel of one SIMULATION UNIT, all instances except one switch to listening mode.

In listening mode all inputs and outputs of the SIMULATION UNIT are displayed, but none of them can be operated. Downloading is blocked. The background colour of all modules and channels is grey. In the hardware view you will see the text "**Controlled by SimWhatever.exe**"

- If SIMULATIONUnit and another simulation program (e.g. Simit) are accessing the same channel, then SIMULATIONUnit changes to listening mode.
- If two or more instances of the same simulation program are accessing the same channel, then the first started instance is active and all others switch to listening mode.
- Connections from different PCs to the same SIMULATION UNIT channel are blocked.

3.2.3.3. Save an autostart project in the SIMULATION UNIT Box

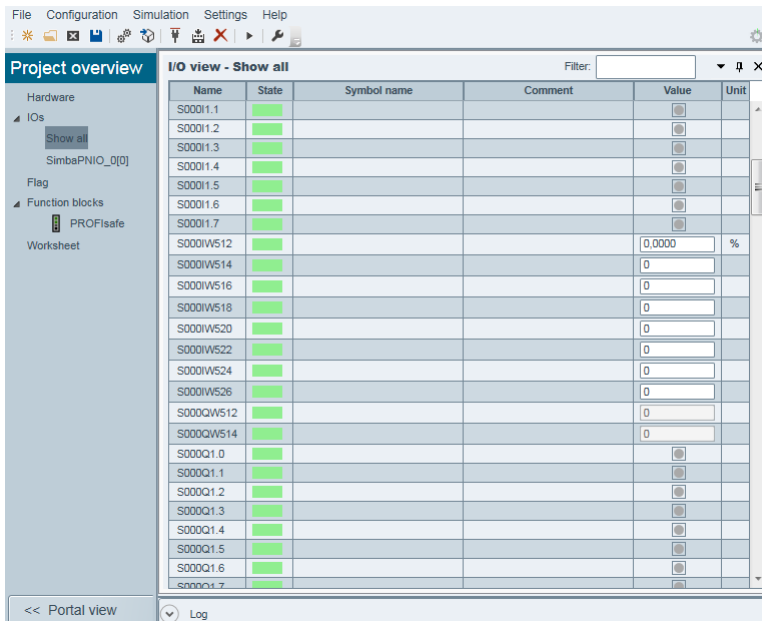


Via a context menu in **hardware view** it is possible to save a project into online connected boxes. That opens a possibility to configure ready to use simulation boxes. The box loads that autostart project after power on. In hardware view this is signed by a comment **"AutoStartProject"**.

The autostart project is stored in flash memory of the box. If a USB stick is plugged into a **SIMULATION UNIT PN Box**, the project will be stored on the USB stick.

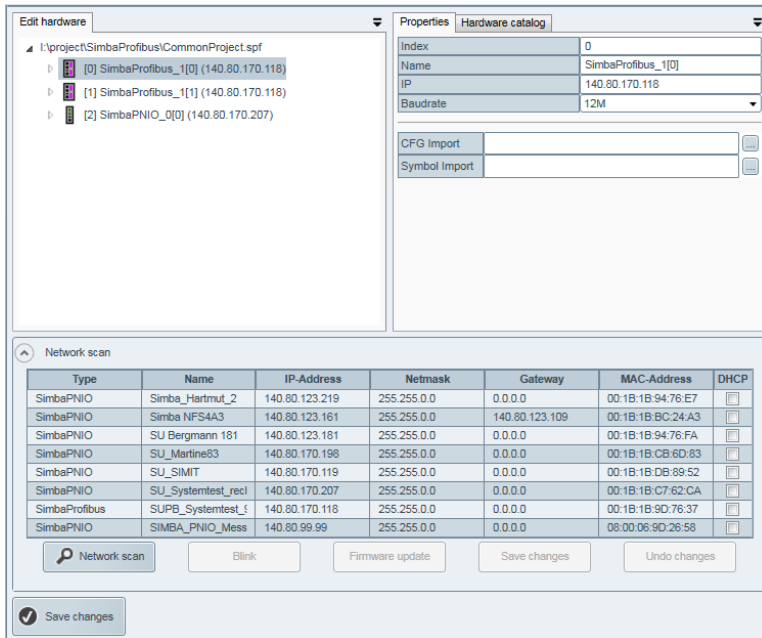
If a micro SD card is plugged into the backside of a **SIMULATION UNIT PN Box**, the project will be stored on the SD card.

3.2.4. The IO view



In the IO view are shown all IO values configured in the project in a List. Additional here are shown logical addresses and possibly imported signal names or comments. Control or observation of the IO's is also possible here. The IO view can be limited by using a filter.

3.2.5. The hardware editor



The menu Configuration → "Edit hardware" opens the hardware editor. Here you can see in the left part the hardware tree of the project (analog to the hardware view).

In the right part are shown the properties of a left selected hardware node. Most of these properties are editable here.

Example

If a SIMULATIONUnit box is selected left, the IP address is shown at the right and editable. This can be edited or found by network scan. After a network scan in the bottom part of the window the SIMULATIONUnit boxes found in network are shown. Per drag and drop the IP address will be assumed into the properties of the box.

When a **SIMULATION UNIT PN** is selected in hardware tree, it is also possible to edit [topology settings](#). Set up the device number and port number of the first device that is connected to the PNIO controller here.

If the project contains additional redundancy these values have to be edited also for the second controller. If no topology is set up in TIA or Step 7 you can use the default properties.

Optionally you can also import an export file (.cfg) into your hardware. This imports all configured names of devices and modules from the STEP-7 project. If you don't import a cfg file, the original names from the SIMULATIONUnit hardware database will be shown.

You can also combine the imported hardware with a symbol file from PCS-7 to show logical IO symbols.

The second tab on the right side contains the hardware catalogue of SIMULATIONUnit. You can use it to configure hardware configurations by hand. Drag and drop devices or modules into the hardware tree with the mouse.

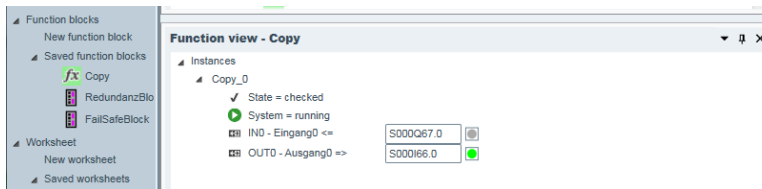
A click with the right mouse button onto a device/module in the hardware tree opens a menu for editing:

- **Remove** removes the selected elements
- **Copy** copies the selected elements into the clipboard
- **Cut** copies the selected elements into the clipboard and deletes them
- **Paste** pastes the elements in clipboard into the hardware tree

Ctrl + mouse click selects an additional element in the tree Shift + mouse click selects all elements from the previous selected to the new element

With the button **Save changes** the edited properties will be saved and the additional files will be imported.

3.2.6. Function blocks



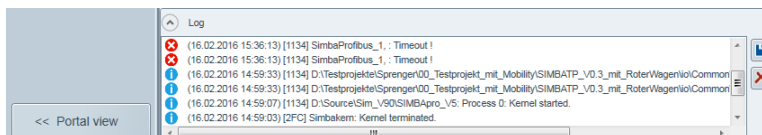
By double click on a saved function the view of the instances of this function will be opened for control or observation. More to function blocks please look at [runtime system](#).

3.2.7. Worksheets



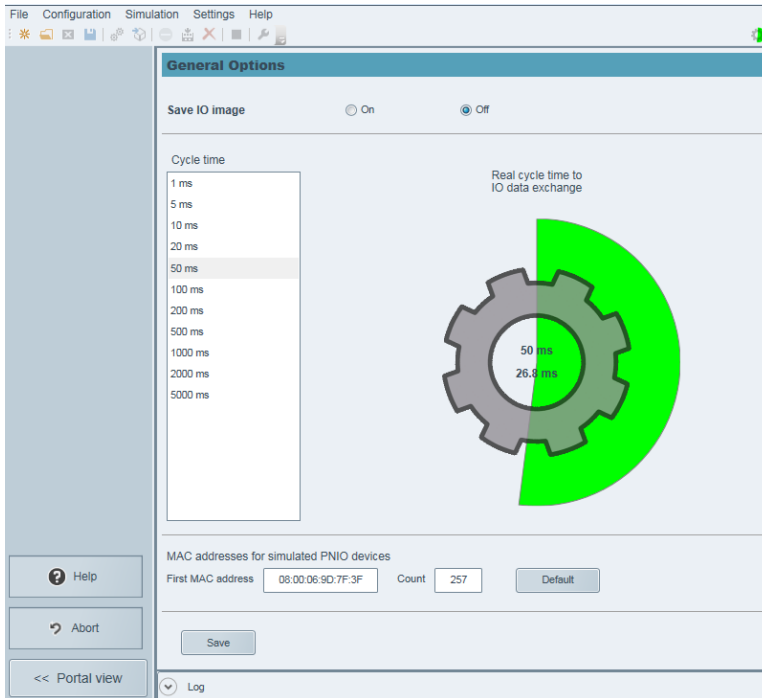
Worksheets are free configurable windows containing IO's or function instances. You can place them on the worksheet by "drag and drop". Worksheets deliver the possibility to group IO's and function instances by functionality to get a better overview over the simulation project.

3.2.8. The log window



At the bottom of the project view window you can find the view of the log file. In the log file there will be written important events, warnings an errors during the simulation. The log file can be saved into a text file for further diagnostics.

3.2.9. System settings (options)



The menu **Settings** → **Options** opens a view for adjusting global program options.

In the upper part of that view you can select, if SIMULATIONUnit saves the input IO image. That is useful to get the same scenario when opening the project again.

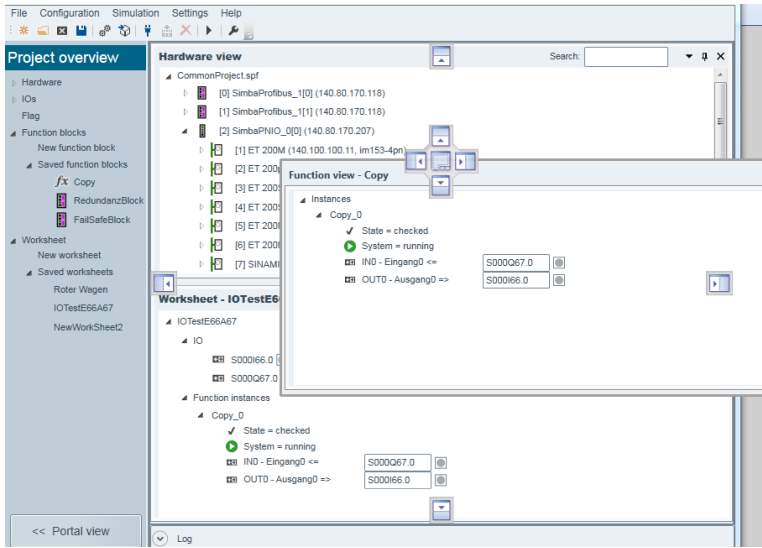
It is also possible to adjust the internal program cycle.

The program cycle should be adjusted so that a safe transmission of all telegrams within one cycle is guaranteed!

In the bottom part the band of used MAC addresses for the import of PNIO devices can be edited.

3.2.10. Window arrangement

The windows in the right part of the program can be grouped as you want by holding their headline with the mouse and dropping them into the marks afterwards to see.



4. Simulation of PROFIBUS Components

The Simulation of Profibus systems is possible with SIMBAProfibus box / SIMULATION UNIT PB box connected to a real Profibus DP master. Data exchange between SIMBAProfibus box / SIMULATION UNIT PB box and the simulation computer occurs over ethernet network at control port 1 of the box. Maximal bus requirements at the Profibus lines are:

- maximum number of devices per line: 125
- maximum baud rate (DP): 12 MBaud

4.1. Remarks for creating a simulation

The SIMBAProfibus box / SIMULATION UNIT PB box must not be configured in the SIMATIC S7 hardware configuration. The box is not a Profibus device. The SIMBAProfibus box / SIMULATION UNIT PB box acts in the SIMATIC-System system-neutral. You can configure up to 32 Profibus lines in one simulation projekt.

4.2. SIMULATION UNIT PB



4.2.1. Cables

4.2.1.1. Power connection

L+	Connect to a power supply with a voltage of 24 V, min 1 A here.
M	Ground.

4.2.1.2. Connection to Simulation computer

Connect a Simulation PC via an ethernet network to the port CTRL of the SIMULATION UNIT PB box. This port is used to control the box. Polarity and speed of the connection will be detected automatically.

4.2.1.3. Connection of the Profibuses

Connect up to 2 profibus cables to the SUB-D jacks at the top of the box. The profibus connected to the upper jack is called SIMBAProfibus_x[0] in the simulation project, the lower one is SIMBAProfibus_x[1].

4.2.2. Installation and heat

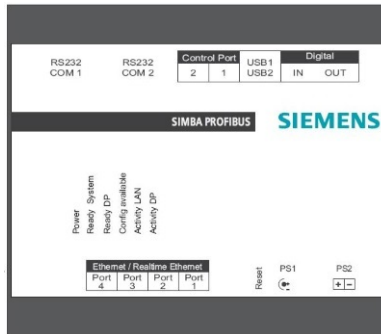
The SIMBAProfibus box / SIMULATION UNIT PB box can be placed on a table or snapped to a cap rail with the clamp on the back side. The housing is also used for chipset cooling. Please make sure that the heat can be emitted to the surrounding air. The environment temperature should not exceed 70°C.

4.2.3. Description of the LEDs

LEDs on top	
PB1, PB2	green light when activity at the Profibus ports

LEDs on top	
CTRL	green light, when connected to a network, blinks orange at data exchange
PWR	green light, if the power supply is on.
RDY	green blinking while booting after switching on or reset, green light, when system is operational. orange light, if a project is loaded to the box.

4.3. SimbaProfibus



4.3.1. Cables

4.3.1.1. Power connection

The box can be connected to power over PS1 or PS2.

PS1	Connect to a power supply with a voltage between 9 and 36 V, 2 A here.
PS2	The connection has to be made with 2 wires linked to a power supply with a voltage between 9 and 36 V, 2 A (for example 24V supply PS307 from Siemens).

4.3.1.2. Connection to Simulation computer

Connect a Simulation PC via an ethernet network to Control Port 1 of the SIMBAProfibus box. This port is used to control the box. Polarity and speed of the connection will be detected automatically.

4.3.1.3. Connection of the Profibuses

Connect up to 8 profibus cables to the SUB-D jacks at the sides of the box. The profibus connected to CH0 is called SIMBAProfibus_x[0] in the simulation project, CH1 is SIMBAProfibus_x[1], etc.

If you want to simulate a redundant system, you have to connect the redundant Profibus lines to 2 ports which at the opposite side of the box. You can simulate H systems at channels 0 and 1, or at channels 2 and 3, at channels 4 and 5, or at channels 6 and 7.





4.3.2. Installation and heat

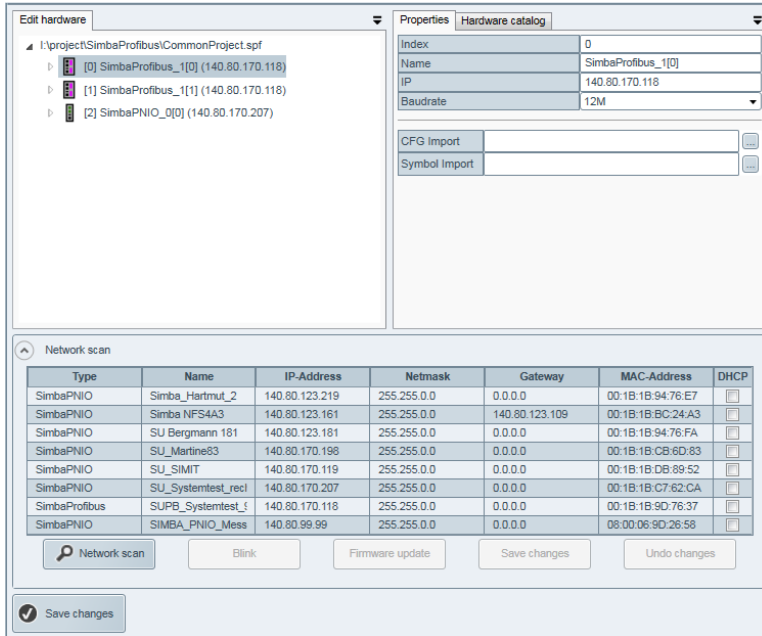
The SIMBAProfibus box can be placed on a table or snapped to a cap rail with the clamp on the back side. The housing is also used for chipset cooling. During normal operation, the module can warm up to more than 40°C. This has no negative effect on functionality. Please make sure that the heat can be emitted to the surrounding air. The environment temperature should not exceed 40°C.

4.3.3. Description of the LEDs

LEDs on top	
Power	orange light, if the power supply is on.
Ready System	green blinking while booting after switching on or reset, green light, when system is operational.
Ready DP	green light, if the Profibus DP Simulation module is activated.
Config available	green light, if a project is loaded to the SIMBAProfibus.
Activity LAN	green light, when connected to a network.
Activity DP	green light at activity at one of the Profibus ports.

4.4. Addressing the box


IP addresses for the SIMBAProfibus box / SIMULATION UNIT PB box can be configured in SIMULATIONUnit in the Hardware-Editor or at [Portal](#) → [System](#) → [scan network](#). Scan the connected network and configure an IP address in in your IP band.



Enter the IP address by hand or scan the network and drag/drop a found SU in the list to the imported Simba.

4.5. Creating a new project

Process the following steps to create a simulation projekt:

Select menu **"File"** → **"Create project"** or press Button  in [portal view](#).

In the following file box select a folder and edit a name for your project. SIMULATIONUnit creates a new folder for the simulation project (extension ".spf") and copies the necessary files into it.

Remark:

Don't create a (*.spf) project folder inside of a *.spf folder!

4.5.1. Hardware configuration

The hardware configuration can be done manually with the integrated configuration tool "Configure Hardware" or automatically with the import function.

