



Engineering



Content

About this Manual	4
Overview.....	7
Overview for Advanced programmers	10
Installation of Multiple SQL server versions in parallel	12
Global Communication View	13
Log files and how to view them.....	15
Database Synchronization	16
Default Usernames and Passwords	17
Station Settings and Projects.....	18
BatchXpert SDK.....	21
Project Engineering Tool	22
Taglist and importing.....	24
PLC code generation.....	29
Creating a New Project.....	32
Copy data from another Project's database.	34
Tips for handling Projects	36
Special Considerations for Simatic S7-1500	37
Connecting to a S7-1500 or S7-1200 PLC	40
Special Considerations for Simatic S7-300/400.....	44
Simulation of PLC.....	45
Required PLC Hardware Configuration Settings.....	48
PLC retentive data	50
PLC Time Synchronization	56
PLC Overview	58
The "Trans xx" blocks for IO signal transfer	62
TIA-Portal.....	64
TIA Portal: Recommended way to modify Control module data blocks	67
Global Signals and Symbols	69
General Structure of Control Module DBs	72
Actuators (Act).....	74
Actuator Interlocks	82
Digital Inputs (DIIn).....	87
Analog Input (AIIn).....	92

Analog Input Scaling	98
Polygon Tables.....	103
PID Regulator (PID)	105
PID Regulator Output Scaling	111
Counter Module (Cnt).....	113
Message Module (Msg)	117
Alarm Groups.....	120
Software Switch (Switch).....	124
Frequency Converters	126
Material Modules	135
Process Unit.....	140
Unit Data block.....	145
Unit Function Block	147
Unit Parameter Module.....	154
Unit Phases	158
Unit Status Signals: "Phase Active"	168
Subphases (Phases inside Unit Phases)	171
Recommendations when implementing a Phase	176
Unit Properties	178
Unit Status	180
Starting a unit with an existing or new Batch.....	181
Shared process Resources	184
Recipe Options.....	187
Run and Hold timers.....	189
Event recording	190
Unit-to-Unit Communication.....	192
Some Common Help functions.....	203
"Class" Programming.....	205
Production Planning System.....	217
PLC-to-PLC communications.....	218
Extending of Control Module Data blocks.....	224
Overview of HMIs	227
"Visu Extern" System (Touch Screens)	229
Process Graphics with VisXpert	231

BatchXpert Process Screens	236
Process Graphics Style Guide	240
Application Start Script.....	243
The HMI Main Menu	244
Configure communication with the PLC.....	246
Diagnosing PLC communication problems	248
Endianness.....	250
HMI Tag names.....	251
Status Tag Definitions of Control modules.....	252
The HMI Library	260
Adding a new PLC to the HMI.....	264
The Batch Module	267
Show Trends with VisXpert.....	268
Record custom Trend Value	271
Adding Batch Number, Step number and Phase	272
Adding Batch Number to Trends	274
PLC backup.....	276
Operating Station Backup.....	277
Recommended Settings for Touch Panel use	278
System Hardening and Operating system Security	280
Virtualization	283

About this Manual

The BatchXpert system is a system for controlling, managing, and visualizing batch processes, incorporating batch reporting, protocols and recipes, batch tracking with materials, trends, alarms, BatchXpert station and/or HMI. BatchXpert is a modular system that can be adapted to a wide variety of use cases, and can manage whole production plants, such as breweries, but also be reduced for controlling single CIP stations, or Pasteurization units. The system is fundamentally divided into three categories, which work together in an automation project.



Target Audience of this Manual

This manual is directed towards Project engineers that have a basic understanding of the fundamentals of the BatchXpert system, and process engineering in general. A fundamental understanding of the functionality and usage of the BatchXpert system is assumed for the purpose of this manual. Furthermore, basic knowledge of Automation technology and Programming with Simatic S7 style controllers is also assumed.

This manual tries to teach the basics of how these concepts are applied in the BatchXpert control system and is not intended as an introduction into PLC programming in general. Basic knowledge of PLC Programming and the usage of their tools, such as “Simatic Manager” and “Simatic TIA Portal”, are required.

Version of this Manual

Version 2.0	Initial Version
Version 2.1	Addition of Visu Statuses section
Version 2.2	Addition of Alarm Groups Addition of Hardware Settings Addition of Taglist Addition of PLC-to-PLC Communications
Version 2.3	Added many more Actuator Interlock examples Added better Unit Function block example
Version 2.4	Greatly improved and extended HMI sections of this manual Added Security chapter Added Virtualization chapter Added Frequency Converter Status Word Descriptions
Version 2.5	Added chapter about PLC simulation
Version 2.6	Added chapter about “Starting Units from the PLC”
Version 2.7	Added PID parameter Examples Improved “Trans xx” chapter for IO transfer blocks Added Special Explanations for Analog Input and Output Scaling Mentioned Incompatibility of “PLC Sim” for S7-1200 from Siemens Added Database Synchronization Chapter with Time Synchronization Added Chapter about Analog Input and Regulator Scaling
Version 2.8	Added Chapter about “communication settings” for S7-1200/1500 PLC’s Added Chapter about “Diagnosing Communication Errors”
Version 2.9	Added Chapter about Sub-Phases inside of Unit Phases
Version 2.10	Added Chapter about data Remanence Added Chapter about Web server Added Chapter about “Phase Active” and its equivalent in other Process control systems. Added Chapters about PLC Backup and PC Backup Added Chapters about PLC Time synchronization Added Chapter to explain the “Application.Start” script Improved Actuator IO Assignment example and added more explanations Added Chapter about HMI Main Menu Added Chapter about adding Batch Number, Step number and Phase Minor improvements in other chapters Improved Chapter about HMI library, HMI Main Menu, Adding a new PLC to the HMI
Version 2.11	Improved Trend Configuration section Added explanation of “Batch Module” Added “Process Graphics Style Guide”
Version 2.12	Added TIA-Portal Recommendations Reordered some chapters to make more logical sense Added chapter on how to modify existing Control module data blocks Improved Project Engineering tools and SDK Chapters Added Log files chapter

	Added Item "Communication load" to S7-1500 hardware settings Added item about "Touch monitor" recommended settings Added Overview for Advanced programmers coming from Braumat or Botec
--	---

Overview

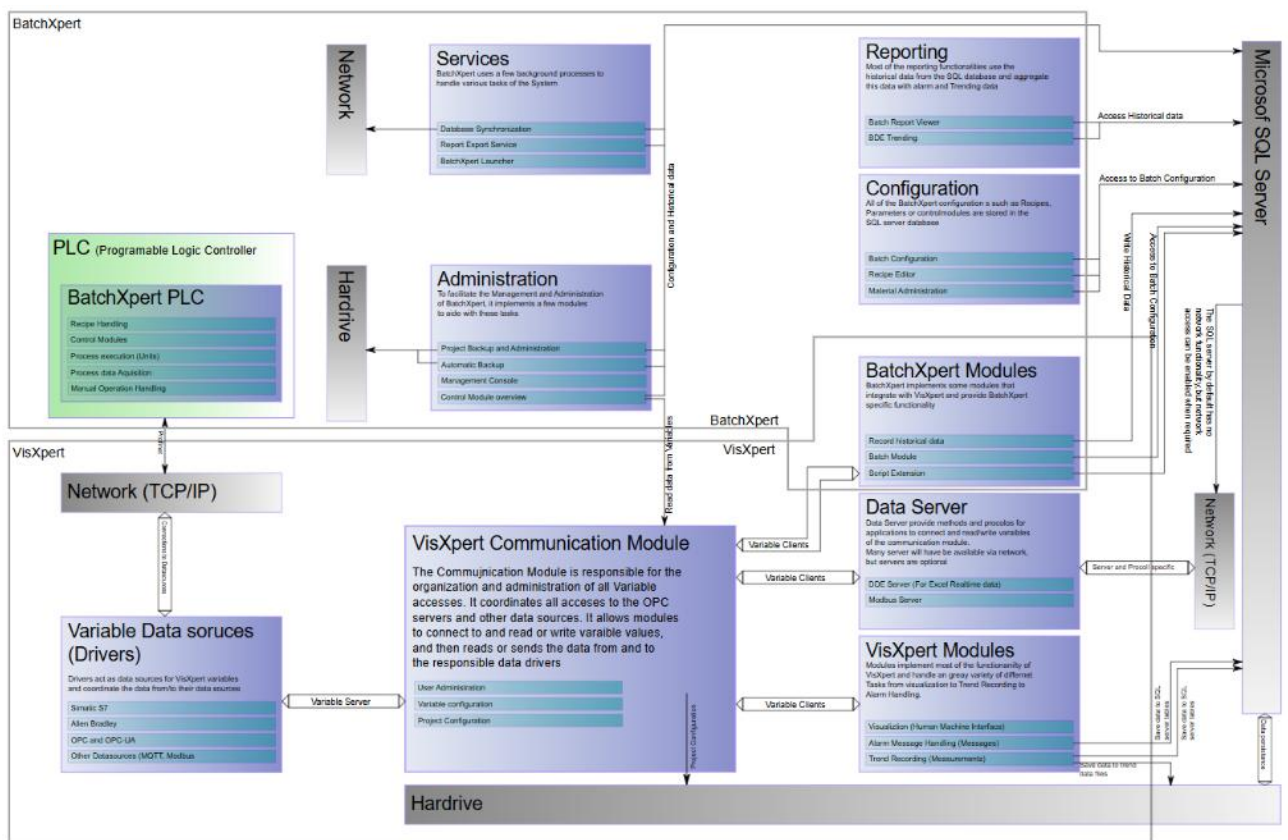
VisXpert Scada

BatchXpert is based on the VisXpert Scada system. This system is used as the main SCADA component, and manages communication with the Plc. It is responsible for the Process graphics, Alarm management, Trend recording but also the recipe downloads and historical data recording. This means that BatchXpert fundamentally requires the VisXpert SCADA to work properly.

VisXpert comes with an already prepared HMI library, which incorporates all functionality of BatchXpert and allows you to rapidly create process graphics with all the functionalities available. The Process symbol library also supports updating from a library, which enables you to easily update the symbol library in a project with newer and improved versions, and with versions that have bug fixes applied.

Software Architecture

VisXpert is a modular SCADA system that allows extension by “Modules” which can interact with the Scada system. BatchXpert integrates several modules into VisXpert to allow its specific functionality. These modules give BatchXpert access to the variables defined in VisXpert and to interact with all its functionality. A log of functionality is implemented using VisXpert modules.



The Database

BatchXpert stores most of its configuration and historical data in an SQL database, where all the tools have access to this data. As database the “Microsoft SQL Server Express” is used, which will be installed from the Installation Center.

While working with the database you must keep in mind that these database files will be updated when started with newer versions of BatchXpert or the SQL server, and after that are not compatible anymore with previous versions.

Considering this you must keep in mind two cases:

1. If you start a project with a newer version, the database may be updated to support new functionality. In that case it will not be compatible anymore with the previous BatchXpert versions. For example, if you start a project created for BatchXpert V1.8 with V1.10, some database updates will be applied, and after that you cannot open the project anymore with V1.8.
2. If you installed a newer SQL server version, the server itself will upgrade your database file, which makes it incompatible with previous versions. For example, if you have a database file for SQL server 2019 and open it with SQL server 2022, it will be upgraded and afterwards cannot be used with previous versions of SQL server.

When starting the project, both cases will give you a clear warning and the option to abort or allow the upgrade. Keep this in mind when you are doing engineering for projects that are already deployed.

To mitigate the first problem, you should always update the deployment on site when doing engineering or temporarily downgrade your BatchXpert installation when doing engineering.

For the second problem you can follow the instructions below to support multiple SQL server versions.

PLC

The PLC program was made as a program based on S7 platform, so it can be used on all S7 PLCs. For BatchXpert, a special standard was implemented in programming, with modularization, standardization of programming with the creation of standardized interfaces, to achieve considerable improvement in process engineering times. In this Program, several Criteria were considered – from short cycle times to the preparation of the program with various prepared and standardized processes.

The PLC programming follows an “Imperative/Declarative” programming style, although you can apply techniques as you see fit.

Tia Portal and Simatic Manager

BatchXpert is programmed in “AWL” (Anweisungsliste) or “STL” (Statement List) in English. Both programming environments support this language. You can choose whatever programming environment you prefer. Most of the Examples are based on the older “Simatic Manager” but should work without problem on “Tia Portal.”

We Recommend that you use “Tia Portal” for all new projects but not upgrade existing projects.

By default, SQL Server is always Local

By default, each BatchXpert station has its own installed SQL Server. This means that each station only utilizes its locally installed SQL Server service. Network functionality is usually not enabled on these SQL Server instances, since all database accesses are always local.

This is being done to improve the system's reliability, by eliminating a common point of failure which would be a centralized SQL Server. To maintain the same system configuration between multiple stations, BatchXpert includes a synchronization service that synchronizes the project database between all stations.

This means that all database access is always located on the computer, and do not need network access for its configuration data.

You can however configure batch expert to use a centralized server, but this is not recommended, and the system is not explicitly designed for this use case.

Overview for Advanced programmers

In this section we want to give a small overview for programmers that already have experience in other control systems such as Braumat, Botec, BrewMax or PCS7.

General

BatchXpert PLC programming is like other control systems such as Braumat or Botec systems. Programmers familiar with these systems will find it exceedingly easy to transition to BatchXpert.

BatchXpert uses the VisXpert SCADA system as its main SCADA application. This system was first developed by Gefasoft under the name of "GraphPic", and then later acquired by Mlogics for further development and support in the BatchXpert Control system.

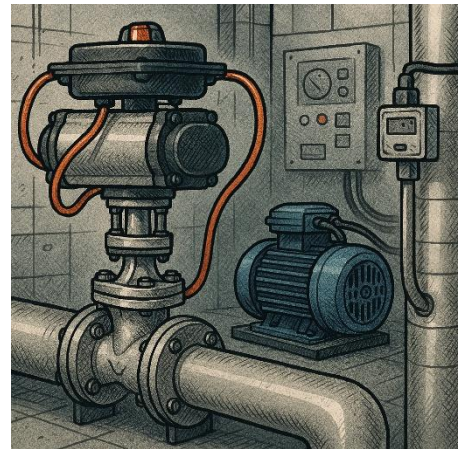
Recipe execution

Configurations such as faces parameters and recipes are all stored in an MS-SQL database. If a unit requests a recipe, this request is observed by the Skype application which in turn loads didn't require data from the database compiles the recipe and downloads it to the requesting unit. This data includes steps to be executed and set point parameters for each of these steps. From then on, the PLC takes over the execution of this recipe step by step by executing each of the programmed phases in the PLC. This is like systems like Braumat or Botec.

PLC Control modules

In BatchXpert, all control modules of all types are automatically executed and processed by the PLC framework. This means that you do not need to call a function for any of the control modules, BatchXpert internally executes all existing control modules that are available in the control module data blocks. The system checks the amount of control modules present in these data blocks and executes every one of them automatically, without the need for the programmer to call any control module functions. This means that you can consider control modules as a commodity resource that you can just use in your PLC application.

All control modules of the same type are all contained in one single data block. This means that all actuators, for example, are all in the same data block and are a global resource. You can assign control modules to specific units; this however only affects reporting and alarm groups. Control modules are always considered global and do not belong to a specific unit in and of itself. Of course, actuators are usually assigned to the automatic status of a specific unit, they can however be used by any other unit that wants to activate the automatic control (Aco) for example.



SCADA Control modules

A similar concept applies to the communication between the PLC and the HMI application. Since the PLC always executes all control modules that are available in the PLC, the VisXpert SCADA system has all the variables for all controlled modules already preconfigured. This means that if you need a new control module you can simply use it in your HMI application and since the PLC already executes the appropriate module and all variables already exist you do not need to call the control modelling appeal C nor do you have to import variables into your SCADA application. All variables for all control modules always exist in all the SCADA applications. This is possible because VisXpert does not limit the number of configured variables by its licensing model.



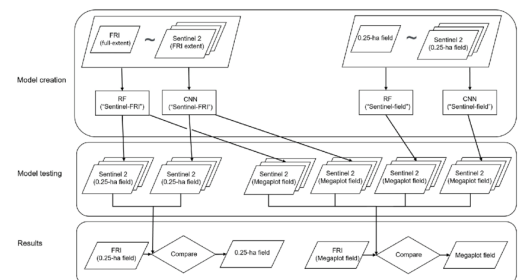
Unit Execution in the PLC

When you work in the PLC, and a unit is executed, the unit will always copy its unit data block onto a generic unit data block called "Bx Uxx". This means that if you want to refer to your unit data block from inside of your unit function code, you can and should do so by always referring to the generic unit data block "Bx Uxx", which will always contain the current data of the unit that is currently executing.

This has the advantage that you can write the same code in different units, without the need for adapting data block numbers and such since the unit itself will always copy its unit data to the generic unit data block and back after the unit function code is executed. You can find further information here [The "Current Unit Data block" \(DB100\)](#)

Unit to Unit Communication

BatchXpert Includes a concept for communication between multiple units in a standardized way. This system allows you to select a communication partner, send signals to this communication partner, receive signal from the selected communication partner and allocate the communication partner for exclusive use. This system automatically handles unit allocation, and thus eliminates problems caused by multiple part and units trying to communicate with the same unit. More information here [Unit-to-Unit Communication](#).



Installation of Multiple SQL server versions in parallel

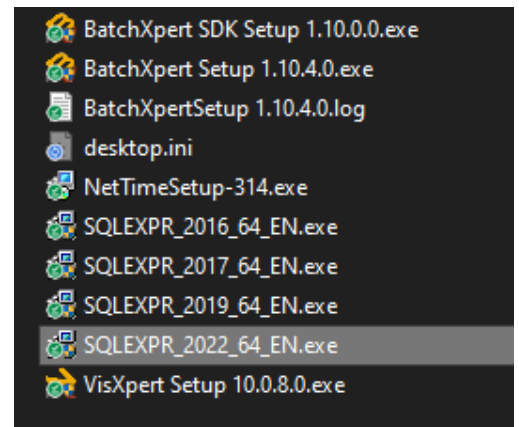
For more details about the installation, you should review the “Manual BatchXpert Installation on Windows” manual. However, there are complementary things that should be mentioned for project engineers.

Since the SQL server database files cannot be used once upgraded to a newer SQL server version, you should always do engineering with the same version as you have deployed on site. Since this may mean that you have multiple versions of SQL server, you should consider installing multiple SQL server versions in parallel.

If you get the SQL server warning when opening a project, you can then just change the server version you want to use in the “Management Console” and start the project, with the same version as deployed on site.

Manual Installation

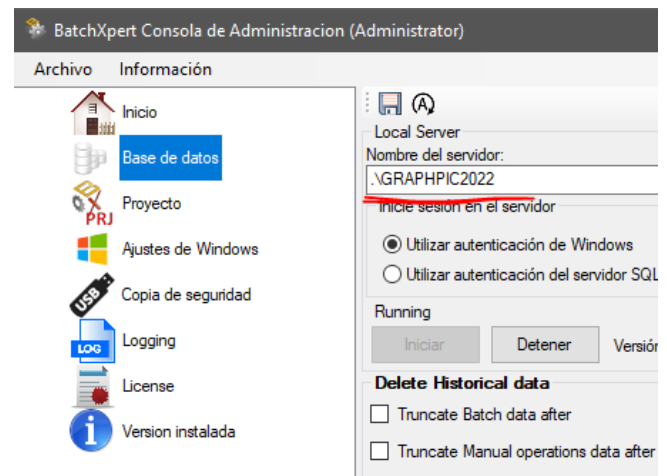
By default, the database server is called “GraphPic,” which is the instance name of the server as the “Installation Center” will install it. If you want to install a different version, you will have to choose a different “Instance Name” for this server, for example “GraphPic2022” etc. To be able to choose different Instance name, you cannot use the Installation center to install the SQL Server, as it always installs with the default “Instance Name.” You can download the SQL server installer from the installer, but then execute it manually, which gives you access to the configuration dialog where you can configure the “Instance Name” among other settings. After downloading, the installer file will be in a sub directory called “Setups” of your Installation center.



Select current SQL server instance.

After installing the different versions, you must specify which “Instance Name” and by extension which version of SQL server you want to use for engineering.

You can change the SQL server instance that is being used by all BatchXpert applications by opening the “Management Console” and changing the “Server Name” in the “Database” settings.



Global Communication View

The BatchXpert system uses standard communication interfaces and supports various communication systems from the PLC to the Automation Field. The following is a summary of the communications:

The BatchXpert system supports connecting "BatchXpert Stations", "HMI Displays" and other equipment through an Ethernet network with TCP/IP, which can be managed with standard tools for Ethernet network management. It is also possible to connect laptops through "Wi-Fi" access points and provide a form of remote control for smartphones or tablets.

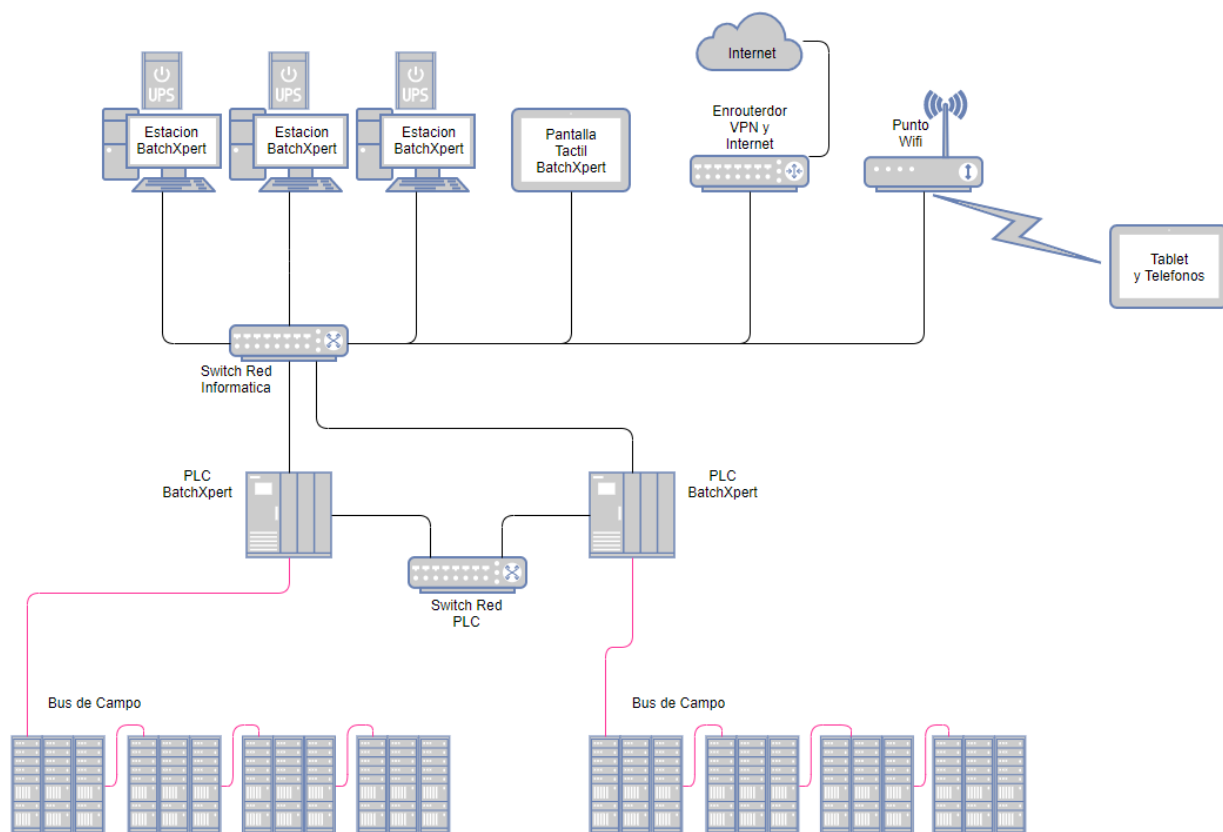
Communication between PLCs should take place at a separate PLC port (if your plc has two Network interface controllers), thus preventing BatchXpert network communications from interfering with the exchange of data between system controllers.

The customer has the option of connecting the system to their corporate network and/or the internet to access the databases and to enable the possibility of Teleservice (through "TeamViewer", please read the "TeamViewer Teleservice Manual").

The field communication is conducted through "Profibus" or "ProfiNet" interfaces that are industrial standards and allow us to connect to a wide variety of equipment such as IO systems, olive harvesters, meters, etc.

Example Simple Network Layout

This is a simple layout, and does not include hardware firewalls, DMZs, or other advanced IT infrastructure. Of course you Can extend this simple diagram to include these options.



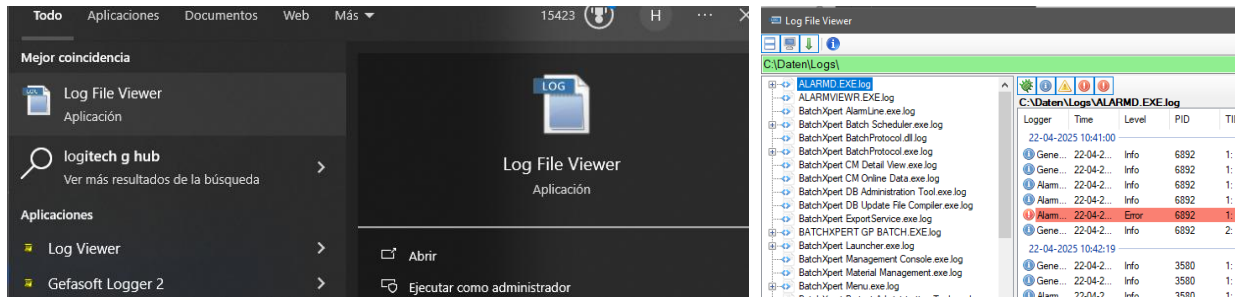
Default Communication Interfaces

BatchXpert uses and supports the following communication interfaces.

- **PLC to Operating Station**: S7-Connection through a Simatic driver. Optimized data blocks of S7-1500 are NOT supported.
- **Fieldbus**: Profibus, Profinet, HART and ASI and any other fieldbus implementable with Simatic plc's
- **PLC to PLC**: Can use any connection supported by your plc. Usually these are S7-connections or ISO-TCP connections. We recommend S7-Connections because they are easier to use.
- **Operating Stations to Operating Stations**: The stations use a proprietary protocol based on TCP/IP

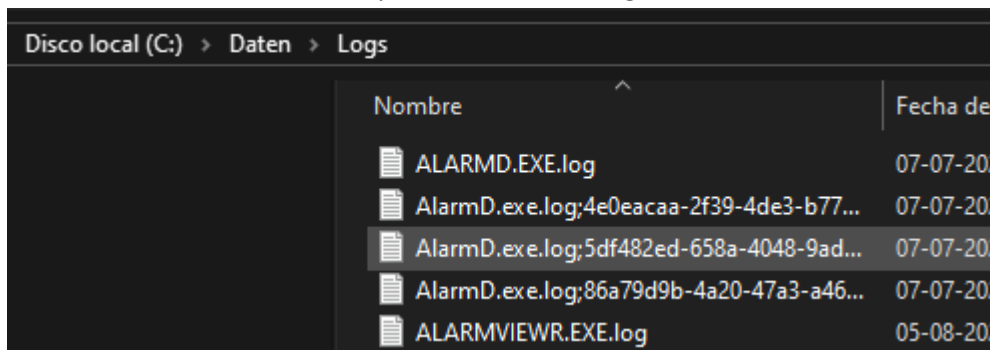
Log files and how to view them

All BatchXpert applications write log files where you can review errors and debug messages to find out if there have been some problems or if the system is functioning as intended. You can view these log files by using the “Log File Viewer”, which can be easily found in the start menu.



Location on Disk

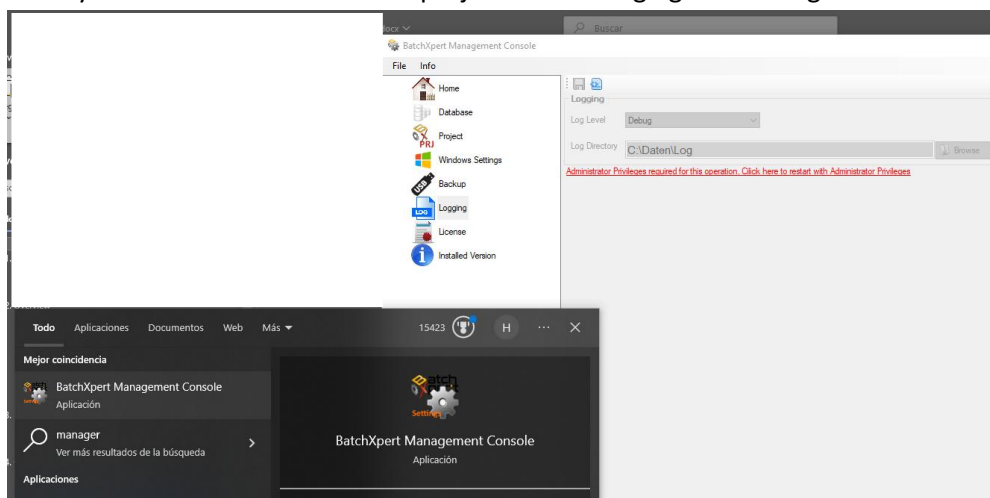
All log files are stored in the “C:\Daten\Logs” directory in your Hard drive. Log files can be opened in any regular text editor; however it is usually easier to use the log file viewer.



Setting the Log level

The amount and depth of the log messages that are getting written by the applications depends on your log level setting. You can change this setting by opening the “BatchXpert Management Console” and opening the “Logging” section.

However keep in mind that after changing the log level setting you usually have to restart your applications order for this setting to take effect, since the setting is only read once when an application starts. This means that it is usually easier to restart the whole project after changing this setting.



Database Synchronization

Since each BatchXpert station uses its own local SQL Server and configuration database, BatchXpert includes a synchronization service that synchronizes all databases between each other.

The data Synchronization is being performed by the “BatchXpert Synchronization Service” and operates in an “Peer to Peer” fashion. Each BatchXpert Station connects to all other Stations in the Network. Each Station checks

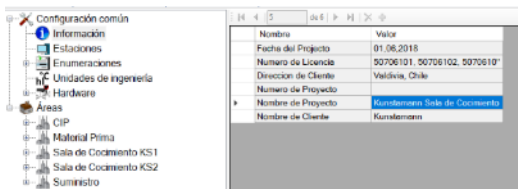


Figure 1 The Project Name is used as Identifying Number

all other station if they have actualized data, and if so, downloads them from the station with the most current data.

Each Project that is started on a BatchXpert station has an Identifying Number, and only Stations with the same Identifier are allowed to Synchronize data between each other. This allows various BatchXpert Applications to run on different stations, without mixing data during Synchronization.

You can find more detailed information in the “Manual BatchXpert Data Synchronization” Manual.