4.5.1.1. Manual configuration


Menu "Configuration" → "Edit hardware"

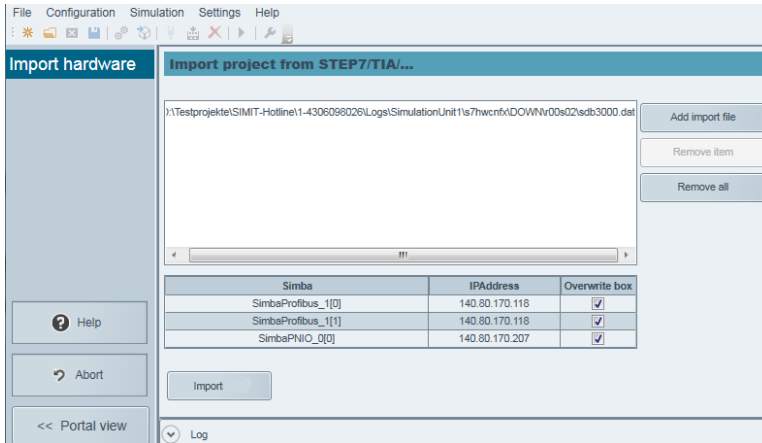
When configuring SIMULATIONUnit with [Edit hardware](#) please insert the hardware components from the hardware library into the hardware tree. SIMULATIONUnit only checks the logical consistency of the configuration (for example: slave number).

Remark:

SIMULATIONUnit does not check the consistency of the logical addresses like STEP7 Hardware-Config! That can cause unwanted changes of signals during the simulation!

4.5.1.2. Import hardware

Menu **"Configuration"** →  **"Import hardware"**.

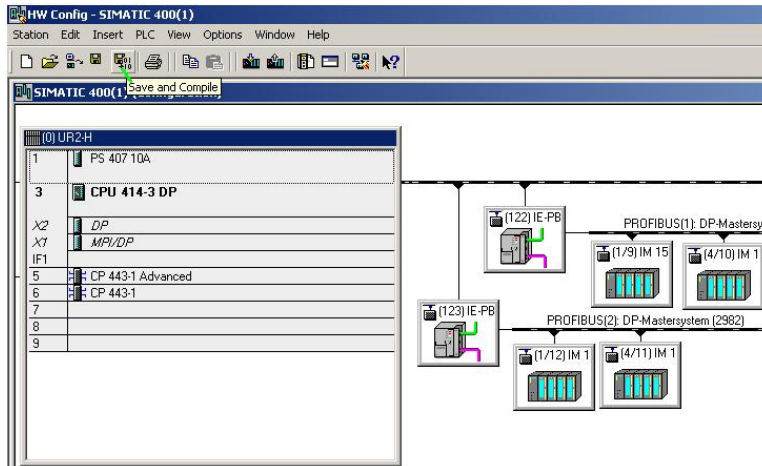


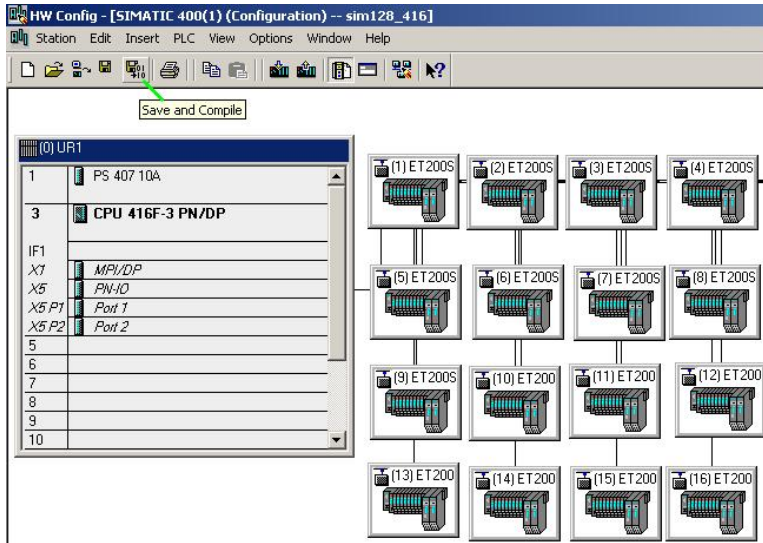
In the program view you can add import files as system datablocks (.dat), OMS files (.oms) or XML files to the import filebox and then import these files into the SIMULATIONUnit project.

4.6. Step 7 Hardware config export

4.6.1. SDB file export

In Simatic-Manager, open the hardware configuration and click "Save and compile". This action will create system data blocks (sdb?????.dat files) which can be imported into SIMULATIONUnit.





For the hardware import function, one system file (SDB1xxx or SDB2xxx.dat) will be needed for each channel. The file SDBxxx.DAT is located on the SIMATIC PC under:

~ /Siemens/Step7/S7tmp/SDBData/program/DOWN/rXXsXX (or ~/Siemens/Step7/S7tmp/SDBData/s7hwcfx/DOWN/rXXsXX).

If the sdb files cannot be found in the folder above, HW-Config deleted them after compiling or downloading them into the plc. To prompt S7/PCS7 to keep the files after compile or download it is necessary to execute the registry file "SDBView.reg" (for 32 bit systems) or "SDBView_Win64.reg" (for 64 bit systems) delivered by SIMULATIONUnit installation on the S7/PCS7 computer. This will be done automatically if you install SIMULATIONUnit on the same computer as S7/PCS7.

After a restart of S7/PCS7 and after compilation or download of the hardware into the plc you will find the SDB files in the folder above.

If you copy the files to another destination, please make sure that all files sdb*.dat will be copied because SIMULATIONUnit needs all cross references between the SDB-files. Each file SDBxxx.DAT contains the hardware information for one Profibus network. There exists no rule between number of SDB-files and PROFIBUS-lines. Begin the import with the biggest file.

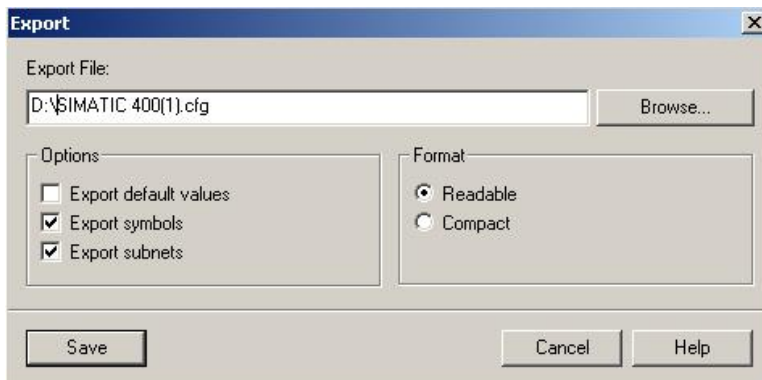
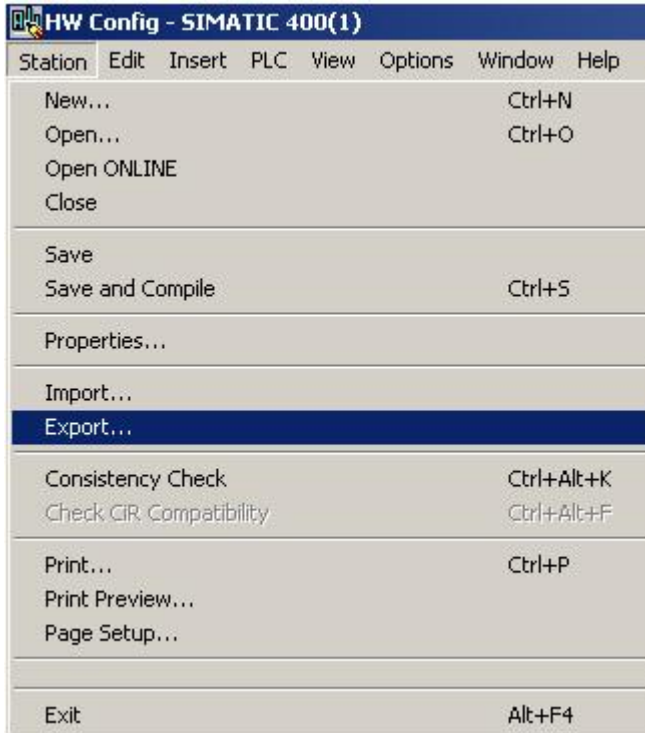
For import of the SDB's of a technology-plc (f. i. 315T-2DB) please use in the following file box the numerical range of SDB2xxx for the plc part and the range SDB1xxx for the technology part.

4.6.2. cfg file export

Optional an export file in S7/PCS7 project (*.cfg) can be imported. In this case all station names from the S7-Project will be imported to the SIMULATIONUnit-Project. Also Failsafe or Redundant slaves will be recognized from *.cfg file.

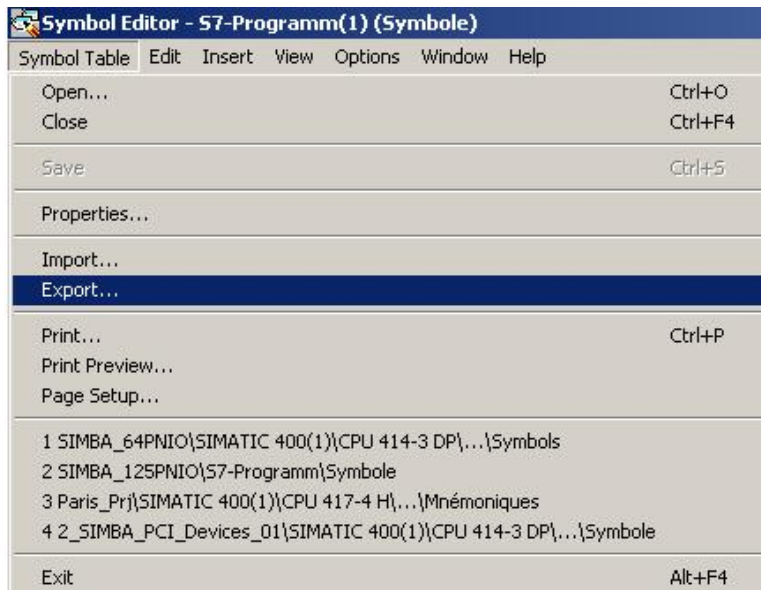
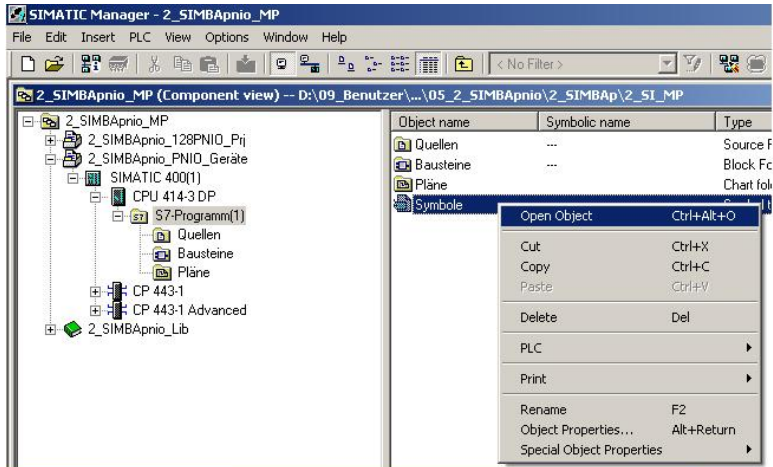
Remark:

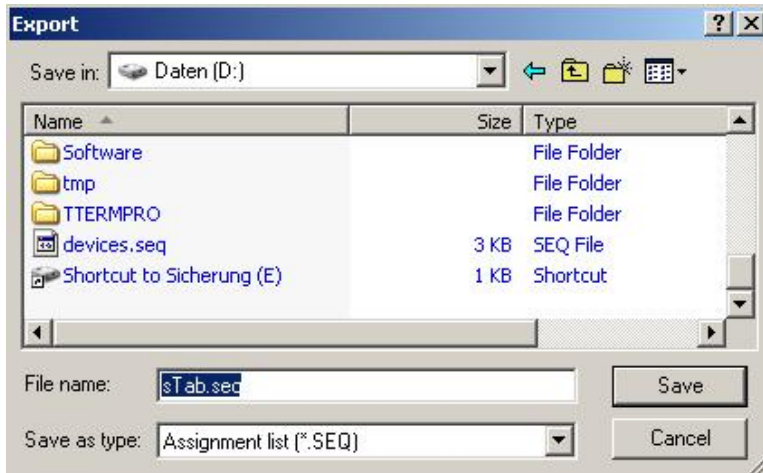
If you use digital IO modules with packed IO's in your hardware configuration, it is necessary to import SDB's and *.cfg file! Otherwise you will not have access to all IO channels!



4.6.3. Symbols

Optionally a symbol table (.seq or .asc) can be imported. A later import of symbol table is also possible without using the SDBfile at the same time.





This step is only useful, if you have configured the symbols and you want to use them in SIMULATIONUnit. In SIMATIC Manager open the symbol table of the STEP 7-Project and export it in *.seq-format. The language set up in SIMULATIONUnit must be the same as the project language of the STEP 7 Project.

4.6.4. Copy the export files over to the simulation computer

Copy the .cfg file (if desired), .seq file (if desired) and the complete folder `~\Siemens\Step7\S7tmp\SDBData\program\DOWN\rXXsXX` (or `~\Siemens\Step7\S7tmp\SDBData\s7hwcfnx\DOWN\rXXsXX`) over to the SIMULATIONUnit computer.

4.7. TIA-Portal hardware config export

Data export from TIA-Portal is possible only from TIA version V13.x and higher!

If you use SIMULATIONUnit and TIA-Portal on the same computer, just select "TIA portal import" at the SIMULATIONUnit installation. You will find then the exported hardware data files ("OMS" files) after compile process by TIA-Portal in the folder "C:\TIAExports". Select button "OMS files" in the hardware import box and navigate to the folder "C:\TIAExports".



If you use SIMULATIONUnit and TIA-Portal on different computers, please select "TIA portal import" at the SIMULATIONUnit installation also. Under `%SIMULATIONUnit install folder%\TIAImport` you will find the setup program "TIA2SUSetup.exe". Please install that on your TIA portal computer. After that you can convert the exported "OMS" files with the program "TIA2SUConvert" into an XML file, which you can import into SIMULATIONUnit on your simulation computer.

4.7.1. Symbols

You can export the symbols (PLC-Tags) from TIA Portal as an Excel file (.xlsx). This can be imported also by SIMULATIONUnit.



Go to PLC tags → Show all tags. Then press the Export button.

4.8. Hardware download, display and diagnostic

After importing the hardware data the SIMULATIONUnit program switches into the [hardware edit](#). You have to complete the configuration f. i. with the IP address of your SIMULATIONUnit box (look at [Addressing the box](#)).

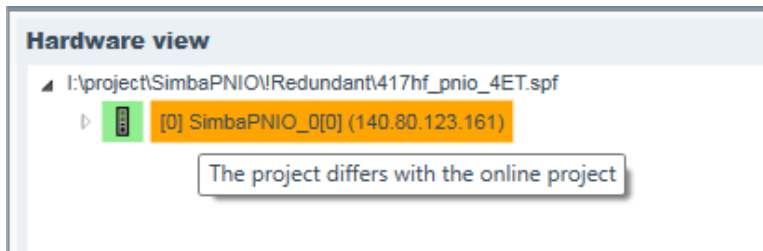
After that you can connect the computer with the box and download the hardware configuration.

4.8.1. Hardware icons

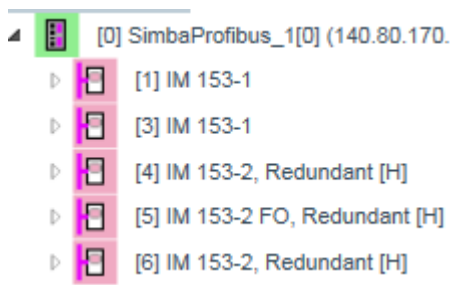
- After connecting successfully to the box icon changes into green colour.



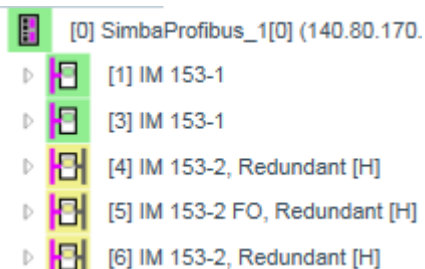
- If there is another project loaded to the SIMULATION UNIT, the SIMULATION UNIT will be marked orange.



- After downloading the hardware configuration the Profibus devices will be shown as "defect" (red colour).



- when the master has parametrized the devices, they will be shown in green colour.



- In the last case additional will be shown if devices are redundant (two busses) or single (one bus), and if in case of redundancy the device is the active (green colour) or the reserved one (yellow colour).

Attention:

No check of the slave or module type is done here! Slaves, which are configured in the simulation module but not configured in the automation system are not recognised. In case of doubt you should download the hardware configuration again!

4.9. Redundant systems

The simulation of ET200M-H or DP-Y-Link connected to a redundant system is possible. Connect two channels of one SIMBAProfibus box / SIMULATION UNIT PB box to the two Profibus lines of one redundant master system.

Use the automatic import function to create a redundant simulation system. Refer to chapter [Import hardware](#) for the automatic import. It is important to import both Profibus lines at the same time. The import function will detect any redundant slave. The special card typical for redundancy will be generated automatically. They will be added to the sequence control with their necessary parameters by SIMULATIONUnit software.

Attention:

When configuring an H-system for 4- or 8- channel Profibus boxes it is necessary to configure the system always for an "even" channel (0,2,4,6) and the next channel.

Simulation of two "half" H-machines on 2 consecutive channels is not possible!