How do Stations Discover each other

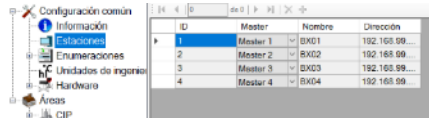


Figure 2 Station Configuration in the Batch Configurator

Each station tries to establish communication with each of the configured Stations in the Batch Configuration database. In the Batch Configuration Database one can define all stations existing in a network with their corresponding IP Address and Network Names. After connecting the configured stations, the stations start a Discovery mechanism on the network to find Stations that are not configured in the Database. If a Station is discovered, communication is being established to this station.

When a Connection is established, the Station first checks if the Stations are compatible for Synchronization, and if not, the connection is rejected, and a message is sent to the user. During connection, the following is checked for compatibility:

- System time of both stations is within 1 minute of time difference
- Both Stations have the same SQL-Server Version
- Both Station have the same Project Name

By default, the discovery mechanism is using UDP Port 3702.

System Time

The system time is very important on the Stations, since this is required for accurate Timestamping of all changed data. To avoid problems during synchronization, the System Time difference between the stations is permanently monitored between them. If any station has a difference of more than 1 minute, it is rejected by the other stations and can no longer participate in the synchronization. In the case of a Time difference, a Message is raised to the operator, to indicate this kind of error.

For this reason, it is important to keep the System time synchronized between the Stations.

To set up the time synchronization you should view the “Manual BatchXpert Time Synchronization” manual, which describes the process in more detail. The manual can be found on our Knowledgebase (www.docu.mlogics-automation.com/batchxpert-2/engineering-2/)

For more information about how to adjust the Time in the PLC please review [PLC Time Synchronization](#)

Default Usernames and Passwords

BatchXpert by default uses the following passwords that should be changed before deploying the application.

Application	Username	Password
VisXpert	Admin	<no password, leave empty>
VisXpert	Admin	MLogics
PLC	None	None
Database (old)	sa	Delphi
Database (new)	sa	GraphPic2023
S7-1500 Web Server	Admin	BatchXpert1

Station Settings and Projects

BatchXpert divides its settings between “Station Settings” which are station global and settings that are project dependent. BatchXpert includes an “Management Console” that allows you to adjust most of these global station dependent settings, such as the stations Master Number, or the automatic backup system.

Projects contain all the configuration of BatchXpert applications, such as the SCADA images, Database configuration, plc programming and historical data.

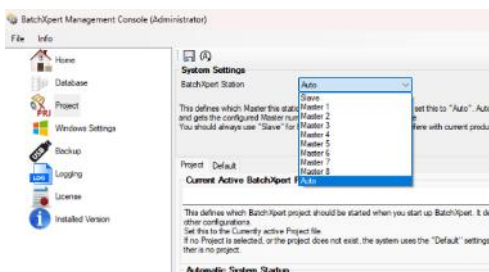
Master and Slave Stations

BatchXpert stations are divided into up to 8 Masters and as many Slaves as you need. Each Operating Stations is assigned a Master number (Master 1 up to Master 8), and all other stations will have the “Slave” designation. Usually, all the BatchXpert stations are masters, so slaves are only used if there are more than 8 operating stations in a system.

Each master will use one of the 8 communication channels, designated by their master number. This communication channel allows the master to receive Recipe requests, record historical data and perform other actions.

To avoid problems with these communication channels, programmers should always use “Slave” on their engineering stations and programming notebooks when executing the System. This setting can be set to Slave from the “Management Console.”

Setting the Master Number



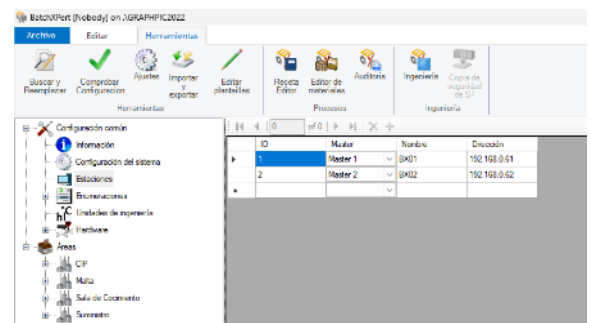
In the “Management Console” you can set the stations master number to use or set it on “Auto”.

If set to “Auto” it will look up the configured master number by using the stations Hostname in the “Computer list” of your Project configuration.

Usually, it is best to leave this setting on “Auto” and fill in all computer names into the “Computers” list in the “Batch Configuration.”

In a BatchXpert system, the default operating stations are set to the following names.

- BX01 for the first server
- BX02 for the second server
- BX03 for the third server
- BXSlave1 for the First Slave
- ...



Important Directories

BatchXpert relies on a few important folders to function properly:

C:\Daten

There are several folders that correspond to different system functions.

- **Backup:** Contains system backups
- **Logs:** contain information about the events of each tool

C:\Program Files (x86)\BatchXpert

There are folders where we can find information and the executables of the BatchXpert tools.

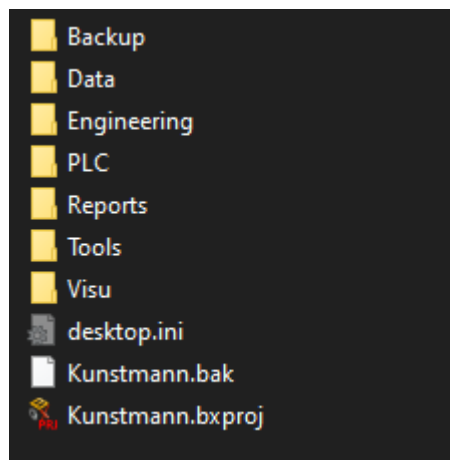
- **Documentation:** contains manuals for programmers and operators in different languages on how to use BatchXpert, as well as information on the BatchXpert license.
- **Reports:** in this folder you can find the templates of the different reports that can be generated by BatchXpert (Batch Sumy, Recipe Report, Report with Menu...).
- **Tools:** These are additional tools from BatchXpert (Command line Tools).

C:\Program Files (x86)\BatchXpert SDK

There are additional BatchXpert tools and templates for the programmer, it is oriented to the compatibility of the different software with the BatchXpert system and HMI, to minimize the engineering time, the **installer of the BatchXpert SDK** is required.

- **Documentation:** contains manuals for programmers on how to use the BatchXpert, HMI and PLC tools.
- **PLC:** contains templates for the S7 PLC (Vipa compatible)
- **Tools:** Contains templates for project documentation (Taglist, Project Information) and tool executables (Installation Center, Command line Tools).
- **Visu:** you will find the templates for the visualizations (GraphPic, Movicon, WinCC Flexible).

The Project Directory



Each Project has a similar sub directory structure and contains the following directories:

- **Backup:** if backups are created, they are stored in this directory.
- **Data:** contains all the database “*.ldf” and “*.mdf” files that form the databases that hold the Configuration, and the historical data
- **Engineering:** is the directory where the “Project engineering tool” creates its generated data. It also holds the Tag lists that are imported into BatchXpert.
- **PLC:** Contains the current Plc Project(s). You should keep your plc programming projects in this directory, so it always forms part of the project.
- **Reports:** is where all report templates that a user creates are stored. Report templates are used to create report printouts and for exporting tabular data.
- **Tools:** if you have custom tools that you use for your project, you should put the source code of them in this directory. This may include for example tools that send recipe lists to WinCC Comfort panels or Profinet monitoring applications, or similar custom tools.
- **Visu.:** This is where your HMI systems projects are located. This is the directory where your “VisXpert” Project is located.

Command line Tools.

BatchXpert comes with some command line tools that may help you accomplish common tasks related to BatchXpert. These tools can be found in:

C:\Program Files (x86)\BatchXpert\Tools

In this directory you can find many tools, such as “Command line tools” that you can use. All the Command line tools can be executed with the “-?” para meter to get an explanation of the command lines they support.

BatchXpert SDK

To facilitate the generation and execution of an automated project with BatchXpert, there is the "BatchXpert Software development Kit" also called "SDK". This package installs all the engineering tools and templates for both the controllers and the display systems.

The most recent version of the SDK can be obtained from the following link:

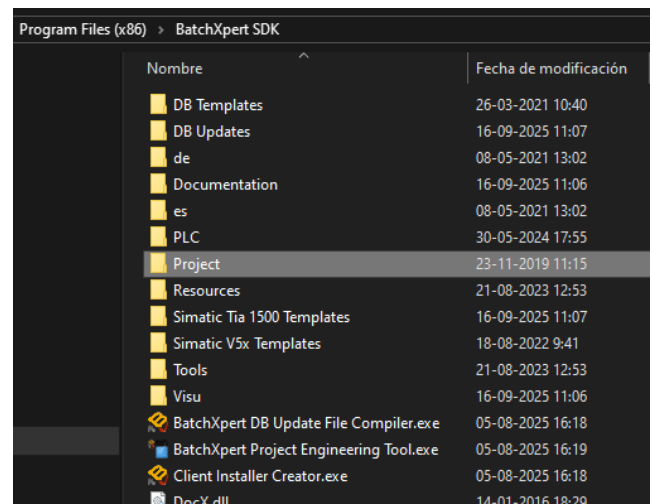
[MLogics Documentation – VisXpert and BatchXpert \(mlogics-automation.com\)](https://mlogics-automation.com)

It is recommended that you use the same version of the SDK with the same version of BatchXpert that is used in your project. For new projects, it is recommended to use the latest versions, to take advantage of system improvements.

Installation directory

Once installed, you will find the following folders inside the installation folder:

- **Visu:** This folder contains the project templates for all supported visualization systems of the BatchXpert system. The appropriate file should be copied to the engineering folder, extracted, and adjusted to suit the needs of the project.
- **PLC:** Contains the templates of the supported PLCs of the BatchXpert system. As with the visu, you need to copy it to the engineering folder, extract it, and adjust it.
- **Tools:** Contains useful engineering tools, such as Taglist Templates, a SQL Database Management Tool, and several other tools
- **Documentation:** Contains many Manuals, which are not included in the normal installation of BatchXpert, as they are intended for an Engineer and not for Operators.

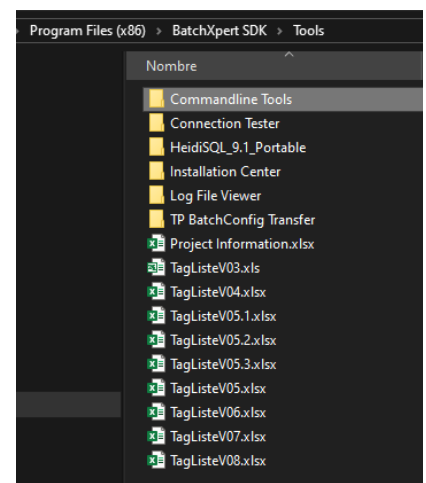


Tools

Several tools will also be installed, such as the "BatchXpert Project Engineering Tool" that allows importing, exporting, and generating data for PLCs and HMIs based on the system's current engineering database. This tool can generate alarm messages for the HMIs, data blocks for the PLC and others.

Here you can find more information about the [Project Engineering Tool](#).

In the "Tools" directory you can find other tools that can be helpful when engineering projects.



Project Engineering Tool

One of the main tools of the “BatchXpert SDK” is its “Project engineering tool.” This application incorporates many functionalities that help you to solve many engineering problems that you will face when implementing projects with BatchXpert.

The functions include:

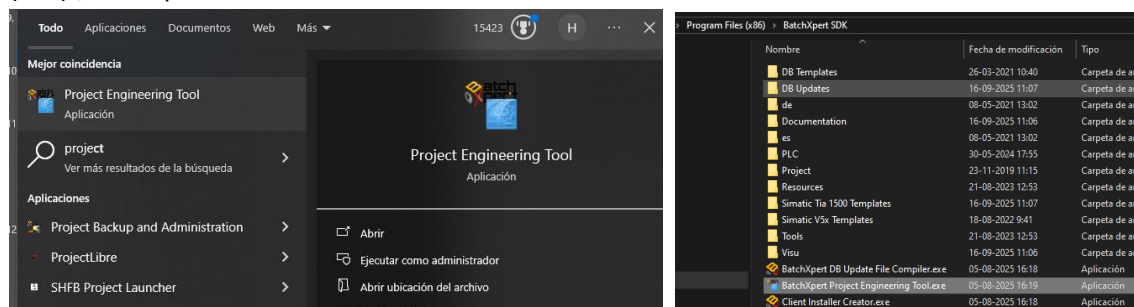
- Import Taglist
- Export and generate Taglist from current configuration.
- Generate IO Symbols for your PLC programming.
- Create IO Transfer blocks.
- Create Alarm groups.
- Create Tag variable files for many HMI systems, such as WinCC Comfort, Movicon, etc.



Where to find the “Project Engineering Tools”

After installation of the BatchXpert SDK. You can directly type project engineering tools into your windows and start menu to start deep project engineering tools. To be able to use most of the code generation functionality, you must open a project's pet project first. If you don't have a project yet you can create a new one [Creating a New Project](#), and then opening or starting it.

You can also find the tool inside the “BatchXpert SDK” installation directory which usually is: C:\Program Files (x86)\BatchXpert SDK



What data does the Project Engineering tools use

Most if not all code generation and other utilities that the project engineering tools provides, are based on the data configured in your configuration database. You can use the “Batch Configuration” to edit your database, create classes, phases, recipes etcetera.

All operations that the project engineering tool provides are based around this configuration database. There are tools to efficiently update this configuration, such as importing tag lists, or deleting all controller modules of a PLC.

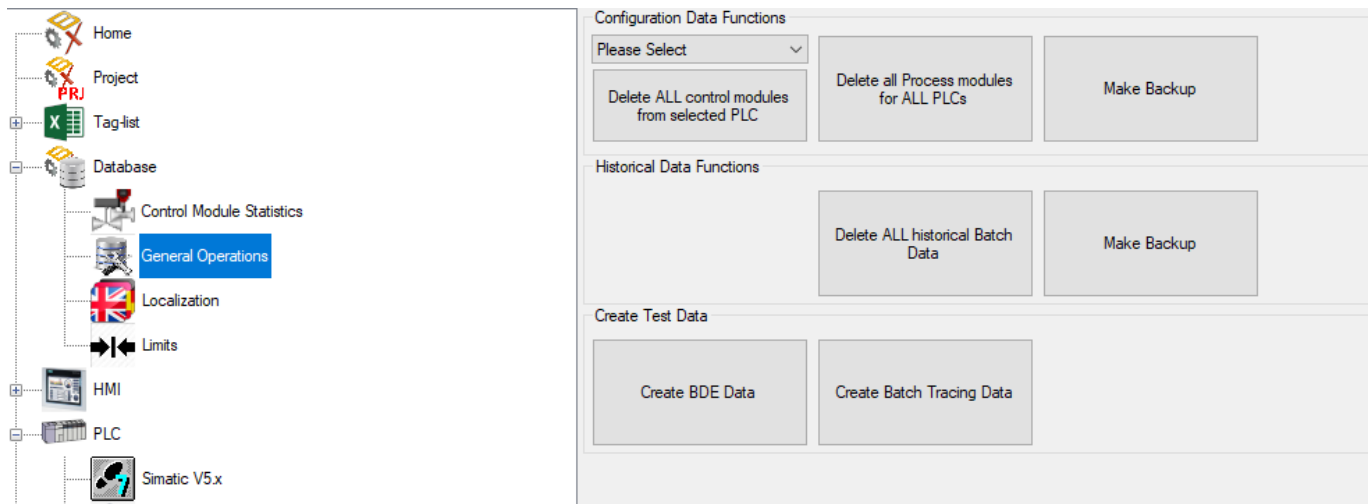
General Database Operations

One of the most useful functions can be found in the database general operations section of project engineering tools. There you can find useful actions that should only be activated during engineering of a project, since they

can delete all configured process modules, such as phases, recipes and classes, and they can also delete all configured control modules.

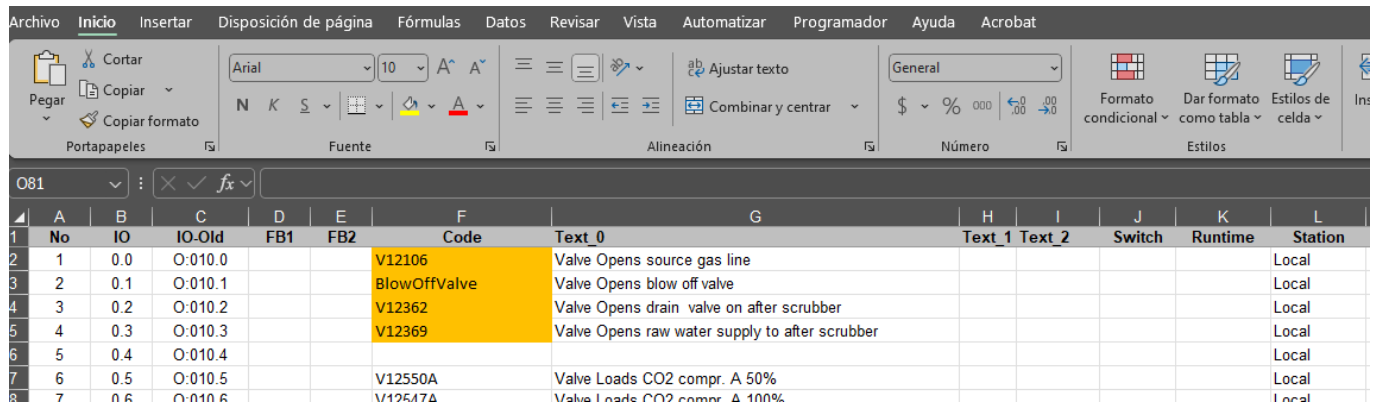
These functions are useful when starting a new project where you must delete all the existing configuration, or when you want to delete all control models of PLC, to import them again by using the tag list import function.

There's also an option to delete all the historical recorded data, which should always be done before starting commissioning on new project, so that all recorded historical data that you have recorded during testing it's not going to be deployed to the client. Of course you should never delete historical data on site.



Taglist and importing

A Taglist is an Excel file that contains the configuration of each of the control modules of a project. This Taglist can be imported, exported or an existing one can be updated. A Taglist allows you to efficiently modify existing control modules, or add new modules to a PLC. All Taglist belong to one single PLC and have one “Worksheet” for each Control module type. Each “Worksheet” has one column per property, and each row represents one single control module of a type.



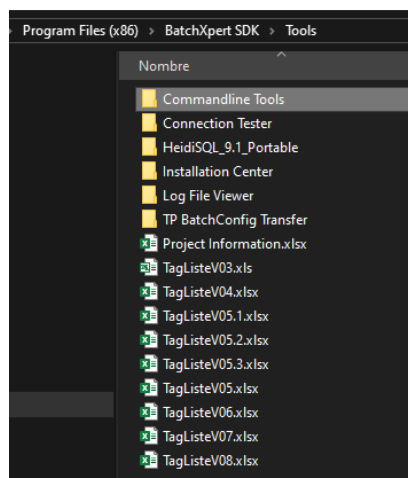
	A	B	C	D	E	F	G	H	I	J	K	L
	No	IO	IO-Old	FB1	FB2	Code	Text_0	Text_1	Text_2	Switch	Runtime	Station
2	1	0.0	O:010.0			V12106	Valve Opens source gas line					Local
3	2	0.1	O:010.1			BlowOffValve	Valve Opens blow off valve					Local
4	3	0.2	O:010.2			V12362	Valve Opens drain valve on after scrubber					Local
5	4	0.3	O:010.3			V12369	Valve Opens raw water supply to after scrubber					Local
6	5	0.4	O:010.4									Local
7	6	0.5	O:010.5			V12550A	Valve Loads CO2 compr. A 50%					Local
8	7	0.6	O:010.6			V12547A	Valve Loads CO2 compr. A 100%					Local

New Tag lists

And the template for Taglist is included in the BatchXpert SDK. You can find it in the following directory of your BatchXpert SDK installation:

D:\Program Files (x86)\BatchXpert SDK\Tools

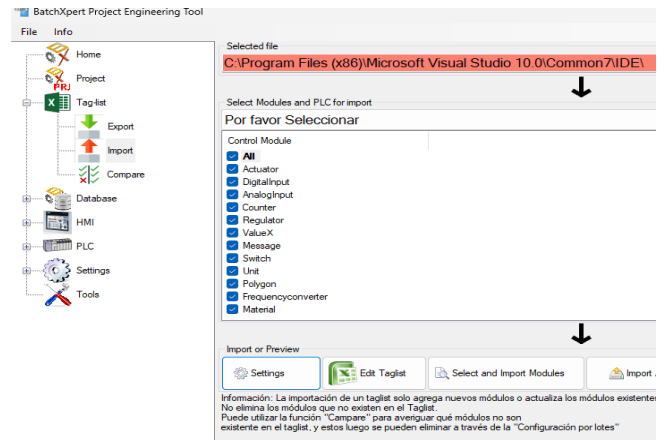
Alternatively, you can also create a new tag list, from an existing project by using the project engineering tools ([Taglist Export](#)). This allows you to create new tag lists in case they are lost or update existing tag lists in case they may not be up to date. In the case of an export the provided tag list will be updated with the current information stored in the database, but custom data is still retained in the existing Taglist. This way you can update existing lists without the fear of losing existing data.



Taglist import

When importing, the “Project engineering Tool” will go through each column of each worksheet and look up the data for specifically named columns. This means that you must follow the following rules when working with Taglist :[Rules to follow when working with tag lists](#)

To import a file, select the file to import, select the PLC where the data should be imported into, Check the modules types you want to import and then click “Import.” The import starts by creating a backup and placing it in the backup directory. After that It will import each row with the following rules:



- If a Control module with the same “Number” of a Type already exists, the information will be overwritten with the information from the Taglist.
- If no control module with the same “Number” of a Type already exists, a new module will be created with the information defined in the Taglist.
- No Module will ever be deleted from the database. So, if some modules are missing from the Taglist, they will NOT be deleted from the project. Since they do not appear in the Taglist, they will simply remain untouched.
- If you want to remove modules, this must always be done manually by using the “Batch Configuration.”

Taglist Export

Exporting works similarly to importing, except it transfers data from the project into the Taglist. It allows you to select an existing Taglist, which will be updated, instead of overwritten. This allows you to “Update” Taglist from the Taglist, without losing cell styling or additional columns. The export functionality will go through all project control modules, try to find a corresponding row in the excel Taglist, and update the corresponding columns, without removing or changing custom columns, that you might have added.

One of the first tasks when engineering and BatchXpert system, is the creation of a so-called tag list. A tech list is a Microsoft Excel file that contains definitions for all control modules of a single PLC. It defines the control modules number, IO assignment, names and descriptions. Checklists can easily be imported, exported and even be updated by using project engineering tools. For more information about how to import Tag lists: [Taglist import](#).

NOTE: Creating A tag list should always be the first step when starting a new batch expert project, since it forms the basis of all further work on the HMI and the PLC. Most importantly, the data that you define in Tag lists, will form the basis for further code generation particularly the generation of trans IO blocks.

General Structure

Taglists are designed to be easy to work with and contain one “Excel Work Sheet” per control module type, which defines all control models of a specific type. These control module specific work sheets contain one column for each metadata of a module, and one row per control module of the specific type.

No	IO	FB1	FB2	Code	Text_0	Text_1	Text_2
125	90.5	85.7		LT_AK_02	valve CIP lauter tun	Valvula CIP Cuba filtro	Ventil CIP Läuterbottich
126	90.6	86.0		LT_AK_03	valve cloudy wort to lauter tun	Válvula de mosto turbio a cuba de filtración	Ventil Trubwürze im Läuterbottich
127	94.0	86.1		LT_AK_04	valve weak wort to weak wort tank	Valvula de mosto debil hacia Tanque	Ventil Glatwasser zum Tank

One of the most important metadata information of the control modules is there I assignment generally done in the IO column, of each control module type.

Rules to follow when working with tag lists

Detect lists are designed to be extendable and easy to work with. You can add custom columns and even reorder existing columns, however you should follow the following rules when working with tag lists, to be able to import Tag lists using the project engineering tool.

- **The default Column names cannot be changed and must remain as they are.** If they are changed, the Engineering tool cannot identify the column, and the import will fail.
- **The first row of each “Worksheet” must contain the column names,** and all subsequent rows will be imported.
- “Empty” rows can exist. If the Import detects that there is no “Symbol” (also known as Tag name or Code) defined for this control module row, it will simply ignore it. Your Taglist is not required to be contiguous.
- You can reorder the Columns if the names stay the same.
- You can add custom Columns to your Taglist. These columns will simply be ignored. The columns can have a name if they do not collide with an existing columns name. This function is particularly useful to add an “Tests” column, so you can keep track of tested columns during IO testing.
- You can apply any “Cell Style,” font, size, border, or any other style you like. As said before, only the names of the first-row matter.
- The Taglist can be if you like. However, keep in mind that you will not be able to have 50.000 Actuators in your Plc.

The Unit Assignment Column of all Control Modules

All control modules allow you to assign them to a specific Unit. This assignment is used to automatically generate Alarm groups and for reporting errors to specific Units. All control modules can be assigned to a Unit, by inputting the unit number into the “Unit Assignment” column of a tag list.

K	L	M
ation	Station Address	Unit Assignment
		1
		1
		1

To be able to assign a control module to multiple units, you can write the values as a list, separated by a Coma.

L	M
ation Address	Unit Assignment
	2,3
	2,3
	4

In the Above example, the control module will be assigned to both, unit 2 and unit 3 of the PLC of the control module.

All Unit numbers in the Unit Assignment field, always refer to the same PLC where the Control module lives in. However, BatchXpert supports assignments of Control modules to Units in different PLC's, for reporting purposes, but currently you cannot use the Tag list to make these assignments to Units in different PLCs than the Module. To assign Modules to Units in different PLCs, you must assign them from the "Batch configuration".

Class definitions of Units

The tag list contains an excel worksheet for units, that allows you to define the corresponding processing class that is defined in a batch configuration tools. In this column you can input the class identifier that you can obtain from the batch configuration utility, or the literal name of the class that you want your unit to be assigned to. If any class name is ambiguous, you can optionally add the Batch area name separated by period.

Class identified by its Class name:

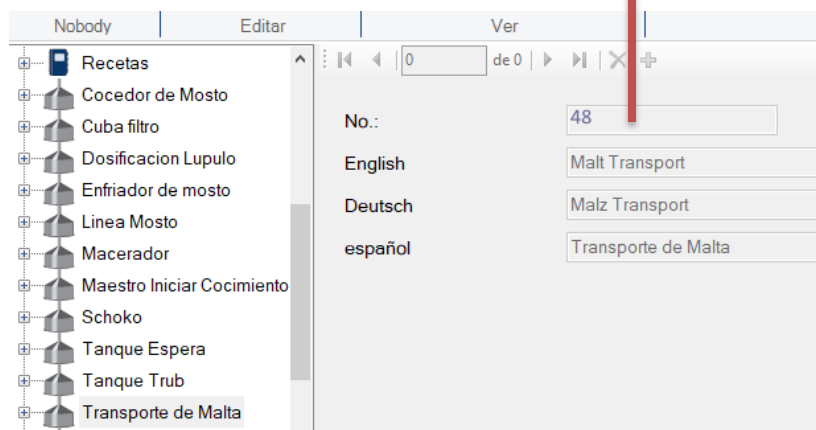
No	Code	Class	Text_0
28	UT59	Unittank	Tanque CC UT59
29	UT60	Unittank	Tanque CC UT60
30	DT11	Tanque Druck	TK Cerveza filtrado DT11
31	DT12	Tanque Druck	TK Cerveza filtrado DT12

Class identified by its Batch Area and Class Name:

41	WLKS2	Sala de Cocimiento KS2.Linea Mosto	Linea Mosto KS2
42	WLKS1	Sala de Cocimiento KS1.Linea Mosto	Linea Mosto KS1

Class Identified by its Class ID

No	Code	Class	Text_0
1	Unit [2/1]		44 Start Master
2	Unit [2/2]		48 Malt Transport

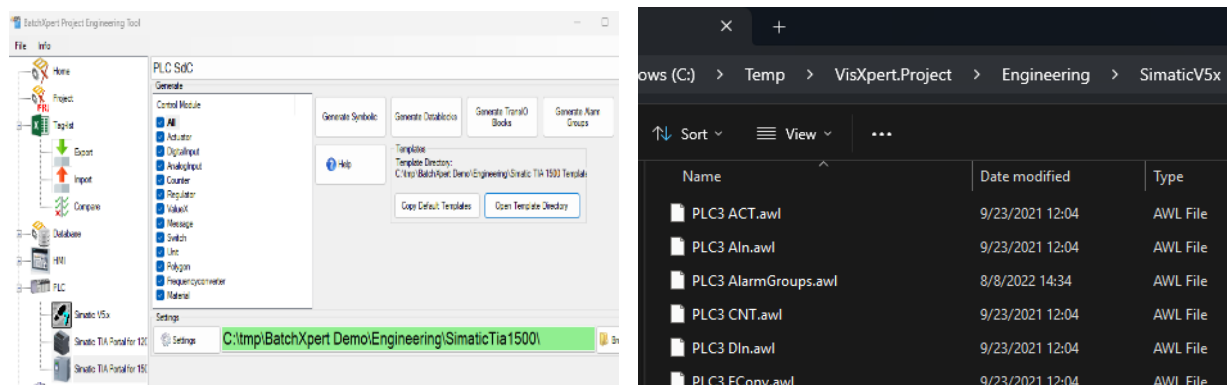


PLC code generation

This option allows you to generate a variety of code blocks, which can be imported into either Simatic manager or Tia Portal. The data block and function block code generation is based around applying data to templates. You can customize these templates and Project engineering tool will use these templates for their code generation. The generated files can then be imported easily into your PLC programming software, usually tia-portal.

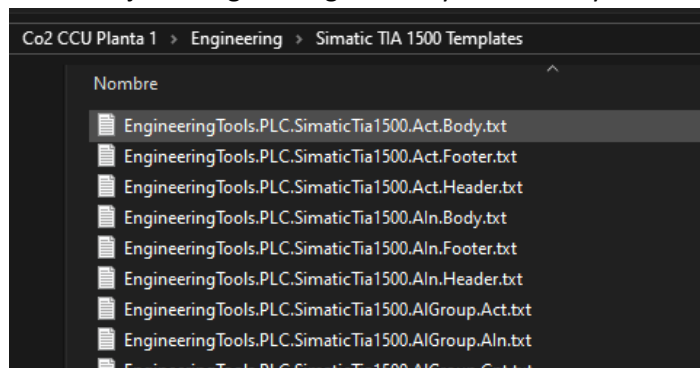
All generated code files are being stored in the engineering subdirectory of your project. From this directory you can import them into your engineering software. The process of importing and compilation of these files depends on the type of software used for your PLC programming.

You should read [TIA-Portal](#) on a guide on how to import files into “TIA-Portal”.



Customizing Templates

The tool supports custom “Plc code templates” that are being used to generate the data blocks with. This allows you to customize the code being generated. The template is stored in the “Engineering” directory of your Project. If no templates exist, the system will use the system default templates, which should be good for about 99% of all cases. If you want to customize them, you must copy the default templates by clicking “Copy default templates” to the Project’s Engineering directory. After that you can customize them.