4.9.1. Handling and data transfer of the H-system

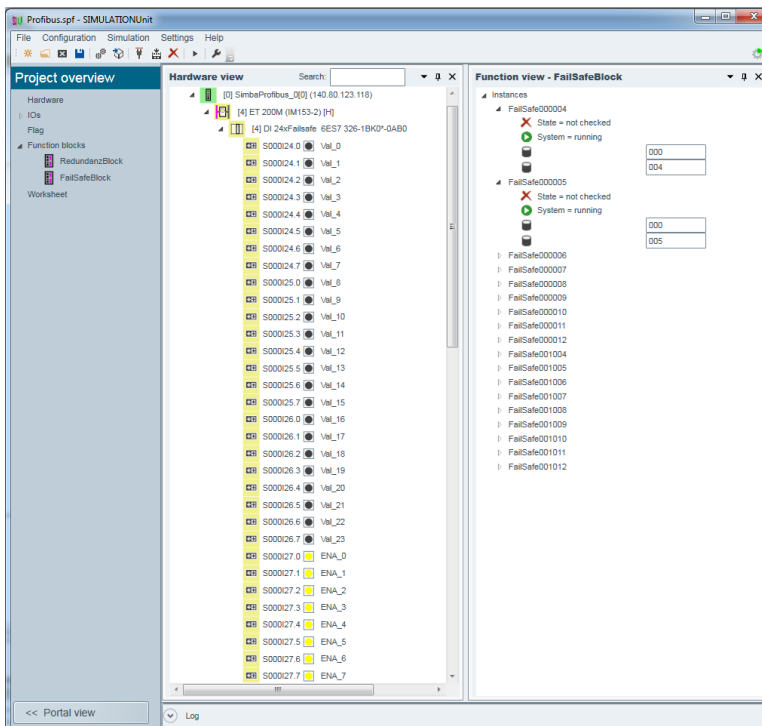
INput and OUTput module exist twice in SIMULATIONUnit for a redundant system. Important for handling, monitoring, functionality of sequence control or use of programming interface: **only even channel of the SIMBAProfibus box / SIMULATION UNIT PB box is relevant for use!** SIMULATIONUnit software takes care to get always the process values from the active channel of the H-system. Inputs will be generated automatically for both channels. The display of the active/standby channel is indicated by different colours for the slave icons in "Hardware view". (see also [Hardware icons](#)).

4.10. Failsafe

With the SIMBAProfibus box / SIMULATION UNIT PB box you can also simulate Failsafe functionality. It is possible to simulate Failsafe on a single channel as in combination with redundancy ("FH" systems).

4.10.1. Configuration of Failsafe components

The import function will detect any Failsafe component. The special card typical for "FailSafeBlock" will be generated automatically. They will be added to the sequence control with their necessary parameters by SIMULATIONUnit software.



In a digital failsafe module you can find additional to the data bits an *enable* bit for switching the channel on or off.

4.10.2. Load and start SIMBAProfibus box / SIMULATION UNIT PB box Failsafe functionality

After the download of the hardware configuration with Failsafe components it might be necessary to restart the Profibus master. Some masters only send failsafe parametrizations during startup.

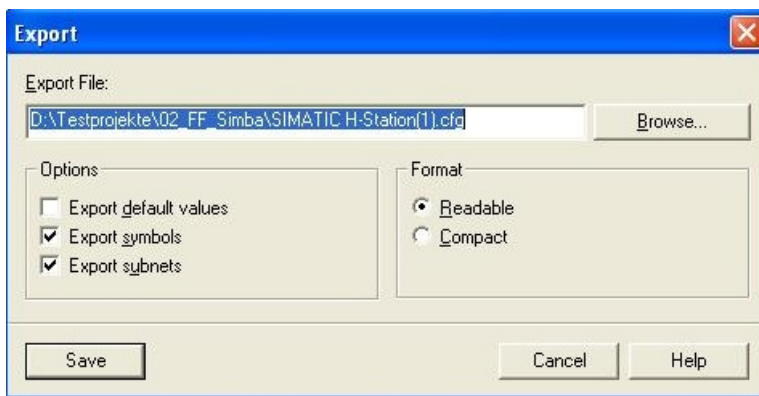
4.11. Foundation-Fieldbus systems

SIMBAProfibus boxes are able to simulate FF systems behind a DP-FF link. The hardware configuration will be imported as described under [Redundant systems / Failsafe components](#). After this step the data exchange of the complete FF system can be simulated.

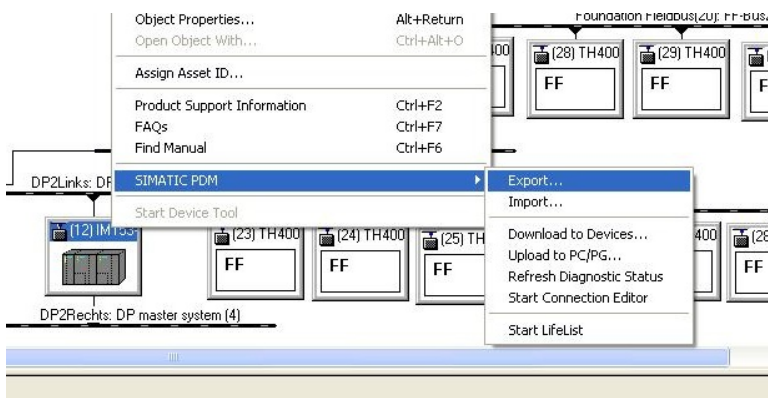
4.11.1. Data import from PDM

For more diagnostics and functionality of the FF system it is necessary to import more informations from PDM.

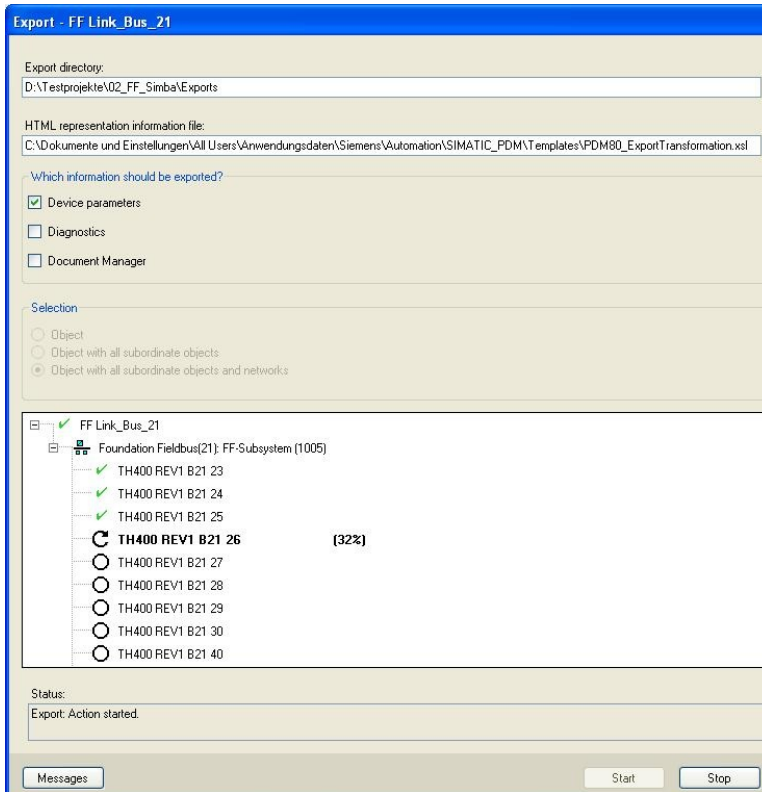
At first you have to export the HW-Config configuration.



After that export the PDM configuration into the same folder. It's usable to delete older PDM export files by hand because PDM doesn't do that.




Please select at PDM export box on the lower left side "Object with all subordinate objects and networks".



And last but not least make an hardware import in SIMULATIONUnit together with the cfg-file(See also [Import hardware](#)).

4.11.2. Download into the SIMBAProfibus box / SIMULATION UNIT PB box and start of the FF-functions

After the download of the hardware configuration into the box please select the button  "Start module function" to activate the FF functions in the box.

Attention:

This time it's not possible to simulate full functionality of FF systems in SIMBAProfibus! It's possible that you get error messages when using FF functions in your plc because SIMBAProfibus will generate negative FF system responses in case of missing data or functions.

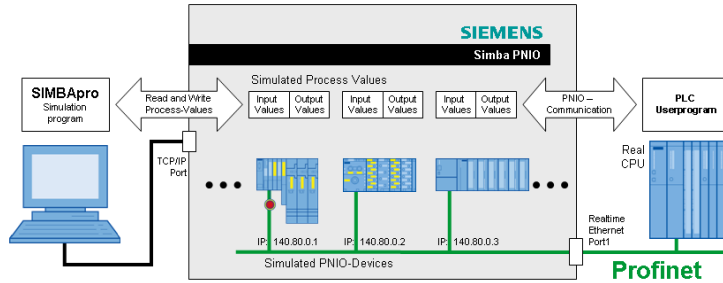
4.12. Limitations in simulating Profibus components

The SIMBAProfibus boxes / SIMULATION UNIT PB boxes support connections for MSAC2 services only in a very basic way. That can cause error messages from driver blocks that try to read diagnostic informations of underlaid devices or modules via MSAC2 services. The simulation of the IO data of these devices or modules is full warranted.

During the download of a hardware configuration into a SIMBAProfibus box, networks in this box with a speed above 1,5 MBd will be disconnected for a short time. This is necessary for configuration performance and will cause a short bus failure in that networks.

5. Simulation of Profinet

5.1. Introduction to the Simulation of PROFINET



Profinet components are simulated with SIMBA / SIMULATION UNIT PN modules connected to a real PROFINET IO controller. Data between a SIMBApro / SIMULATION UNIT PN module and a PC is transmitted via a TCP/IP connection. The hardware can be configured to simulate a limited number of PROFINET IO devices (128 or 256 devices).

The simulation module SIMBApro / SIMULATION UNIT PN is not available as a simulation block in the hardware configuration of the SIMATIC S7 project and therefore not inserted in HWConfig.

SIMBApro / SIMULATION UNIT PN is not inserted as a PROFINET participant. Instead, after the hardware engineering in S7/PCS7/TIA, the actual Profinet structure of S7/PCS7/TIA project is imported into the SIMULATIONUnit software.

5.1.1. Features and restrictions

The following features are supported by SIMBApro / SIMULATION UNIT PN:

Feature	SIMBA PNIO standard	SIMBA PNIO premium	SIMULATION UNIT PN 128	SIMULATION UNIT PN 256
Order Nr.	9AE4120-1AA00	9AE4120-1AB00	9AE4120-2AA00	9AE4120-2AB00
Max number devices	128	256	128	256
Profinet interfaces	1	1	1	2 z)
I/O Data manipulation	✓	✓	✓	✓
Minimum cycletime	250 µs	250 µs	250 µs	250 µs
Records	✓	✓	✓	✓
Device and module failure	✓	✓	✓	✓
Bus failure	✓	✓	✓	✓
Channel diagnostic	✓	✓	✓	✓
IRT class 2	✓	✓	✓	✓
IRT class 3	✗	✗	✓	✓
Profisafe V1.0 (PN/PN Coupler)	✓	✓	✓	✓
Profisafe V2.4	✓	✓	✓	✓
Profisafe V2.6	✗	✗	✓	✓
Fast Startup	✓	✓	✓	✓
Advanced startup (CPU 1200/1500)	✗	✗	✓	✓
Multidevices (IE/PB-Link)	restricted 2)	restricted 2)	restricted 2)	restricted 2)
I&M Data	✓	✓	✓	✓

Feature	SIMBA PNIO standard	SIMBA PNIO premium	SIMULATION UNIT PN 128	SIMULATION UNIT PN 256
Shared device	✓ 1)	✓ 1)	✓ 1)	✓ *)
Redundancy MRP	✗	✗	✓ 3)	✓
System redundancy S2	✗	✗	✗	✓
System redundancy R1	✗	✗	✗	✓ *)
System redundancy R2	✗	✗	✗	✗
Save project in flash	✓	✓	✓	✓
Save project to USB/SD	✗	✗	✓	✓
integrated switch	✗	✗	✗	✓
extended diagnostics via SNMP	✗	✗	✗	✗
Dynamic Frame Packaging	✗	✗	✗	✗
Failsafe values at CPU Stop	✗	✗	*)	*)
High-precision time stamping	✗	✗	✗	✗

*) Feature will be available in the future. The SIMULATION UNIT PN can be upgraded by firmware update.

1) Projects with shared devices can be loaded into the SimbaPNIO / SIMULATION UNIT PN, but only one PNIO controller can access the simulated devices.

2) Some devices support the ISO protocol for exchange of configuration and diagnosis data (e.g. IE/PB Link). The ISO protocol is not supported by SimbaPNIO / SIMULATION UNIT PN, but the PNIO part of the device will work. Simulation of IE/PB Link with S7-300 CPU doesn't work.

3) MRP configurations are supported, but the ring cannot be closed with SIMULATION UNIT PN 128.

z) You can not simulate two separate PNIO networks. You can simulate one network with 2 interfaces to 2 PNIO controllers.

5.2. SIMULATION UNIT PN



5.2.1. Connectors

Power supply L+ : DC +24V, min 0,5A M : Ground

Ethernet ports network P1 and P2

For communication between the communication processor (CP module) of the PLC and the SIMULATION UNIT PN box (directly or via a switch).

The SIMULATION UNIT PN 128 can be connected with the upper port P1 only. The SIMULATION UNIT PN 256 can use both ports P1 and P2. P2 only works if a cable is plugged to P1.

Control port

Ctrl: To load the project into the SIMULATION UNIT PN and to exchange simulation data with SIMULATIONUnit.

The TCP/IP port can be connected directly to your simulation computer or indirectly via one or more switches. The baud rate (100MBit/s or 1GBit/s) and cable polarity will be set automatically, normal or cross over cables can be used.

USB Host

The hardware contains one USB host port. Mass storage devices can be connected here.

The standards USB 1.1 und USB 2.0 are supported.

The USB interface provides a power supply for connected devices with max. 500mA.

Micro SD card

At the back side of the housing there is a metal cover which can be removed. There is a micro SD card slot behind the cover.

The standards SD and SDHC are supported with a maximum capacity of 32 GB.

Digital I/O

The hardware has one connector which can be used as a digital input or a digital output.

DI/DO: digital input or output SGND: signal ground.

Input:

```
log. 0: no input signal or 0V
log. 1: 12 - 48V, 5 - 100mA
```

Output:

```
The output can drive 100mA maximum.

log. 0: internal 4k7 resistor to ground
log. 1: output pulled to ground by photo transistor
```

Button

There is a reset button below the USB port.

Function:

After a reset, the box is reset and switches to startup mode. While the startup LED is flashing, the box cannot be accessed. A reset is necessary in case of new parametrization/firmware update of the module or in case of unexpected behaviour of the module. After a reset all debug and log informations are lost. If possible and necessary, the user should back up those informations before resetting the module.

5.2.2. Installation and heat

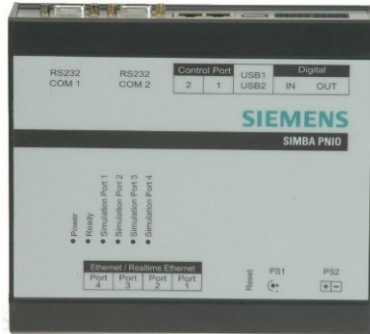
The SIMULATION UNIT PN can be placed on a table or snapped to a cap rail with the clamp on the back side. The case is also used for chipset cooling. Please make sure that the heat can be emitted to the surrounding air. The environment temperature should not exceed 70°C.

5.2.3. Description of the LEDs

Power (PWR)	Lights up green when the power supply is connected.
Ready (RDY)	Flashes green during startup, when the power is restored or after a reset, and lights up green when the module is ready.
	orange blinking, if a simulation project is loaded in the module.
	Lights orange, if the cyclic communication between one simulated device and the profinet I/O Controller is activated.

Control (CTRL)	Lights up green if a link to the network has been established. Flashes orange at data exchange.
LEDs for Ethernet ports P1, P2	Lights up green if a 100MBit/s link at the network has been detected.
	Flashes green if a 1GBit/s link has been detected. Flashes orange when data is received.

5.3. SIMBA PNIO



5.3.1. Connectors

For simulation, the following connectors have to be connected:

Control Port 1 to load the project into the SIMBApnio and to exchange simulation data with SIMULATIONUnit.

Realtime Ethernet 1 for the communication between the communication processor (CP-module) of the PNIO module (direct or indirect via a switch).

Power supply

The power can be supplied at PS1 or PS2.

PS1	Connector for a standard power supply (9 - 36V, min. 1,5A).
PS2	two wire connector, for instance for a 24 V power supply "PS307" from Siemens.

Realtime ethernet line

The box comes with 4 Real Time Ethernet ports.

RT Ethernet 1	RT Ethernet 1: This port has to be connected. It simulates a complete Profinet I/O bus or part of it. It can be directly connected to a Profinet I/O controller (for example CP443-1) or indirectly over one or many switches (for example Scalance X208). In case a switch is used, you have to pay attention on the fact that the used port can work with 100 Mbit/s. The polarity of the ethernet cable will be automatically recognised, i.e. normal or cross cable can be used.
RT Ethernet 2	deactivated.
RT Ethernet 3	deactivated.
RT Ethernet 4	deactivated.

Control Port (TCP/IP)

The box comes with 2 Control Ports (TCP/IP).

Control Port 1	This port has to be connected. It is used to control the SIMBA-PNIO. A PC with SIMULATIONUnit software is used for this.
----------------	--

Control Port 2

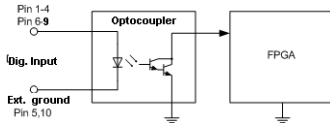
unused.

The port could be directly connected to the PC or indirectly over one or many switches. The polarity and the speed of the ethernet cable will be automatically recognised, i.e. normal or cross cable can be used.

Digital I/O

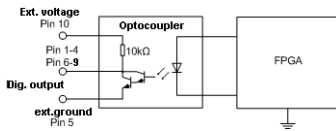
The box has 8 digital inputs and 8 digital outputs. These are electrically isolated from the rest of the box using optocouplers.

Inputs



Input voltage < 2,7 V : logical "0"
 Input voltage 3 - 48 V : logical "1"

Outputs



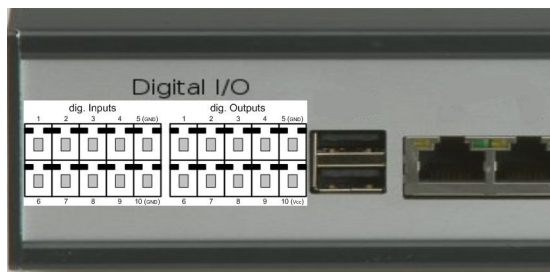
8 open-collector outputs are provided:

Max. operational voltage: 40 V, 1 A

logical "0": Voltage at pin 10 is present at the output pin via a 10k resistor.

logical "1": The output pin is switched through to ground (at pin 5).

Socket assignment on the left of the enclosure



Inactive connectors

This module was not only developed for Simba. It contains other connectors which are not used here:

Serial 1

This port is only used for the development and should not be used otherwise.

Serial 2	deactivated.
USB	The module comes with 2 USB host connectors. They are not active.