Generate symbolic.

This will generate IO and block Symbolic files that can be imported into either Simatic Manager or Tia Portal. It generates all symbolics for Inputs, Outputs of all used control modules. It allows you to assign symbols to all your Inputs and outputs.

You should read [TIA-Portal](#) for recommendations about import settings for “TIA Portal”

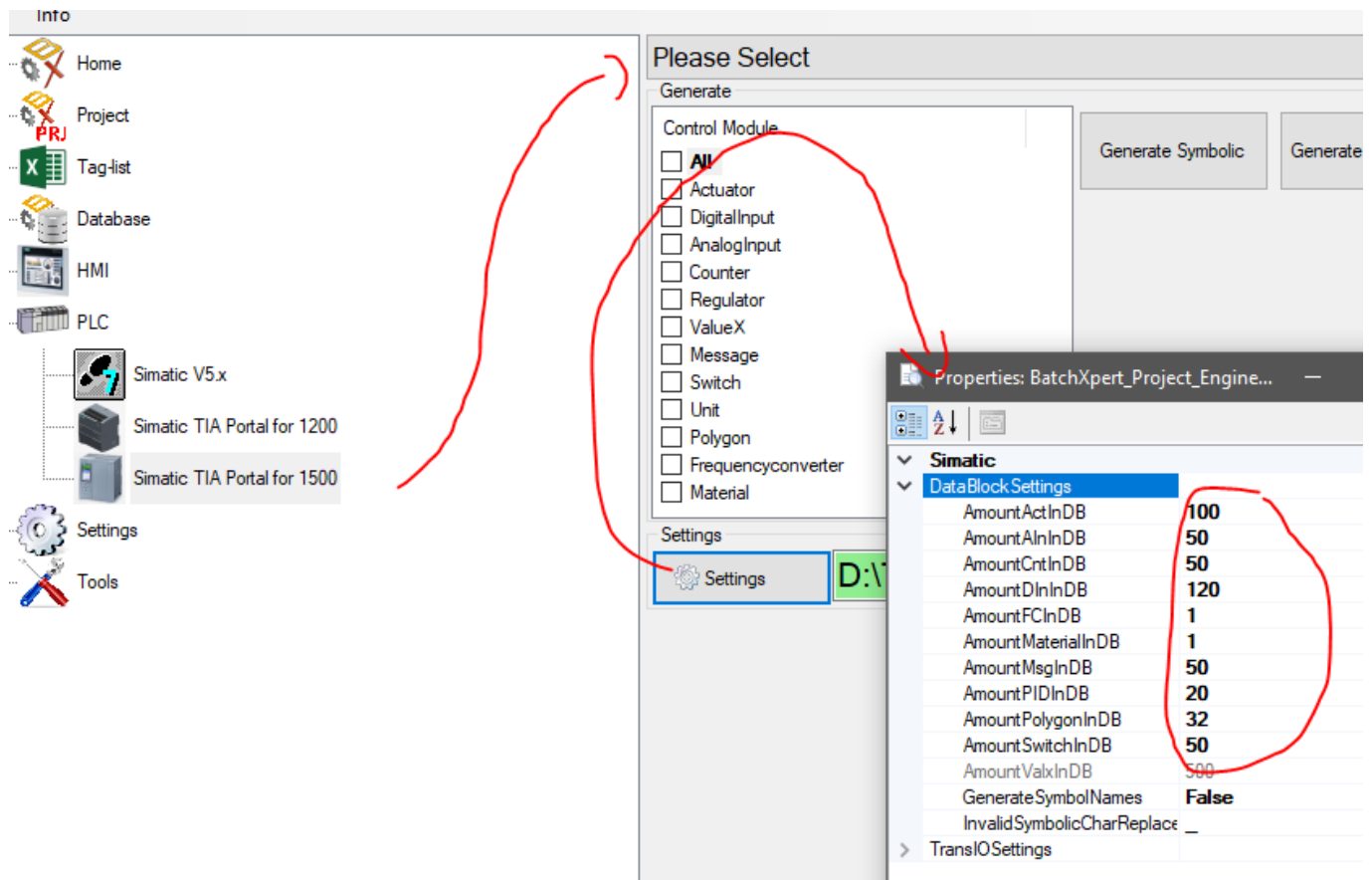
Generate Data block.

This will create the initial Control module data blocks with the configured module description from the database. This is usually done only at the beginning of a project, since later updates of the data blocks are non-trivial since they require a data block download, which should not be performed.

As a rule, you should use this functionality only at the beginning of a project, and then update the data blocks manually, or carefully plan a data block download.

When generating data blocks for control modules you can set the amount of control models to be created in these data blocks in the settings of your project engineering tool. This setting basically allows you to set the number of spare modules to be created in your data blocks. You should evaluate a tradeoff between memory consumption of your control module data blocks against already existing spares that can be easily used in your PLC application without downloading the control module data block.

D data block generator will always create all your configured control modules in your current project and then will append spare modules until reaching is set number of modules to create. You should choose these values according to your project needs and available memory in your PLC.



Generate TransIO Blocks

This generates all IO assignment blocks, which will link your control modules to the configured Inputs and outputs of your PLC. How this works is described in more detail in the [“Trans xx” blocks for IO signal transfer](#) section of this Manual.

Generate Alarm Groups

BatchXpert allows you to assign control modules to units. Keep in mind that you can assign one control module to multiple units, in which case it belongs to all the configured units. Based on this information, this option will automatically generate the function block that assigns all these control modules to the “Unit Alarm Group,” so the alarm group can be used in the HMI system.

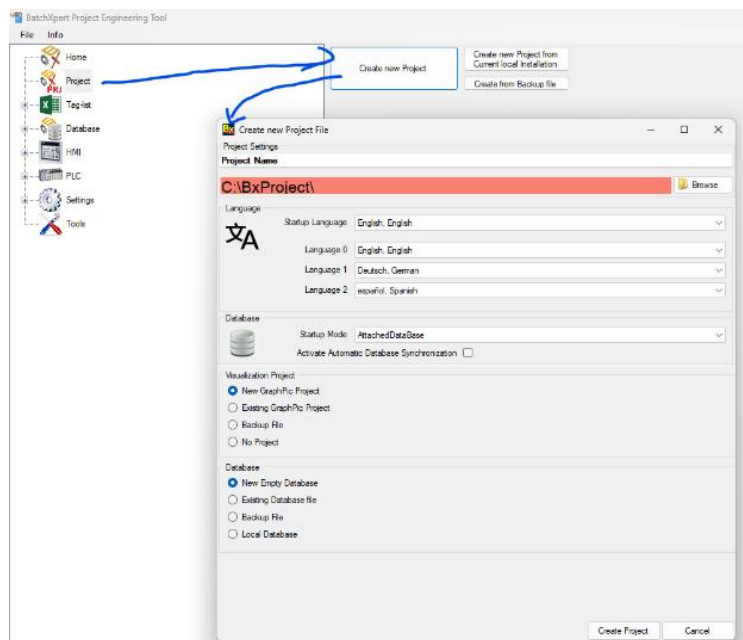
Creating a New Project

When starting the implementation of a BatchXpert application, you first must create a new Project. This can be done easily by using the “Project Engineering Tool.”

When opening the Engineering tool, you may encounter an error related to a nonexistent server, project, or database. This is because you have not started any BatchXpert project yet, so there is no database yet.

Create new projects.

“Create new Project” is the easiest way to create a new Project. The other two options “Create from backup file” and “Create from current installation” are meant for legacy project conversion and not relevant for new projects.



When creating a new project, you can create it based on existing databases and Hmi projects, use default Hmi and database or none. You should choose the default “Empty” options. This will create a project with a default database and HMI Project for you. The default database already contains some example configurations which you can easily remove if needed.

Clear existing database (Optional)

If you want to remove all configurations from a database, for example to remove the whole example configuration, you can use the “Project Engineering Tool” for this. In the “Database/Database operations” you will find options to clear several parts or all the configuration from your database.

There is also an option to remove all historical data, which of course should never be done on site, but always be done before commissioning begins.

Plc Template

After creating a project, you must prepare your plc application. The BatchXpert SDK distinct options of Projects for you in the “C:\Program Files (x86)\BatchXpert SDK\PLC” directory.

You must choose your project based on the Project architecture and programming environment you want to use. There are projects for:

- S7-300/400 in Simatic Manager
- S7-300/400 in Tia Portal
- S7-1500 in Tia Portal
- S7-1200 in Tia Portal

After selecting an appropriate solution for you, you must copy and extract the template folder into the “PLC” directory of your project.

Hmi Template

This step is only necessary if you are using other HMI systems apart from the integrated VisXpert SCADA. If you are using for example a WinCC Comfort panel, you must extract the appropriate project from the HMI Template folder located in:

“C:\Program Files (x86)\BatchXpert SDK\Visu.”

This project should also be copied into a sub directory of your “Visu” directory in your Project directory.

If you are only using the integrated VisXpert SCADA, this is already done by the project creation itself and no further action is required.

Database

The database lives in the “Data” directory inside your project directory. BatchXpert uses “Attached” Sql server databases, which means that the files will be attached to the selected database server when the project is opened. If no project is open, there is no database since no file is attached to the server. When BatchXpert is shut down, the database will be automatically detached from the server.

This methodology makes it easier to create backups from projects, since all the data of a project is always contained in the project directory, including the database, plc, and auxiliary information.

For example, if you need to restore the database manually, you can simply shut down BatchXpert, replace the “Data” directory of your project from a backup and start it up again with the database from the backup.

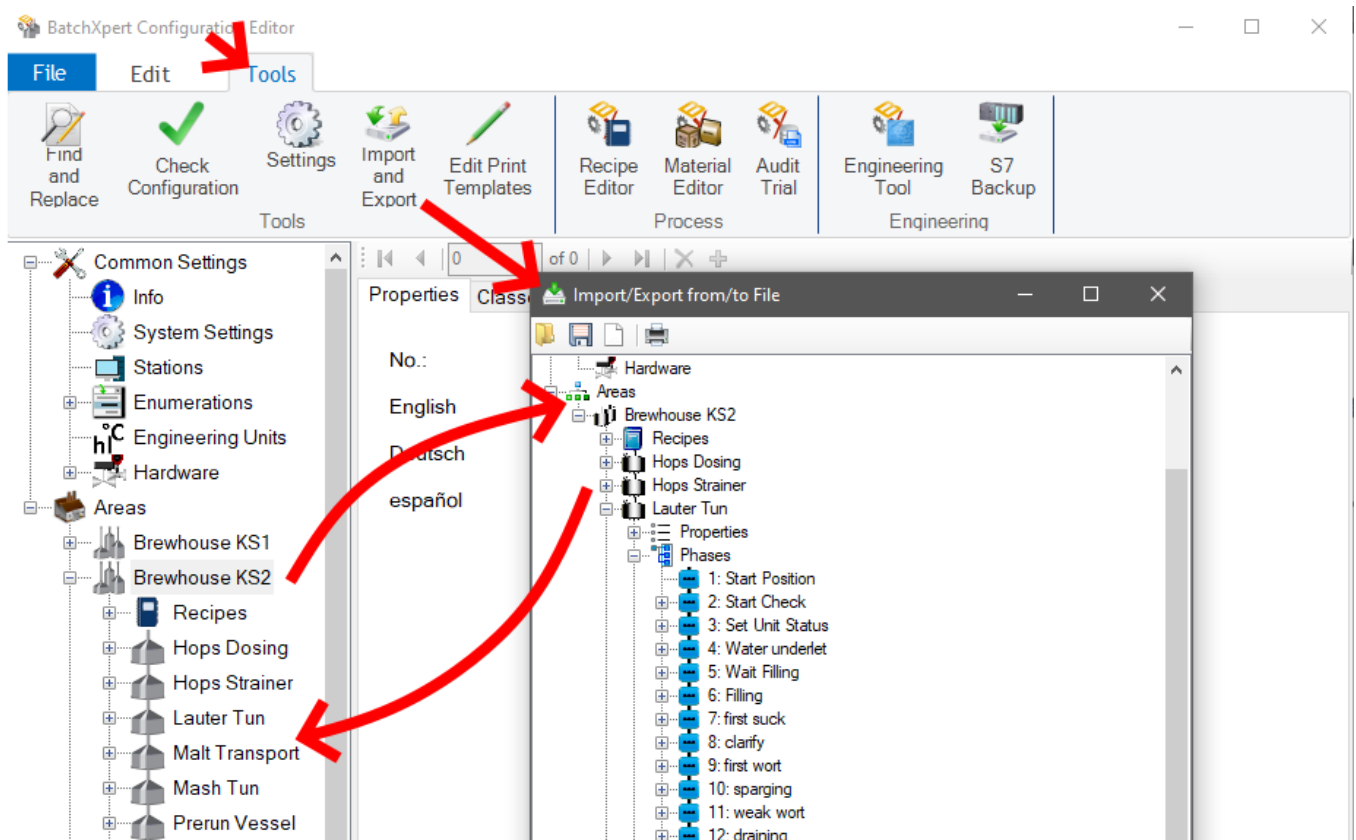
Copy data from another Project's database.

After creating a new project, you may want to copy data from an existing project into this project's batch configuration. This works by utilizing the "Import/Export" functionality of the Batch Configuration utility of BatchXpert.

In the Batch Configuration, you can create an "Import Export" file onto which you can "Drag" batch areas that you want to export. After saving these can be loaded into another project by opening and "Dragging" them back into the main configuration window.

You should always export the whole Batch Area, but you can import only fractions of this into your project. You can import only a single phase, parameter, or whole classes and even batch areas.

Overview



Procedure

To copy data from one project into another, you should follow the following procedure:

1. Open your Source Project and open the Batch Configuration
2. Open the Import/Export window as shown above and click "New" in this window to create a new file.
3. Drag all batch Areas you wish to export into this window. You need to be logged in to BatchXpert with your username to do this. Info. If you export large Batch Areas, this may take up to 30 seconds and the application may appear "hung" for a short amount of time.
4. Save the Import/Export file to disk, using its save functionality.
5. Close your source BatchXpert project.

6. Open your destination BatchXpert project and open Batch Configuration
7. Open the Import/Export window and open the file that you just saved to disk.
8. Drag the parts of the exported file into your configuration that you wish to import.

Tips for handling Projects

Keep the plc program in project directory.

In your project directory you should have a sub directory called “PLC” where you should always maintain your current PLC application project. In fact, you should leave it in the project directory and work directly from there.

You should not create separate “PLC Project Backups,” but rather “Full Project Backups” of your project directory that includes all the project’s data, not just the plc or everything except the plc project.

This way you will always have a consistent project and the plc project, and all other engineering material is always included in all manual and automatic backups.

Keep tag lists in the project directory.

For tag lists we recommend the same procedure as for plc programs, but inside the “Engineering” sub directory of your project.

Create “Project information” files.

We recommend creating “Project Information” files for each project where you can register all installed versions, Network addresses and other additional information of your project, including Passwords.

If you also record passwords, you must treat this archive as “Confidential” since it contains sensitive information.

Keep Tools source code in the project directory.

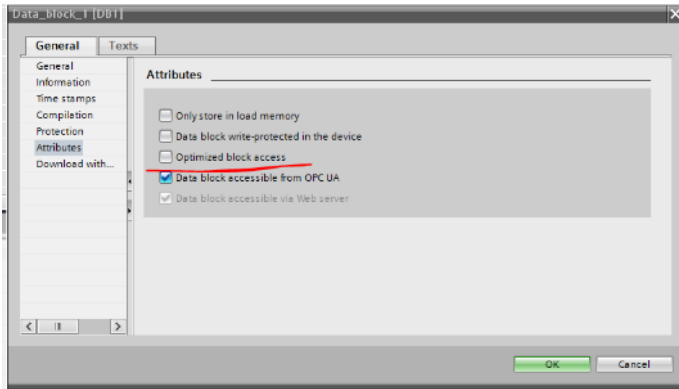
For the same reason as stated above, you should always keep the full source code of tools you have written for a project in the “Tools” directory of your projects. This way you will always have the exact source code of your tools available directly in the project directory

Special Considerations for Simatic S7-1500

The S7-1500 and S7-1200 have special requirements for setting up Communications with BatchXpert which you can review in detail here [Connecting to an S7-1500 and S7-1200 PLC](#)

The settings described in the Chapter are needed for running BatchXpert in the PLC, but you also must consider the special connection settings.

Only non-optimized blocks can be read from any HMI.



The BatchXpert and VisXpert systems only support reading of “non-optimized” data blocks. All data blocks that are exchanged with BatchXpert cannot have the “Optimized” attribute. You can still have optimized blocks; those just cannot be read by the HMI. Please keep in mind that Function Blocks FB’s also have these settings.

You can find this setting in the Datab block and function block properties under “Attributes.

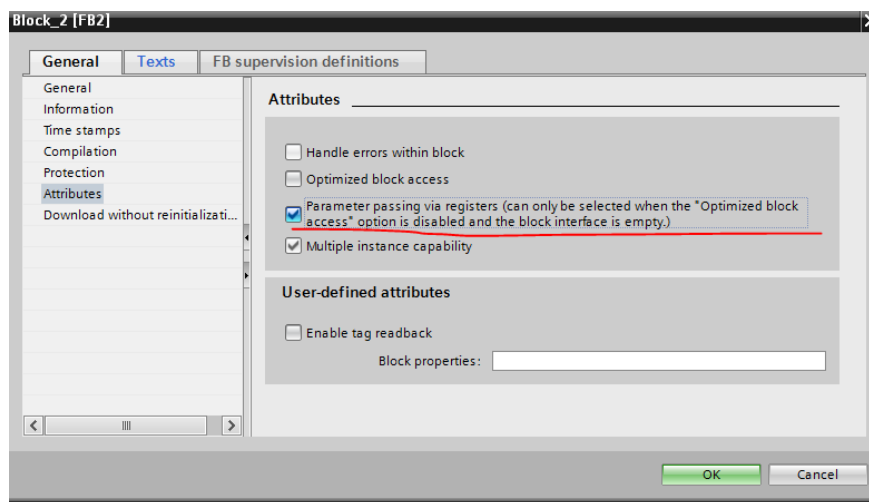
Activate “Parameter Passing via registers” for Unit Function Blocks

Only for Unit Function blocks you should activate the “Parameter Passing via registers” option in the function blocks property/attributes.

Tia Portal requires this setting to enable you to call the unit function blocks via “UC” (unconditional call) instructions without specifying an Instance data block to the function block.

If you do not activate this setting, you must specify an instance data block for the Unit function and call it via the “Call” instruction. This is possible as can be done, even though you will need to create additional instance data blocks, which are otherwise not necessary.

We recommend activating this setting, but you can also use “standard Simatic Step 7 calling conventions”.



Remanent Memory

Remanent memory is an important consideration for S7-1500 PLC's, as they have comparatively a small remanent memory. Even though they may have megabytes of data ram, only about 200 kb of that is remanent, which for most applications is just not enough.

We recommend the use of the “Battery buffered Power supply” “6ES7505-0RB00-0AB0”. This power supply converts all Data RAM into remanent Data ram, thus eliminating the remanent data restriction of S7-1500 PLC's.

As an Alternative you can use the “Persistence” functionality of BatchXpert, which utilizes the Memory card. This results in considerable “Wear” on flash memory. The “Persistence” functionality is optimized for this but still results in about “20-year lifetime” of the flash memory. The Persistence functionality will save modification at most every 20 minutes, so the “Resolution” is limited to this interval.

A typical Simatic Memory Card will allow 500.000 writes per cell, which at most every 20 minutes will result in about 20 years of Memory Card life.

See [PLC retentive data](#) for more information.

Memory Card

All Simatic PLCs require the use of a memory card, so the 1500 series also require a memory card to function and will not work without an appropriate memory card. The Memory Card holds non-runtime relevant data from the downloaded project and should be about 12x the code AND data size of your project.

If you have a project that uses 200kb of Code ram and 300kb of data ram, you should use at least a memory card with 6MB of Memory.

We recommend a 12MB card since this has enough memory for most projects.

Objects	Load memory	Code work-memory	Data work-memory	Retain memory
	44 %	34 %	12 %	9 %
Total:	12 MB	614400 bytes	2621440 bytes	2621440 bytes
Used:	5535715 bytes	209780 bytes	324602 bytes	234450 bytes

Figure 3 Example Project with about 525 kB of RAM usage and 5.5 MB of memory card usage.

Test DB

Test_DB call can be changed with ATTR_DB Calls. They are similar. Note that the DB_Number is now an UInt, which makes sense, the DB_Length is now an UDINT and the ATTRIB a Byte as it contains more attributes.

```
CALL ATTR_DB
REQ    :=TRUE
DB_NUMBER :=#UnitDBNo
RET_VAL :=#SFC_RetVal
DB_LENGTH :=#SFC_Length
ATTRIB :=#SFC_WriteProt
```

```
Check Datablock
CALL ATTR_DB
REQ    :=TRUE
DB_NUMBER :=#UnitDBNo
RET_VAL :=#SFC_RetVal
DB_LENGTH :=#SFC_Length
ATTRIB :=#SFC_WriteProt
SPBI    nobb
L        DINT#0
T        "By UnitDB No" DB ChkMe
```

Auf DB calls

Especially important. From Siemens Documentation

The data block register DB is set to "0" after each access to the data block with the specification of a fully qualified address (%DB10.DBW10, "MyDB.Component", for example). A subsequent partially qualified access leads to an error during compiling.

What that means is that you MUST use an AUF before each indirect Transfer or Load instruction! Yes, every time. On old Controllers the DB "Stayed" open. Not anymore

Webserver

The new generation of PLC's integrate a Webserver which allows you to gather diagnostics data, create online backups and other maintenance tasks. Some BatchXpert utilities, especially the Backup utilities, utilize this Web server. It is thus recommended that you activate the Web Server. Please see [Activate Web Server](#) for more information.

Connecting to a S7-1500 or S7-1200 PLC

The S7-1200 and S7-1500 PLC series require some settings to be adjusted and considered when setting up communication with the BatchXpert system.

Since 2023, Siemens added encryption and certificates to the PLC Communication channels. Since BatchXpert utilizes the “traditional” S7-Connections to exchange data with the Controllers, some settings must be adjusted in the PLC to communicate with the plc.

If you have problems communicating with an PLC, please check the following settings and review the [Diagnosing PLC communication problems](#)

Ping will still work

Please keep in mind, that the S7-1200/1500 series of PLC’s incorporate more advanced communication and protection mechanisms than the S7-300/400 did. This means that you may be able to “Ping” an PLC, but any communication request will fail. It may even happen that you can connect to the PLC, but every read and write to any of the data blocks will fail, because the Communication channel does not have “[Set “Full Access” to HMI applications](#)”.

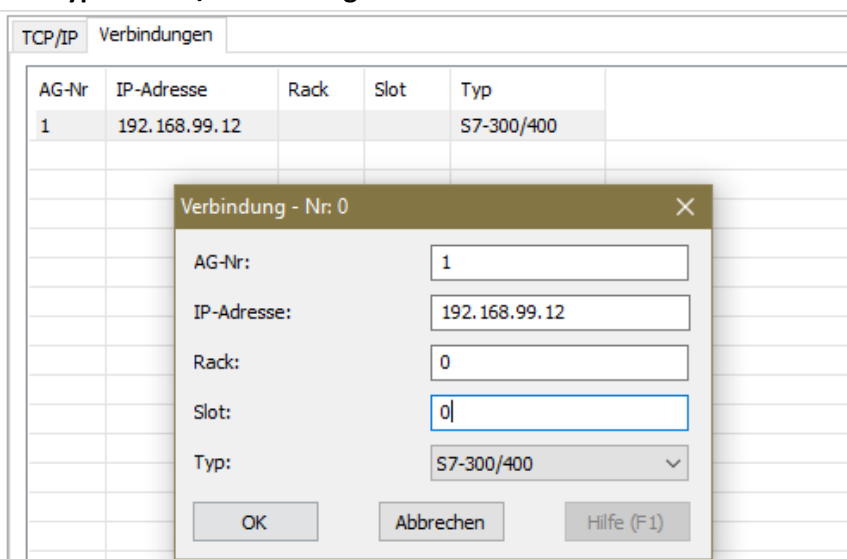
This manifests itself in the situation where you might have communication for one second, but as soon as BatchXpert tries to read data blocks, you get errors.

As mentioned before, in all these situations, a Ping will still work properly, even though communication is restricted or blocked by the PLC. This means that an “PING” is NOT a definitive check for communication with the PLC, but rather a check for general IP reachability from our computer.

Rack and slot should both be 0

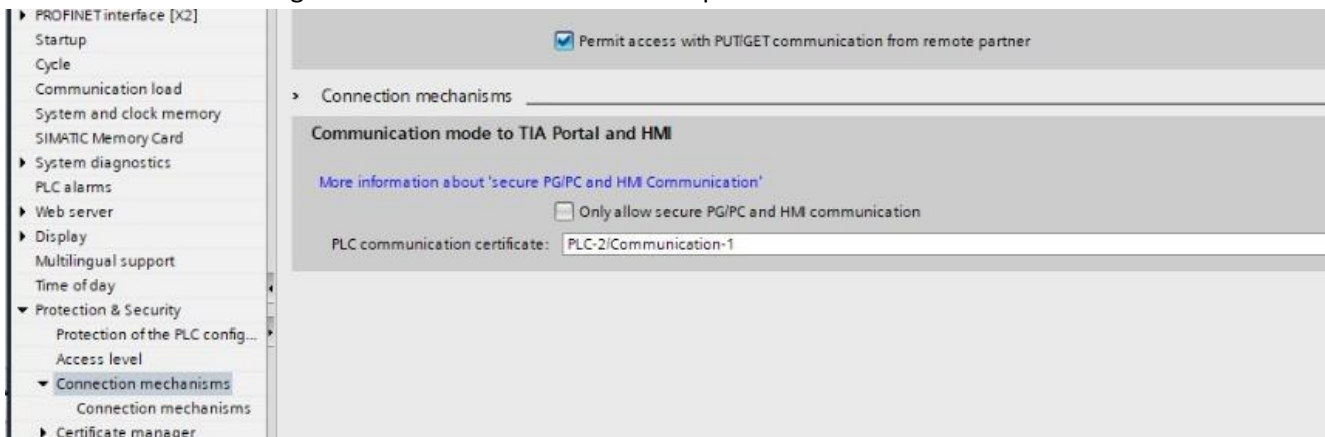
The traditional S7-Connection requires the definition of a Rack and Slot on the PLC to be used for data exchange. For S7-1500/S7-1200 this does not apply anymore, and thus the Rack and Slot should always be 0 in the communication settings of the VisXpert driver. You should not set this to any other value, since the PLC might reject communication and not respond to Communication requests.

The Typ “S7-300/400” setting does not matter.



Allow “Get/Put”

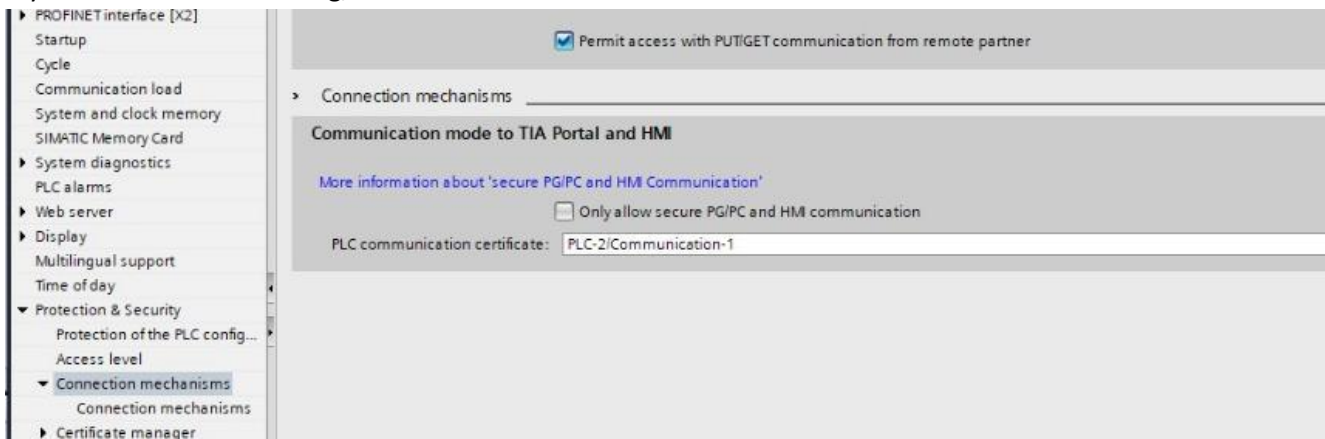
The S7-1500 by default are blocking “Get/Put” connection, which is what S7-connections use for communication. This can impact your ability to communicate with your plc via HMI devices and other PLC via any S7-Connection. We recommend allowing these connections at all S7-1500 plc’s.



Deactivate “Only allow secure PG/PC and HMI communication

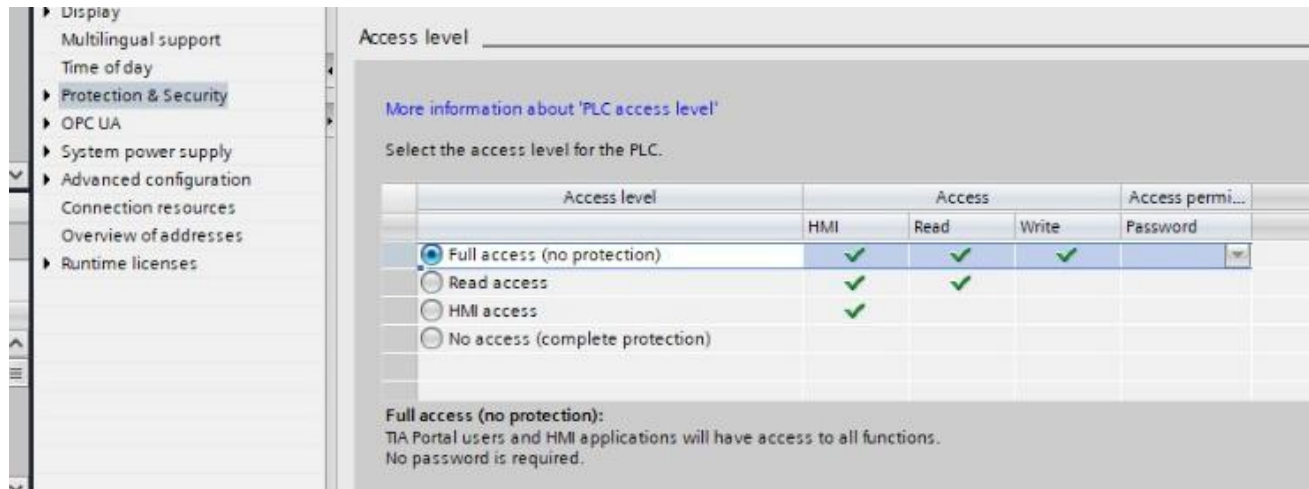
Since firmware version 3.0 of the S7-1500 controllers, the option to only allow “Secure” communication to HMI systems is active by default. This settings means that you MUST exchange certificates between the HMI and the PLC to communicate with each other. This effectively deactivates the “Traditional” S7-connection mechanism.