5.3.2. Installation and heat

The SimbaPNI/O box can be placed on a table or snapped to a cap rail with the clamp on the back side. The housing is also used for chipset cooling. During normal operation, the module can warm up to more than 40°C. This has no negative effect on functionality. Please make sure that the heat can be emitted to the surrounding air. The environment temperature should not exceed 40°C.

5.3.3. LED description

LEDs on the top

Power	orange light, if the power supply is on.
Ready	green blinking while booting after power on or reset.
	green light, when system is operational.
Simulation Port 1	green blinking, if a simulation project is loaded in the module.
	lights green, if the cyclic communication between one simulated device and the profinet I/O Controller is activated.
Simulation Port 2	always off.
Simulation Port 3	always off.
Simulation Port 4	always off.

LEDs near the Ethernet ports

Green LED	lights, if a physical link is present.
Yellow LED	blinks at data exchange.

5.3.4. Reset

The box comes with one Reset button. After Reset the box switches into startup mode. While StartUP LED is blinking, the module can't be accessed by the web overview and by SIMULATIONUnit. A reset is necessary in case of new parametrization/firmware update of the module or in case of unexpected behaviour of the module. After a reset all debug and log informations are lost. If possible and necessary, the user should back up those informations before resetting the module.

5.4. Configuration

5.4.1. Accessibility

To access the module via web interface, the Control port has to be linked with a PC (see: [hardware connections SIMBApni/o](#)/[hardware connections SIMULATION UNIT PN](#) and [connection examples](#)).

The startup phase has to be finished, i.e. Power and Ready LEDs are lighting. An access by the web interface is also possible during simulation.

First of all, an IP address has to be parametrized for the module. This address will be then used each time to access the module.

Attention!

The subnet of the IP address must be another one as the one of the IP addresses of the simulated devices!

Example:

IP address of the simulation PC	140.80.123.1	Subnet mask: 255.255.0.0
IP address of SIMBApni/SIMULATION UNIT PN	140.80.123.110	Subnet mask: 255.255.0.0
IP address of simulated Devices	140.90.x.y	Subnet mask: 255.255.0.0

5.4.1.1. Setup the IP address with SIMULATIONUnit

Scan the network in portal view with [Scan network](#) or in [Hardware edit](#) in the project view.

If no box can be found please check if the firewall of the simulation PC is deactivated.

Select a box out of the list. The box can be identified by clicking the button "Blink". The LEDs on the box blink until you press button "Stop". Here you can configure IP address, Subnet mask, if necessary Default gateway and the name of the box. With the button "Save changes" these informations will be saved in the box.

5.4.2. Web interface

With the web interface, the operator can: - verify the hardware and firmware versions of the module - watch Log files

Start a webbrowser (for example Internet Explorer) and give the IP-Address of your SIMBApnio/SIMULATION UNIT PN module:



It opens the home window.

5.5. Network connection

The best solution is to separate the Profinet network from the control network.



SIMULATIONUnit and Engineering station can be on the same PC or on different PCs.



You can also connect all networks with a switch.

This solution works, but is not the best one. **The external network may disturb the Profinet I/O communication and slow down the SIMULATION UNIT Box - SIMULATIONUnit-PC communication.**

The IP address of the SIMBApnio/SIMULATION UNIT and the IP addresses of the simulated devices must be in different subnets !

see also [Simulation of the Topology](#)

5.5.1. Network connection with redundant systems

If you have a redundant machine, connect one PLC to SIMULATION UNIT PN 256 Port P1 and the other PLC to port P2. The SIMULATION UNIT PN 128 has only one active simulation port. It cannot be used for redundancy simulation.



Connecting two ports of a single CPU to the ports P1 and P2 of a SIMULATION UNIT PN 256 creates a ring.



You have to set up a MRP configuration in Step 7 / TIA Portal to use this configuration.

ATTENTION: Do not connect both ports P1 and P2 of a SIMULATION UNIT to a single CPU without a MRP ring configuration going through the SIMULATION UNIT. This will generate a loop and lead to very high network load.

see also [Simulation of Redundancy](#)

5.5.2. Network connection with multiple SIMULATION UNITs and real Profinet devices


Real Profinet devices can also be connected to the Profinet network. They must be deleted from the SIMULATIONUnit project.



- Multiple SIMULATION UNITs can be connected to one SIMULATION PC by using a network switch.

- The SIMULATION UNIT PN 256 has two ports (P1 and P2) which can be used to cascade SIMULATION UNITS or to connect real devices before or behind the SU.
- The SIMULATION UNIT PN 128 has only one active simulation port (P1) and must be connected to the end of a line topology.

5.6. Creating a new project


Menu "File" → select "create project" or Button  in the portal view.

First create an empty project. Choose a folder on your Windows system and a name for the new project. SIMULATIONUnit will create a new folder with .spf ending for the simulation project.

5.6.1. Hardware configuration

The hardware configuration can be done manually with the integrated configuration tool "Configure Hardware" or automatically with the import function ([Import hardware](#)).

5.6.1.1. Manual configuration

Menu "Configuration" →  "Edit hardware"

When configuring SIMULATIONUnit via [hardware edit](#) the hardware components have to be inserted from a library by "drag and drop". SIMULATIONUnit checks here only the logical consistency of the configuration (for example: device number and IP address).

Attention:

SIMULATIONUnit does not check the consistency of the logical addresses. That means that in case of using logic addresses in a double way inconsistencies or unrecognized signal changes may occur during simulation!

To configure packed address (for ET200S):

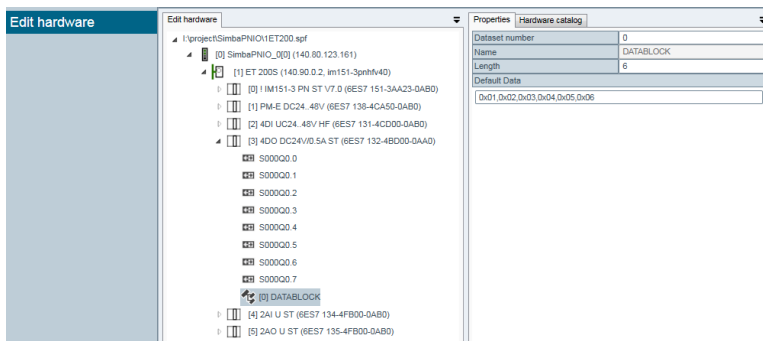
configure first module as normal

select further modules with * attribute in HW lib.

PNIO Records

In "Hardware edit" it is also possible to define records for PNIO modules and fill them with default data.

To add a new record, click with the right mouse button on a PNIO module or a subplot and select **Add dataset** from the context menu:

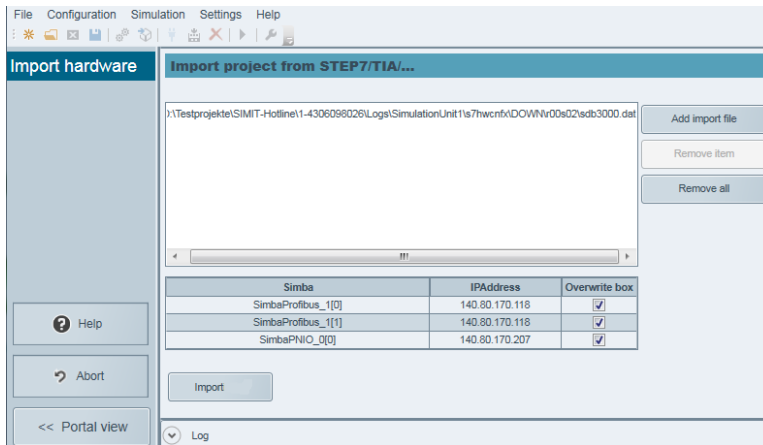


When one record is added you have to select and configure it in the properties window.

5.6.1.2. Import hardware

You can import hardware configurations from Step 7, TIA Portal or other SIMULATIONUnit projects.

Menu "Configuration" → "Import hardware".



A hardware configuration can be imported from a Step 7 project, an OMS file from TIA-Portal, or a XML file exported from other SIMULATIONUnit projects.

Getting the hardware data from S7 or TIA-Portal is the same as in Profibus projects. Please look at

[STEP7 Hardware config export](#)

[TIA-Portal hardware config export](#)

For getting hardware data from other SIMULATIONUnit Projects you need one or more *Project.xml* files out of the .spf project folders.

For a correct import of **packed addresses** (f. i. ET200S) it is necessary to import the .cfg file together with the .dat file!

For a correct import of **shared devices** into two SIMULATION UNIT boxes it is also necessary to import the .cfg file of both lines together with the corresponding .dat files!

It is also possible to import symbol files when the hardware was configured by manual configuration.

Avoiding MAC address conflicts

If you use more than one SIMULATION UNITS in the network and load them from different computers, it is important to avoid conflicts by using unique MAC addresses. You can set up the band of MAC addresses used by your SIMULATION UNIT before you import the hardware with the dialog "[Settings → MAC Addresses for simulated PNIO devices](#)".

5.6.2. Import new devices to the SIMULATIONUnit hardware library

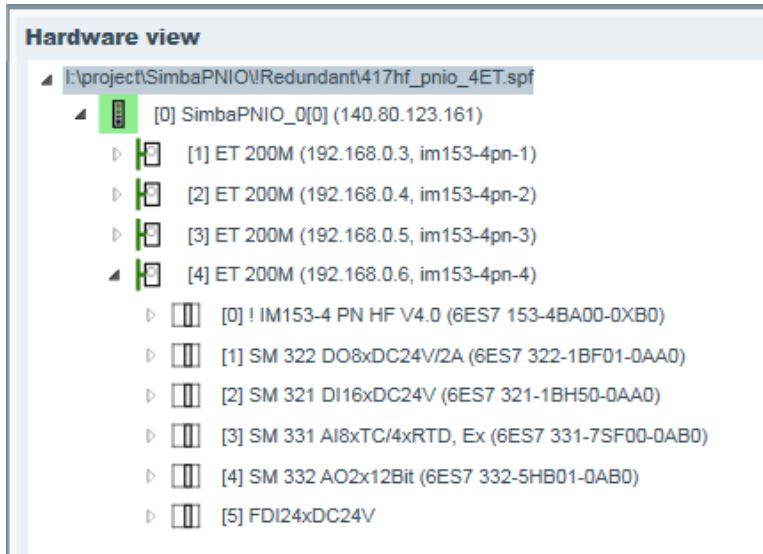
You can import a device description file into SIMULATIONUnit. Under "Portal view" → "System" you can insert devices from GSDML***.xml files into your installed device database. The new devices are available if you create a new project or (re)import SDBs.

5.7. Hardware download and diagnostics

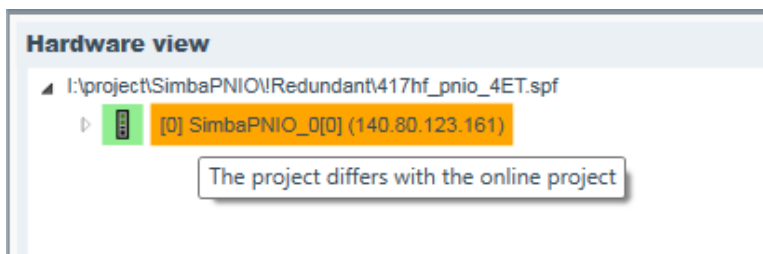
After a successful import of a hardware configuration every PNIO line have to be connected to a physical SIMULATION UNIT box. Please select the SIMULATION UNIT box in hardware edit. On the right side of the window you can edit the properties of the box as f. i. the IP address. You can also find out the IP address by scanning the network (see also [addressing the box](#)). After that you can connect the box and download the configuration.

5.7.1. Icons

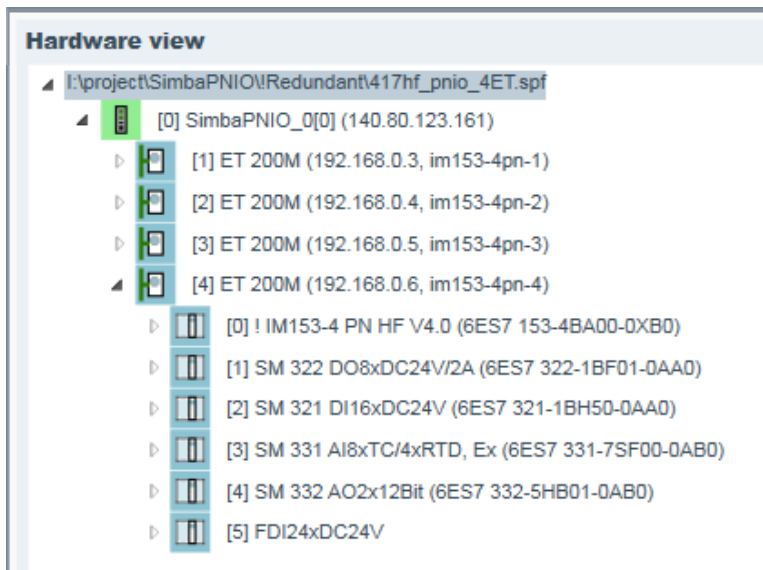
- After connect you will see the SIMULATION UNIT box switching to green in hardware tree.



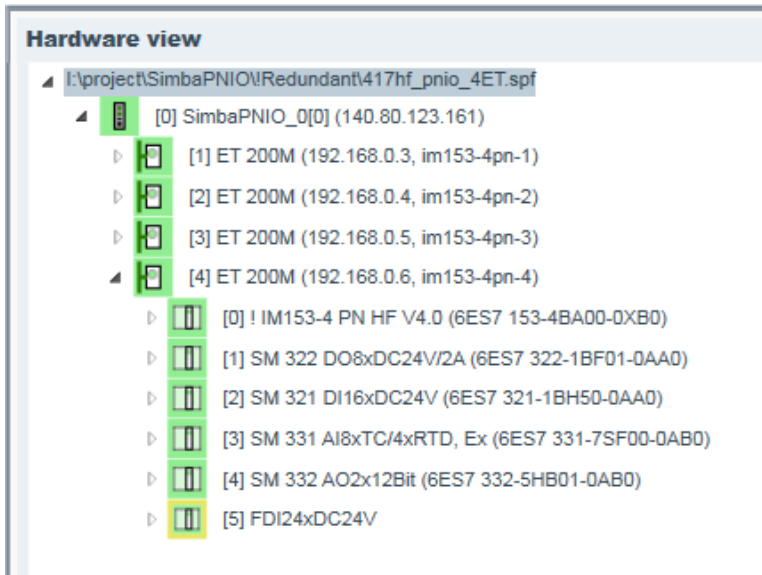
- If there is another project loaded to the SIMULATION UNIT, the SIMULATION UNIT will be marked orange.



- After a successful download of the configuration the devices are shown as "loaded" (light blue).



- If the communication between master and devices is established, devices switch to green color.



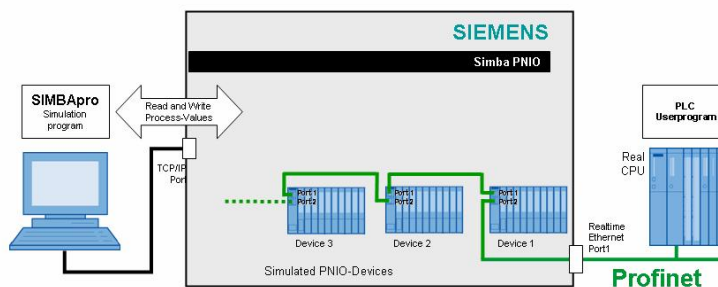
- In the last case you will see additionally the failsafe state of modules (yellow)

Attention:

No check of the device or module type is done here! Devices, which are configured in the simulation module but not configured in the automation system are not recognised. Delete real devices from the project.

5.8. Simulation of the Topology

When simulating IRT the topology of the network is important.



SIMULATIONUnit-PNIO simulates the behaviour of a Profinet network at a single point in the network:

This is one port of the first simulated device.

When creating a simulation project, the number of the first device and its port number that is connected to the outer world must be provided. In the example above device 1 / port 2 has to be configured.

The exact timing of all Profinet nodes is supplied by the PNIO controller during startup. SIMULATION UNIT PN set up the timing parameters for all simulated devices according to the parametrisation that is sent to the first device.

That means:

- The topology of all devices behind the first device does not matter when you are creating a simulation project.
- SIMULATION UNIT PN can switch on the devices behind the first device only when the first device has got its parametrisation.
- If you simulate a device failure of the first device, the whole bus will fail.

5.9. Simulation of IRT Devices

The simulation of Profinet IRT (Isochronous Real Time) devices is possible.

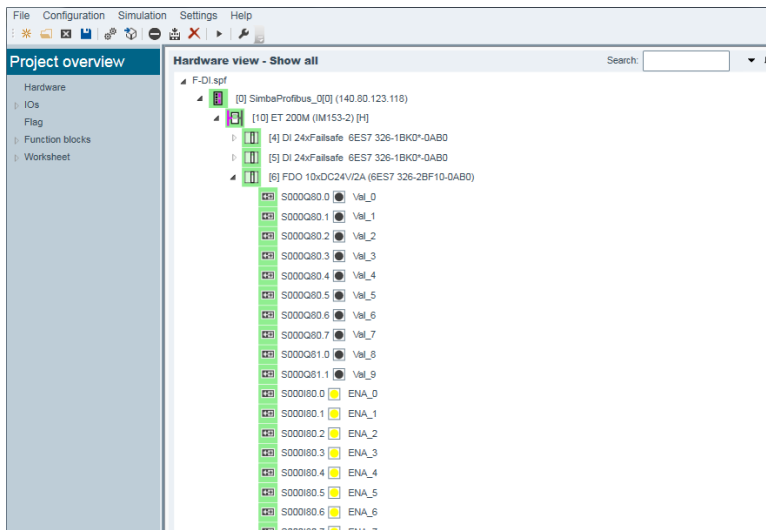
You have to set up a correct [topology](#) to get IRT running.

5.10. Simulation of Failsafe Modules (Profisafe V2)

SIMULATION UNIT PN supports simulation of the Profisafe V2 protocol.

When a hardware configuration with failsafe modules has been imported, a function block **"PROFIsafe"** appears in the project overview tree.

This function block contains an instance of every F-module in the project. When Downloading the project the functions are loaded to the SIMULATION UNIT PN and activated.



Digital Failsafe modules contain additional to the data bits an "Enable" bit for each channel. To prevent the Simulation from channel errors these bits have to set to "1" by default.

In hardware view you can see also as the last channels of every module some state bits. They deliver the information to the state of the Failsafe communication.

Meaning of the most important status bits:

activate_FV	Activate Failsafe Values Request from PNIO controller. IOs are invalid.
Device_Fault	Failure in the simulated device.
CE_CRC	Error in F-checksum
WD_Timeout	Watchdog Timeout: The PNIO Controller has not activated Profisafe.
FV_activated	The F-Module deactivated the IOs. (WD_Timeout may be the cause).

When the failsafe mode is active, all bits are zero.

If an F module does not work, you may try to increase the parameter F_WD_Time in the STEP7 hardware configuration for the F module.

5.11. Simulation of Redundancy

There are different types of redundancy at Profinet:

5.11.1. MRP ring redundancy

A ring topology is set up. There is a MRP master who opens the ring if it is closed.

SIMULATION UNIT PN supports the simulation of MRP clients.

The ring cannot be closed with the one channel SIMULATION UNIT PN 128.

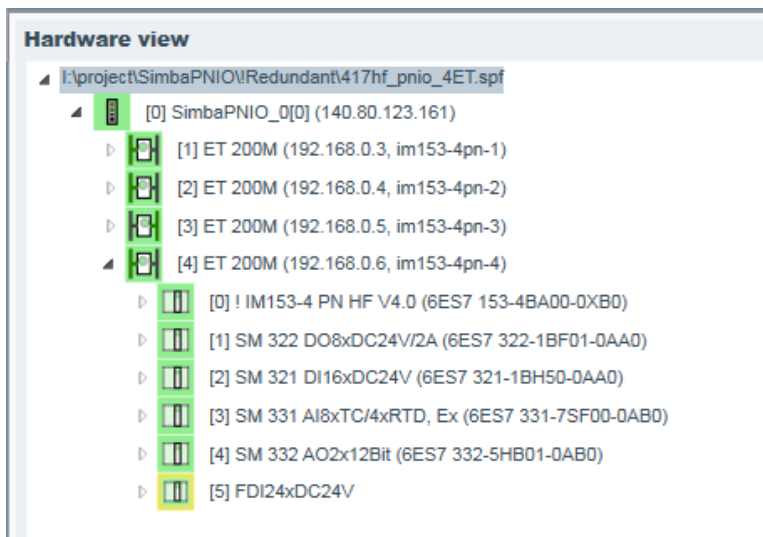
With a SIMULATION UNIT PN 256, a complete ring can be simulated by using both ports P1 and P2. The ring can not be opened by software.

5.11.2. System redundancy S2

Every redundant PNIO device can establish two connections, one to each PNIO controller of a H machine.

The SIMULATION UNIT PN 256 supports the simulation of system redundancy S2. One PNIO controller has to be connected to simulation port P1 and the other one to port P2. A mix with non-redundant devices is possible.

In the hardware view you will see a redundant device symbol with an active and a passive side. At the left side there is the connection to port P1, right side is P2. Green is active, black is passive.



You can simulate a redundancy switch by simulating a device failure at port P1 or P2. Click with the right mouse button onto a device in the hardware view and select device failure at port P1/P2.

5.11.3. System redundancy R1 and R2

The PNIO devices have two heads which can establish connections to two PNIO controllers independently.

This kind of redundancy is currently not supported.

5.12. Fast startup

Some PNIO devices use the feature "Fast Startup", activated from the PNIO controller, for running a parametrized device faster after Power On. One simulated device has to be parametrized from the PNIO controller on the normal way. After that you can simulate fast startup via "Device failure" and "Device return" or "Line failure" and "Line return".

6. Manipulation of the simulated devices

6.1. I/O-Values

I/O-values in SIMULATIONUnit can be controlled and monitored at the [hardware view](#), the [I/O view](#), at the [function block instance view](#) in the runtime system or at self configured [worksheets](#).

The same IO value is consistent in all views at the same time. In the GUI, you will not see every change of the I/Os, depending on the selected cycle time.

6.1.1. I/O-names

The names of the I/O-values in SIMULATIONUnit contain the simulation hardware index and the address name as in a STEP 7-Project. They have following format:

```
S<hardware index><I/O type><log. address>
```

For example:

S001QW512 - SIMULATION UNIT Hardware Nr. 001, Output from AS view, word, logical address 512

S000I17.2 - SIMULATION UNIT Hardware Nr. 000, Input from AS view, logical address 17, Bit Nr. 2

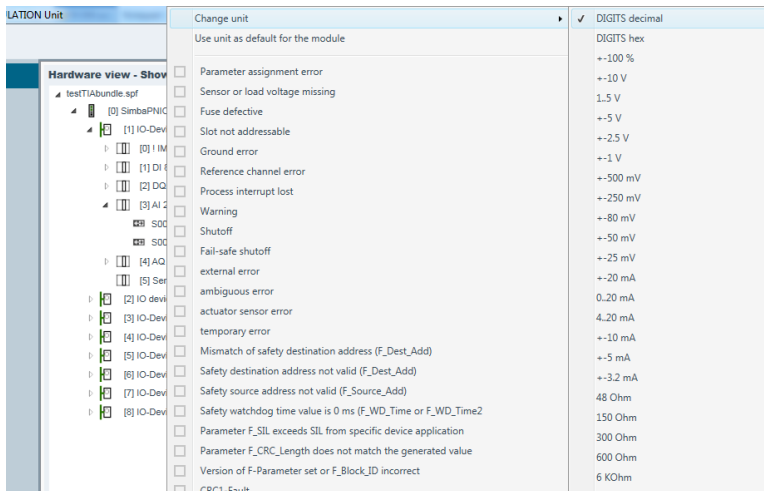
6.1.1.1. Symbolic addresses

Symbolic addresses can be also imported, edited and shown in SIMULATIONUnit (see [IO view](#), second column).

6.1.2. Analog values

Analog values can be configured depending on the simulated hardware or as a technological value in SIMULATIONUnit. Physical units are predefined in the system. The unit of the value is the same in the [hardware view](#), the [I/O view](#), the [function block instance view](#) of the runtime system or in the [worksheets](#).

To change the unit please select the channel and open a context menu by right mouse click.



The menu shown shows the unit definitions for the most used sensors and actors. Additionally, you can show the "raw value" as a decimal or hexadecimal value.

If you have selected an analog value in the hardware tree, you will see also a menu item "Use unit as default for the module" to set all analog units of the selected module to the same unit.

6.2. Alarm generation and triggering

6.2.1. For Profibus

In SIMULATIONUnit for every slave the different alarms can be edited in a type-related XML file and then triggered by context menu or by runtime system.

6.2.1.1. Device fault

Open the [hardware view](#). With a right click on an element of the hardware tree you will get a context menu of possible actions. F. I. you can simulate a line fault on a SIMULATION UNIT box or a device fault on a device.

Additionally it is possible to simulate device faults by [runtime system](#). For that you can use the basic function "SlaveAusfall".

The two inputs of this block are to configure with the slave/device number (Slv) and a control bit (Control). F. I. you can configure the slave number 5 as following:

- Slv = 5 (Profibus address) + (number of the line in SIMULATIONUnit project x 65536)
- Control = 1 : slave will be simulated
- Control = 0 : slave fault

6.2.1.2. Profibus alarm description in the XML file

The alarm file for a slave type is located in the folder "<SIMULATIONUnit program folder>\ALM". The filename will be composed from the Profibus Id of the slave:

```
f. I.: ET200M type (hexadecimal): 0x801D -> filename: "Diag801D.xml".
In a SIMULATIONUnit project folder you can find the Profibus Id in the file
project.xml under the XML key "TYPE" of every slave tag.
```

The alarm file has the following syntax:

```
<Diagnose>
  <ALARM>
    <NAME>Modul Ausfall</NAME>
    <INDEX>0</INDEX>
    <LEVEL>1</LEVEL>
    <DATA>$DExt,43,$SlotBit,09,03,$Slot,10,$DS0 </DATA>
    <ACTIVE>2</ACTIVE>
  </ALARM>
  <ALARM>
    ...
  </ALARM>
</Diagnose>
```

For every alarm a data block has to be created .

Meaning of the keys for a Profibus simulation:

NAME	Alarm text in context menu.
------	-----------------------------

INDEX	Identifier of the alarm within the file. All identifiers in a file have to be unique.
LEVEL	Level in the hardware tree. If level is "0", the alarm is shown in the context menu of the slaves that shall trigger the alarm. Level "1" identifies a module dependent alarm.
DATA	Diagnostic data beginning from byte 7 in slave diagnostics. The standard diagnostic bytes of a slave cannot be changed. Data format is hexadecimal format divided by comma. Inside of a data string you can use some extra keys for dynamic data. These always start with a "\$" sign. This time there are supported the following extra keys: - \$DExt: initiates the setting of the "Ext_Diag-Bit" in standard diagnostics of the slave. This bit advertises an extended diagnostic information for the master. If used this key has to be the first entry in the data string. - \$DS0: initiates the module identification (like data block 0) for the module. It is active only on levels > 0. - \$Slot: slot index or module index - \$SlotBit: reserves 2 bytes in the slave diagnostic information. For the actual slot a bit will be set in this bytes.
FUNCTION	Please don't change! Instead of the "DATA" key you can use the "FUNCTION" key to generate the data by an integrated C function dynamically. The C function name has to be implemented in SIMULATIONUnit.
ACTIVE	rules the shown state of the object after triggering the alarm. Valid values: 0 - inactive 1 - active 2 - fault

In the example above the Ext_Diag_Bit will be set. The identifier dependent diagnostics starts with a 0x43 and then a bit follows to identify the slot (f.i for slot 4 the slot field is "08, 00"). "09" is the length of the alarm part, "03" the alarm type (this: pull/plug), followed by the number of the slot (f.i. "04"), "10" is the number of the alarm sequence and \$DS0 triggers 5 byte of record 0 for the actual slot (depends on the module).

6.2.1.3. Triggering the alarm by menu

In the same way as the device fault the alarm can be triggered by context menu of a module in the hardware tree of SIMULATIONUnit.

6.2.1.4. Triggering the alarm by runtime system

For an event triggered alarm you have to create a runtime function block and include the integrated basic function "**SlaveAlarm**".

The basic function "SlaveAlarm" triggers alarms. For that you need 4 parameters: an I/O address of the object, which should trigger the alarm, a control bit, and two constants representing the alarms triggered at the changes of the control bit. These constants represent the value of the "INDEX" key for the alarm in the XML file.

6.2.2. For ProfinetIO

In SIMULATIONUnit for every device the different alarms can be edited in a type related XML file and then triggered by context menu or by runtime system.

6.2.2.1. Device fault

Open the [hardware view](#). With a right click on an element of the hardware tree you get a context menu.

In that you will find different entries for simulation of device faults depending on the device type (f.i. for Profinet: device fault and device return).

Additional you can simulate a fault of the whole line. For that you make a right click on the SIMULATIONUnit module and click on "Line fault".

It is also possible to simulate device faults by runtime system. For that you have to use the basic function "SlaveAusfall".

Entries of the function are device number (Slv) and control bit (Control):

- Slv = Profinet address of the device + (number of the line the SIMULATIONUnit project x 65536)
- Control = 1 : device simulating
- Control = 0 : device fault

6.2.2.2. Description of a ProfinetIO alarm in the XML file

The alarm file for a Profinet device is located in folder "<SIMULATIONUnit-Program folder>\ALM\PNIODEV". The filename is built from the VendorID and the DeviceID of the device:

f. i.: ET200S type (hexadecimal): VendorID=0x002a, DeviceID=0x0301 -> file name:
"Diag002a0301.xml"

In a SIMULATIONUnit project the VendorID or DeviceID can be found in the
"project.xml" under the XML key "TYPE".

The alarm file looks like this:

```

<Diagnose>

    <MENU>

        <INDEX>0</INDEX>

        <NAME>short circuit</NAME>

        <SUBTYPE>CHANNEL</SUBTYPE>

        <FUNCTION>PNIOALARM </FUNCTION>

        <ACTIVE>2</ACTIVE>

        <ERRTYPE>1</ERRTYPE>

    </MENU>

    <MENU>

        ...

    </MENU>

</Diagnose>

```

For every alarm there has to be a data record.

Meaning of the keys:

NAME	Description of the alarm in the context menu.
INDEX	Identifies the alarm in the file. All identifiers have to be unique in the .xml file.
FUNCTION	Please don't change! Instead of the "DATA" key you can use the "FUNCTION" key to generate the data by an integrated C function dynamically. The C function name has to be implemented in SIMULATIONUnit.
ACTIVE	set the display state for the object after triggering the Alarm. Valid values are: 0 - inactive 1 - active 2 - fault
ERRTYPE	predefined ProfinetIO Errorcode

6.2.2.3. Triggering alarms by menu

The alarm can be triggered by context menu of device, slot or channel in hardware tree of SIMULATIONUnit.

6.2.2.4. Triggering alarms by runtime system

For triggering alarms by event you have to create a new function block in function edit. They use the integrated basic functions for Profinet: **"PNIO_Module_Alarm"** and **"PNIO_Channel_Diag"**.

The basic function "PNIO_Module_Alarm" triggers a module dependent alarm.

It has to be configured by 4 parameters:

- line number
- device number
- slot number
- control bit triggering the alarm (Control= 1 → all OK, Control = 0 → Module fault)

The basic function "PNIO_Channel_Diag" triggers a channel dependent alarm.

It has to be configured by 3 parameters:

- an I/O address of the object which should trigger the alarm
- the alarm index (see xml file)
- the control bit triggering the alarm.

6.3. Profisafe V1 (F_SND_DP, F_RECV_DP)

This chapter contains the description of the simulation of a fail safe master-master communication (Profisafe V1) with SIMULATIONUnit. For that in SIMULATIONUnit will be simulated the opposite part of a real in plc integrated F_SENDDP/F_RECVDP- function block. The SIMULATION UNIT Profibus or Profinet simulates the configured DP/DP-coupler or PN/PN-coupler with the universal modules, the F-communication will be simulated in the user data of the modules.

6.3.1. Import of a configuration with Profisafe

The hardware import in SIMULATIONUnit is the same as in a standard project. In HW-Config of the S7-Project is to configure a correct length of the universal modules for the F communication interface in the coupler.

```
+-----+   ---12 Bytes---> +-----+
| F_SENDDP |                   | F_RECVDP |
+-----+   <---6 Bytes--- +-----+
```

When a wrong length is configured (f.i. bigger amount of bytes), SIMULATIONUnit configures the module as a standard I/O module.

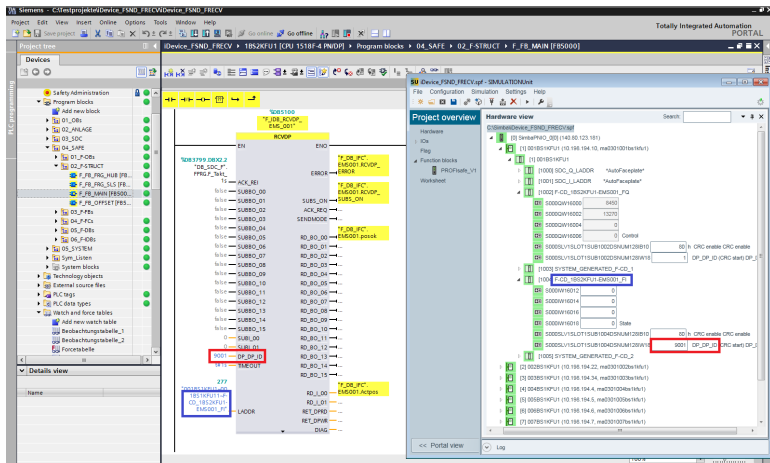
In SIMULATIONUnit a function PROFIsafe_V1 will be created automatically. Every F_SENDDP/F_RECVDP function block in the real CPU has in SIMULATIONUnit an opposite part.

6.3.2. Necessary parameters in the SIMULATIONUnit function instances

For a correct connection you have to configure two parameters in the function block:

DP_DP_ID System unique connection ID. This parameter has to be identical to the same parameter of the partner block in the plc.

CRC_enable To enable the CRC creation in the SIMULATION Unit this parameter has to be configured with the value 80h.



6.3.3. Simulation F_RECVP

For simulation of a F_RECVP you have to configure the values DP_DP_ID and CRC_enable only.

6.3.4. Simulation F_SENDDP

When simulating the send block F_SENDDP you have to configure the values DP_DP_ID and CRC_enable. If the values are correct before plc starts the connection will be established after starting the plc. If the plc is running before, the F_RECVP block has to be reintegrated by his input ACK_REI.

Attention: In case of a broken connection (f.i. cable defect), the plc has to be restarted! In that case the internal telegram counter will not be reset.

6.3.5. Troubleshooting

The output byte "DIAG" of the F_SEND/F_RECV block in Step 7/TIA shows the error state:

DIAG (hex)	Meaning
0	ok
10	Timeout - may be parameter TIMEOUT in HW-Config is too less.
40	CRC error - parameter error in SIMULATIONUnit
50	CRC error + Timeout

7. Runtime system

The integrated Run Time system manages all simulation tasks for all defined functions. The simulation typicals for these functions can be created by using the basic function library. These typical can be easy duplicated by copy/ paste function. The typicals can be triggered either by I/O or timer status. Each typical can be assigned to only one run time group.

If you are using the SIMBA PNIO or the SIMULATION UNIT PN hardware, you can download functions to the hardware and execute them in a Realtime Run Time System. The cycle time is the Profinet send clock.

How to connect I/Os:

- [Create or load typical from the typical library](#)
- [Implementation of a new instance](#)
- [Start Run Time Sequence](#)

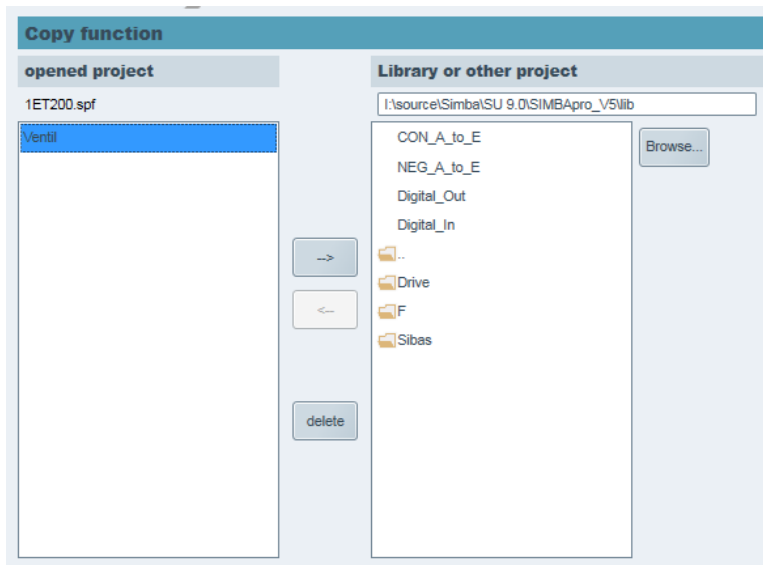
7.1. Insert block type into the project

There are two ways to get a typical.