If you deactivate this setting, the PLC behaves as bevor Firmware version 3.0



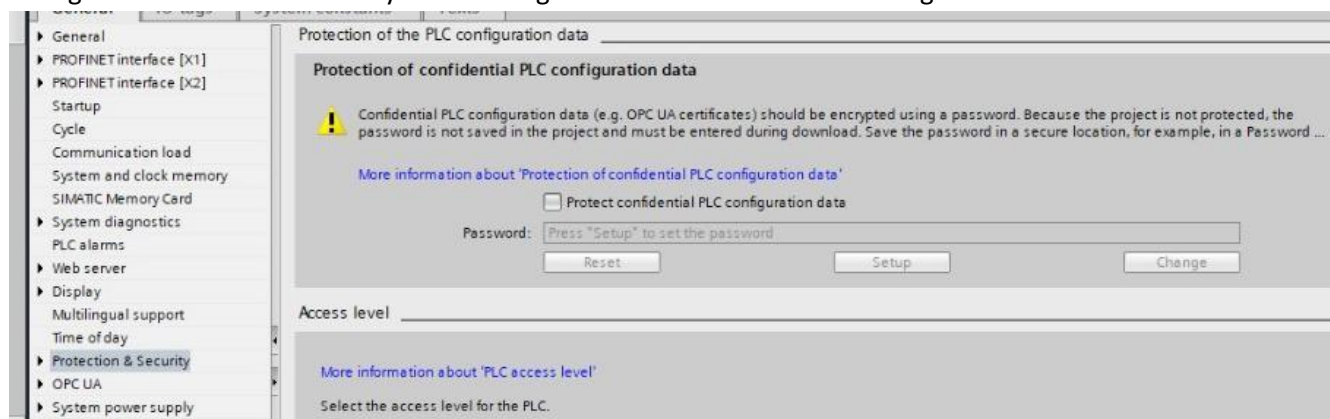
Set “Full Access” to HMI applications

This setting limits the actions that an HMI application can do in the PLC. You should set the access to at least “HMI Access” or even better “Full Access” to allow HMI applications to exchange data with the controller. If you do not set this access level, the PLC will reject all communication requests.



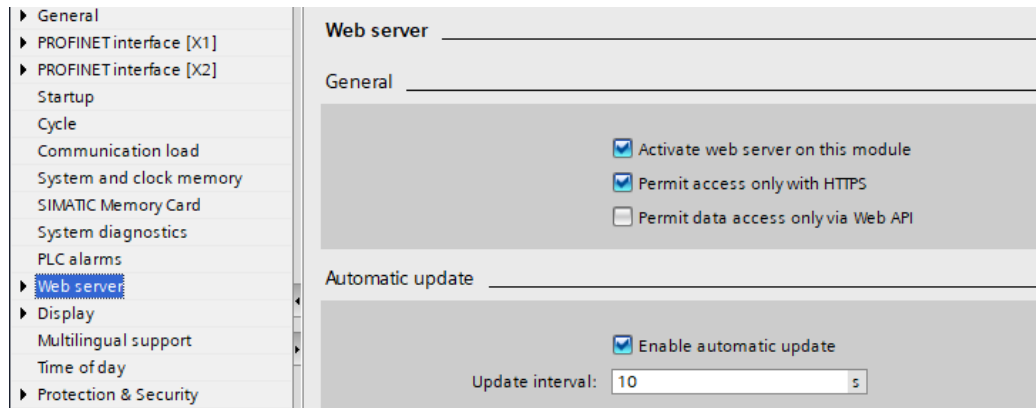
Deactivate “Protect of confidential PLC configuration data

This setting is not directly related to communication, but we still recommend that you do not encrypt the configuration data in the PLC. By deactivating “Protect confidential PLC configuration data”

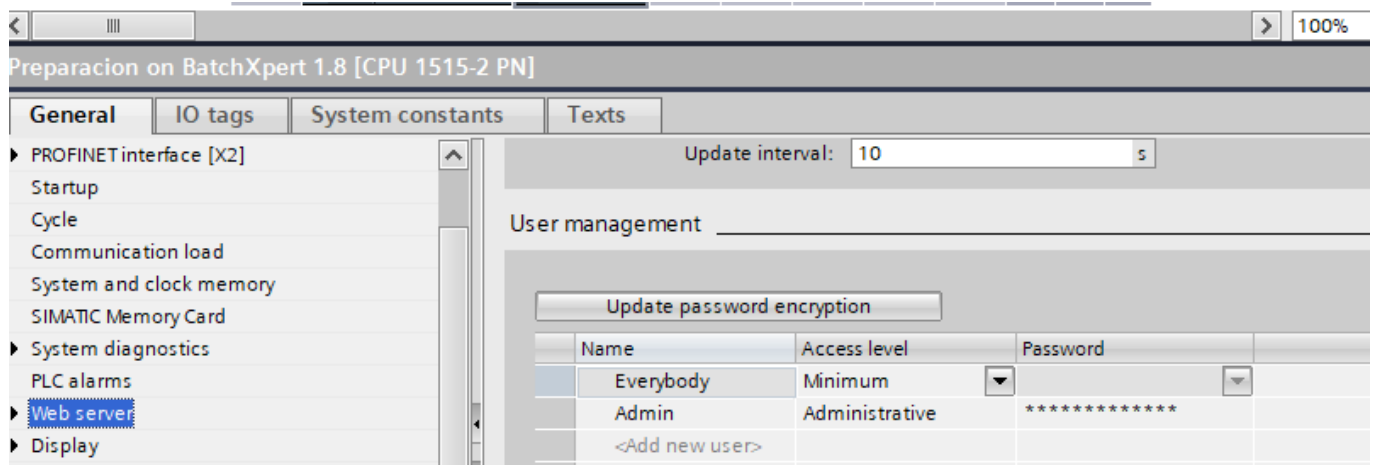


Activate Web Server

The new generation of PLC's integrate a Webserver which allows you to gather diagnostics data, create online backups and other maintenance tasks. Some BatchXpert utilities, especially the Backup utilities, utilize this Web server. The Web Server allows the BatchXpert Backup utility to connect and download online Backups of your PLC program.



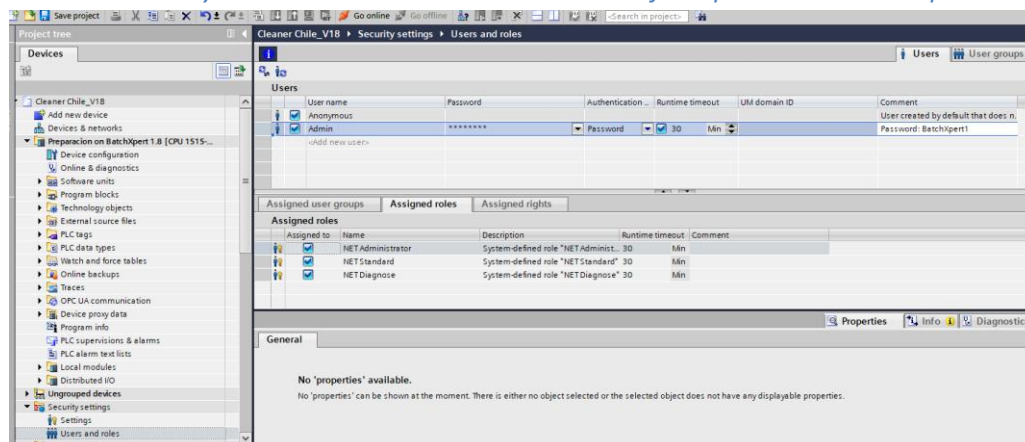
You should always create an "Admin" user with the default password "BatchXpert1"



Assigning User Rights to your Users

As for TIA Portal V19 and firmware version 3, the plc allows for the creation of different users and access rights. Please Create your users and assign access rights to them so that you can use these users with our backup utilities.

You should always create an "Admin" user with the default password "BatchXpert1"



Special Considerations for Simatic S7-300/400

BatchXpert was originally developed to target the Simatic S7-300/400 platform and thus is compatible with this platform. However, you must consider the following:

Discontinued

“The S7-300 CPUs and associated FEPROM and RAM memory cards MC 951 will be finally discontinued, following a 10-year phase-out period, as per October 1, 2020. With this discontinuation, these CPUs and memory cards will no longer be available as a spare part and can no longer be repaired. State-of-the-art successors are available for all CPUs.”

This Press Release from Siemens means that these PLCs should not be used for new projects anymore and an alternative should be used.

This statement means that the S7-300 series PLCs should not be used for new projects as of 2020 and will not be available anymore for purchase as of 2030.

The S7-400 series is not affected by this statement, as it continues to be supported by Siemens, and is NOT! Discontinued.

Ram amount

The most important requirement for S7-300 and S7-400 PLCs is the available RAM. The amount of RAM needed from the PLC-Frame is strongly dependent on the number of Control Modules that are used in the Application. To reduce the Memory footprint of the PLC-Frame, one can simply reduce the amount of use Control Modules and adjust them to fit the needs of the actual automation.

Memory Card

For the S7-300 platform from Siemens you always require a Memory card that you must consider in your project engineering. The S7-300 platform from Vipa does not require any memory card.

Simulation of PLC

While engineering, you will need to simulate the Program that you are writing for testing and debugging of your Process logic. To simulate an Simatic PLC, exist different software solutions, that depend on the PLC series that you are developing for. Of course, if it is possible, you can always simulate with a real plc, if this is available, however since this is not the case in most cases, a software simulation PLC is recommended.

PLCSim

Siemens develops an Plc Simulation application, for its S7-300, 400, 1200 and 1500 series, that it calls "PLCSim". This application must be differentiated from "PLCSim Advanced", which is an entirely different product, described in detail below.

The "Normal PLCSim" has no network functionality whatsoever. It may appear that it has, but this is trick that the Simatic infrastructure does. If you turn on PLCSim, you get the message that All PLC connection will only work with PLCSim from now on, this is because Simatic enters an "PLCSim" mode from where it uses "Named Pipes" to communicate with their integrated PLCSim. Simatic will show you an IP-Adress from the PLCSim but, it is never used.

This means that only HMI systems that use the integrated Simatic Driver will ever be able to communicate with PLCSim. No Intouch, Iltis, Botec, or any other HMI system can ever communicate with PLCSim via TCP/IP.

"PLCSim" can NOT be used for simulating PLC applications in BatchXpert.

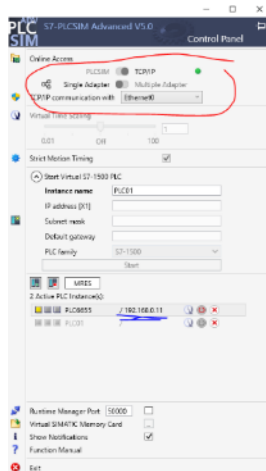
S7-300/400 series

There are several Software PLC solutions that can be used with BatchXpert, that easily allow you to simulate a Plc application on your computer. Usually these simulate one single PLC per computer, which means that you will need to have multiple computers, or multiple virtual machines to be able to simulate multiple Plcs.

- IBH Softec: Simulation PLC
(www.ibhsoftec.com/epages/63444704.sf/en_GB/?ObjectPath=/Shops/63444704/Products/1302)
- ABC-IT: ABC X-CPU-4 w57 (www.abcit.de/en/abcprodukte/abc-x-cpu-4-w57/)
- NetToPlcSim: www.nettoplcsim.sourceforge.net/index.html
- Simatic WinAC

We recommend the "IBH Softec" solution, since this is a cost-effective simulation plc that works well with BatchXpert and allows you to easily simulate all processes.

S7-1500 Series



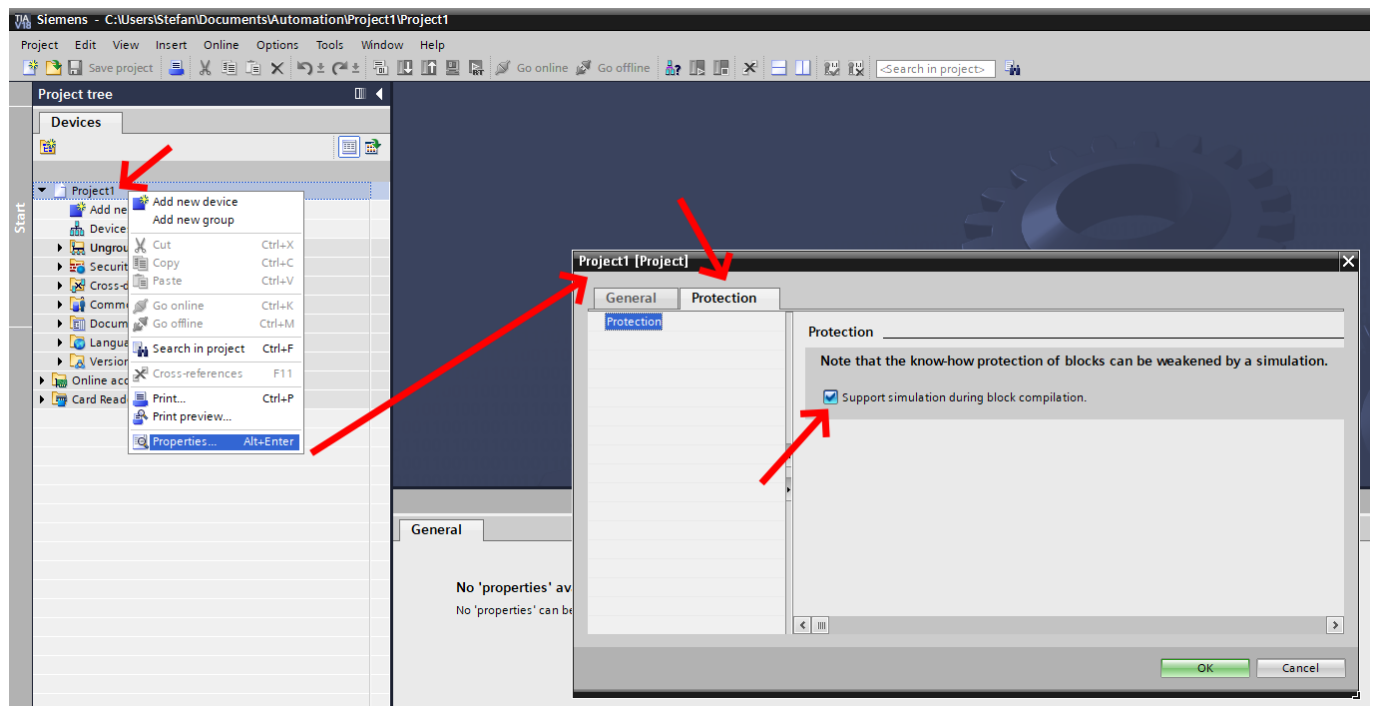
For simulation of S7-1500 series Plc, we recommend “PLCSim Advanced”, which is an entirely different product to the “PLCSim” mentioned above. The “PLCSim Advanced” allows you to simulate all aspects of an PLC, including networking functionality and allows you to define a unique IP-address for each PLC instance. This also means that you can run multiple PLC instances on one single computer.

For “PLCSimAdv” to be accessed via any network interface, you MUST change the “Online Access” from “PLCSim” to “TCP/IP” in the Control panel. If you do not do this, the PLC will only work as an “normal” PLCSim without any network functionality whatsoever.

The IP Address of the PLCSimAdv Instance is NOT the IP of your machine. It is the one that you set when starting the machine, even if it is in a completely different network.

It is also very important that you enable “Simulation” for you project in the Project settings, so that the project can be downloaded into “PLCSimAdv”. You do this by accessing the “Properties” of your project and activating the “Support Simulation during Block Compilation” attribute, found in the “Protection” tab.

NOTE: YOU MUST OPEN THE PROPERTIES OF THE PROJECT, THE TOPMOST NODE IN THE TREE VIEW, NOT THE PROPERTIES OF THE PLC.



After that, the plc can be used as if it were an “real” plc.

S7-1200 series

Currently there is no Simulation for S7-1200 plc available the support network functionality. Since TIA Portal V18, PLCSim includes support for the S7-1200 series, but not “PLCSim Advanced”, which means, that you cannot have any external HMI connections to your Simulation plc.

Siemens “PLC Sim” for S7-1200

Siemens offers an “PLC Sim” for the S7-1200, however this application has some severe Problems, which make it unsuitable for testing an BatchXpert, and any other application for that matter.

- **PLC Sim for S7-1200 does NOT! Simulate PID controllers.** If you call any of the “Technology” functions such as PID controllers, you will not get an error, but the blocks does NOT get processed. The PID’s output will always remain 0% no matter what your inputs are. PLC Sim does not process PID function blocks
- **No Network functionality.** It may seem that you can connect an HMI to the PLC Sim for S7-1200, however this is not the case. Plc Sim for S7-1200 does NOT support any network functionality whatsoever. You can NOT connect any HMI to this simulation PLC. Only WinCC does work on the same computer, due to special handling by TIA Portal. **You cannot connect any HMI, including BatchXpert, to this Simulation PLC, even if installed on the same computer as PLC Sim.**
- **You can also not simulate any blocks that utilize Communication functions, such as Get, Put or Modbus functions.** These blocks get called, but do not get processed by “PLC Sim” for S7-1200.

Use a real plc for testing

Due to the low-cost nature of S7-1200 Plcs, you can use real plc for simulation purposes. However, keep in mind that the S7-1200 has many hardware iterations, which makes it difficult to maintain compatible Plcs in stock for testing.

Simulate with S7-1500 on “PLCSim Advanced”

Since about 98% of the code is compatible between S7-1200 and S7-1500, especially when programmed in “Ladder”, you can convert the project to use an S7-1500 CPU, which then can be simulated with “PLC-Sim Advanced”.

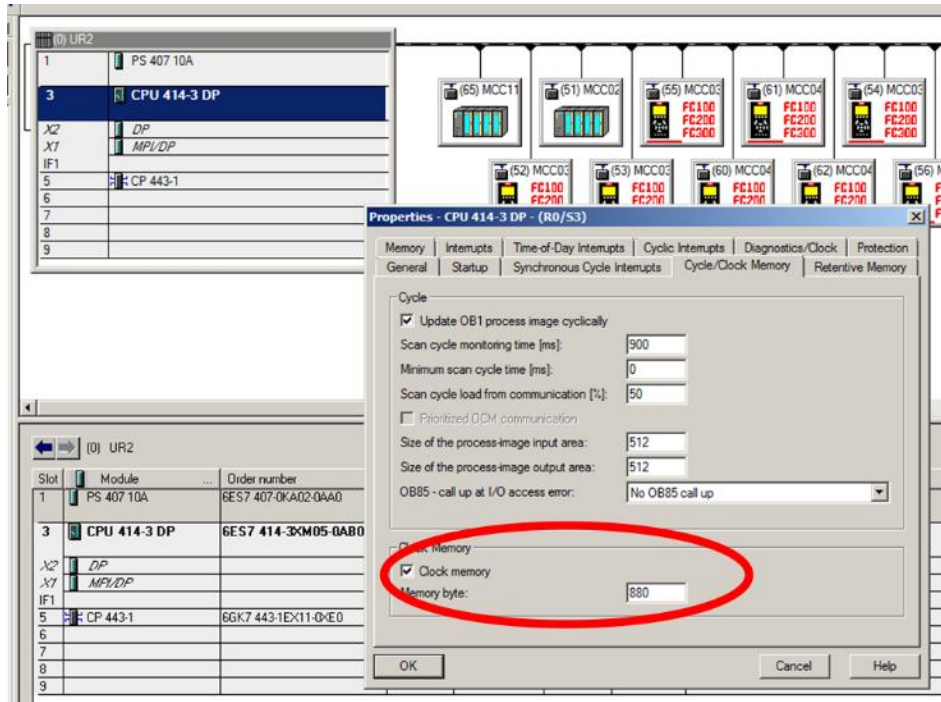
However, keep in mind that not all blocks and functions may be 100% compatible between the two series, and some adjustments may be needed to be able to simulate with S7-1500. System functions from the Simatic library, such as “Continuous regulator” blocks and communication blocks, are generally not compatible.

Required PLC Hardware Configuration Settings

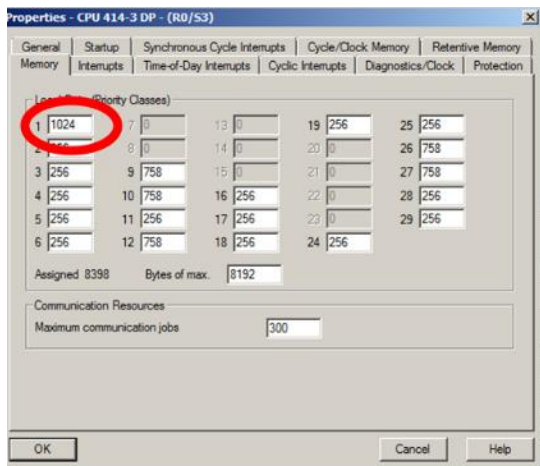
BatchXpert requires some specific settings to be adjusted in the Hardware configuration of the PLC. The specific settings depend on the Type of PLC used and may be different between PLC series that is being used.

S7-300/400 series, and compatible

You must set the “Clock Memory” of the PLC to the Memory byte 880.

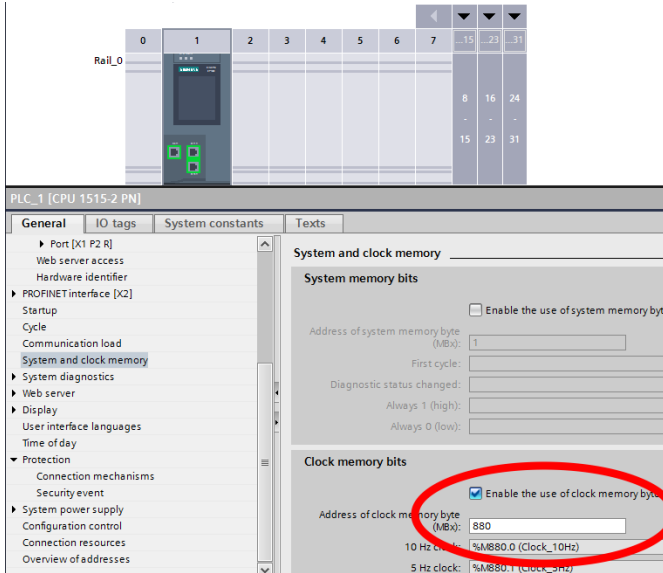


The “Local Data” should also be adjusted to at least 1024 for Priority Class 1, on PLC’s where this can be adjusted. BatchXpert can run with less local memory, but generally this amount of memory is required.



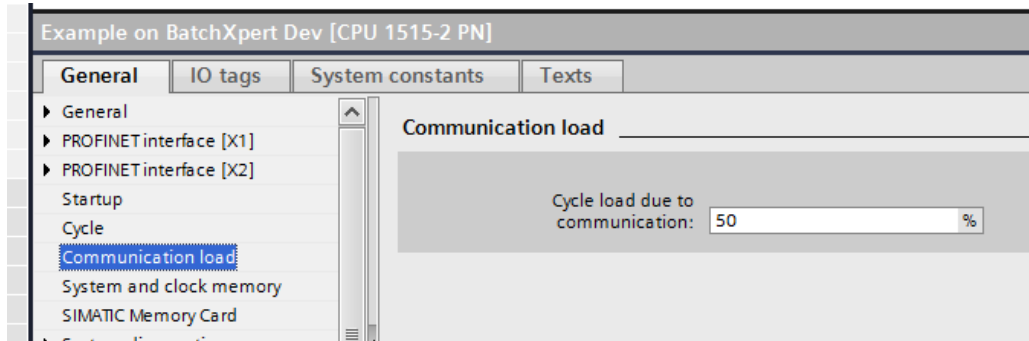
S7-1500 series

You must set the “Clock Memory” of the PLC to the Memory byte 880. All other settings can be set as needed by your implementation.



Communication Load

Since The S7-1500 series is sufficiently fast to execute a typical BatchXpert application in generally less than 20 milliseconds, you should increase the Communication load to 50%, to make communications with the Controller more responsive.



Webserver

The new generation of PLC's integrate a Webserver which allows you to gather diagnostics data, create online backups and other maintenance tasks. Some BatchXpert utilities, especially the Backup utilities, utilize this Web server. It is thus recommended that you activate the Web Server. Please see [Activate Web Server](#) for more information.

Connection Settings

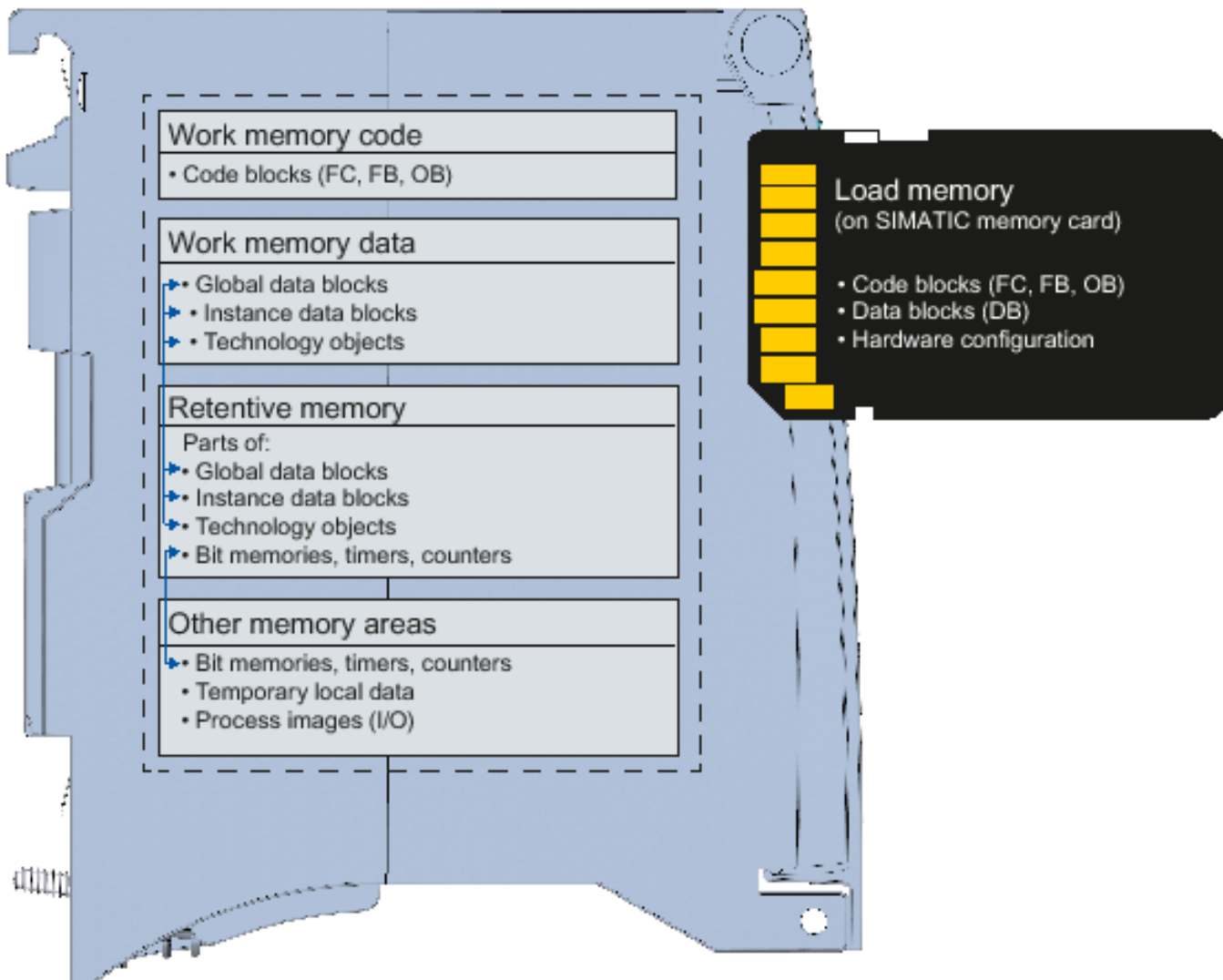
The S7-1200 and S7-1500 PLC series require some settings to be adjusted and considered when setting up communication with the BatchXpert system.

Please review [Connecting to a S7-1500 or S7-1200 PLC](#) for more information.

PLC retentive data

Chapter we are going to talk about data remanence in the programming logic controller, but we are mostly focusing on the current PLC generations, meaning the 1200 and 1500 series of PLCs. The older 300 and 400 series PLC's have similar mechanisms, but we will not go into detail about these PLC series.

In this chapter we want to give an explanation about this type of data, the limitation of schematic PLC's, and options to solve these limitations. **We recommend you use “[Using a Battery buffered Power supply](#)”.**



What is Retentive data, remanence and Retain

Retentive data, remanence and retain data are all synonyms when talking about Simatic PLCs. Retentive data is the data that remains in memory after power cycling PLC. Usually when a PLC is power cycled, it will set all data blocks to their starting values, except data and data blocks that are explicitly marked as retentive.

This means that the controller basically resets its memory back to its starting values every time it is power cycled. For machine parameters, recipe parameters and many other applications, this behavior is not desirable. Simatic PLC's offer a separate memory area which can be used for certain data blocks, which retain their values even through power cycles. This means that the data in these retain areas are simply kept even after power cycles and are not reset to their starting values.

Retentive data in BatchXpert

Retentive data is extensively used in the BatchXpert control system. All adjustments for control modules such as delay times, limit values and regulate the parameters of control modules, as well as unit statuses and downloaded recipes need to be kept in retentive data regions in your PLC.

Otherwise, all your control module settings would reset to their starting values after you power cycle your PLC.

This basically means that BatchXpert cannot function properly without retentive data. The amount of retentive data that you need for your project depends heavily on the amount of control modules and units that you are using, but it's generally relatively high compared to simpler applications.