- By creating new functions using the Global modules, Basic functions and Module functions
- Copying existing functions from a library or another simulation project.

7.1.1. Copy from library or other simulation project

In order to use your newly created typical as a library function, you can put it in the library for using in your other projects.

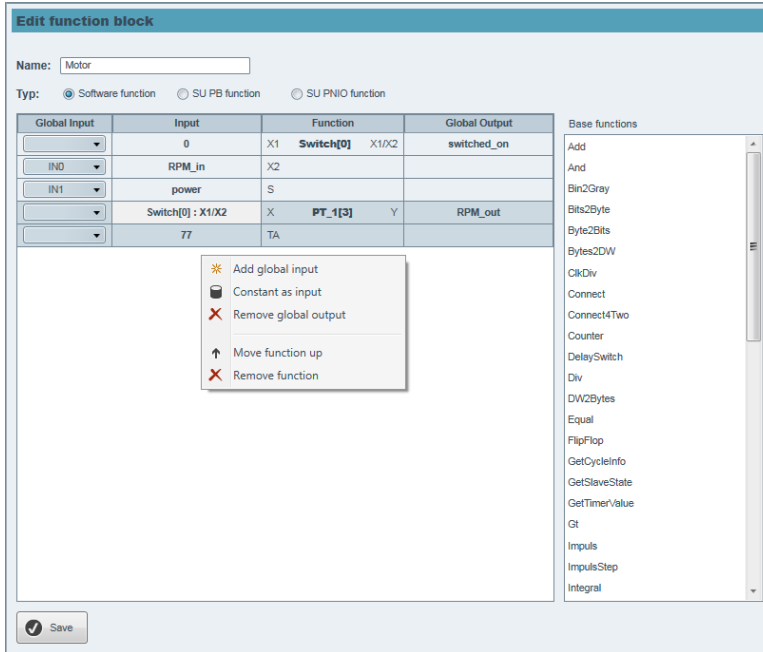


Menu Simulation → Copy function
or Right click on "Function blocks" → "Copy function".

7.2. Creating a function typical

You can create a function from a set of basic functions.

To create a new function block, double click onto "Function blocks" in the navigation column or select menu Simulation → New function block
To change a present function, click with the right mouse button onto a function in the list of function blocks and select "Edit function"



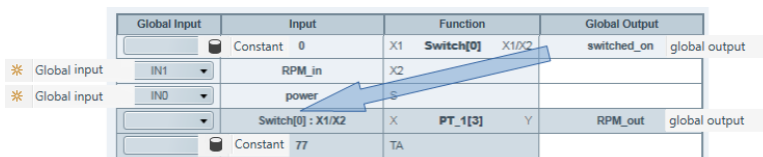
Name: Name of your new function. If another function with the same name already exists, it will be overwritten at Save.

Type: The place of execution of the function instances.

	Software function	Simulation PC
	SU PB function	Simulation UNIT PB
	SU PNIO function	Simulation UNIT PN

Some of the basic functions can be executed at different places. See [Implemented basic functions](#) for more information.

Usage of basic functions



Select one or more basic functions from the list at the right side and move them into the main window. The selected functions will be executed from top to bottom.

In this example there are two basic functions **Switch** and **PT_1**

Switch has 3 inputs and 1 output,

PT_1 has 2 inputs and 1 output

Global inputs

Global inputs are inputs to the function block. You can connect I/Os, flags and constants to the global inputs.

Create a global input by clicking with the right mouse button onto a line in the table an select *** "Add global input"**

In the column **Global input** you will see **IN0, IN1, IN...**

You can select the same global input for more than one input of a basic function from the drop box.

Click into the column "Input" to change the name of the global input.

Global outputs

Global outputs are outputs to the function block. You can connect I/Os or flags to the global outputs.

Create a global output by clicking with the right mouse button onto a line in the table an select *** "Add global output"**

Click into the column "Global Output" to change the name of the global output.

Constants

You can connect constants to all inputs of the basic functions.

Click with the right mouse button onto an input of a base function and select  "Constant as input". In the column **Input** the value 0 shows up. Click onto the value and change it to another numerical value.

Connections between base functions

In this example, the output of the basic function **Switch** is connected to the first input of **PT_1**.

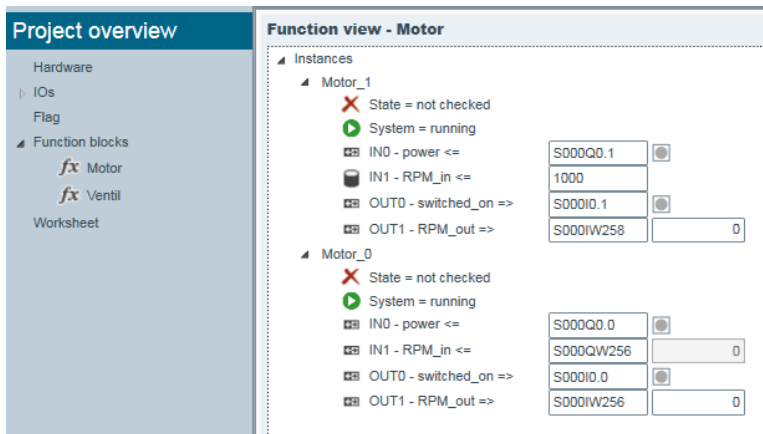
Click onto the output and pull it with the mouse onto the input.

Note: The basic functions will be executed from top to bottom.

Click Save to finish the function block. It will be displayed in the navigation column.

7.2.1. Implementation of function instances


After double click at the respective typical, all instances of the same typical will be displayed. For each instance following informations are displayed:



The screenshot shows the 'Function view - Motor' window with two instances, Motor_1 and Motor_0. Each instance has the following I/Os and values:

Instance	I/O	Value	Status
Motor_1	IN0 - power <=	S000Q0.1	On
	IN1 - RPM_in <=	1000	Off
	OUT0 - switched_on =>	S000I0.1	On
	OUT1 - RPM_out =>	S000IW258	0
Motor_0	IN0 - power <=	S000Q0.0	On
	IN1 - RPM_in <=	S000QW256	0
	OUT0 - switched_on =>	S000I0.0	On
	OUT1 - RPM_out =>	S000IW256	0

creating a new instance:


- After selection of a typical (right mouse button) select the function  "new instance".

- Define a new unique name.
- The new instance will be displayed in the tree.
- All I/Os of the typical must be defined with a flag(e.g. S000MW2000) or a logical address(e.g. S000I7.0). A constant can be assigned to an input.



You can enter an I/O address or constant manually or click onto an I/O from the hardware view or I/O view and pull it with the mouse onto a function instance I/O.

The instance will be saved if all I/Os have a correct value assigned.

(de)activate an instance

 /  activate / deactivate instance. Deactivated instances will be ignored by the run time system.

check an instance

 /  The user can set a checked mark to an instance. This does not affect the simulation.

rename an instance Click with the right mouse button onto an instance name and select "rename instance"

Enter an arbitrary instance name.

7.2.2. Import and export of Instances

Function instances can be exported to and imported from a .csv file, which is editable with Excel.

Click with the right mouse button into a function instance window and select from the menu: "Export instances" or "Import instances"

The .csv file has 5 columns:

1. function name

2. instance name
3. I/O address
4. parameter number
5. I/O type:
 - 1 = input (from AS point of view)
 - 2 = output
 - 3 = flag at input
 - 4 = flag at output
 - 5 = constant

The csv file for the example above contains these entries:

function name;	instance name;	log. address;	parameter number;	I/O type;
Motor;	Motor_0;	S000A0.0;	0;	2;
Motor;	Motor_0;	S000AW256;	1;	2;
Motor;	Motor_0;	S000E0.0;	0;	1;
Motor;	Motor_0;	S000EW256;	1;	1;
Motor;	Motor_1;	S000A0.1;	0;	2;
Motor;	Motor_1;	1000;	1;	5;
Motor;	Motor_1;	S000E0.1;	0;	1;
Motor;	Motor_1;	S000EW258;	1;	1;

7.3. Start / Stop of the runtime system

If the runtime system is started (🟢 "System = activated" is on) all instances of the typical will be activated.

If the option 🟡 "System = deactivated" is on, the runtime system won't execute the instance.

The runtime system requires a successful connection with the SIMULATION UNIT module.

- Menu "Simulation" "stop runtime system"
- ▶ Menu "Simulation" "start runtime system"

If the runtime system is stopped all instances of the typical will be deactivated.

7.4. Activating of a typical in a SIMULATION UNIT box

Instances of a typical which include a reference to the SIMULATION UNIT PB firmware (e.g. redundancy or failsafe) or the SIMULATION UNIT PN will be executed directly in the simulation hardware. The instances are active after download. If an instance is deactivated, it will not be downloaded.

An icon next to the function name in the navigation column shows the place of execution.

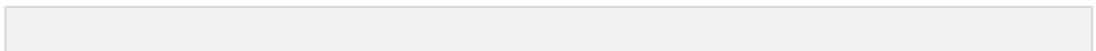
	Software function	Simulation PC
	SU PB function	Simulation UNIT PB
	SU PNIO function	Simulation UNIT PN

The cycle time of the SU PNIO functions is the Profinet cycle.

Most of the runtime system functions that run on a simulation PC can be downloaded to a SIMULATION UNIT PN hardware. All instances of a function will be downloaded to the SIMULATION UNIT PN if "SU PNIO function" is selected in the [function editor](#).

Functions can be converted from PC software function to SIMULATION UNIT PN hardware function or in the other way if all basic functions exist in both function types.

See [List of supported Implemented basic functions](#).



Restrictions:

All inputs, outputs and flags that are connected to function inputs/outputs must belong to the same SIMULATION UNIT PN hardware. Constants in function inputs are allowed. Flags can be used for controlling inputs, but if a flag is connected to a function output it will only be used internally and will not be displayed in SIMULATIONUnit.

7.5. Implemented basic functions**Arithmetic:**

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
Add	Adder	✓		✓	X, Y	X+Y	Add X+Y
Sub	Subtractor	✓		✓	X, Y	X-Y	Subtract X-Y
Mul	Multiplier	✓		✓	X, Y	X*Y	Multiplier X*Y
Div	Divider	✓		✓	X, Y	X/Y	Divider X/Y
Pow	Power of	✓		✓	X, Y	X^Y	X power of Y

Comparer:

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
Equal	=	✓		✓	X, Y	X==Y	if X == Y: output = 1, if X != Y: output = 0
Gt	greater then	✓		✓	X, Y	X>Y	if X > Y: output = 1, if X # Y: output = 0
Limit	limiter	✓		✓	X0, X, XU	Y	if X > X0: Y = X0, if X < XU: Y = XU, else Y = X
LevelTrigger	level trigger (greater then or equal)	✓		✓	IN, Level	Trig	if IN >= Level: Trig = 1, else Trig = 0

Copy/Convert function

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
Bits2Byte	Copy 8 Bits to Byte	✓		✓	X0, X1, X2, X3, X4, X5, X6, X7	Y	Copy 8 bits to one byte (X0 = LSB)
Byte2Bits	Copy Byte to 8 Bits	✓		✓	X	Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7	Copy one byte to 8 Bits (Y0 = LSB)
Bytes2DW	Copy 4 Bytes to Doubleword	✓		✓	X0, X1, X2, X3	Y	copy 4 bytes to a doubleword X0 = LSB)
DW2Bytes	Copy Doubleword to 4 Bytes	✓		✓	X	Y0, Y1, Y2, Y3	copy a doubleword to 4 bytes (Y0 = LSB)
Connect	Copy	✓		✓	X	Y	Copy X to Y

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
Connect4Two	Copy value to 2 outputs	✓			X	Y0, Y1	Copy X to Y0 and Y1
SlaveMirror	Slave Mirror	✓			Slv		Copy all OUTputs of a slave with addr. Slv to its INputs

Digital:

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
And	log. AND	✓		✓	X, Y	X&Y	AND bitwise
Neg	Inverter	✓		✓	X	/X	If X=0: Y=1, else Y=0
Or	log. OR	✓		✓	X, Y	XorY	OR bitwise
XOR	log. exclusive OR	✓		✓	X0, X1	Y	exclusive OR bitwise
Counter	Counter / Ramp	✓		✓	Step, Clk, Clr	Y	Counter preset with value "Period" "Clk" is trigger for count to zero during counter value is zero, "Y" get high pulse, "Period" will be set to preset value
Bin2Gray	Binary to Gray-Code converter	✓		✓	BinCnt	GrayCnt	Binary counter BinCnt will be converted to Gray Code GrayCnt
Shift1	1-bit-shift register	✓		✓	X, CLK	Y	With the L-H flank the actual value on "Clk" will be took over; the stored value will be disposed at "Y", X → Mem0 → Y
Switch	Multiplexer	✓		✓	X1, X2, S	X1/X2	S=1: X1/X2 = X1, S=0: X1/X2 = X2
OneOf8	1 of 8 Multiplexer	✓		✓	X	Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8	X=1: Y1=1, others=0, X=2: Y2=1, others=0, X=3: Y3=1, others=0, ..., X>8 or X<0: all outputs =0
FlipFlop	RS-FlipFlop	✓		✓	Set, Reset	Q	If R=1: Q=0 (S will be ignored),

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
							if S=1, R=0: Q=1, if S=0, R=0: Q keeps last value

Time functions:

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
PT_1	PT 1	✓		✓	X, TA	Y	Start value Y = 0, Output get delayed signal from Input. After time TA: Y will be 63% from endvalue.
Impuls	Impuls	✓		✓	Trigger, Time[ms]	Imp	When "Trigger" = positive edge(L → H): "Imp" will be high for "Time"[ms] afterwards "Imp" = 0
DelaySwitch	Delayed switch	✓		✓	X, Time[ms]	Y	When "X" = positive edge (L → H): after "Time[ms]" "Y" will be set to H if "X" = 0 : "Y" = 0
SqareWave	Square wave	✓		✓	Period[ms]	Y	generates a square wave with period Period[ms]
Integral	Ramp function	✓		✓	Step, Clk, Clr	Y	A L-H flank at Clk increases Y by Step, Clr=1: Y=0
ZigZag	Ramp function	✓			Clk, Reset, StepUp, StepDown, Min, Max	Y	Reset=1: Y=Min, L→H flank at Clk: Add StepUp to Y, Y>Max: Y=Max, now add StepDown to Y at Clk flank (StepDown should be negative), Y < Min: Y=Min, now add StepUp to Y
ImpulseStep	increase/decrease	✓			Reset, ImpUp, ImpDown, StepUp, StepDown	Y	if Reset=1: Y=0, if L→H flank on ImpUp: Y=Y+StepUp,

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
							if L→H flank on ImpDown: Y=Y-StepDown.
SysClk	System timer	✓			Enable	Y	When "Enable" is set to High system timer will change status of "Y" according to the adjusted Cycle time of SIMULATIONUnit.
ClkDiv	Clock divider	✓			Clk, Period	Y	Counter preset with value "Period", "Clk" is trigger for count to zero during counter value is zero, "Y" get high pulse, "Period" will be set to preset value.

Slave functions:

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
SlaveAddress	Profibus address	✓			EA	Slv	returns the Profibus address of the slave that contains the I/O EA
SlaveAlarm	PROFIBUS alarm	✓			EA	Control	"EA" = IO Start Address of the Alarm-Slave, "Control" Bit enables an alarm, L → H Idx : ALARM coming, H → L Idx : ALARM going
SlaveAusfall	Slave failure/return	✓			Slv, Control		Control = 1: Slave will be simulated, Control = 0: Slave is dead.
PNIO_Module_Alarm	Plug/plug PNIO module	✓			HWIdx, DEVIdx, MODIdx, active		active 0→1: plug module, active 1→0: pull module.
PNIO_Channel_Diag	PNIO channel diagnostics	✓			IOAddr, DiagIdx, active		IOAddr: channel address (S0001.2),

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
							DiagIdx: Index from diagnosis file, active 0→1: enable channel failure, active 1→0: disable channel failure.
GetSlaveState	get slavestate	✓			Slv	State	Slv: Slaveaddress, in State these bits can be set: - 0 : is not simulated - 1 : active - 2 : failure - 4 : in data exchange with controller - 8 : redundant - 16 : active redundant partner
TM_COUNTER	Teleperm counter	✓			HWid, BG, Ch, Cnt, Dist	Y	
TM_SQRT	Teleperm square root	✓			X	Y	

Special functions:

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
GetTimerValue	Global-Control-Operate Time	✓			Slv, Ena/Res	ActTime, MinTime, MaxTime, DsTime	
GetCycleInfo	Cycle info of the runtime system	✓				Cycle, Time	Configured cycle time[ms], measured used cycle time[ms]
NullFunction	Dummy, does nothing	✓		✓	Value		

Firmware functions:

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
F_SLAVE	Failsafe function		✓		HW_ID, Slv, Slot4..11		will be automatically created during hardware import
H_DIAGNOSE	Redundancy function		✓		HW_ID, Slv		will be automatically created during hardware import

Name	Description	PC function	SIMULATION UNIT PB function	SIMULATION UNIT PNIO function	Inputs	Outputs	Function / Comment
MC_FIRMWARE	Motion control function		✓				used internally only
WATCHDOG	Watchdog		✓				used internally only
PROFIsafe	Profisafe V2			✓	A, A_Len, E_Len, Record1, E		will be automatically created during hardware import
PROFIsafe_V1	Profisafe V1			✓	A, A_Len, E_Len, Record1, E		will be automatically created during hardware import
sinamicsSim	Motor axis according to Profidrive specification			✓	TelIn	TelOut	will be automatically created during hardware import
DigitalIn	reads the digital inputs of the box			✓		InByte	reads the digital inputs of a SimbaPNIO box
DigitalOut	switches the digital outputs of the box			✓	OutByte		switches the digital outputs of a SU PNIO box

7.6. Adjust runtime cycle

Setting of basic-time-cycle for the runtime system is available at menu [Settings/Options](#).

7.7. Flags for applications

For use with runtime system SIMULATIONUnit provides 4096 bytes of flags for each installed SIMULATION UNIT module. These flags can be used for read and write functions.

Attention:

The user is responsible for the data management of these flags.

The addressing of the flags is equal to the I/O addressing (e.g. "S000M0.1", "S000MB10", "S000MW256", "S003MD500" ...).

Flag view

For monitoring and control of the flags a special menu is available: "flag view". In this window the flag addressing can be free defined by the user. Independent of this window the flags are available for the runtime system.

8. Programming interface of SIMULATIONUnit

8.1. General

The comfortable and user friendly programming interface of SIMULATIONUnit supports you in creating almost any simulation configuration. The simulation system handles the system hardware functionality.

8.2. Project Functions

The following commands can be used to access SIMULATIONUnit modules from applications other than SIMULATIONUnit. This is the API interface of SIMBAkern.dll for 32 bit systems and SIMBAkern64.dll for 64 bit systems.

The commands may be executed even if SIMULATIONUnit.EXE has not been started before. If SIMULATIONUnit.exe is supposed to run in the background while some other application takes over control, then SIMULATIONUnit switches to [Listening mode](#)

The operating System is WINDOWS 32- or WINDOWS 64-Bit. Exported functions have the call format PASCAL or cdecl, depending on the requirements of the application program (example: EXCEL uses the format PASCAL, whereas LabView uses the format cdecl).

Elements in structures will have byte alignment (close-packed).

The terms Input and Output are defined according to CPU notation.

The return codes are hwindexical for all functions ([Return codes](#)).

The simbakern.dll exports more functions than those which are mentioned in this documentation. For different reasons, you can only use the functions described below.

8.2.1. Set Simbakern.dll path

The **simbakern.dll** must know the path of the SIMULATIONUnit Installation.

```
int SIMBA2_SetSimbaproPath(char *path)
```

If path is empty, the current working directory is used.

8.2.2. Load SIMULATIONUnit project

```
int SIMBA2_OpenProjectFile(char *FileName)
```

As file name, the complete path of a SIMULATIONUnit project (folder with ending .spf) must be entered. SIMULATIONUnit opens projects in the background. A project is opened when the project state equals 3. The project state can be checked using **SIMBA2_CheckProjectState**.

8.2.3. Check project state

```
int SIMBA2_CheckProjectState(int *state)
```

state	Denomination	Meaning
0	EMPTY	no project present
1	CLOSED	project closed
2	OPEN	project opens
3	READY	project is opened
6	FAIL	project faulty

8.2.4. Close Project

```
int SIMBA2_CloseProjectFile()
```

closes the project. This function is executed when calling `SIMBA2_OpenProjectFile(...)`

8.3. Online Functions

8.3.1. Establish connection to simulation modules

```
int SIMBA2_ConnectAll(int hwindex)
int SIMBA2_ConnectAllEx(int hwindex, bool wait)
```

A connection to all simulation modules in the current project will be established.

hwindex [in] is the index of a SIMULATION UNIT channel. `hwindex = -1` connects to all SIMULATION UNITS in the project.

`SIMBA2_ConnectAll(...)` and `SIMBA2_ConnectAllEx(with wait = true)` are blocking until the connection is established or a timeout occurred.

8.3.2. Check connection status

```
int SIMBA2_IsConnected(int hwindex, int *active)
```

The connection status can be checked with `SIMBA2_IsConnected`. Over `hwindex`, the hardware index is given. `active` can return following values:

- `active = 0`: no connection
- `active = 1`: connected
- `active = 2`: module deactivated

8.3.3. Disconnect simulation modules

```
int SIMBA2_DisconnectAll(int hwindex)
```

All simulation modules in the project will be disconnected. The parameter `hwindex` must be set to `-1`.

8.3.4. Load hardware configuration into simulation modules

```
int SIMBA2_DownloadAll(int hwindex)
```

The hardware configuration in the current project will be loaded into the simulation module.

hwindex [in] is the index of a SIMULATION UNIT channel. `hwindex = -1` downloads to all SIMULATION UNITS in the project.

The function returns immediately. Check the progress of the download action with `SIMBA2_GetDownloadState`.

8.3.4.1. Download progress

```
int SIMBA2_GetDownloadStatus(int hwindex, int *percent)
```

hwindex [in]: index of a SIMULATION UNIT channel. `hwindex = -1`: all channels

percent [out]: Download-progress as percent value 0-100

return value = `-1`: module is not connected, `0`: percent value is valid

8.3.5. Read Data from I/O image

```
int SIMBA2_ReadIO(char *name, char *value)
int SIMBA2_ReadIO_Bin(char *name, long *value)
int SIMBA2_ReadIO_Double(char *name, double *value)
```

```
int SIMBA2_ReadIO_Unit(char *name, char *value)
```

The **name** consists of the index of the SIMULATIONUnit module and the I/O designation (e.g. S000IW512). The index of the module always starts with S. The triple-digit number that follows is taken from the hardware view of SIMULATIONUnit. The following I/O names are valid: IDx, IWx, IBx, Ix.y, QDx, QWx, QBx, Qx.y (according to SIMATIC notation), and IFx, QFx for float values.

SIMBA2_ReadIO_Double converts the value to type double.

SIMBA2_ReadIO_Unit converts the values to the selected unit and adds a unit symbol.

Attention:

When using **SIMBA2_ReadIO** or **SIMBA2_ReadIO_Unit**, sufficient buffer space for the resulting string **value** must be assigned in the application program! **SIMBA2_ReadIO_Bin** transfers the required information in un-scaled format to the defined double word.

8.3.6. Write Data into image

```
int SIMBA2_WriteIO(char *name, char *value)
int SIMBA2_WriteIO_Bin(char *name, long value)
int SIMBA2_WriteIO_Double(char *name, double value)
int SIMBA2_WriteIO_Unit(char *name, char *value)
int SIMBA2_WriteIO_BinMask(char *name, long value, long mask)
```

For parameter description refer to SIMBA2_ReadIO. Within the function SIMBA2_WriteIO_BinMask, the additional parameter "mask" will be used to select the several bits (Bit n in mask=1 → Bit n will be processed).

8.3.7. Enable/Disable simulation of a single slave

```
int SIMBA2_Activate(int hwindex, int slave, int active)
```

hwindex [in]: index of a SIMULATION UNIT channel

slave [in]: slave or device index

For **active** use these values:

- 1 - Active
- 2 - Slave is not simulated

8.3.8. Enable/Disable simulation of a module

```
SIMBA2_ActivateModule(int hwindex, int slave, int module, int active)
```

The hardware index must be entered for **hwindex**, **slave** must be set to the slave number, **module** to the slot number of the module.

For **active** use these values:

- 1 - Active
- 2 - Module is not simulated

Attention:

SIMBA2_ActivateModule is not available for Profibus boxes!

8.3.9. Check hardware state

```
int SIMBA2_IsActive2(char* indexstring, int *active)
```

indexstring [in]: Comma separated indexes of the hardware tree

examples for indexstring:

- "1" addresses Simulation hardware Nr 1. - "0,4,3" addresses the 3. module of the 4. slave in SIMBAProfibus 000.

active [out]: 0: not simulated, others: possible combinations below.

The following bits can be read together:

- 1 = activated
- 2 = failure

The following bits can be set in slaves/devices in the first layer:

- 4 = dataexchange present
- 8 = redundant
- 16 = active redundant partner
- 128 = listening mode

8.3.10. Start firmware functions

```
int SIMBA2_StartCardFunction(void)
```

```
int SIMBA2_StartCardFunction2(int hwindex)
```

hwindex [in]: Index of the simulation hardware, or -1 for all

SIMULATIONUnit recognizes automatically by import if H- or F- devices are present and creates the respective information files. These informations are transferred to the module during download.

To process these functions in Profibus boxes, the functions have to be activated explicitly after each download.

When using PNIO boxes the function blocks (f.i. Profisafe) will be also loaded to the box during download and started automatically.

```
int SIMBA2_CheckFunctionDownload(int hwindex, int *present, int *active)
```

hwindex [in]: Index of the simulation hardware, or -1 for all **present** [out]: number of functions in simulation project **active** [out]: number of functions loaded into the box

After start of SIMBA2_StartCardFunction you can check, if all functions are loaded and activated (present = active).

8.3.11. Reading or writing of complete I/O images of slaves

```
int SIMBA2_ReadSlaveOutputImage(int hardware_hwindex, int slave, BYTE *buffer)
int SIMBA2_ReadSlaveInputImage(int hardware_hwindex, int slave, BYTE *buffer)
int SIMBA2_WriteSlaveInputImage(int hardware_hwindex, int slave, BYTE *buffer)
```

These functions write or read the complete I/O images of DP slaves or PNIO devices direct from/to the hardware. The buffer management of the box will be considered (f.i. for F functions). These functions are relevant only for Profibus- or PNIO boxes and deliver when called for other hardware definitions a negative errorcode.

8.3.12. Records

The following functions can be used for handling of acyclic data.

8.3.12.1. Define record in project

```
int SIMBA2_DefRecordDataExt(int hwindex, int dev, int slot, int sub, int index, int
length, int offset)
```

8.3.12.2. Delete record from the project

```
int SIMBA2_DeleteRecordDataExt(int hwindex, int dev, int slot, int sub, int index)
```

8.3.12.3. Write default data into a record

```
int SIMBA2_WriteDefaultRecordDataExt(int hwindex, int dev, int slot, int sub, int
dsnum, char* data)
```

Parameter **data** is a string with hex values divided with comma ("0x01,0x22,0xab,0xcd").

8.3.12.4. Read/write records

```
int SIMBA2_WriteRecordDataExt(int hwindex, int dev, int slot, int sub, int index,
char* data, int maxlen)

int SIMBA2_ReadRecordDataExt(int hwindex, int dev, int slot, int sub, int index,
char* data, int *maxlen)
```

hwindex [in]: hardware index

dev [in]: device number

slot [in]: slot number of the module

sub [in]: subslot nr. (ignored at Profibus, when PNIO module without subslot set sub = 1)

index [in]: record number

data [in at write, out at read] data bytes, byte array

maxlen [inout]: number of bytes in data

8.3.12.5. Read/delete record event queue

```
int SIMBA2_ReadDataBlockQueueExt(int hwindex, int* slave, int* slot, int *subslot,
int* dsnum, BYTE* buffer, int* len)
int SIMBA2_ResetDataBlockQueue(int hwindex)
```

When the box receives a new record from PNIO-/Profibus-controller this will be added to a queue. The application can read the queue for getting these records.

Returnvalue: 0 = OK, -1 = data set queue is empty

hwindex [in]: hardware index

dev [in]: device number

slot [in]: slot number of the module

sub [in]: subplot nr. (ignored at Profibus, when PNIO module without subplot set sub = 1)
dsnum [in]: record number
buffer [in at write, out at read] data bytes, byte array
len [inout]: number of bytes in data

8.3.12.6. Register WINDOWS system event for record queue

```
int SIMBA2_RegisterDataBlockQueueEvent(int hwindex, HANDLE eventhandle)
int SIMBA2_UnRegisterDataBlockQueueEvent(int hwindex)
```

Parameters:

hwindex: hardware index
eventhandle: WINDOWS system event handle.

The application can register/unregister a WINDOWS system event channelwise for the record queue. Whenever records will be added to the queue the registered event will be raised. The application is full responsible for the validity of the event handle.

8.3.13. Reading or writing a unique project ID

The SIMULATION UNIT PN and SIMULATION UNIT PB boxes support the load of a unique generated project ID to check the load state of a box. In SIMULATION UNIT PB boxes this will be done per line. In addition to that you have the possibility to store the download sequence of the project permanently in the box and start this sequence after power on. This offers the possibility of creating pre-configured simulation boxes.

```
int SIMBA2_GetProjectIDs(int hwindex, DWORD *phardwareProjectID, DWORD
*pspfProjectID, char *AutoStartProjectName)
```

When the box is connected this function returns a unique ID of the loaded project (or one line of the project), the ID of the project/line opened in software and the name of a permanently stored project.

8.3.14. Autostart project

```
int SIMBA2_SaveProjectFlash(int hwindex)
int SIMBA2_DeleteProjectFlash(int hwindex)
```

hwindex is the index of the channel in the SIMULATION UNIT project.

Attention:

hwindex == -1 (for all channels) is not possible here!

These functions handle the store and delete of the actual project permanently in flash. For store a project it is necessary to call **SIMBA2_SaveProjectFlash** with the hardware index first and then call **SIMBA2_DownloadAll** with the same hardware index. That stores the loaded configuration into the flash memory of the box. You can watch the progress of the download with **SIMBA2_GetDownloadStatus**.

8.3.15. Export / import the I/O image

```
int SIMBA2_SaveProcessIOFile(char *PathName)
```

This function saves all I/O data of the opened project into a .csv file.

```
int SIMBA2_LoadProcessIOFile(char *PathName)
```

reads I/O values from a .csv file.

8.3.16. Reading configuration

```
int SIMBA2_ReadStringConfiguration(char *configurationstring, int maxlen, int first)
```

This function delivers offline the IO configuration of the project. There is no check of the online project included.

Parameters:

configurationstring [out]: string that represents an I/O- or datarecord-channel as an hierarchical path description.

example: "HARDWARE0\PNIODEV1\PNIOSLOT2\E20.0"

maxlen [in]: size of the buffer for the path in bytes.

first [in]: parameter has to be set by the user program to 1 at the first call and to 0 at any next call.

The user program has to allocate the buffers to fill in the string by the function. The user program calls the function until a negative return code occurs.

```
int SIMBA2_Get_Channels(char *IOName, int *direction, int *datatype, char
 *SymbolName, char *Comment, int first)
```

direction [out]: 0 = readable signal, 1 = writable signal

datatype [out]: 0 = binary, 1 = integer 8-32 Bit, 2 = float

IOName [out]: signal name (f.i. S000IW512)

SymbolName [out]: optional symbolic name of the signal

Comment [out]: signal comment

first [in]: parameter has to be set by the user program to 1 at the first call and to 0 at any next call.

The user program calls the function until a negative return code occurs.

8.3.17. Alarms and other device operations

It is possible to raise module- or channel alarms for some Profibus and PNIO devices. The alarm structure is defined in .xml files in the folder <SIMULATIONUnit installation path>\Alm. The following API functions can be used to support various user defined device operations in future.

```
int SIMBA2_ReadMenuName(char* indexstring, char *name, int menuindex)
```

reads a possible alarm for the defined object in the hardware tree (module or channel)

indexstring [in]: path index of the object over all hierarchical levels, divided by comma. (f.i. has slave 3 in SIMULATIONUnit 0 the indexstring "0,3", the module in slot 5, slave 31, SIMULATIONUnit 1 has the indexstring "1,31,5").

name [out]: Description of the alarm

menuindex [in]: ID number of an alarm

return value: 0 = OK, -1: no alarm configured

To read all alarms for an object of the hardware tree, the function has to be called until return value is not 0. The menuindex has to set to 0 at the first call and incremented at every function call.

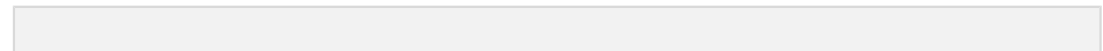
```
int SIMBA2_SetMenuCommand(char *indexstring, int menuindex)
```

raises an alarm. Parameters are the same as at SIMBA2_ReadMenuName().

The channel diagnostic alarms for PNIO devices are toggled alarms. The first call of SIMBA2_SetMenuCommand activates the alarm and the second call deactivates it. If the alarm is activated, the character * is the first character in the parameter *name* at SIMBA2_ReadMenuName().

8.3.18. Using Windows-Events

An effective application can use Windows events that are generated for special purposes.



Attention:

If you are running your application as a second instance together with SIMULATIONUnit.exe, you can only use **SIMBA2_Register...NamedEvent** functions.

8.3.18.1. End of I/O data transfer event

An application that calls [Synchronisation](#) can receive an end of data transfer event:

```
int SIMBA2_RegisterCyclicIOReadyNamedEvent(int hwindex, HANDLE eventhandle)
int SIMBA2_RegisterCyclicIOReadyEvent(int hwindex, char *eventname)
int SIMBA2_UnRegisterCyclicIOReadyEvent(int hwindex)
```

Parameter:

hwindex [in]: Hardware index **eventname** [in]: global named event **eventhandle** [out]: WINDOWS system event handle

8.3.18.2. Download ended event

After calling [SIMBA2_DownloadAll](#) the end of the download can be signalled:

```
int SIMBA2_RegisterDownloadReadyEvent(HANDLE eventhandle)
int SIMBA2_RegisterDownloadReadyNamedEvent(char *eventname)
int SIMBA2_UnRegisterDownloadReadyEvent()
```

Parameter:

eventname [in]: global named event **eventhandle** [out]: WINDOWS system event handle

8.3.18.3. New record event

An application can register a WINDOWS system event for the [Datensatzqueue](#) per channel. This event is fired, when a new record has been added to the record queue.

```
int SIMBA2_RegisterDataBlockQueueEvent(int hwindex, HANDLE eventhandle)
int SIMBA2_RegisterDataBlockQueueNamedEvent(int hwindex, char *eventname)
int SIMBA2_UnRegisterDataBlockQueueEvent(int hwindex)
```

Parameter:

hwindex [in]: Hardware index **eventname** [in]: global named event **eventhandle** [out]: WINDOWS system event handle

8.4. Functions for using the DLL with your own GUI

The following functions are only callable from the "first instance program" that has loaded the "SIMBAkern(64).dll". These are functions for creating a project, hardware import and visualisation of the I/O's.

8.4.1. Create an empty project

```
int SIMBA2_CreateProjectFile(char *PathName)
```

The function creates a new project folder and the minimal necessary files in it. The folder has the extension "*.spf".

8.4.2. Save a project under a different name

```
int SIMBA2_SaveAsProjectFile(char *OldPathName, char *NewPathName)
```

The project **OldPathName** will be copied into **NewPathName**. **NewPathName** will be opened.

8.4.3. "Save" project

```
int SIMBA2_SaveProjectFile()
```

By definition the "Project.xml" always consistent. The call of that function only sets the project state to "SAVED" in order to make a new compilation when the project will be opened next time (new creation of the necessary binary files).