Retentive data in S71500 and S71200 series

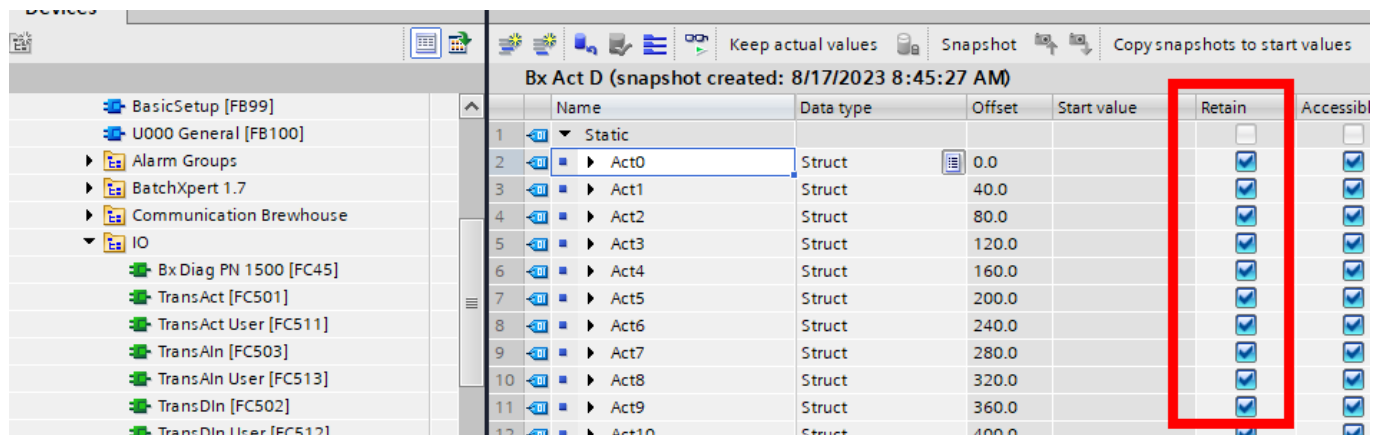
The S7-1500 and S7-1200 series have comparatively little retentive memory integrated into their CPU's. This PC series also has a higher memory footprint overall and does consume generally more memory and also retentive memory than the previous S7-300 and S7-400 series of PLCs.

This means that the retentive data area is a much more important feature of the CPU that must be considered when selecting a PLC for your project. Even though they may have megabytes of data ram, only about 200 kb of that is remanent, which for most applications is just not enough.

It is also important to note that your retentive data area cannot be extended on Simatic Controllers, not even by using memory cards.

Using Retain in your Project

For smaller projects, typical for S7-1200 series PLCs, it may be enough to simply use the existing retentive memory areas off your PLC. In that case all you must do is activate the "Retain" option in your data blocks of all your control modules and units.



	Name	Data type	Offset	Start value	Retain	Accessibl
1	Static				<input type="checkbox"/>	<input type="checkbox"/>
2	Act0	Struct	0.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	Act1	Struct	40.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4	Act2	Struct	80.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5	Act3	Struct	120.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6	Act4	Struct	160.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	Act5	Struct	200.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
8	Act6	Struct	240.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
9	Act7	Struct	280.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
10	Act8	Struct	320.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
11	Act9	Struct	360.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
12	Act10	Struct	400.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Please keep in mind that you must activate this option for all data blocks of all control modules and all unit data blocks that you are using in your project. Blocks that do not have this activated will get reset after a power cycle.

In BatchXpert this option can only be activated for the whole data block, and not for only individual modules. This basically means that you must mark all your control modules with "Retain", which makes this option unsuitable for medium and large projects, which generally have more control modules than would fit inside the retentive data area of your PLC.

How much retentive data is required

The amount of required retentive data can be checked in your project on the left side of the project on the option "program info". This gives you a total overview of all memory areas that you are using in your PLC, including retentive data.

Objects	Load memory	Code work-memory	Data work-memory	Retain memory
	44 %	34 %	12 %	106 %
Total:	12 MB	614400 bytes	2621440 byte	221856 byte
Used:	5548656 bytes	210692 bytes	324602 byte	234450 bytes
Details				
OB	35664 bytes	2719 bytes		
FC	2751479 bytes	198978 bytes		
FB	197120 bytes	8995 bytes		
DB	2476494 bytes		324602 bytes	234450 bytes
Objects for Motion Technology	-	-	-	0 bytes
Data types	61600 bytes			
PLC tags	26299 bytes			0 bytes

Ilustración 1 Not enough Retentive data

Using a Battery buffered Power supply

We recommend the use of the “Battery buffered Power supply” “6ES7505-0RB00-0AB0”. This power supply converts all Data RAM into remanent Data ram, thus eliminating the remanent data restriction of S7-1500 PLCs.

This power supply includes an internal battery which keeps the memory of your PLC alive even while your power supply is powered off. This is essentially the equivalent off the power supplies of the outgoing S7-400 series of PLC's, which used buffer batteries to keep the memory of the controllers alive. **You must also activate the “Retain” option for all your Control module and Unit data blocks.**



If you are using this type of power supply, please do not forget that you must activate certain options in your parameters of your hardware configuration.

The screenshot displays the HW Config interface for a power supply module. At the top, a rack diagram shows the module in slot 0 of rack 100. Below this, the 'Properties' window for 'PS 60W 24/48/60VDC HF_1' is open, showing the 'General' tab. The 'Name' is 'PS 60W 24/48/60VDC HF_1', 'Author' is 'Stefan', and 'Comment' is empty. The 'Rack number' is 0 and 'Slot' is 0. The 'Description' field states: 'System power supply 60W, 24/48/60VDC HF; supplies the operating voltage for the S7-1500 backplane bus and supports additional retain memory for the CPU.'

Ilustración 2 the Battery backed power supply in your Hardware configuration

The screenshot shows the 'System power supply' configuration for 'PLC01_Fermentacion [CPU 1513-1 PN]'. The 'General' tab is selected, showing two radio button options: 'Connection to supply voltage L+' (unselected) and 'No connection to supply voltage L+' (selected). Below this, the 'Power segment overview' table is displayed:

Module	Slot	Power consumpti..
PS 60W 24/48/60...	0	60.00W
PLC01_Fermenta...	1	-5.50W
Summary		54.50W

Ilustración 3 You MUST select "No connection to supply voltage L+" to enable full memory retention

Objects	Load memory	Code work-memory	Data work-memory	Retain memory	Me
	44 %	34 %	12 %	9 %	
Total:	12 MB	614400 bytes	2621440 bytes	2621440 bytes	
Used:	5548658 bytes	210692 bytes	324602 bytes	234450 bytes	
Details					
► OB	35664 bytes	2719 bytes			
► FC	2751479 bytes	198978 bytes			
► FB	197120 bytes	8995 bytes			
► DB	2476494 bytes		324602 bytes	234450 bytes	
Objects for Motion Technology	-		-	0 bytes	
► Data types	61600 bytes				
PLC tags	26301 bytes			0 bytes	

Ilustración 4 All memory is now "Remanent"

Using the memory Card

As an Alternative you can use the "Persistence" functionality of BatchXpert, which utilizes the Memory card. This results in considerable "Wear" on flash memory. The "Persistence" functionality is optimized for this but still results in about "20-year lifetime" of the flash memory. The Persistence functionality will save modification at most every 20 minutes, so the "Resolution" is limited to this interval.

A typical Simatic Memory Card will allow 500.000 writes per cell, which at most every 20 minutes will result in about 20 years of Memory Card life.

For this option to work you must call "Bx Persist" at some point of your OB1 "Cycle_exc". You can turn off the "Retain" option for most data blocks, and only activate them for the most important blocks, since all data will be stored to memory card after some time interval or changes have been made to the data.

The screenshot displays the SIMATIC Manager interface for the 'PLC01_Fermentation [CPU 1513-1 PN]' project. The left pane shows the project tree with the 'Persistence' folder highlighted. The right pane shows the 'CYCL_EXC [OB1]' program block. The 'CYCL_EXC' block contains a table of variables and a list of networks. A red box highlights the 'Persistence' folder in the project tree, and a red arrow points to the 'CALL "Bx Persist"' instruction in the program block.

Name	Data type	Offset	Default value	Comment
Temp				
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Ever
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of

Block title: "Main Program Sweep (Cycle)"

Network 1: Init Data

Network 2: System Begin

Network 3: Assign Global Control Modules

Network 4: Units and user programmes

Network 5: Communications

Network 6: Alarm Groups

Network 7: System functions

CALL "Bx SysEnd"

CALL "Bx Persist"

Details about “Bx Persist”

The BatchXpert persistence function is optimized to minimizing “write cycles” on your memory card. For this BatchXpert listens to parameter changes from the user and then triggers a write to the memory card only if values have changed after waiting for a few minutes. This means that the memory card is only written when some value in any of the control modules or units have changed.

The memory card “write” conditions are as follows:

- If a Parameter is changed from the HMI, after 5 minutes the corresponding module is written
- All modules are written at least once each day.

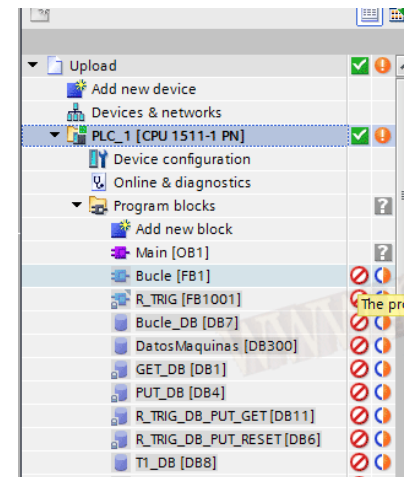
This mechanism should result in at least 15 to 20 years of service life for your memory card.

Disadvantage of “Bx Persist”

When using “BatchXpert”, we always recommend that you use a battery backed power supply. You should only use the “Bx Persist” function if no other option is available.

Since “Bx Persist” rights data blocks on the memory card with their current values, this means that these data blocks will appear as “Different” or “Changed” when going online with your project. The reason for this is that you are technically changing the data block starting values on your memory card which then do not match the starting values that are stored in your project anymore.



To resolve this situation, you should then always upload the data blocks from the controller which contain the current starting values that correspond to the last values that have been saved to the memory card.





“Keep Actual Values”

The S7-1500 PLC's, offer the option of “Keep Actual Values”. This is not a reference to data remanence but refers to the option to be able to download a data block without resetting its current values. Even if you activate this option for the data block will still get wiped when a power cycle happens. This option is just a programming convenience feature, then you can use it for extending and modifying data-blocks.

TestProjekt ▶ PLC_1 [CPU 1516-3 PN/DP] ▶ Program blocks ▶ MyGlobalDB [DB1]

Keep actual values  Snapshot 

Copy snapshots to start values  Load start values as actual values 

MyGlobalDB (snapshot created: 08.02.2017 10:13:02)

	Name	Data type	Start value	Snapshot	Monitor value	Retain
1	Static					<input type="checkbox"/>
2	value	Real	22.22	22.22	15.88	<input type="checkbox"/>
3	timeCheck	Time	T#450MS	T#450MS	T#225MS	<input type="checkbox"/>
4	tempMax	Byte	16#AB	16#AB	16#2F	<input type="checkbox"/>

PLC Time Synchronization

The system time is very important on the Stations, since this is required for accurate Timestamping of all changed data. The time synchronization between stations is usually done via NTP and you can find more information here [System Time](#).

However, the PLC must also have the same system as the BatchXpert stations. This means that you must implement some time synchronization mechanism in your project. You can implement this mechanism in one or two ways.

Time zones

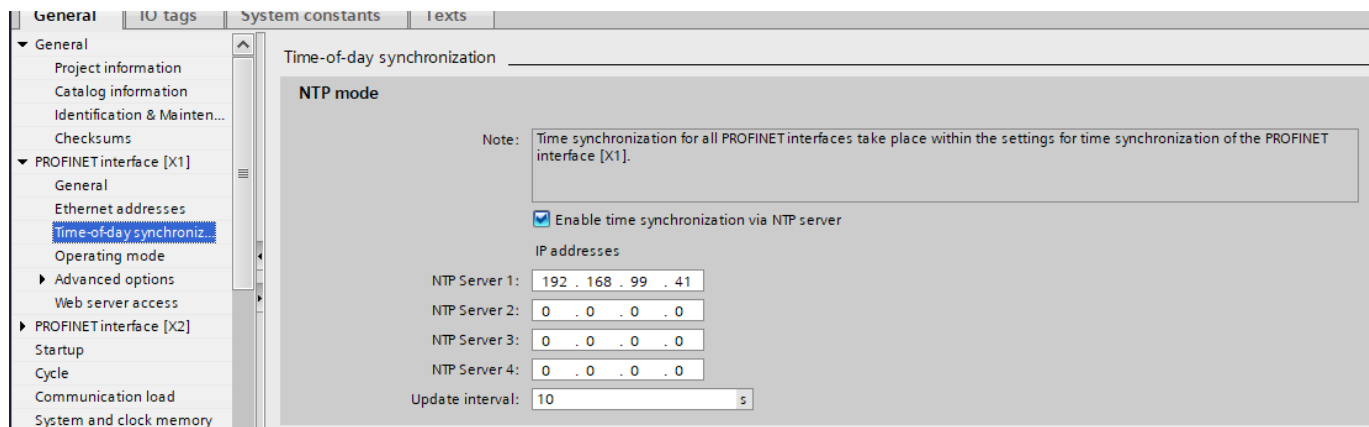
Due to the mechanism that BatchXpert registers its historical data, the Time Zone of the PLC is not relevant, and all-time stamps are always considered as local time of the operating stations. This is done to make the time stamping in the PLC more robust, since most of the time, the PLC time zones are not adjusted by the commissioning engineer or the Maintenance department.

Using the NTP server

Since the BatchXpert system already must implement a time synchronization mechanism so that all operating station can synchronize their time between each other, usually you already have an NTP server installed on usually the “Bx1” operating Station. You can find more information here [System Time](#).

This means that you can set the PC to use this operating station as its NTP server so that the PC also connects to the server and synchronizes its system time from it. This is the preferred way to implement time synchronization to the PLC, since it uses standard mechanisms that are well understood by any IT personnel. However, keep in mind that adjusting these settings in the PC requires you to download the hardware configuration which means that you must restart your controller, which will affect your processes that are running.

The required settings in your PLC are shown below. The settings below correspond to the settings required for a connection to the standard batch expert NTP server “NetTime” that is described in its own manual in detail.



You must select the Interface that is connected to the BatchXpert stations and set the IP address of the BatchXpert stations that are running as NTP server, usually the “Bx1” station.

Using the HMI Script

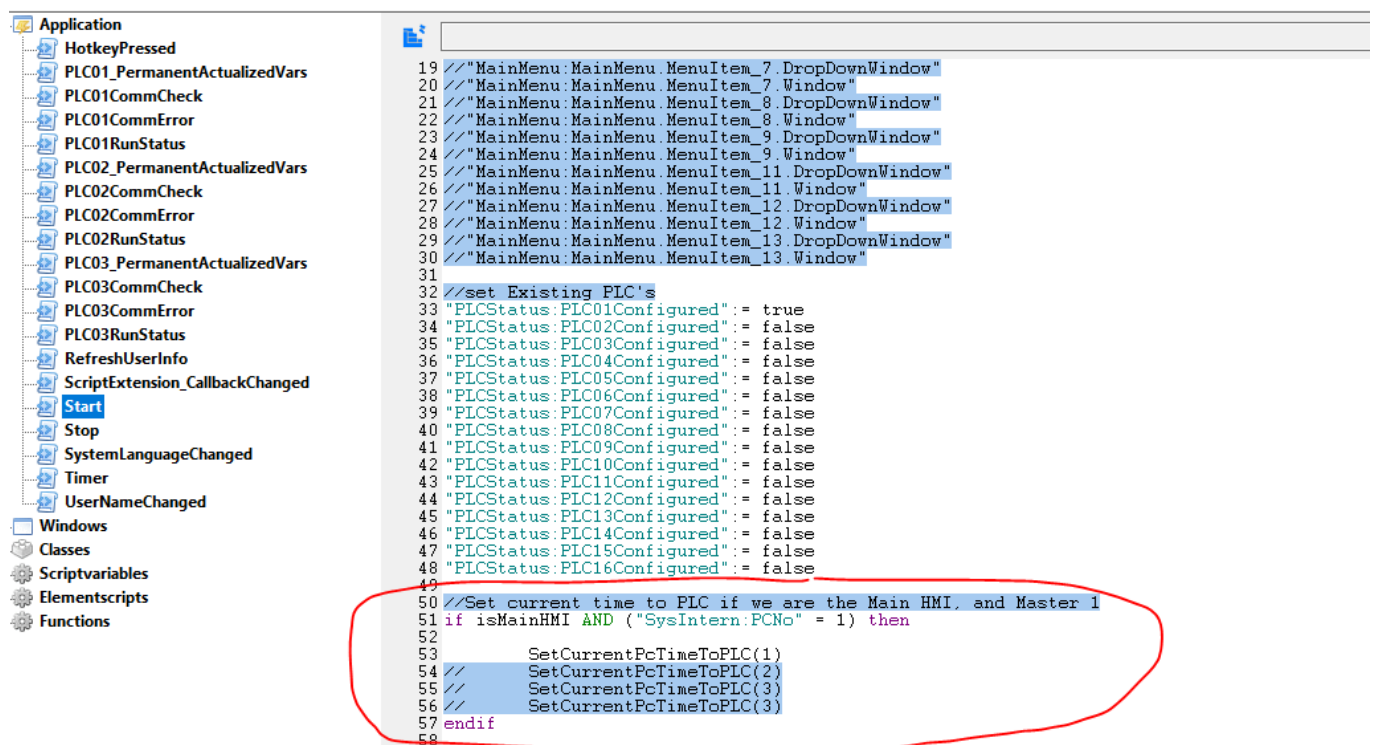
The BatchXpert PLC framework includes a mechanism to set the local module system time of your PLC from an HMI application. The HMI application provides several scripts to automatically send the current time to the PLC and request setting this time as system time on the controller. Usually this is only done by the user by clicking on the PLC and manually adjusting the time of your controller.

However, you can automate this functionality and send a “adjust PLC Time” requests to the PLC every time your HMI application starts up. This means that at least when your BatchXpert station starts up the current time will be sent to the PLC and assumed synchronized with the BatchXpert.

To avoid multiple times synchronization requests to the PLC, you should only enable this automatic time synchronization on one of the operating stations and only on the main HMI system. Usually, this time synchronization is only done on the “BX1” operating station.

This script for current time to the controller is placed in the application start script so that it runs every time your application starts up. Please refer to [Application Start Script](#) chapter about more details about the Application start script.

If the script functions are not available, you can import them from the library. Please refer to [The HMI Library](#).



```
19 //MainMenu:MainMenu MenuItem_7.DropDownWindow"
20 //MainMenu:MainMenu MenuItem_7.Window"
21 //MainMenu:MainMenu MenuItem_8.DropDownWindow"
22 //MainMenu:MainMenu MenuItem_8.Window"
23 //MainMenu:MainMenu MenuItem_9.DropDownWindow"
24 //MainMenu:MainMenu MenuItem_9.Window"
25 //MainMenu:MainMenu MenuItem_11.DropDownWindow"
26 //MainMenu:MainMenu MenuItem_11.Window"
27 //MainMenu:MainMenu MenuItem_12.DropDownWindow"
28 //MainMenu:MainMenu MenuItem_12.Window"
29 //MainMenu:MainMenu MenuItem_13.DropDownWindow"
30 //MainMenu:MainMenu MenuItem_13.Window"
31
32 //set Existing PLC's
33 "PLCStatus:PLC01Configured" := true
34 "PLCStatus:PLC02Configured" := false
35 "PLCStatus:PLC03Configured" := false
36 "PLCStatus:PLC04Configured" := false
37 "PLCStatus:PLC05Configured" := false
38 "PLCStatus:PLC06Configured" := false
39 "PLCStatus:PLC07Configured" := false
40 "PLCStatus:PLC08Configured" := false
41 "PLCStatus:PLC09Configured" := false
42 "PLCStatus:PLC10Configured" := false
43 "PLCStatus:PLC11Configured" := false
44 "PLCStatus:PLC12Configured" := false
45 "PLCStatus:PLC13Configured" := false
46 "PLCStatus:PLC14Configured" := false
47 "PLCStatus:PLC15Configured" := false
48 "PLCStatus:PLC16Configured" := false
49
50 //Set current time to PLC if we are the Main HMI, and Master 1
51 if isMainHMI AND ("SysIntern:PCNo" = 1) then
52
53     SetCurrentPcTimeToPLC(1)
54 // SetCurrentPcTimeToPLC(2)
55 // SetCurrentPcTimeToPLC(3)
56 // SetCurrentPcTimeToPLC(3)
57 endif
58
```


PLC Overview

BatchXpert uses an Operating system for a PLC which provides facilities and functions to be used in user programs. This Operating system or “PLC Frame” depends on your specific PLC type and enables the system to send recipes, record historical data, handle phases and steps and provides all the different Control Modules.

The PLC Program is structured as follows:

- FC 1-100: Fixed System Functions (Block Numbers Cannot Be Reassigned)
- DB1-100: Fixed system data, with no possibility of reassigning.

The rest of the functions and FB not mentioned are free for the use of the user (programmer). However, there are many auxiliary functions that occupy the FC 400-600 range, but which can be redirected by the user, if necessary.

Requirements to the PLC

The BatchXpert system requires certain features of the PLC. The system mostly requires a lot of RAM to be able to function. For more information, please refer to the system's "System Requirements" manual.

The program is compatible with the following PLC systems:

- Siemens Simatic S7-300 series
- Siemens Simatic S7-400 series
- Vipa Speed7 300 series
- Vipa Slio series
- Vipa Micro series, although in a limited capacity as it has little ram
- Siemens Simatic S7-1200 series, for smaller projects
- Siemens Simatic S7-1500 series

The Projects can be created in “Simatic Manager,” but also “Tia Portal” starting from Version 16 are supported.



Program Structure

The following shows the structure of the system's general calls.

The functions are colored according to the following categories:

- System functions, not modifiable
- Generated function from Project engineering tool. To not manually modify
- IO-related functions, Adjustable if required.
- User Modifiable Block

OB1 (CYCL_EXC)		
	FC10 (Bx SysTime)	
	FC1 (Bx SysBegin)	
	FC8 (Bx SysInit)	
	FC86 (Bx UnitProtSend)	
	FC50 (Bx RecLoader)	
	FC96 (Bx UnitPc)	
	FC94 (Bx UnitProgWin)	
	FC97 (Bx UnitStatusInfoWin)	
	FC7 (Bx ManuProtSend)	
	FC45 (Bx DiagDP)	
	FC502 (TransDIn System)	
	FC512 (TransDIn User)	
	FC16 (Bx DIn)	
	FC503 (TransAIn System)	
	FC512 (TransAIn User)	
	FC21 (Bx AIn)	
	FB101 (U001 config)	
	FC100 (Bx Unit)	
		FC101 (U001 Phases)
	FB102 (U002 config)	
	FC100 (Bx Unit)	
		FC102 (U002 Phases)
	FC2 (Bx SysEnd)	
	FC11 (Bx Act)	
	FC31 (Bx PID)	
	FC36 (Bx Msg)	
	FC39 (Bx Switch)	
	FC4 (Bx SVal)	
	FC504 (TransPID System)	
	FC514 (TransPID User)	
	FC501 (TransAct System)	
	FC511 (TransAct User)	
	FC5 (Bx WinOrder)	

User programs are programmed into the FB1xx and FC1xx of the sequences. IO-related blocks are generated by the Engineering tool of the BatchXpert system.

Control Modules

BatchXpert programs do never access any IO signal directly but do so by using an “Control Module.” A control module abstracts these IO Modules with functionality accessible to the user programmer. Instead of “Activating an Output” you send a signal to an “Actuator” control module to activate the output, if possible. These modules all implement extended configuration, simulation, alarm condition and time delays, depending on the module. They also implement a Symbol library for the HMI with functionality, configuration and so on.

The most important control modules are the “Actuator,” “Digital Input”, “Analog Input” and “PID Regulator” modules, as they directly correspond to physical PLC OI modules.

However, not all Control modules correspond to physical IO of the PLC, but nevertheless provide useful functionality to the user, such as the “Material Module,” the “Special Value”, and the “Counter” modules. The Modules also provide signals, which describe their current state, and allow the programmer to write logic if any state changes. These states depend on each module, but are:

- General Alarm
- Automatic Mode
- Signal
- Etc.

Each module of a Type has a unique “number” that identifies it in the PLC. For example, there might be “Act10”, an “Ain3”, and so on. The names that correspond to each module are assigned in the database, usually by importing an Taglist via the “Project engineering Tool.”

PLC communication Resources

Some important resource in a plc that is often overlooked are the communication resources it has. Each connection from either an HMI station, plc-plc communications, BatchXpert or a programmer will consume one of these resources. Different CPU types and different Communication processors will have different amounts of available connections.

When connecting equipment to plc, you must keep this communication resource limitation in mind. Usually, the amount of communication is 16 or even 32, but for some Communication processes may be as low as just 4 connections.

Memory Reset (Factory Reset)



A "Memory Reset" of the PLC will delete all the data stored in the PLC's memory! This includes all data, such as process logic and parameters. **After a complete deletion, a subsequent download of the program to the PLC is always necessary!** For guidance on how to restore a PLC backup, please read the manual "PLC Restore Manual".

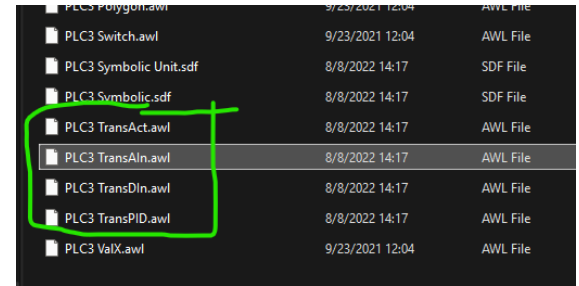
It is NOT recommended to perform "Memory Reset". Please note the following:

- *All logic will be deleted until you download it back into the controller again. This means that there won't be any logic at all running on the controller. There are no functionalities of any Valve or Motors, not even Manual operations will be possible; No HMI will be able to connect, since the communication channel data blocks will also be deleted.*
- *The parameters and settings of all control modules (such as regulators, valves, etc.) will be reset to the values that they had at the time the backup was taken.*
- *When downloading a backup to the PLC, ALL data and processes are reset to the moment the backup was made. This means that ALL sequences of ALL Processes lose their state and are reset, making it impossible to continue with a process that was in progress!*
- *After loading, the statuses of the processes must be reset, when the corresponding recipes are started, and the steps of the Units must be advanced to the corresponding Steps.*

The “Trans xx” blocks for IO signal transfer

Many control modules are meant to work as abstractions for the IO level of an PLC, such as Actuators, Digital inputs, or Analog Inputs. This means that there must be some code that links the IO points of the PLC to the individual control modules.

This is what the “Transxx” blocks do. They assign the Inputs of the configured feedback to the actuator modules and then copy the “Output” signal of the actuators, to the corresponding Plc outputs. This same happens similarly for all Control Modules.



PLC3 Polygon.awl	9/23/2021 12:04	AWL File
PLC3 Switch.awl	9/23/2021 12:04	AWL File
PLC3 Symbolic Unit.sdf	8/8/2022 14:17	SDF File
PLC3 Symbolic.sdf	8/8/2022 14:17	SDF File
PLC3 TransAct.awl	8/8/2022 14:17	AWL File
PLC3 TransAln.awl	8/8/2022 14:17	AWL File
PLC3 TransDIn.awl	8/8/2022 14:17	AWL File
PLC3 TransPID.awl	8/8/2022 14:17	AWL File
PLC3 ValX.awl	9/23/2021 12:04	AWL File

Taglist

Usually, you assign the IO's of the modules by editing the Taglist in Excel, import them into the database, and then regenerate the “Trans IO” blocks, so they can be imported, compiled, and downloaded into the plc. This means that fundamentally these blocks are being regenerated every time you have a change in any of the IO's of a module, or you must add a new module.

This Generation, Importing and downloading is fundamentally a manual process and does not happen automatically.

Import, compile and download

The “Project Engineering” tool creates “STL” (or “AWL”) source files must be manually imported into your project and then be compiled and downloaded into your controller. Usually, you can override the existing transfer blocks with the ones created from the project engineering tool, however you are advised to check that the new and old blocks are compatible.

If the blocks are not compatible you must manually combine the old and new blocks. Overwriting the existing blocks is possible because you should never modify the “System” transfer blocks but create all modifications that you need in the corresponding “User” blocks. This way the generated code will never override manual modifications.

Modifications in generated code

However, there are situations where the generated code must be adapted to work properly. However, since the “Trans IO” blocks are automatically generated, you cannot modify them, since all your changes would be overwritten the next time, somebody regenerates them. To avoid this problem, all “Trans IO” Blocks are generated as pairs, of an “System” block and an “User” block. Both blocks are called by the BatchXpert plc frame automatically, by first calling the “Trans IO System” block, which is the one you regenerate, and then the PLC will execute the “Trans IO User” block, which allows you to overwrite whatever logics that was executed in the “System” block or add your own custom IO Logic.

Since the “User” blocks are never generated automatically, every code that you write in them will never be overwritten by any code generator. Also, since they are called immediately after the “System” blocks, you can adjust all IO Transfer in them before the Control modules can process them. You can, for example:

- Send Outputs to entirely different outputs.
- Use different Inputs, or no inputs at all.

- Create any custom logic that is required, for example implement simulations, pulses, or even communications with external equipment.

The “Trans IO User” block is where your customized IO to control module transfer logic goes.

Special Considerations for Analog Modules

Analog Input and Output modules, which correspond to Analog Input and PID Regulator Control modules, sometimes have special considerations to adjust their scaling. These special settings are described in the Chapter describing the Control module itself. Please refer to the [Analog Input Scaling](#) and [PID Regulator Output Scaling](#) chapters for more information.

TIA-Portal

While the older SIMATIC Manager is still relevant for existing projects, all newer projects should be created using its successor “TIA-Portal”, since support for Simatic Manager will not be available much longer (as of 2025).

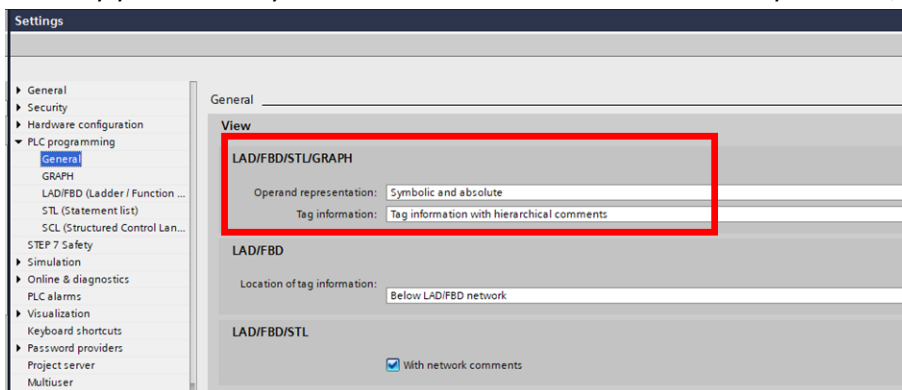
In general, we recommend that you use the following setting when working with TIA-Portal since most code generation is designed around these settings.

Use Hierarchical Data block comments

You should activate the “Tag information with hierarchical comments” in your “General-View” settings. The BatchXpert Engineering tool generates Control module data blocks, by using UDT (user defined datatypes), that have modules symbols and comment in their comment field.

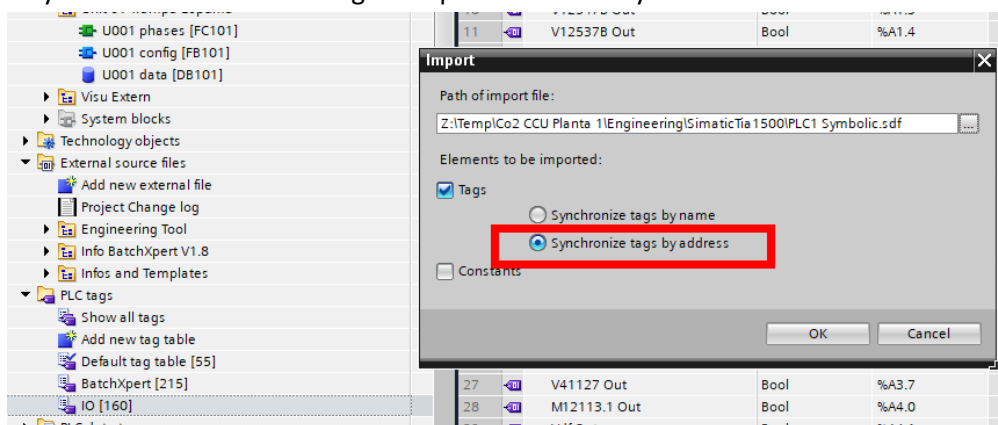
What this setting does is, in your code editor, it shows you the comment of the variable from the control modules UDT first, and then appends the comment for its parents, which contain the symbol and comment of the corresponding control module.

This way you can always view the control modules comments in your code, greatly improving code readability.



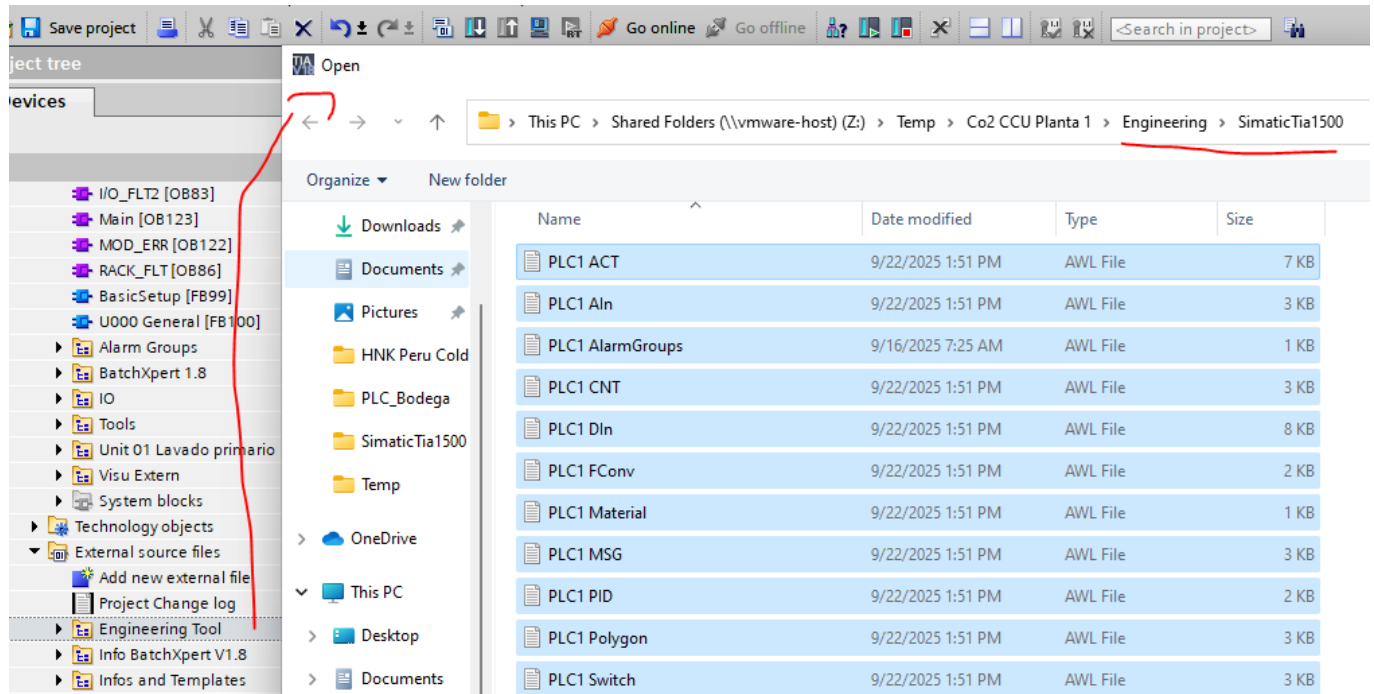
Import Symbols always “By Address”

When importing symbols generated from the Project Engineering tool, you should always import “By Address” so that changed symbol names do not result in duplicate addresses, but rather in updated information. This is the way the older “Simatic Manager” import functionality worked.



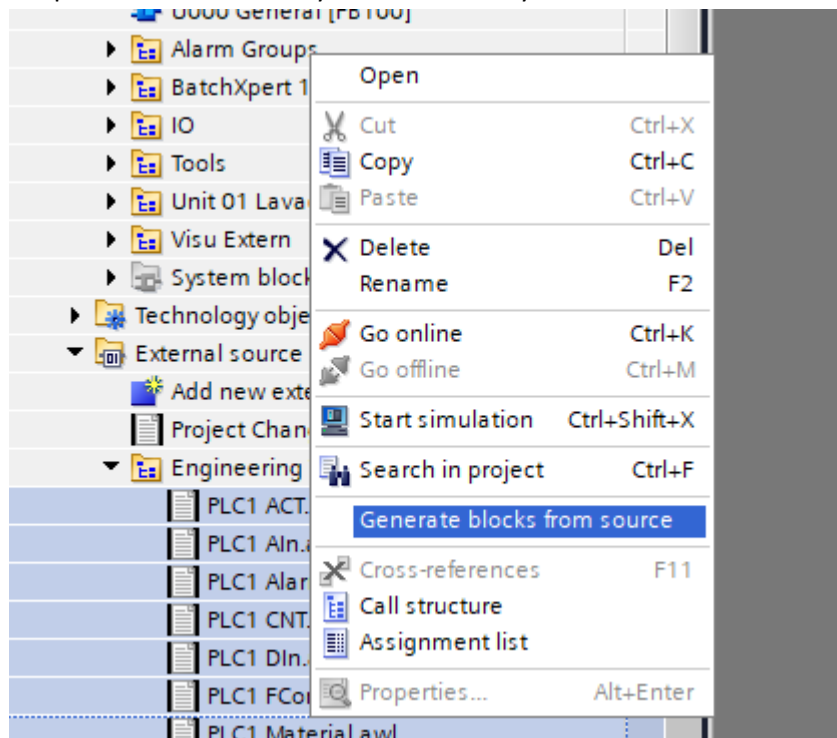
Importing and compilation of Generated block sources

In order to use the generative blocks from the project engineering tool, you first have to add a new external source file into your external source files for your project. You do that by right clicking on the external Swiss files and then selecting all your generated code files from the appropriate engineering directory.



after you have imported all your source files you can mark all of them and right click to “Generate Blocks from sources”, which starts the compilation of all your blocks and will overwrite existing blocks in your project with the newly generated ones.

Keep in mind that this may also mean that you must download modified code and data blocks.



Control module data-blocks

When compiling control module data blocks, the existing data block of the corresponding control module will be completely overwritten by the newly generated 1 which contains the new control modules that you may have added, removed or modified. However, since this is a new data block it will need to be downloaded into your PLC, which means that all current control module data will be overwritten by default values stored in your newly generated data block.

This is a problem that always exists when downloading data blocks into a PLC, and it's not easily solvable. You can use tools such as [“S7-Backup”](#) to make backups of your values that you can later then restore into your PLC, however this is a manual process, which cannot easily be automated.

Generally, it is preferable that you add new modules manually into your control module data block by using features that already exist in TIA-Portal.

TIA Portal: Recommended way to modify Control module data blocks

Because of the problems that I mentioned above, it is generally not a trivial task to override an already existing and deployed control model data block by compiling and source into a new data block. However, the portal provides functionality that more easily allows you to modify and even extend already existing data blocks, without overwriting the current control module's configuration with default values.

The first thing to note is that you should never compile an already existing data block, since this will override the existing data block, and requires a download with default values into your PLC.

Summary

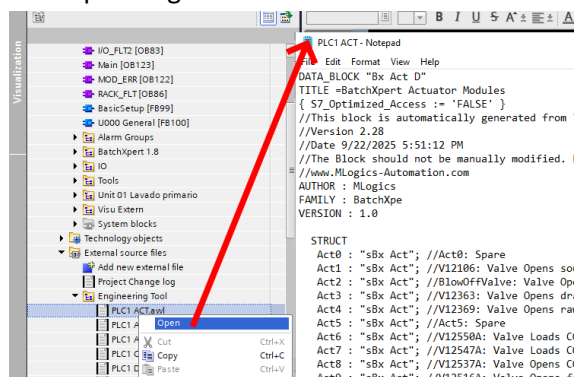
you take a snapshot of your current control module startup block, assign these snapshot values as starting values on your control module data block.

Then you change the name of the newly generated data block in the source file, compile it into a temporary new data block, from which you can copy all your modified control modules into your existing control module data block.

After that you can download your modified control module startup block without losing your current settings of all your control modules contained in this data block.

1. Open the data block source file

Right click on the corresponding source and click open. This will open the text editor so that you can edit the corresponding source file.



2. Change the Blocks symbolic name

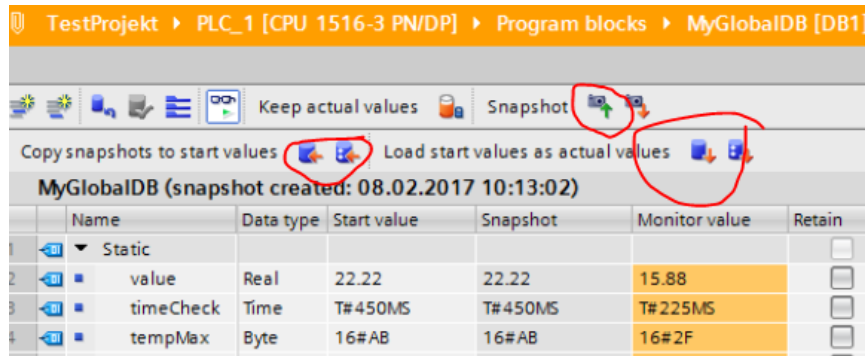
Then change the data blocks symbolic name for example append running number to it. This means that if you compile this data block, it will generate a completely new data block without overwriting the existing one. This means that you can then manually handle new control modules by copying them into your existing data block.

*PLC1 ACT - Notepad

```
File Edit Format View Help
DATA_BLOCK "Bx Act D 2"
TITLE =BatchXpert Actuator Modules
{ S7_Optimized_Access := 'FALSE' }
//This block is automatically gener
//Version 2.28
//Date 9/22/2025 5:51:12 PM
```


3. “Snapshot” current values of your existing control module data block

You can then open your existing data block, go online and create “Snapshot” of all current values of your data block, which you then assign as “Starting Values”. This way if you download the data block the control module will start up with the currently up to date values.



4. Manually copy new or updates modules

You can then open your newly generated data block, which will have a new name and copy new or modified modules manually into your existing control modules data block.

5. Delete newly generated data block

Since you have already copied all new control modules from the newly generated data block into your actual control module startup block, you don't need the temporarily generated data block anymore, and you can delete it from your project.

6. Download Modified Control module data block

After that you can download the modified control module data block. This will of course override all current value stored in the data block with the values stored in your “offline” data block, but since you uploaded and snapshot and assigned this snapshot as starting values, the data block essentially starts up with the snapshot values that you took as starting values. This basically means that you have reset your data block to the point in time where you took your snapshot values, which should only be one or two minutes in the past, end user usually has no effect on the control modules.

Global Signals and Symbols

The BatchXpert system provides the following signals for use in the user's program. All signals presented are "READ ONLY" and should not be written by the user. The following Signals are global and can be used throughout the program in any user block.

General PLC status

These signals give you information and commands about the global status of the PLC and some global command you can trigger from your code.

Symbol	Address	Data type	Description
PLCRestart	M 878.0	BOOL	PLC restart (Stop => Run) Will be TRUE for one cycle wafter restarting the PLC. It allows you to detect restarts, for example to preset the "Emergency Stop" alarm to activated, so that every restart, by default the Estop needs to be confirmed.
PLCRunning	M 878.1	BOOL	PLC Running after restart. After the first cycle of the plc, this will become TRUE and remain so until you stop the plc
ToDo_Read	M 878.2	BOOL	To Do - read signal. This is an "Marker" signal, with which you can mark sections of code and then later find it by doing an "Cross Reference" search for this signal.
ToDo_Write	M 878.3	BOOL	To Do - set signal. This is an "Marker" signal, with which you can mark sections of code and then later find it by doing an "Cross Reference" search for this signal.
SimTest	M 878.4	BOOL	Only simulation test must be "0" in production mode. For deployed projects, this signal should remain "False." But you can deactivate certain Estop alarms, and other safety related things when you are running in simulation.
QuittAll	M 878.7	BOOL	reset all alarms. If set to TRUE it will confirm all alarms in the PLC. Will be reset automatically.

Timers and Clock signals

- Clocks are periodic signals that have a periodic on and off time. For example, the “Clk10” will stay ½ second TRUE, and ½ second FALSE. These signals are used for different “flashing” rates for illuminated buttes etc.
- Clock edge signals turn on for just one Plc cycle after their time expired. They are a convenient way to count time or perform calculation ever xxx time. For example, the “Clk1E” will be true for one cycle every 1 second.
- Clock cycle edge signals are like the “Clock Edge” signals but operate on PLC cycles instead of time. So, the “clk16CE” will become TRUE every 16 plc cycles.

Symbol	Address	Data type	Description
Clk2CE	M 879.0	BOOL	clock 2 cycle (edge)
Clk4CE	M 879.1	BOOL	clock 4 cycle (edge)
Clk8CE	M 879.2	BOOL	clock 8 cycle (edge)
Clk16CE	M 879.3	BOOL	clock 16 cycle (edge)
Clk32CE	M 879.4	BOOL	clock 32 cycle (edge)
Clk64CE	M 879.5	BOOL	clock 64 cycle (edge)
Clk128CE	M 879.6	BOOL	clock 128 cycle (edge)
Clk256CE	M 879.7	BOOL	clock 256 cycle (edge)
Clk01	M 880.0	BOOL	clock 0,1 sec (10 Hz)
Clk02	M 880.1	BOOL	clock 0,2 sec (5 Hz)
Clk04	M 880.2	BOOL	clock 0,4 sec (2,5 Hz)
Clk05	M 880.3	BOOL	clock 0,5 sec (2 Hz)
Clk08	M 880.4	BOOL	clock 0,8 sec (1,25 Hz)
Clk10	M 880.5	BOOL	clock 1,0 sec (1 Hz)
Clk16	M 880.6	BOOL	clock 1,6 sec (0,625 Hz)
Clk20	M 880.7	BOOL	clock 2 sec (0,5 Hz)
Clk1E	M 881.0	BOOL	1 second (edge)
Clk1E1	M 881.1	BOOL	1 second (edge), 1 cycle later
Clk1E2	M 881.2	BOOL	1 second (edge), 2 cycle later
Clk6E	M 881.3	BOOL	6 second (edge)
Clk10E	M 881.4	BOOL	10 second (edge)
Clk60E	M 881.5	BOOL	60 seconds (=0.1-minute, edge)
Clk1DayE	M 881.6	BOOL	1 day (edge)

Time values

These values give you access to the currently passed cycle time of your OB1 execution and are used to easily build your own counter, by summing up the value each cycle.

Symbol	Address	Data type	Description
CycleCnt	MB 879	BYTE	cycle counter
CycleTimeSec	MD 900	REAL	Cycle Time in Seconds
CycleTimeMin	MD 904	REAL	time minutes
CycleTimeHour	MD 908	REAL	time hours
CycleTimeDay	MD 912	REAL	time days

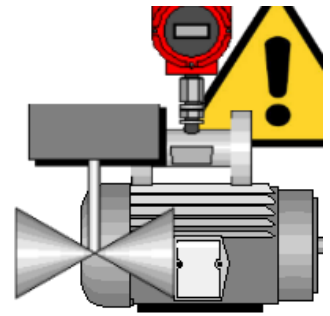
```
//Count timer
L UxxD.User.TimerDelayMem          //Accumulated Time
L CycleTimeSec                     //Current time in seconds: 0,012 = 12 msec
+R
T UxxD.User.TimerDelayMem          //Save new sum to the accumulation

//Check timer
L UxxD.User.TimerDelayMem
L 30.0                             //30 seconds
>R
SPBN TiDo
L 0.0
T UxxD.User.TimerDelayMem          //Reset Accumulated Time to restart it

//Do something here
TiDo:
```

General Structure of Control Module DBs

In the PLC, the data of control modules (actuator, PID, ...) is kept in Arrays, which may also be “unrolled” into their individual elements, to be able to put individual comments on them. It is only important that the internal structure of the objects is maintained. You can generate these unrolled data blocks where each element has its own structure with appropriate descriptions, taken from the control modules description in the database. We recommend that you choose this option because it creates the most readable plc code.



Execution of the Control Modules

In the BatchXpert plc framework, the control modules are executed by the system, without the need for user intervention. There is no need for control modules functions to be called by the programmer in any of the user function blocks. The PLC frame always executes all control modules that are present in the data blocks of all the control module types.

You can modify the number of control modules of a type by reducing the number of modules contained in the data block of this type.

Additionally, you can limit the maximum number of modules to be executed in the “Bx SysBegin” function by adjusting the amount for each of the control module processing calls.

Network 4: Digital Input / Analog Input

```
// Digital Input
UC  "TransDIn System"
UC  "TransDIn User"

CALL "Bx DIn"
Amount:=L#400

// Analog Input
UC  "TransAIn System"
UC  "TransAIn User"

CALL "Bx AIn"
Amount:=L#240
```

Data Structure

There is one data block for each control module type, which contains all the modules of this type. The data block contains a list of structures of a control module type, where each element corresponds to a specific module, ordered by their module number. This structure contains Commands, Status, and parameters as well as help values of each module.

Commands

This section describes the data that functions as commands from the user's program to the BatchXpert system. The signals described in this section can be written into the user's program with the corresponding restrictions for each signal.

These signals are usually commands that activate the corresponding function in the control modules. Generally (with a few exceptions), these are "Write only" signals.

All commands automatically reset each plc cycle, so as soon as you do not set them to TRUE, they reset automatically. This is to ensure that no Command can ever become permanently activated by programmer error.

Status

These are data and signals that the BatchXpert system provides for use in the user's program.

From these statuses, the user can obtain different information about the general status of the control modules. Generally (with a few exceptions), these are "Read only" signs.

Parameters

These are configurations of the different control modules. Normally this configuration is done through the "Faceplate" of the BatchXpert HMI systems. Normally, they are not written by the PLC.

However, for some signals there is the possibility of overwriting the values in the plc, so that the user cannot change them in the control modules faceplates. This is not encouraged, as it runs contradictory to the BatchXpert control module philosophy, where the user can manipulate these values. However, it may still be desirable for safety relevant parameters, for example, reset simulations at emergency stop.

Alarms

In the plc many control modules can have an Alarm condition. This alarm condition is represented by the "Gal" (General Alarm) status signal. This "Gal" signal stays true if the alarm condition is still present.

There is also the "Gals" (General Alarm Saved) signal. It activates at the same time as the Gal signal but stays TRUE until reset by the operator either by "Acknowledge" on the specific Module, or by triggering a global "Acknowledge All" by setting the "QuittAll M878.7" global signal.

How these Alarm conditions are evaluated, and the Gal signal is generated depends on the control module type and includes a delay timer before the alarm is triggered.

Ignore

Most control modules also include an "Ignore" mode which can be activated by the Operator (Ign Parameter) and means that the "Gal" and "Gals" signals are not set, even if the alarm condition of the control module is met.

Simulation

Most control modules also include an "Simulation" mode which can be activated by the Operator (Sim Parameter) and means that the modules process value or signal is not connected to the Plc Inputs and outputs anymore but can be defined from the operating interface. How this simulation mode is implemented depends on the specific Control Module type.

External signals

Many control modules have signals that they must process that have to be supplied "externally," meaning from outside of the control module. These signals are marked with an "x" for "External" in their symbol name.

For example, the actuator needs to know its physical feedback status signals, which must be supplied by the user program to the actuators. These signals are "xFba1" and "xFba2" which will get converted to "FbaOn" and "FbaOff". Since these latter two do not have the "x" in their symbol, they are statuses, not external signals.

Actuators (Act)

Actuators are all elements that can be activated from the automation point of view. These may be Valves, Pumps, or simple Lamp or other single digital outputs. They can have one Output, and optionally a closed and an open Feedback. The actuator associated with the physical output is realized in the FC 501 "TransAct System" and the "Transxx User".

An Actuator may also be a purely virtual actuator, which has no real IO connections, neither output, nor feedback. This is useful for example for Regulation values, Frequency converters etc. They usually do not have a digital output, but rather a communication or and 4-20mA loop, but it is still useful to have a discrete Actuator to activate and manipulate them.

General Principle

An actuator has one digital output and two digital inputs to represent the Feedback. Feedback is optional. It constantly monitors feedback against the desired state, indicated by the output.

- If the Output is TRUE, the actuator is opening, but the ON feedback is not TRUE, it starts the alarm timer.
- If the output is FALSE, the actuator is closing, but the Off feedback is not TRUE, it starts the alarm timer.
- If both the On and Off feedback are TRUE, the alarm timer starts.
- If the Alarm Time expires, it activates an Alarm.

If the Feedback signals matches the expected state, the internal "On" and "Off" signals are activated, respectively. These are the Control modules signals, which should be used in the user program to check if the actuator is in the "Activated" or "Deactivated" state.

Simulation

If an Actuator is simulated, it will always generate the correct "On" and "Off" feedback signals for the modules. This means that it will ignore the actual feedback signals connected to it, and always assume the correct state, thus never generating any errors.

Ignore

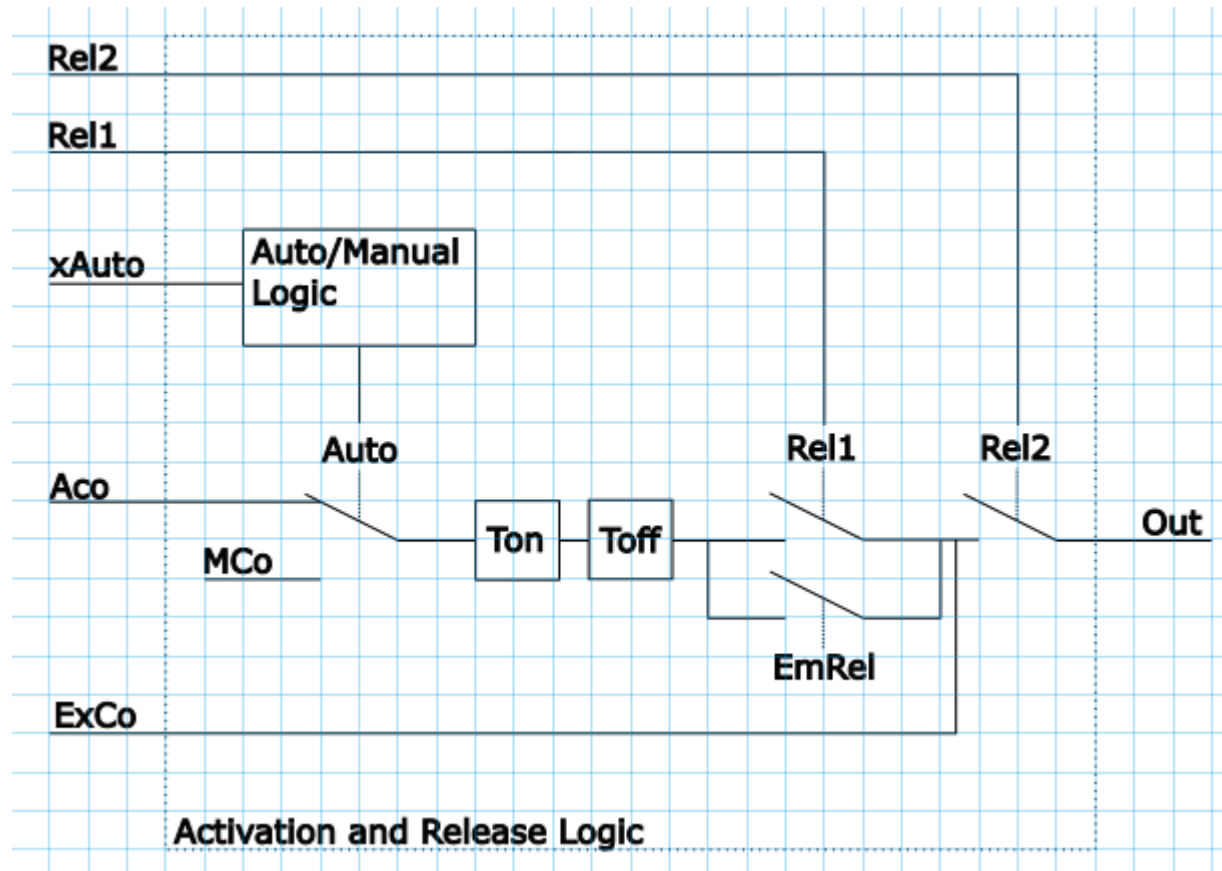
The Ignore mode blocks the activation of the alarm condition "Gal" and "Gals" signals from activating. However, the On and Off signals will remain connected to the connected feedback, although they will never generate any error condition anymore.

For Actuators, you should prefer the "Simulation" mode, because "Ignore" may hide error conditions, without simulating the Feedback signals.

Interlocks

Programming interlocks (also called Releases) for the actuators is one of the more complicated parts of writing a BatchXpert application. [You can find some examples here.](#)

Function diagram



Structure

The Structure of this control modules is as follows.

Address	Symbol	Type	Remark
0.0	ACo	BOOL	automatic control
0.1	ExCo	BOOL	extern control
0.2	SCS	BOOL	status check start
0.3	xFBa1	BOOL	feedback 1
0.4	xFBa2	BOOL	feedback 2
0.5	Rel	BOOL	release
0.6	Rel2	BOOL	release 2
0.7	xAuto	BOOL	External automatic
1.0	ACoHM	BOOL	automatic control help memory
1.1	ExCoHM	BOOL	Extern control help memory
1.2	FBaOn	BOOL	feedback ON intern
1.3	FBaOff	BOOL	feedback OFF intern
1.4	FBaChange	BOOL	change extern feedback (0 FBa1=OFF FBa2=ON / 1 FBa1=ON FBa2=OFF)
1.5	FBa1Active	BOOL	feedback 1 active
1.6	FBa2Active	BOOL	feedback 2 active
1.7	xAutoHM	BOOL	extern automatic old
2.0	GAIQuitt	BOOL	general alarm acknowledges
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	Auto	BOOL	automatic mode
2.4	MCo	BOOL	manual control
2.5	EmRel	BOOL	emergency release
2.6	InterlockGAI	BOOL	interlock by alarm
2.7	Maint	BOOL	maintenance
3.0	GAI	BOOL	general alarm
3.1	GAIS	BOOL	general alarm saves
3.2	SCE	BOOL	status check error
3.3	Mov	BOOL	Actuator is moving for visu
3.4	On	BOOL	actuator is ON
3.5	Off	BOOL	actuator is OFF
3.6	Out	BOOL	output
3.7	User	BOOL	free for user program
4.0	TOnVal	REAL	turn on delay value
8.0	TOnSp	REAL	turn on delay setpoint
12.0	TOfVal	REAL	turn off delay value
16.0	TOfSp	REAL	turn off delay setpoint
20.0	ADVal	REAL	alarm delay value
24.0	ADSp	REAL	alarm delay setpoint
28.0	TInterlock	REAL	time interlock before restart
32.0	SwCntVal	DINT	switch counter value
36.0	RunTimeVal	DINT	duty timer value (seconds)

Commands

Symbol	Remark
ACo	Actuator control, only effective in Automatic
ExCo	External control (e.g., by a switch), effective in automatic and manual. Production-related interlocks are ignored. Safety interlocks are still considered.
SCS	status check start
xFBa1	feedback 1
xFBa2	feedback 2
Rel	release
Rel2	release 2
xAuto	External automatic This signal is usually connected to the "Run" status of its corresponding unit.

Status

Symbol	Remark
FBaOn	feedback ON intern
FBaOff	feedback OFF intern
GAI	general alarm
GAIS	general alarm saves
SCE	status check error
Mov	actuator is moving
On	actuator is ON
Off	actuator is OFF
Out	output

Parameters

Symbol	Remark
FBaChange	change extern feedback (0 FBa1=OFF FBa2=ON / 1 FBa1=ON FBa2=OFF)
FBa1Active	feedback 1 active
FBa2Active	feedback 2 active
GAIQuitt	General Alarm Acknowledge
Ign	ignore alarm
Sim	simulation
Auto	automatic mode
MCo	manual control
EmRel	emergency release
InterlockGAI	interlock by alarm
Maint	maintenance
TOnSp	turn on delay setpoint
TOfSp	turn off delay setpoint
ADSp	alarm delay setpoint
TInterlock	time interlock before restart

Faceplate

42	11.01V11	outlet valve lauter tun	P: 5.1 / 205.1	Win Time / sec:	999.00	52.36
----	----------	-------------------------	----------------	-----------------	--------	-------

Function

1 **Auto** 0

Auto Control

1 **Manu Control** 0

Extern Control

Prod. Release

1 **Maunual Rel** 0

Security Rel

1 **Maintanance** 0

FBa OFF Output **FBa ON**

Alarm Status

☐ **Alarm** 0

Status Check

1 **Ignore** 0

1 **Simulation** 0

1 **Lock by Alarm** 0

Input parameters

1 **FBa 1 active** 0 **FBa 1 Status**

1 **FBa 2 active** 0 **FBa 2 Status**

1 0

Switch Counts + Running time **Reset**

Switch Counts	0	42
Running time / h:	0.00	6.29
Delay ON / sec:	0.00	0.00
Delay OFF / sec:	0.00	0.04
Delay alarm / sec:	0.00	75.50
Time interlock / sec:	0.00	

Special Configurations

In addition to the system window of the actuators, the default parameter settings are made, there is a window for the mouse parameterization. This determines what should happen when you click your mouse over the item. In addition, in the mouse parameterization you can even set the Manual/automatic behavior generally:

	Mouse Click	
	SET	RESET
Quitt Alarm:	1 0	
Ignore	1 0	1 0
Simulation	1 0	1 0
Automatic:	1 0	1 0
Man. Control	1 0	1 0
Lock by Alarm:	1 0	1 0
Maintenance	1 0	1 0
Emerg. Rel:	1 0	1 0
FBa 1 active:	1 0	1 0
FBa 2 active:	1 0	1 0
FBa 1 <-> 2:	1 0	1 0

Automatic Philosophie

ACO set Auto 1 0

Unit Auto Impuse set Auto 1 0

Unit Auto set Auto 1 0

- Automatic control by actuator. If you drive an actuator, it is usually in automatic mode. Switching to manual mode is not always possible when the actuator has a program effect. This corresponds to the automatic philosophy of many programs in the fermentation cellar.
- Automatic Edge Unit (RUN) is the only actuator mode in Automatic. Disabled, the RUN flank can be activated manually at any time.
- Auto Unit (RUN) sets the automatic mode of the actuator. While the corresponding unit is on RUN it cannot be switched into manual actuator mode.
- If one of these options is selected, the operator can manually interrupt it at any time.
- Switching from manual mode to automatic mode is always possible at any time.