8.4.4. Import hardware

```
int SIMBA2_SDBImport(char *simbaname, char *sdbpath, char *cfgpath, int clear, int
start_idx, int end_idx, char* parameter)
int SIMBA2_SDBImportwait(int hwindex, char *sdbpath, char *cfgpath, int clear, int
start_idx, int end_idx)
```

If a SIMULATION UNIT called **simbaname** is present in the project, it will be overwritten, if **clear** = 1. If **clear** = 0, a new SIMULATION UNIT channel will be created.

sdbpath - path of a sdb30???.dat for Profinet or sdb20???.dat for Profibus

cfgpath - optional, path of a HWConfig export file (*.cfg)

Parameters **start_idx**, **end_idx** and **parameter** will be ignored when importing Profibus slaves from a sdb20xx file.

For the SIMULATION UNIT PN the parameters **start_idx** and **end_idx** select which devices shall be imported. The device indexes have to be the indexes from Step7 or -1 for import of all devices.

The parameter **parameter** can be the IP address for the communication with the simulation PC. The IP address is to configure in ASCII format (f.i. *strcpy(parameter, "192.168.0.1")*); If a blank string is passed here, the IP address can be added later with *SIMBA2_SetHWParameter(...)*

Attention:

For SIMULATION UNIT PN the IP address is to configure in a different IP band as the simulated devices!

A system data block (.dat) is necessary, configuration file (.cfg) is optional.

8.5. Change parameter in project

```
SIMBA2_SetHWParameter(char* IndexString, char* key, char* value)
```

IndexString [in]: Hardware Index

key [in]: Parameter name, see below

value [in]: value, see below

Eingestellte Werte auslesen:

```
SIMBA2_GetHWParameter(char* IndexString, char* key, char* value, int index)
```

IndexString [in]: Hardware Index

key [in]: Parameter name, see below

value [out]: value, see below

index [in]: 0 = first parameter. If index >= max. number of parameters, the return value is -1, otherwise 0 = ok.

Possible parameters:

Key	value
IPADDR	IP Address (e.g., 192.168.0.1")

Key	value
CHANNELNR	channelnumber of a SU Box
FIRSTDEVICE	Devicnr. of first devices behind the PNIO Controller at SU port P1
FIRSTPORT	Portnr. of first devices behind the PNIO Controller at SU port P1
SECONDDDEVICE	Devicnr. of first devices behind a second PNIO Controller at SU port P1 (SU with 2 channels only)
SECONDPOR	Portnr. of first devices behind a second PNIO Controller at SU port P1 (SU with 2 channels only)

8.5.1. Import of TIA portal projects

The tool TIA2XMLConverter can convert .oms files from TIA Portal to .xml files. You can import the .xml files.

```
int SIMBA2_ImportXMLProject(char *XMLpath, char *simbaname, char* parameter)
```

XMLpath [in]: path of the XML file. The file type will be read from the file itself.

simbaname [in]: f.i. *SimbaPNIO_0[0]*

parameter [in]: IP address of the SIMULATIONUnit in ASCII format

8.5.2. Import symbols

```
int SIMBA2_SymbolImport(char *simbaname, char *symbolpath)
```

A symbol file exported from PCS7 (ASCII format) will be imported into the project.

8.5.3. Last used projects / SDB's / CFG / symbol lists

```
int SIMBA2_GetRecentProjectPath(int index, char *path)
int SIMBA2_GetRecentSDBPath(int index, char *path)
int SIMBA2_GetRecentCfgPath(int index, char *path)
int SIMBA2_GetRecentSymbolPath(int index, char *path)
```

Delivers up to 16 used project names (also SDB / CFG / SYM, selected by *index*), but only if the project is found.

8.5.4. Language selection

```
int SIMBA2_GetLanguage()
int SIMBA2_SetLanguage(int language)
```

In SIMULATIONUnit there is a global language selection used for the GUI, messages but also for generating correct I/O names. **Language** codes are 0 for "German", 1 for "English".

After changing the language code the project has to be closed and the *SIMBAkern.dll* has to be unloaded and reloaded!

8.5.5. Messages

```
int SIMBA2_GetErrorString(char *error, int *typeicon)
```

SIMBAkern.dll created messages (especially in case of errors). These messages can be read and shown using the API. After reading a message it will be deleted from the archive. The actual archive of messages just exists in memory and will be lost when the application ends.

Parameter *typeicon* contains a classification of the message (0 = error, 1 = warning, 2 = notice).

8.5.6. Adjust baudrate on Profibus

```
int SIMBA2_SetBaudRate(int hwindex, int BaudIdx)
```

Adjusting the baudrate of the Profibus. The following baudrates are defined for Profibus:

BaudIdx	Baudrate
0	9600 Bd
1	19200 Bd
2	93,75 kBd
3	187,5 kBd
4	500 kBd
5	750 kBd
6	1,5 MBd
7	3 MBd
8	6 MBd
9	12 MBd

8.5.7. Adjust MAC adress band

8.5.7.1. Read or write MAC addresses range for simulated PNIO devices

```
int SIMBA2_MACListRead(char* firstmac, int* number)
int SIMBA2_MACListWrite(char* firstmac, int number)
```

firstmac: first MAC address in format String with column (default ="08:00:06:9d:34:3f") **number**: max. number MAC addresses

8.5.7.2. Get MAC addresses from current project

```
int SIMBA2_GetProjectMACs(char* firstmac, int* number)
```

firstmac: first MAC address in format String with column (default ="08:00:06:9d:34:3f") **number**: number MAC addresses used in the opened project

8.5.7.3. Regenerate MAC Addresses in projekt

```
int SIMBA2_GenerateMACs()
```

Call this function to apply the settings made by `SIMBA2_MACListWrite(...)` to the opened project.

8.5.8. Import of GSD(ML) files into SIMULATIONUnit hardware library

8.5.8.1. Import of Profibus GSD files

```
int SIMBA2_GSD_Import(char* gsdfilepath, char* SlvName, char* SlvTyp, char* SlvRev)
```

reads a GSD file into memory.

gsdfilepath [in]: path of the GSD file **SlvName** [out]: slave name **SlvTyp** [out]: slave ID **SlvRev** [out]: slave revision

```
int SIMBA2_GSD_SaveXML(char* xmlfilepath)
```

saves the data imported with SIMBA2_GSD_Import.

xmlfilepath [in]: file name of the XML file in the SIMULATIONUnit hardware library.

format: <SIMULATIONUnit folder>\hwlib\SLAVE\slv<SlvTyp>.xml

<SlvTyp> has to be a four digit hexadecimal number

8.5.8.2. Import of Profinet GSDML files

```
int SIMBA2_PNIOXML_Import(char* xmlfilepath, char* SlvName, char* SlvTyp, char*
  SlvRev, char* existingFile)
```

reads a GSDML file into memory.

xmlfilepath [in]: path of the GSDML file **SlvName** [out]: slave name **SlvTyp** [out]: slave ID **SlvRev** [out]: slave revision **existingFile** [out]: if the file has been imported yet, the filename of the imported GSDML file will be delivered.

```
int WINAPI SIMBA2_PNIOXML_SaveXML(char* xmlfilepath, char* gsdmlname)
```

saves the data imported with SIMBA2_PNIOXML_Import

xmlfilepath [in]: file name of the XML file in the SIMULATIONUnit hardware library.

Format: <SIMULATIONUnit folder>\hwlib\PNIO\slv<SlvTyp>.xml,

<SlvTyp> has to be an eight digit hexadecimal number.

- gsdmlname: name of the GSDML file without path.

8.5.9. Synchronizing I/O daten with the simulation hardware

```
int WINAPI SIMBA2_SetThreadCycle(int milliseconds)
```

In SIMULATIONUnit there runs a thread cyclic synchronizing the I/O image with the hardware. The cycle time can be adjusted by SIMBA2_SetThreadCycle.

Parameter:

milliseconds [in]: valid values are 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 5000, 10000 and -1

SIMBA2_SetThreadCycle(-1) switches off the cyclic synchronization. In that case the user application has to call SIMBA2_Synchronize() to synchronize the I/O's.

```
int WINAPI SIMBA2_Synchronize(int hwindex)
```

Writes input data to the hardware and requests output data. The output data are valid for the user after the next call of SIMBA2_Synchronize.

Parameter:

hwindex [in]: index of the simulation hardware or -1 for all.

```
int SIMBA2_GetCommunicationDiagnostics(int hwindex, DWORD *syncintervall, DWORD
  *synctime, int *telegramsleft)
```

Reading the performance info of the synchronisation.

Parameter:

hwindex [in]: index of the simulation hardware. **syncintervall** [out]: adjusted synchronizations intervall (see SIMBA2_SetThreadCycle) **synctime** [out]: measured time needed for synchronizing I/O's. **telegramsleft** [out]: number of synchronization blocks not synchronized yet in current cycle.

8.5.10. Reading versions

```
int WINAPI SIMBA2_GetVersion(char *version, int para)
```

With para = 0 the version of the SIMBAkern.dll will be delivered, with para = 1 the version of the hardware library. SIMULATIONUnit boxes send their versions after SIMBA2_ConnectAll was called. After connection is established versions can be read by SIMBA2_GetErrorString.

8.6. Load sequence (example)

Please call the following functions step by step to open a simulation project, load it into the hardware and start the simulation:

```
1. int SIMBA2_OpenProjectFile(char *FileName)

2. int SIMBA2_CheckProjectState(int *state) ... wait until state == 3

3. int SIMBA2_ConnectAll(-1)

4. int SIMBA2_DownloadAll(-1)

5. int SIMBA2_GetDownloadStatus(-1, int* percent) ... return value has to be 0,
   percent has to be 100
```

Only for Profibus projects:

```
6. int SIMBA2_StartCardFunction()

7. int SIMBA2_CheckFunctionDownload(-1, int *present, int *active) ... wait until
   present == active
```

8.7. Return codes

Denomination	Value	Meaning
SIMBA_QUIT_POS	0	O.K.
SIMBA_QUIT_NEG	-1	Internal failure, unexpected program error
SIMBA_PARAMETER_INVALID	-2	Wrong parameter in call (syntax or value not available)
SIMBA_NOT_SUPPORTED	-3	not used
SIMBA_WRONG_SIMBATYPE	-4	function not supported by this SIMULATION UNIT
SIMBA_SYS_DRIVER_CLOSED	-5	hardware driver already closed
SIMBA_THREAD_NOT_STARTED	-6	task-thread not launchable (WINDOWS-problem)
SIMBA_CHANNEL_LOCKED	-7	channel locked by another application, no read-access possible.
SIMBA_MULTI_ACCESS_ERROR	-8	multi access from other applications causes WINDOWS-failure

Denomination	Value	Meaning
SIMBA_MAINPROG_NOT_RUNNING	-9	main procedure of driver was stopped
SIMBA_BUSY	-10	timeout / delay caused by SIMULATIONUnit.exe
SIMBA_HARDWARE_NO_ACCESS	-11	No access to hardware, hardware index is not present in project
SIMBA_TIMEOUT	-12	Time out
SIMBA_QUIT_NOT_CONNECTED	-20	Module not connected
SIMBA_WRONG_FIRMWARE	-23	wrong firmware version used
SIMBA_QUIT_FULL_BUFF	-29	request buffer is full
SIMBA_REQUEST_TIME_OUT	-30	SIMULATIONUnit.exe causes time out of call.
SIMBA_DRIVER_ERROR	-40	SIMULATIONUnit driver not available or not working
SIMBA_MEM_ERROR	-41	Driver cannot allocate memory for I/O-image
SIMBA_PROJECT_FAIL	-48	SIMULATIONUnit cannot open project or project not available
SIMBA_PROJECT_CLOSED	-49	no project opened
SIMBA_FILEIO_ERROR	-50	file access error

9. OPC server

9.1. Purpose

The OPC server for SIMULATIONUnit enables an OPC client to read and write simulated process values of a currently open project.

SIMULATIONUnit OPC server supports communication links according OPC data access (OPC DA) specification V2.0.

9.2. Installation

With the installation of SIMULATIONUnit the OPC server get registered to the host PC (the PC running SIMULATIONUnit). All OPC clients running on the same computer are now able to access the SIMULATIONUnit OPC interfaces.

For clients running on other external systems it may be necessary to register the server there.

For this purpose the file **SimbaOpcServerProgld.reg** is provided. This file is located in the SIMULATIONUnit program folder in the subfolder OPC. (Usually **C:\Program Files\SIMULATIONUnit v8.xx.xx\OPC**).

To register the server at the client please copy the file **SimbaOpcServerProgld.reg** to the client and double click it to merge it with the local registry.

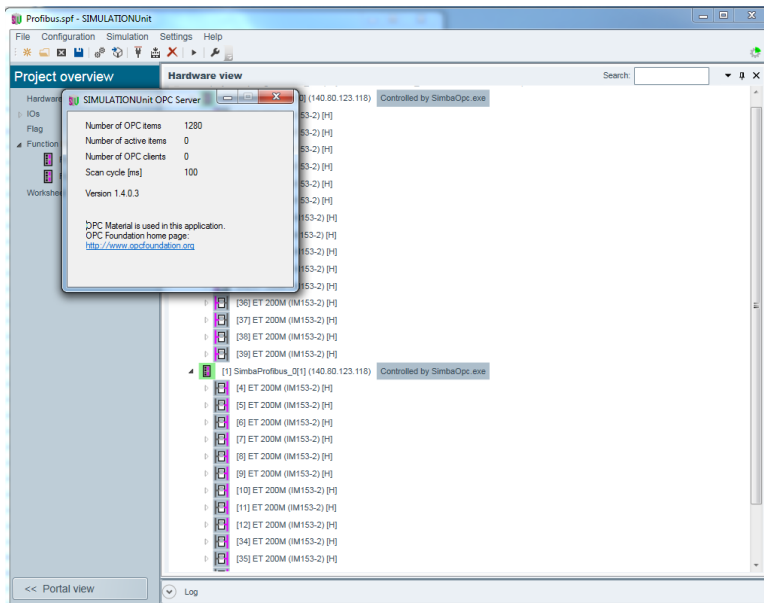
This gives the OPC client all necessary information to connect with the server.

For remote OPC communication it is usually required to adjust the DCOM security settings on the client and on the server computer. OPC settings are adjusted with the standard Microsoft program dcomcnfg.exe. Please refer to the documentation provided by Microsoft for more information about DCOM settings.

9.3. OPC server start / stop

To start the OPC server it is necessary to start SIMULATIONUnit and to load a project file first.

Menueitem "Simulation → Start OPC server".



The OPC server starts with the opened project and takes the control over the IO's. It connects to the SIMULATIONUnit boxes automatically and transfers the data.

Attention:

OPC server cannot modify or download any configuration, this has to be done in SIMULATIONUnit before OPC server starts!

The server terminates automatically when the last client disconnects.

To successfully start the OPC server it is necessary to start SIMULATIONUnit and to load a project file first. If a client connects to the server while SIMULATIONUnit is not running or no project was yet loaded the server will refuse the connection.

9.4. User interface

The OPC Server provides a simple user interface to display the following statistic values:

Number of OPC items	This is the number of process values available in the SIMULATIONUnit project.
Number of active items	Displays the number of process values that one or more OPC clients are currently connected to.
Number of OPC clients	The number of active client connections.
Scan cycle[ms]	Scan cycle in milliseconds. The OPC server is scanning the process values for changes within this cycle.

9.5. Cycle Time

The cycle time for scanning changes of the I/O values can be changed in SIMULATIONUnit in menu "Settings → Options".

Restart the OPC Server to activate the changes.

In case the SIMULATIONUnit cycle time is less than 100 ms the OPC server will use 100 ms as a minimum.

10. FAQs (Frequently Asked Questions)

10.1. In Simba Project more devices are present after a DP/PA-Link as in the Step7 project. Why is the simulation not working for the PA- Slaves which are present in both projects?

The PA- Slaves are processed as an image in the IM157. SIMULATIONUnit simulates this image and not the structure of the PA bus. This is why the image is changing by inserting or deleting a PA-device. Moreover the length of the IO-data of the IM157 changes, so that the bus master exchanges no cyclic data anymore. This is why a complete PA-link can not be simulated anymore. Help : Use here an original IM157 with a PA-SIMBA directly connected (or use a consistent project).

10.2. A module has 3 bytes (EBO - EB2), the following device has 2 Bytes (EB3 - EB4). What happens if we write EDo ?

It will write over the 3 first bytes of the first device and over the first byte of the second device. S7 works with big-endian format: i.e. the MSB will be written and a multiple of 256 will then be displayed for the second device.

10.3. Can I simulate with a SIMBAProfibus box each active Bus of 2 H-systems ?

No! A H-System uses 2 CPU's, which are communicating with each other informations over the periphery status and parametrization. For this reason, it exists also such a relationship between the channels of a redundant slave in the SIMBAProfibus box. This relationship will be processed by activating a firmware block. In the case mentioned above, one of both H-system would always have a bus failure, thus SIMBAProfibus firmware always activate only one channel. The other one is then passive. Moreover the SIMBAkern.dll alone recognizes a H-system in the configuration and then always reads the CPU-outputs from active channel. It also always writes CPU-inputs from channel 0 to both profibus channels.

10.4. How do I parametrize failsafe modules in SIMULATIONUnit ?

Failsafe modules will be imported automatically.

For Profibus failsafe see [Configuration of Failsafe components](#)

For Profinet failsafe see [Simulation of Failsafe Modules \(Profisafe V2\)](#)

10.5. How can I simulate a device that SIMULATIONUnit does not know ?

You can import Profibus and Profinet device description file into SIMULATIONUnit.

see [Device GSD import](#)

The new devices are available if you create a new project or (re)import a hardware configuration.

10.6. How can I get rid of a topology error ?

The topology cannot be imported from Step 7 or TIA. You have to set the device and port number of the first simulated device that is connected to the outer world.

see [Simulation of the Topology](#)

10.7. Is it possible to simulate multiple Profinet networks with one SIMULATION UNIT ?

You can copy the devices of several Profinet lines within one Step 7 / TIA project into one SIMULATION UNIT project if

- all device numbers, devices names and IP addresses are unique
- no topology is projected

Copying devices from different Step 7 / TIA projects into one SIMULATIONUnit project does not work. This will generate I/O address conflicts.

10.8. My project is correct, but the simulation does not work. What can I do ?

Contact the technical support at <https://support.industry.siemens.com/My/ww/en/requests#createRequest> Give a detailed description of your problem and attach your Step 7 project if possible. An engineer will contact you for support.