IO Assignment Behavior

If the "xFba1" or "xFba2" are not written to in any of the "Transfer IO blocks", the internal status of the control module always assumes the correct internal status as to not generate an error. This means that if you have actuators that for example do not have "On-Feedbacks", you just must leave the "xFba1" signal "Floating", meaning never write to it. The Actuator module will then automatically assume a status as to not create an alarm and your actuator module will always produce the correct ".On" signal for your Actuator. If you have an Actuator module, that does not have any feedbacks, just do not write on neither "xFba1" nor "xFba2".

Please keep in mind that each Actuator Module provides an ".On" and ".Off" signal, which should be used in your applications. Please do not use ".xFba1" or ".Fba1" for any logic in your application and rather use the ".On" and ".Off" signals.

It should be noted that the IO assignment of control modules is generally done by generating this code using the "project engineering tools". Please refer to [The "Trans xx" blocks for IO signal transfer](#) for more details

Actuator with ON and OFF feedback

```
//Send output to Output card
U "Act". Act[1]. Out           //Signal to be activated the physical output
= A 0.0                        //Physical Output

//Since we are writing a value to both "xFba1" and "xFba2", both of them
//are active and the Module will consider both signals to generate the
//"On" and "OFF" signals of an actuator Module.
U E 0.0                        //Actuator Feedback 1
= "Act". Act[1].xFBa1         //Turning on Feedback 1

U E 200.0                     //Actuator Feedback 2
= "Act". Act[1].xFBa2         //Enable Feedback 2
```

Actuator with only ON and no OFF feedback

```
//Send output to Output card
U "Act". Act[1]. Out           //Signal to be activated the physical output
= A 0.0                        //Physical Output

//Since we are writing a value only to "xFba1" and never to "xFba2",
//the module will always assume that the "Off"-Feedback is true when the
//Actuator Output is turned off. Meaning, it will always have the appropriate
//OFF-Feedback, according to the module status.
U E 0.0                        //Actuator Feedback 1
= "Act". Act[1].xFBa1         //Turning on Feedback 1
```

Actuator with no feedback

```
//Send output to Output card
U "Act". Act[1]. Out           //Signal to be activated the physical output
= A 0.0                        //Physical Output

//Since we are writing neither ".xFba1" nor ".xFba2",
//the module will always have the appropriate Feedbacks, according to the
//module status.
```

Do **NOT!** Simulate Feedbacks like the following

```
//Send output to Output card
U "Act". Act[1]. Out           //Signal to be activated the physical output
= A 0.0                       //Physical Output

//Do not simulate the feedback, by writing the modules Out on the feedbacks.
//While this may work, you should prefer to leave both signals "Floating" as
//described above.
U "Act". Act[1]. Out           //Actuator Feedback 1
= "Act". Act[1].xFBa1         //Turning on Feedback 1

UN "Act". Act[1]. Out          //Actuator Feedback 2
= "Act". Act[1].xFBa2         //Turning on Feedback 2
```

Programming Examples

Automatic Process Control

```
U "PH"                         //activated while in that step and in "Run" (or pause)
S "Act". Act[42]. Aco          //Actuator 42 will be activated automatically.
S "Act". Act[44]. Aco          //Actuator 44 will be activated automatically
```

Note: The "Aco" signals are automatically reset each plc cycle, so even if you use the "S" command instead of an "=", the signal will not remain "TRUE". Using the "S" command makes using the "Aco" signal of actuators much more versatile and easier, as they can be activated by multiple places at the same time without conflicting.

Assign Automatic mode!

```
U "RUN"                        //Process in "Run" mode
S "Act". Act[42].xAuto         //Enable Actuator Automatic Mode 42
S "Act". Act[44].xAuto         //Enable Actuator Auto Mode 44
```

External Control

```
U "Din". Deen[15].Sig          //Safety switch
S "Act". Act[42].ExCo          //Activates the actuator from an external control
```

Release

For more information about Interlocks and Releases, please refer to [Actuator Interlocks](#)

```
//Safety Release
U "Din". Deen[11].Sig          //Manhole
U "Din". Deen[10].Sig          //Emergency Stop
= "Act". Act[42].Rel           //Conditional Safety Release

//Release by process
U "Act". Act[40]. Off           //Valve 1 off
U "Act". Act[41]. Off           //Valve 2 off
= "Act". Act[42]. Rel2         //Process-Conditioned Release
```

Alarm evaluation.

```
U "Act". Act[42].GAIs          //Actuator on alarm
S "HoldReq"                    //Maintains unity
```

Assignment for IO

For more information about IO assignments please refer to "[IO Assignment Behavior](#)".

```
//Actuator 1
U "Act". Act[1]. Out           //Signal to be activated the physical output
= A 0.0                       //Physical Output
```

```
U E 0.0 //Actuator Feedback 1
= "Act". Act[1].xFBa1 //Turning on Feedback 1

U E 200.0 //Actuator Feedback 2
= "Act". Act[1].xFBa2 //Enable Feedback 2
```

Actuator Interlocks

The following describes some of the general interlocking and releasing of the Actuator Modules in common process situation. However, there are also many specific interlocks, which for the sake of brevity are not all described in this manual. The following section should be seen as a guide for implementing common interlocking scenarios with actuator control modules.

Emergency Stop

If an emergency stop is activated all energized equipment related to the emergency stop must be turned off immediately. This includes all pumps, valves and other energized equipment. In the programming you should use the “.Rel” signal for this purpose, since this signal cannot be “bridged” by the user from the faceplate of the control module. Emergency stops should also be programmed using the “.Gals” signal from the emergency stop digital input, since this signal only resets when the operator effectively acknowledges the alarm on the system.

A special case of emergency stops are vessel or equipment specific safety equipment, such as key switches or manhole sensors on tanks. If one of these sensors is not in the correct state all equipment that is energized and directly connected to the inside of the tank or can provoke dangerous situations on the connected machines, must also be immediately shut off.

Pumps

Pump interlocks usually are one of the most complex interlocking scenarios, since they involve many different components and alarm conditions. All process interlocks should be programmed using the “.Rel2” signal, since this signal can be bridged from the faceplate by the user in the case the pump needs to be turned on even though the usual process conditions are not met.

You should separate the interlocks into a suction side and pressure side monitoring where for each side of the pump you can program all valves that have to be open for the pump to be able to run, without running against closed valves. This way you can ensure that the pump can only turn on if and valid suction and invalid pressure side way is opened for the pump.

If the pump includes a flow meter, you should also activate the low low alarm on this flow meter when the pump turns on and use the flow meter as flow monitoring device. If deep flow meter analog input activates the low-low alarm, the pump must stop since no flow is detected.

Another common interlock for pumps is the “dry run protection”. Sometimes pumps include dry run protection sensors in the suction side of the pump, which of course should only allow the pump to run if liquid is present on the suction side. If the pump is a transfer pump of a tank or a vessel, and this vessel includes an empty sensor (LSL), you can use this empty sensor as try to run protection for the pump. Usually, you want to adjust about 20 to 30 seconds of alarm delay on either the dry run protection sensor or the empty level signal of the tank.

Full Signals (LSH)

If a tank or vessel includes and full signal (LSH), all valves and pumps must be blocked to avoid overfilling of that tank on this. Depending on the situation and conditions sometimes along delays may be applied to these full level signals. These full level signals usually are discrete digital level switches, or analog volume measurement devices.

On some vessels you may also have recirculation functionality, in which case you should not block recirculation if the full level sensor detects maximum level.

One other common functionality is that conductive full level sensors may not work during CIP processes, because the CIP solutions may adhere to the sensor and give false full level signal readings. For this reason, usually during CIP you should activate ignore on conductive full level switches or not use conductive full level switches in equipment that may have contact with sodium hydroxide-based cleaning solutions. This problem mostly occurs with sodium hydroxide since this cleaning agent is relatively sticky and very conductive, giving false conductivity LSH sensor readings.

Dry Run Protection (LSL)

Another common interlock for pumps is the “dry run protection”. Sometimes pumps include dry run protection sensor in the suction side of the pump, which of course should only allow the pump to run if liquid is present on the suction side. If the pump is a transfer pump of a tank or a vessel, and this vessel includes an empty sensor (LSL), you can use this empty sensor as dry run protection for the pump. Usually, you want to adjust about 20 to 30 seconds of alarm delay on either the dry run protection sensor or the empty level signal of the tank.

Maintenance Switches

Maintenance switches are interrupters installed near energized equipment such as pumps or motors and allow the maintenance personnel to locally disconnect this type of equipment. Whenever possible you should include feedback signals from these switches so that you can activate alarms and block the actuators from turning on.

Manholes and Key switches

Many tanks include equipment to detect if somebody might be inside the machine or tank, such as key switches or manhole sensors on tanks. If one of these sensors is not in the correct state all equipment that is energized and directly connected to the inside of the tank or can provoke dangerous situations on the connected machines, must also be immediately shut off. Particularly agitators must be blocked immediately if the equipment detects that some person might be operating inside the machine.

Programming Examples

Safety Interlocks

Here we handle all Safety related Interlocks. Safety Interlocks cannot be "Bridged" by the user, and are meant to reflect the interlocks for safety relevant equipment, or where Electrical interlocks exist in addition to the software interlocks.

For Electrical interlocks, "Bridging" an Interlock would not allow the user to activate the Actuator, because it will still be blocked by the safety circuit. To avoid confusion, we can mark them as "Safety Interlock"

```
//Here Are Actuator Interlocks, which depend on the Emergency Stop, but are not
//Connected directly to the interior of the Tank, so the "Vessel" safety does not
//Apply to them
U      #EmStop
=      "Bx Act D".Act[27].Rel
=      "Bx Act D".Act[556].Rel
=      "Bx Act D".Act[555].Rel
=      "Bx Act D".Act[432].Rel
=      "Bx Act D".Act[123].Rel
=      "Bx Act D".Act[334].Rel

//Here Are Actuator Interlocks, which are Connected directly
//to the interior of the Tank, so the "Vessel" safety does
//Apply to them
U      #EmStop
U      #VesselSafe
=      "Bx Act D".Act[28].Rel
=      "Bx Act D".Act[56].Rel
=      "Bx Act D".Act[55].Rel
=      "Bx Act D".Act[42].Rel
=      "Bx Act D".Act[13].Rel
=      "Bx Act D".Act[34].Rel

//Actutors that are not subject to any of the safety interlocks, come here.
//This however usually only applies to "Lamps", "Accoustic indicators" and such,
//And NEVER to Energized equipment such as Valves or Pumps.
//Energized equipment must always be interlocked by at least the Emergency Stop
SET
=      "Bx Act D".Act[13].Rel
=      "Bx Act D".Act[34].Rel
```

Simple Actuator Interlocks

In this example we look at simple Process interlocks.

```
//if a Digital input is in alarm, we block the actuators
UN      "Bx DIn D".DIn[20].GAlS          //High temperature alarm
=       "Bx Act D".Act[27].Rel2          //Heating Valve

//An actuator needs another Actuator to be running first, for example a valve must
//Be open first
U       "Bx Act D".Act[27].On
=       "Bx Act D".Act[27].Rel2

//The actuator may never be active during CIP
UN      #CIP
=       "Bx Act D".Act[27].Rel2
```

More Simple Interlocks

```
NETWORK
TITLE = Act45 - dosing pump CaCl2 lauter tun
//This pump may only run when the signal is true and the actuator 125 is off
U       "Bx DIn D".DIn88.Sig;
U       "Bx Act D".Act125.Off;
=       "Bx Act D".Act45.Rel2;

NETWORK
TITLE = Act 112 - valve CIP return lauter tun
//this Valve may only open if the other one is closed

U       "Bx Act D".Act107.Off;
=       "Bx Act D".Act112.Rel2;

NETWORK
TITLE = Act124 inlet valve 1 lauter tun
//the inlet can only open if the tank is not full
U       #VesselSec;
UN      "Bx AIn D".AIn12.MHHA;
=       "Bx Act D".Act124.Rel2;

NETWORK
TITLE = Act127 - valve weak wort to weak wort tank
//There is no Interlock, it is always enabled

SET     ;
=       "Bx Act D".Act127.Rel2;
```

Pump Actuator Interlocks

This is a more complex Pump interlocking example. Pump interlocks are the most "complex" interlocks, since they involve a lot of components.

Pumps you want to include the following Interlocks/Releases:

- At least one Suction side Way open
- At least one Pressure side Way open
- If a flow meter is in line, activate the LLA alarm that flowmeter, when the pump runs, and then block the pump if the alarm gets activated. This way you can add "Flow Monitoring" to the pump
- If you have Dry run Protection, you want that to be monitored as well
- If you have an LSL of a tank, you want that also

```
//Here we preset the signals, so we can later "S" them and avoid Parentheses and
complex AND/OR constructs
CLR
=      #Suction;
=      #Pressure;

//Suction side
U      "Bx Act D".Act155.On;
U      "Bx Act D".Act156.Off      //This vulve must be cloed
=      #Suction;

//Pressure to PRV
U      "Bx Act D".Act132.On;
U      "Bx Act D".Act143.On;
U      "Bx Act D".Act324.On;
S      #Pressure;

//Circulation
U      "Bx Act D".Act126.On;
U      "Bx Act D".Act324.On;
S      #Pressure;

//Weak Wort
U      "Bx Act D".Act127.On;
U      "Bx Act D".Act324.On;
S      #Pressure;

//General Release
U      #Pressure;
U      #Suction;
UN     "Bx AIn D".AIn84.GAlS;      //Flow Monitoring
UN     "Bx DIn D".DIn84.GAlS;      //Dry Run monitoring, either by a Dry Run switch, or the LSL of the Vessel
=      "Bx Act D".Act304.Rel2;
```

Digital Inputs (DIn)

Digital Inputs are single input points, and used for signals like empty signals, full signals, or other button signals from the process field. They can be on and off delayed, and alarm conditions can be set when the alarm should be activated.

General Principle

A Digital input has one single input assigned to it.

- If the external signal becomes TRUE it starts the On-Delay timer and then activates the internal "Sig," which should be used in the user program.
- If the external signal becomes FALSE, the Off-Delay timer starts,
- If the "EaI0" (Enable alarm on FALSE) is TRUE and the Signal becomes FALSE, the Alarm delay time starts.
- If the "EaI1" (Enable alarm on TRUE) is TRUE and the Signal becomes TRUE, the Alarm delay time starts.
- If the Alarm Time expires, it activates an Alarm.

Simulation

If a Din is simulated, the current internal Signal state can be simulated by the user, and the field signal is completely ignored.

Structure

Address	Symbol	Type	Remark
0.0	EA0	BOOL	enable alarm by 0-signal
0.1	EA1	BOOL	enable alarm by 1-signal
0.2	SCS0	BOOL	status check alarm by 0-signal
0.3	SCS1	BOOL	status check alarm by 1-signal
0.4	xSig	BOOL	signal extern
0.5	B29	BOOL	spare
0.6	B30	BOOL	spare
0.7	B31	BOOL	spare
1.0	AIHM	BOOL	help memory for alarm
1.1	ImpHM	BOOL	help memory for impulse
1.2	xSigHM	BOOL	signal extern help memory
1.3	B19	BOOL	spare
1.4	B20	BOOL	spare
1.5	B21	BOOL	spare
1.6	B22	BOOL	spare
1.7	B23	BOOL	spare
2.0	GAIQuitt	BOOL	general alarm acknowledges
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	iEA0	BOOL	intern alarm by 0
2.4	iEA1	BOOL	intern alarm by 1
2.5	ImpProt	BOOL	write impulse flank to protocol
2.6	ImpNegProt	BOOL	write negative impulse flank to protocol
2.7	Switch	BOOL	convert as switch output
3.0	GAI	BOOL	general alarm
3.1	GAIS	BOOL	general alarm save
3.2	SCE	BOOL	status check error
3.3	Sig	BOOL	signal state
3.4	Imp	BOOL	impulse flank
3.5	ImpNeg	BOOL	negative impulse flank
3.6	B06	BOOL	spare
3.7	User	BOOL	free for user
4.0	TOndVal	REAL	turn on delay value
8.0	TOndSp	REAL	turn on delay setpoint
12.0	TOfdVal	REAL	turn off delay value
16.0	TOfdSp	REAL	turn off delay setpoint
20.0	ADVal	REAL	alarm delay current value
24.0	ADSp	REAL	alarm delay setpoint
28.0	SwCntVal	DINT	switch counter value

Commands

Symbol	Default	Remark
EA0	0	enable alarm by 0-signal
EA1	0	enable alarm by 1-signal
SCS0	0	status check alarm by 0-signal
SCS1	0	status check alarm by 1-signal
xSig	x	signal extern
User	x	free for user

Status

Symbol	Remark
GAI	general alarm
GAIS	general alarm save
SCE	status check error
Sig	signal state
Imp	impulse flank
ImpNeg	negative impulse flank

Parameters

Symbol	Remark
Ign	ignore alarm
Sim	Simulation
iEA0	intern alarm by 0
iEA1	intern alarm by 1
ImpProt	write impulse flank to protocol
ImpNegProt	write negative impulse flank to protocol
Switch	convert as switch output
TOnDsp	turn on delay setpoint
TOfDsp	turn off delay setpoint
ADSp	alarm delay setpoint
SwCntVal	switch counter value

Faceplate

1	08.01.01 LSL	empty signal mash tun 1	I: 400.0	Win Time / sec:	999.00	19.52
---	--------------	-------------------------	----------	-----------------	--------	-------

Input ext.

1 Switch 0

Signal

1 Alarm by 0 0 1 Alarm by 1 0

Alarm 0

Status Error

1 Ignore 0

1 Simulation 0

1 Imp to Prot. 0

1 NegImp to Prot 0

Alarm by 0

Alarm by 1

St. Error by 0

St. Error by 1

Time On Delay / sec:	0.00	0.00
Time Off Delay / sec:	0.00	0.04
Time Alarm Delay / sec:	7.70	0.00
Switch Counter	16	Reset

Special Configurations

In addition to the system window for digital input for default parameter settings, there is a window for mouse settings. This determines what should happen when you click the mouse over the item.

	Mouse Click	
	SET	RESET
Quitt Alarm:	1 <input checked="" type="radio"/> 0	
Ignore	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Simulation	1 <input checked="" type="radio"/> 0	1 <input type="radio"/> 0
Signal	1 <input checked="" type="radio"/> 0	1 <input checked="" type="radio"/> 0
Alarm by 0	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Alarm by 1	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Switch	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Imp to Prot:	1 <input type="radio"/> 0	1 <input type="radio"/> 0
NegImp to Prot:	1 <input type="radio"/> 0	1 <input type="radio"/> 0

- The digital signal of the sensor can be simulated in case of problems (non-process-critical signal, such as a transport sensor).
- Ignore alarms.

Programming Examples

Signal Query

```
U "Din.Din[19].Sig"           //Empty signal
S "PhaseEnd"                  //Finish Step
```

Alarm Assessment

```
U "Step0"                     //Unit is in "StartPosition"
S "DIn".DIn[18].EA0           //Activate Alarm Signal 0

U "DIn".DIn[18].GAlS          //Alarm
S "HoldReq"                   //Stop the current Unit
```

Status check

```
U "PA"                        //Active Step
S "DIn".DIn[12].SCS0          //Status Error Triggered with Signal 0

UN "DIn".DIn[12].SCE          //No Error Status
S "PhaseEnd"                  //Finish Step
```

DIn Assignment

It should be noted that the IOWA assignment of control modules is generally done by generating this code using the project engineering tools. Please refer to [The "Trans xx" blocks for IO signal transfer](#) for more details

```
U E 400.0                      //Physical Entry Address
= "DIn".DIn[1].xSig            //Active Program DiN 1 Signal

U E 599.7                      //Physical Entry Address
= "DIn".DIn[1600].xSig         //Active Program Signal DiN 1600
```


Analog Input (AI_n)

An Analog input represents a single analog measurement. These are usually connected by a 4-20mA current loop, 0-10 V voltage signal, or by an RTD resistor bridge. The module supports Scaling of the input measurement signal to the appropriate process signal and applying an "Polygon table" to the raw input value to perform complex operations.

The Scaling operation can be disabled for values, which do not require this functionality.

General Principle

An Analog input has one single Analog input signal assigned to it.

- If the NPA (no Periphery Adaption) is FALSE, the normal Scaling is applied, and the result is transferred to "PVal"
- If the NPA (no Periphery Adaption) is TRUE, the normal Scaling is applied not applied and the "xPVal" is directly transferred to the "PVal"
- After the scaling is applied, a polygon is applied if one is selected.
- If the "PVal" is above or below the HHval or LLval limits, a Warning is activated.
- If the "PVal" is above or below the HHAval or LLAval limits, the alarm delay is started.
- If the Alarm Time expires, it activates an Alarm

The Analog input has several Limit signals that have Hysteresis applied to them. **The hysteresis** of the values generates a "dead band" to prevent the corresponding signal from fluctuating too much. The hysteresis function with all limit values and is always above and below the set value. If the process value becomes less than the value set at the minus hysteresis limit, the corresponding limit signal is deactivated. The limit signal is lit again, if the process value becomes more than the limit value plus hysteresis.

For example: If the limit value is set to 90 with a hysteresis of 30.

The Limit Signal is turned off when the process value is below $90 - 30 = 60$.

If the process value goes up again, the Limit Signal turns back on when the value reaches $90 + 30 = 120$.

Simulation

If an AI_n is simulated, the current internal Process Value can be simulated by the user, and the field signal is completely ignored

Filters

There are three filters that can be applied to the Analog input. All three filters can be added simultaneously, as they are additives. The filters are "Low Pass Filters" of different "Strength." The higher the filter percentage, the more they "smooth" out signals.

If you have very fluctuating signals, you can add these "Filters" until the signals are usable for your application.

Structure

Address	Symbol	Type	Remark
0.0	ELLA	BOOL	enable low low alarm
0.1	EHHA	BOOL	enable high high alarm
0.2	xAI	BOOL	alarm from extern
0.3	NPA	BOOL	no periphery adaption
0.4	B28	BOOL	spare
0.5	B29	BOOL	spare
0.6	B30	BOOL	spare

0.7	B31	BOOL	spare
1.0	MLLA	BOOL	low low alarm - alarm if enabled
1.1	MLL	BOOL	low low limit - warning if enabled
1.2	ML	BOOL	low limit
1.3	MSp	BOOL	setpoint
1.4	MH	BOOL	high limit
1.5	MHH	BOOL	high high limit - warning if enabled
1.6	MHHA	BOOL	high high alarm - alarm if enabled
1.7	MHWA	BOOL	alarm from hardware
2.0	GAIQuitt	BOOL	general alarm acknowledge
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	iEHWA	BOOL	enable hardware alarm
2.4	iELLA	BOOL	enable LL alarm
2.5	iEHHA	BOOL	enable HH alarm
2.6	iELLW	BOOL	enable LL warning
2.7	iEHHW	BOOL	enable HH warning
3.0	GAI	BOOL	general alarm
3.1	GAIS	BOOL	general alarm saves
3.2	Warn	BOOL	general warning
3.3	Filter1	BOOL	filter 1 on (75%)
3.4	Filter2	BOOL	filter 2 on (88%)
3.5	Filter3	BOOL	filter 3 on (94%)
3.6	ManuInp	BOOL	manual input (no periphery)
3.7	User	BOOL	memory free for user
4.0	PVal	REAL	process value
8.0	Sp	REAL	setpoint
12.0	LScal	REAL	low scaling
16.0	HScal	REAL	high scaling
20.0	LLAVal	REAL	low low alarm value
24.0	LLVal	REAL	low low value (warning limit)
28.0	LVal	REAL	low value
32.0	HVal	REAL	high value
36.0	HHVal	REAL	high high value (warning limit)
40.0	HHAVal	REAL	high high alarm value
44.0	LLAHys	REAL	low low alarm hysteresis
48.0	LLHys	REAL	low low hysteresis
52.0	LHys	REAL	low hysteresis
56.0	SpHys	REAL	setpoint hysteresis
60.0	HHys	REAL	high hysteresis
64.0	HHHys	REAL	high high hysteresis
68.0	HHAHys	REAL	high high alarm hysteresis
72.0	ADVal	REAL	alarm delay value
76.0	ADSp	REAL	alarm delay setpoint
80.0	PoTNo	REAL	positive = polygon table number / negative = offset
84.0	xPVal	REAL	raw value from extern
88.0	iPVal	REAL	process value intern (without polygon)

Command

Symbol	Default	Remark
ELLA	BOOL	enable low low alarm
EHHA	BOOL	enable high high alarm
xAI	BOOL	alarm from extern
NPA	BOOL	no periphery adaption
User	BOOL	memory free for user
xPVal	REAL	raw value from extern

Status

Symbol	Remark
MLLA	low low alarm - alarm if enabled
MLL	low low limit - warning if enabled
ML	low limit
MSp	setpoint
MH	high limit
MHH	high high limit - warning if enabled
MHHA	high high alarm - alarm if enabled
MHWA	alarm from hardware
GAI	general alarm
GAIS	general alarm save
Warn	general warning
User	memory free for user
PVal	process value

Parameters

Symbol	Remark
Ign	ignore alarm
Sim	simulation
iEHWA	enable hardware alarm
iELLA	enable LL alarm
iEHHA	enable HH alarm
iELLW	enable LL warning
iEHHW	enable HH warning
Filter1	filter 1 on (75%)
Filter2	filter 2 on (88%)
Filter3	filter 3 on (94%)
Manulnp	manual input (no periphery)
Sp	setpoint
LScal	low scaling
HScal	high scaling
LLAVal	low low alarm value
LLVal	low low value (warning limit)
LVal	low value
HVal	high value
HHVal	high high value (warning limit)
HHAVal	high high alarm value
LLAHys	low low alarm hysteresis
LLHys	low low hysteresis
LHys	low hysteresis
SpHys	setpoint hysteresis

HHys	high hysteresis
HHHys	high high hysteresis
HHAHys	high hgih alarm hysteresis
ADSp	alarm delay setpoint
PoTNo	positive = polygon table number / negative = offset

Win Time / sec:		9999.00	52.75
1	08.01.21 T1	temperature mash tun 1	P: 1024
Process Value:		10.00 °C	1 Simulation 0
<input checked="" type="radio"/>	High High Alarm:	0.00 0.00	1 <input checked="" type="radio"/> 0
<input checked="" type="radio"/>	High High Warning:	0.00 0.00	1 <input checked="" type="radio"/> 0
<input checked="" type="radio"/>	High Limite:	0.00 0.00	
<input checked="" type="radio"/>	Setpoint:	1.00 0.00	
<input checked="" type="radio"/>	Low Limite:	0.00 0.00	
<input checked="" type="radio"/>	LowLow Warning:	0.00 0.00	1 <input checked="" type="radio"/> 0
<input checked="" type="radio"/>	LowLow Alarm:	0.00 0.00	1 <input checked="" type="radio"/> 0
	High Scale:	100.00	
	Low Scale:	0.00	
	Alarm delay:	0.00 0.00	
		Alarm <input type="radio"/> 0 Warning <input type="radio"/> 1 Ignore 0 1 HWare Alarm 0 1 Manual Input 0 1 Filter 1 0 1 Filter 2 0 1 Filter 3 0 EHHA PLC <input type="radio"/> ELLA PLC <input type="radio"/> No peripherie <input type="radio"/>	
4 ... 20 mA		8.00	0 ... 20 mA 5.00
25.00 →		Poligon / Offset	-15.00 → 10.00 °C

Special Configurations

In addition to the system window for analog input for default parameter settings, there is a window for mouse settings. This determines what should happen when you click the mouse over the item.

In addition to the mouse parameterization, you can determine the overall scale of the inputs:

Mouse Click		
SET	RESET	
<input checked="" type="radio"/>	<input type="radio"/>	Quit Alarm:
<input type="radio"/>	<input type="radio"/>	Ignore:
<input type="radio"/>	<input type="radio"/>	Simulation:
<input type="radio"/>	<input type="radio"/>	Enable HH Alarm:
<input type="radio"/>	<input type="radio"/>	Enable HH Warn:
<input type="radio"/>	<input type="radio"/>	Enable LL Alarm:
<input type="radio"/>	<input type="radio"/>	Enable LL Warn:
<input type="radio"/>	<input type="radio"/>	Hardware Alarm:
<input type="radio"/>	<input type="radio"/>	Filter 1:
<input type="radio"/>	<input type="radio"/>	Filter 2:
<input type="radio"/>	<input type="radio"/>	Filter 3:
<input type="radio"/>	<input type="radio"/>	Manual Input:

Hardware low limite:	0.00
Hardware high limite:	27640.00
Hardware low limite Alarm:	2764.00
Hardware high limite Alarm:	32900.00

- Low Limit Hardware – Scale Division at 4 mA (0 mA)
- High Limit Hardware – Scale Division at 20 mA
- Hardware Alarm Low Limit – If the analog input falls below this value, the wire break alarm is triggered.
- Hardware Limit Alarm High – If the analog input exceeds this value, the overflow alarm is triggered.

Programming Examples

Process Value Transfer

```
L "AIn".AIn[4].PVal      //Temperature Measurement  
T "U002".Para[12].Val    //Unit Parameter 12
```

Alarm Assessment

```
U "Step0"                //Current unit is in Start position  
S "AIn".AIn[18].Ella      //Activate the low low limit alarm  
  
U "AIn".AIn[18].GALS"     //Alarm  
S "HoldReq"              //Maintains Unity
```

AI Assignment

It should be noted that the IO assignment of control modules is generally done by generating this code using the project engineering tools. Please refer to [The “Trans xx” blocks for IO signal transfer](#) and [Analog Input Scaling](#) for more details.

Analog Input Scaling

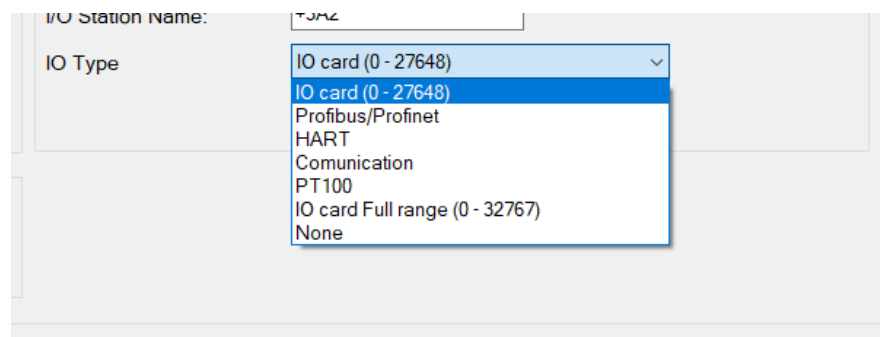
Analog Input control Modules can be assigned from different hardware sources which may have different Scaling values, depending on the Type of Value you are supplying to the Analog Input Module.

Typical Analog input sources

Siemens S7 Analog IO	0 – 27648 correspond to 0 – 100% (4-20mA, 0-10V, ...)	Siemens uses its own Analog input format where all Analog cards are represented by a 14-bit value, and the two upper bits represent error conditions.
Other Analog IO	0-32767 correspond to 0 – 100% (4-20mA, 0-10V, ...)	Some Manufacturer use a different IO Scaling mechanism where they use all 16-bit for its representation, at the cost of not having bits available for error reporting. Most notably “WAGO” IO does this.
PT100	0-1000 correspond to 0.0 – 100.0 °C	If you are using PT100 in two, three or four wire configurations, most of the IO cards give back the current Temperature reading as Integer with one decimal place. So, you must divide by 10 to get the actual temperature reading
Communication	Profinet, ASI-Bus, Modbus, etc.	Most of the time these sources already send a fully resolved and scaled “REAL” value that does not need to be Scaled at all, or you must apply custom scaling factors. Anton Paar measurement devices are an example of this-

Setting Analog input type for Code Generation

BatchXpert allows you to select the type of IO that you want to read and scale according to the Analog inputs parameters. In the “Batch Configurator” or the “Taglist” you can select the type of IO that should be generated for this Analog Input.



This allows the “Project Engineering Tool” to generate an “System Trans Ain” block corresponding to the requirement of the Analog input.

J	K	L
signment	IO Type	Tested
	<div>IO_Card</div> <div>DP</div> <div>HART</div> <div>Communication</div> <div>PT100</div> <div>IO_Card_Full_Range</div> <div>None</div>	

Examples of IO Transfer Code generated by “Project Engineering Tool”

It should be noted that the Io assignment of control modules is generally done by generating this code using the project engineering tools. Please refer to [The “Trans xx” blocks for IO signal transfer](#) for more details

Ain IO-Card (0-27648)

```
L PEW 1024          //We charge the input process value
ITD ;
DTR ;
L      276.48;      //Max Range 27648 for Siemens compatible sensors
/R;                //Range 0 - 100 %
T "Bx AIn D".AIn[1].xPVal //we transfer to the process value of the Ain
```

Ain IO-Card Full Range (0-32767)

```
L PEW 1024          //We charge the input process value
ITD ;
DTR ;
L      327.67;      //Max Range 32767 for Full integer range scaling
/R;                //Range 0 - 100 %
T "Bx AIn D".AIn[1].xPVal //we transfer to the process value of the Ain
```

Ain PT 100 (Value/10)

```
//PT100: Periphery Value is Temperature with one Digit (1000 = 100.0°C)
L PEW 1024;
ITD ;
DTR ;
L 10.0;
/R
T "Bx AIn D".AIn[1].xPVal //we transfer to the process value of the Ain

set
S      "Bx AIn D".AIn[1].NPA;
```

Ain from Communication Sources

Either the Project Engineering Tool or the Programmer can also use the “Per.xxx” functions to transfer Periphery Values to analog inputs. The programmer would write these values in the “User Trans Ain” block.

```
//PT100: Periphery Value is Temperature with one Digit (1000 = 100.0°C)
CALL "Per.DP to AIn"
Value      :=PED1240
AlarmOnNaN:=FALSE //if Input is NaN, it will be 0.0, without an alarm
AIn        := "Bx AIn D".AIn[1]
```


Setting prior to BatchXpert 1.9

Prior to version 1.9 of BatchXpert, the Project Engineering tool did not allow for individual selection of IO types per Analog input control module. There is an “Global” setting that defines the “Scaling Factor” which is set in “OB1” of the PLC.

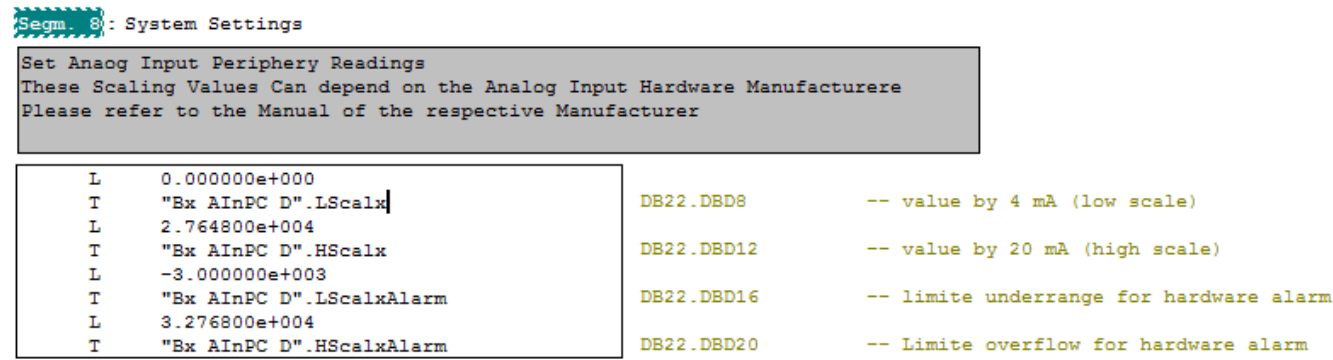


Figure 4 OB1

However, this changes the IO Scaling for all Analog Input modules globally for the whole PLC. This setting is still available, but code generated by the Project Engineering Tool V1.9 or above does not use this setting anymore.

NAN values

If you assign custom values or values coming from a communication source to an Analog input, you should make sure that you do not write “NAN” or any other Invalid Real value onto the Analog Input. The reason for this is that NAN cannot be compared against any other value and thus the “Analog Inputs” High- and Low-level indicators “MHHA” and such do not work!

Because in Simatic S7 NAN values work as follows:

- $\text{NAN} > 0.0 = \text{FALSE}$
- $\text{NAN} < 0.0 = \text{False}$
- $\text{NAN} == 0.0 = \text{False}$

This means all comparisons against an NAN value will always be false, thus never activating either of the Limit indicators of an Analog inputs.

NAN Values usually come from Communication sources, such as “Aton Paar” units, where NAN means, Unit inactive, no Measurement Available.

But they can also come from Arithmetic operations, for example dividing by 0

```
L #SomeProcessValue
L #SomeSetting          //May potentiall be 0.0 if the User inputs 0.0
/R
T #Result                //If #SomeSetting happens to be 0.0, #Result will be "NAN"
```

You should use functions such as “Catch NAN” or “isNAN” to protect against that Possibility

```
CALL "isNan"
Value :=#Value
RET_VAL:=#isNan

U    #isNan
U    #AlarmOnNAN
S    #AIIn.xAl

//Substitute NAN with a "known good Value", as to avoid errors further down the road
//NAN's can really screw things up. Things like this are possible!
//Value == 0 = FALSE, AND Value <> 0 = FALSE! it's not Equal to Zero, but also similar to it!
//it messes up every comparison operator, and most of the "Trend Recording" software, so it may never
//Reach any SCADA system.

UN   #isNan
BEB
L    0.000000e+000          //Replacement value
T    #AIIn.xPVal
```

Custom Values

If you want to send custom values to an Analog input, for example and calculation or similar, you can do that in the “User Trans AIn” function.

Differential Pressure Calculation

```
L "Bx AIn D".AIn[1].PVal    //Pressure Below
L "Bx AIn D".AIn[2].PVal    //Pressure Above
/R                          //Since Pressure Above may be 0.0 there is a chance of NAN
T #PotentialNaNValue

CALL "Catch NAN"
ValueIn      := #PotentialNaNValue
Replacement := 0.0 //We define 0.0 if the result is NAN
ValueOut     := "Bx AIn D".AIn[1].xPVal

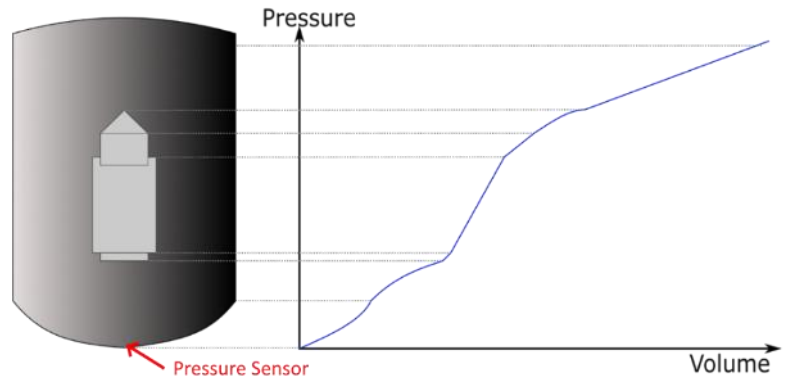
//Since we already did the conversion, we do not need Analog input Scaling
//we set the "No Periphery Adaption"
set
S      "Bx AIn D".AIn[1].NPA;
```

Polygon Tables

Polygon tables are used to convert a value that has a nonlinear dependency to another value. Effectively its “traces” an arbitrary curve and uses the resulting polygon to transform one value to another.

Example premise

An easy example of this would be the calculation of a Tank volume by means of a pressure measurement device at the bottom of the tank. This tank has a non-cone shaped bottom, and on the inside some device that reduces its volume. The Pressure sensor measures the “Water column” above the sensor, and thus essentially measures the water level height. But what we need is the actual volume that is inside the tank. This means:



$$V = f(h)$$

Given the height of the water level derived from the pressure sensor reading, we want to calculate the actual volume of the tank.

Of course, if we had a detailed mechanical construction plan, or even a CAD model, one could determine the volume based on the filling height of the tank. For simpler constructions of tanks, this is straight forward by calculating the volume of a cone and cylinder, but in our example the tank shape is too complex to calculate efficiently. We need a simple solution that we can implement of f in a PLC.

Example Polygon Table

Given the same “pressure to volume” curve in our example above, we can create a list of points where:

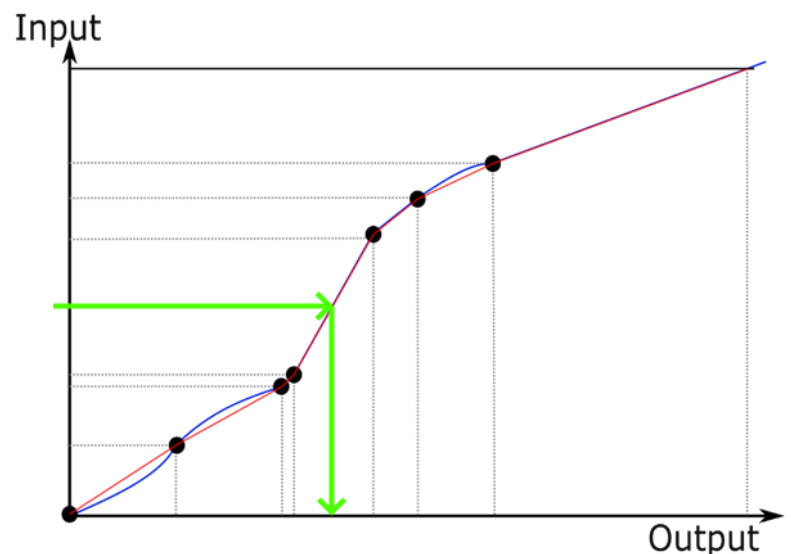
Y = that we measure

X = The Volume that we want

Then we can simply do a linear interpolation between the two nearest Input points that lie above and below our input value (in Green) and then interpolate the output by the same amount.

This effectively gives us an approximation (red) of our original line (blue), with which we can calculate the Output (our Volume) from the input (our pressure) by simply doing some linear interpolation.

Also, the more points we add to this “polygon,” the more accurate our calculation will be.



Creating a Polygon Table

BatchXpert allows you to simply assign these tables for any Analog Input Module by putting the number of the Polygon table to use in the appropriate field of Analog input faceplate. Polygon tables do not need to be created in the PLC, as an amount of 16 already exists by default. If more are needed, the data block can simply be expanded to make room for more polygon tables.

The Polygon window allows you to create as many points as needed (up to 16 points) into a table where you define our “Input” and corresponding “Output” values.

Example for Tank Levels

As an example, to create one of the Tables during commissioning of your project.

Let us assume that you have a Tank or vessel, with a Pressure sensor and a method to fill the tank with a known volume of liquid, usually buy some kind of flow meter or sometimes even by manual transfer (if small enough).

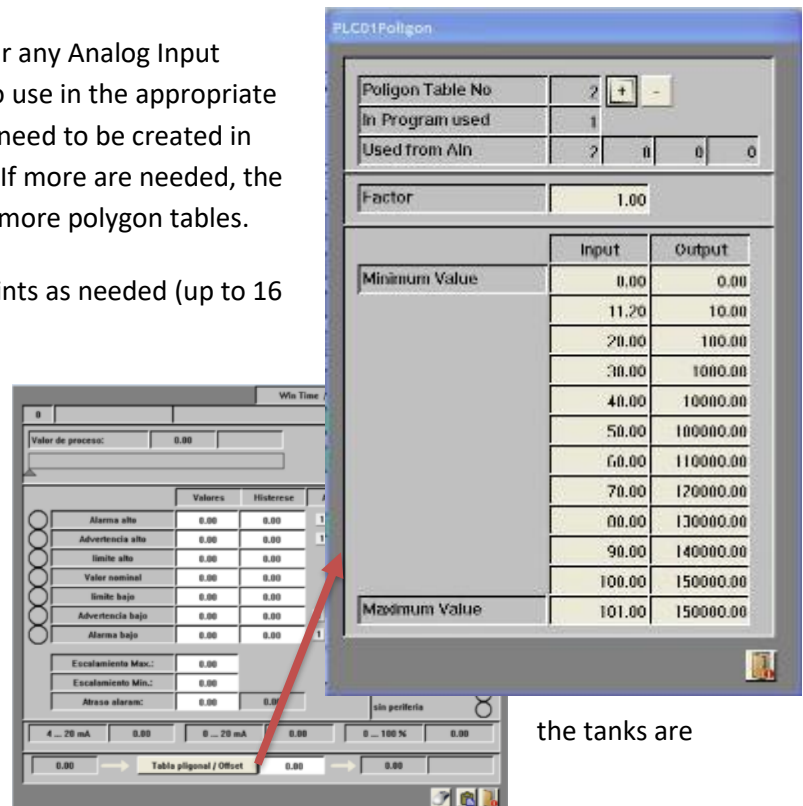
First, we recommend creating an Excel list, where your Pressure readings (your input) and our Flow meter totalizer readings (your output).

Now start with an empty tank. Fill an amount of liquid into the tank, so that you have some change in the pressure sensor reading and stop the transfer. Note the pressure reading and the actual totalizer amount into you excel table, then continue filling the tank, but periodically stop the transfer and recording the pressure and actual totalizer amounts.

Take your measurement about every 10% of the level of the tank, or when some “significant” filling level has been reached, for example when the level reaches the cylindrical part of a tank. After that you should end up with a table like this one:

Input (Pressure)	Output (Volume from Totalizer)
0 mbar	0 liters
10 mbar	324 liters
14 mbar	602 liters
20 mbar	1503 liters
46 mbar	4359 liters
63 mbar	6358 liters
78 mbar	7920 liters

This is precisely the table that you must put into the Faceplate of the polygon table.



the tanks are

you can note

PID Regulator (PID)

An analog output (or PID Regulator) represents all equipment that can be controlled with an analog output signal. This module also implements the full operation of a PID regulator, with all its settings and limit values. If the operation of a regulator is not required, the operation of the regulator can be disabled, and the analog output can be operated directly as a value to an external computer. This module is used for all types of regulations, such as regulating valves, flow regulations, etc. It is also used directly, without regulation, such as nominal values to variable frequency drives, or other equipment.

General Principle

This table sets all the parameters required for the PID algorithm.

- The "**Control Reverse**" function indicates in which direction the PID algorithm acts.
- The **Proportional, Integral, and Differential** values indicate the basic parameters of the regulation algorithm. These values directly affect the behavior of the regulator and should only be adjusted by regulatory experts. For more information, please review the "http://en.wikipedia.org/wiki/PID_controller" articles on PID control.
- "**Banda Muerte**". This sets a dimming range where the regulator does not change its output value.
- "**Output Rampa**". This value limits the modification of the output to a value in %/sec. If the slider modifies its output value faster than this value, the algorithm limits this and only modifies the output with the maximum ramp set. This value can be used for valves or pumps, to avoid damage from abrupt regulation. With the value 0.0, the ramp is disabled and allows instant changes of the output.

Alarms

Alarm limits can be set for the controller. The "Check Delay" value indicates the length of time for which boundary monitoring is delayed. This time starts when the regulator starts. In this way, a time can be set within which the regulator must have regulated its process value.

In the first category, you can set the absolute limits, which the process value cannot exceed. When the process value exceeds these values after the "Backlog Check" was met, the regulator generates a failure.

In addition, a **hysteresis can be set**, which indicates the maximum allowable deviation from the process value to its nominal value. This value will only be activated when the "Checkup Delay" time has expired.

Like the hysteresis alarm, a hysteresis "Warning" can be triggered. The operation is the same, only it generates a warning instead of an alarm.

Simulation

If a PID is simulated, the current internal Process Value can be simulated by the user, and the field signal is completely ignored.

Structure

Address	Symbol	Type	Remark
0.0	EAI	BOOL	Enable alarm
0.1	SCS	BOOL	status check start
0.2	MStC	BOOL	static output value
0.3	MStrt	BOOL	starting value
0.4	MOVMin	BOOL	output value min.
0.5	MOVMax	BOOL	output value max.
0.6	OVOOn	BOOL	output value on
0.7	B31	BOOL	spare
1.0	B16	BOOL	spare

1.1	B17	BOOL	spare
1.2	B18	BOOL	spare
1.3	B19	BOOL	spare
1.4	AIHM	BOOL	help memory for alarm
1.5	AHystHM	BOOL	help memory outside hysteresis
1.6	StrtHM	BOOL	help memory starting value active
1.7	Warn	BOOL	warning
2.0	GAIQuitt	BOOL	general alarm acknowledge
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	MCon	BOOL	mode controller on (0=off)
2.4	MSPExt	BOOL	mode setpoint extern (0=intern)
2.5	DisOut	BOOL	disable output periphery (0=enable)
2.6	EW	BOOL	enable warning
2.7	B15	BOOL	spare
3.0	GAI	BOOL	general alarm
3.1	GAIS	BOOL	general alarm save
3.2	SCE	BOOL	status check error
3.3	Filter1	BOOL	filter 1 on (75%)
3.4	Filter2	BOOL	filter 2 on (88%)
3.5	Filter3	BOOL	filter 3 on (94%)
3.6	CA	BOOL	control acting (1 = inverse)
3.7	User	BOOL	memory free for user
4.0	OVal	REAL	output value
8.0	Sp	REAL	setpoint
12.0	PVal	REAL	process value
16.0	xSp	REAL	setpoint extern
20.0	xPVal	REAL	process value from user program
24.0	LScal	REAL	low scaling
28.0	HScal	REAL	high scaling
32.0	OVMIn	REAL	output value min.
36.0	OVMMax	REAL	output value max.
40.0	StC	REAL	static output value %
44.0	Strt	REAL	starting value %
48.0	StrTVal	REAL	starting time value
52.0	StrTSp	REAL	starting time setpoint
56.0	LLAVal	REAL	low value for alarm
60.0	HHAVal	REAL	high value for alarm
64.0	AHys	REAL	hysteresis band for alarm
68.0	CheckDVal	REAL	check delay value
72.0	CheckDSp	REAL	check delay setpoint
76.0	ADHLVal	REAL	alarm delay high low limit value
80.0	ADHLSp	REAL	alarm delay high low limit setpoint
84.0	ADVal	REAL	alarm delay hysteresis value
88.0	ADSp	REAL	alarm delay hysteresis setpoint
92.0	WHys	REAL	hysteresis band for warning
96.0	WDVal	REAL	warning delay hysteresis value
100.0	WDSp	REAL	warning delay hysteresis setpoint
104.0	KP	REAL	proportional gain (unitless)
108.0	KI	REAL	integral gain (1/sec)
112.0	KD	REAL	derivative gain (sec)
116.0	RampV	REAL	ramp value for OVAL (per second)
120.0	DeadB	REAL	dead band for error

124.0	Fuzzy1G	REAL	fuzzy 1 gain
128.0	Fuzzy1V	REAL	fuzzy 1 variable
132.0	Fuzzy1VOld	REAL	fuzzy 1 variable old
136.0	Fuzzy2G	REAL	fuzzy 1 gain
140.0	Fuzzy2V	REAL	fuzzy 1 variable
144.0	Fuzzy2VOld	REAL	fuzzy 1 variable old
148.0	iOVal	REAL	output value intern
152.0	E	REAL	control error
156.0	DPart	REAL	derivative part

Commands

Symbol	Default	Remark
EAI	0	Enable alarm
SCS	0	status check start
MStC	0	static output value
MStrt	0	starting value
MOVMin	0	output value min.
MOVMax	0	output value max.
OVal	0	output value on
DisOut	0	disable output periphery (0=enable)
User	x	memory free for user
xSp	x	setpoint extern
xPVal	x	process value from user program
StC	x	static output value %
Fuzzy1V	x	fuzzy 1 variable
Fuzzy2V	x	fuzzy 2 variable

Status

Symbol	Remark
GAI	general alarm
GAIS	general alarm save
SCE	status check error

Parameters

Symbol	Remark
Ign	ignore alarm
Sim	simulation
MCon	mode controller on (0=off)
MSpExt	mode setpoint extern (0=intern)
DisOut	disable output periphery (0=enable)
EW	enable warning
Filter1	filter 1 on (75%)
Filter2	filter 2 on (88%)
Filter3	filter 3 on (94%)
CA	control acting (1 = inverse)
OVal	output value
Sp	setpoint
PVal	process value
LScal	low scaling
HScal	high scaling
OVMIn	output value min.
OVMax	output value max.

StC	static output value %
Strt	starting value %
StrTSp	starting time setpoint
LLAVaI	low value for alarm
HHAVaI	high value for alarm
AHys	hysteresis band for alarm
CheckDSp	check delay setpoint
ADHLSp	alarm delay high low limit setpoint
ADSp	alarm delay hysteresis setpoint
WHys	hysteresis band for warning
WDSp	warning delay hysteresis setpoint
KP	proportional gain (unitless)
KI	integral gain (1/sec)
KD	derivative gain (sec)
RampV	ramp value for OVAL (per second)
DeadB	dead band for error
Fuzzy1G	fuzzy 1 gain
Fuzzy2G	fuzzy 1 gain

Faceplate

1		08.01.41 GC		flow water mash tun 1		P: 1024			
Actual Value:	81.239	hl/h	1	Sim	0	Alarm	0		
Setpoint:	82.000	hl/h	1	Sp. Extern	0	Warning			
Output Value:	81.263	%	1	Manual	0	Status Error			
Inverse control	1	0	Delay Check:		0.00	sec	1	Disable Outp	0
Proportional:	2.00000				0.00	sec	1	Enable Warn	0
Integral	1.00000	1/sec	High Alarm Limit:		0.00	hl/h	1	Ignore	0
Differencial:	0.00000	sec	Low Alarm Limit:		0.00	hl/h	1	Filter 1	0
Dead band:	0.000	hl/h	Delay Limites:		0.00	sec	1	Filter 2	0
Output Ramp:	0.000	%			0.00	sec	1	Filter 3	0
Static Output:	0.000	%	Alarm Hysteresis:		0.00	hl/h	Enable Alarm		
Startup Output:	25.000	%	Delay Hysteresis:		0.00	sec	Status Check		
Startup Time	10.00	sec			0.00	sec	Static Output		
	21.82	sec	Warn Hysteresis:		0.00	hl/h	Start Phase		
Fuzzy Control 1:	0.00000	%	Delay Warning:		0.00	sec	Mov MaxVal		
	0.00				0.08	sec	Mov MinVal		
Fuzzy Control 2:	0.00000	%					PID Active		
	0.00								
			Scal Min		Scal Max				
Input Parameter		0.000	200.000		hl/h				
Output Parameter		0.000	100.000		%				

Special Configurations

In addition to the system window for the PID for default parameter settings, there is the window for mouse settings. This determines what should happen when you click the mouse over the item.

In addition to the mouse parameterization, you can determine the overall scale of the inputs:

- Output value 0% – dissipation on output card, 0% PID output
- Output value 100% – dissipation on output card, 100% PID output

Programming Examples

Transfer of Values

```
L "Uxx". Para[12]. Val      //Parameter Unit 12 - Temperature Measurement
L "PID". PID[4].xPVal      //Process Value for PID

L "Uxx". Para[12]. Sp      //Parameter Unit 12 - Temperature Measurement
L "PID". PID[4].xSp       //Nominal value of the PID
```

Alarm Assessment

```
U "Act.Act[45].Out        //Actuator Output
U "CIP"                  //CIP
S "PID". PID[4]. Eal      //Enable Low Limit Alarm

U "PID". PID[4]. GAlS     //Alarm
S "HoldReq"              //Maintains Unity
```

Startup / Static Output of the PID

```
U "Act.Act[45]. Out      //Actuator Output
U "CIP"                  //CIP
S "PID". PID[4]. MStC     //Enable Static Output

U "Act.Act[45]. Out      //Actuator Output
S "PID". PID[4]. MStrt    //Start the PID
```

PID Assignment

```
//This code is usually created by "Project Engineering Tool" and resides in
//"Trans PID"
//The "OutFactor" depends on the type of Analog output card that you are using.
//for some cards may use values from 0 - 27648 (usually Siemens), others
//may use 0 - 32767 (Wago), so you should adjust the OutFactor accordingly

L "PID". PID[1]. Oval      //PID output value 1
L #OutFactor              //is 327.67
*R
RND
T PAW 1024                //Transfer the value to the physical output

L "PID". PID[480]. Oval    //PID 480 output value
L #OutFactor              //is 327.67
*R
RND
T PAW 1982                //Transfer the value to the physical output
```

Parameter Example

Here we are going to list some “common” Parameter values that we recommend as starting values for tuning PID Regulator. It must be noted that the following values are fundamentally empirical values and should only be used as “Guidance”. All PID Regulators must always be “tuned” during startup of your project. This values, however, serve as a good starting point for tuning these regulators.

Temperature Regulator for CIP Station

A Typical heat exchanger that regulates the temperature of an CIP prerun by means of a Steam modulating valve.

Process Value Range	0-110 °C
Typical Setpoint Range	60 – 80°C
Proportional Gain (KP)	3.0
Integral Gain (Ki)	0.3
Derivate Gain (Kd)	0.5
Regulator Startup Value	25%
Regulator Startup Time	5 seconds

Water Flow Regulation

A water flow regulation by means of constant pressure provided by a non-regulating pump and a Modulating valve

Process Value Range	0-200 hl/h
Typical Setpoint Range	80 – 150 hl/h
Proportional Gain (KP)	1.0
Integral Gain (Ki)	0.1
Derivate Gain (Kd)	0.0
Regulator Startup Value	30%
Regulator Startup Time	5 seconds

Wort cooler Regulator

Wort temperature Control by means of a Glycol regulating valve and constant glycol pressure.

Process Value Range	0-100 °C
Typical Setpoint Range	10 °C
Proportional Gain (KP)	1.0
Integral Gain (Ki)	0.2
Derivate Gain (Kd)	0.0
Regulator Startup Value	25%
Regulator Startup Time	5 seconds

Fermenting Tank Temperature Regulator

Wort temperature Control in a fermenting tank by means of digital solenoid valves that are controlled by a two step controller with a duty time of 100 seconds. Duty time means that the glycol valves will open and close once for each 100 seconds.

Process Value Range	-10-40 °C
Typical Setpoint Range	10 °C
Proportional Gain (KP)	1.0
Integral Gain (Ki)	0.03
Derivate Gain (Kd)	0.0
Regulator Startup Value	50%
Regulator Startup Time	5 seconds

PID Regulator Output Scaling

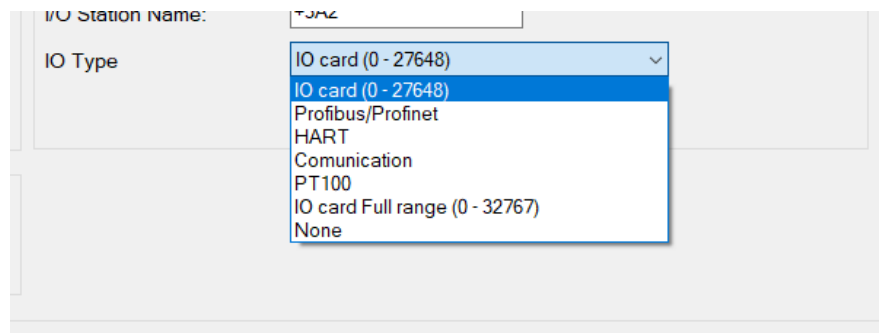
PID Regulator control Modules can be assigned to different hardware destination which may have different Scaling values, depending on the Type of Value you are supplying to the Analog Input module.

Typical Analog input sources

Siemens S7 Analog IO	0 – 27648 correspond to 0 – 100% (4-20mA, 0-10V, ...)	Siemens uses its own Analog input format where all Analog cards are represented by a 14-bit value, and the two upper bits represent error conditions.
Other Analog IO	0-32767 correspond to 0 – 100% (4-20mA, 0-10V, ...)	Some Manufacturer use a different IO Scaling mechanism where they use all 16-bit for its representation, at the cost of not having bits available for error reporting. Most notably “WAGO” IO does this.
Communication	Profinet, ASI-Bus, Modbus, etc.	Most of the time these sources already send a fully resolved and scaled “REAL” value that does not need to be Scaled at all, or you must apply custom scaling factors. Anton Paar measurement devices are an example of this-

Setting Analog input type for Code Generation

BatchXpert allows you to select the type of IO that you want to read and scale according to the Analog inputs parameters. In the “Batch Configurator” or the “Taglist” you can select the type of IO that should be generated for this Analog Input.



This allows the “Project Engineering Tool” to generate an “System Trans Ain” block corresponding to the requirement of the Analog input.

J	K	L
signment	IO Type	Tested
	<input type="text" value="IO_Card"/>	
	DP	
	HART	
	Communication	
	PT100	
	IO_Card_Full_Range	
	None	

Examples of IO Transfer Code generated by “Project Engineering Tool”

It should be noted that the IO assignment of control modules is generally done by generating this code using the project engineering tools. Please refer to [The “Trans xx” blocks for IO signal transfer](#) for more details

AI n IO-Card (0-27648)

```
L    "Bx PID D".PID[1].Out  //0 - 100
L    2.764800e+002;
*R   ;
RND  ;
T    PAW  1024;
```

AI n IO-Card Full Range (0-32767)

```
L    "Bx PID D".PID[1].Out  //0 - 100
L    3.276700e+002;
*R   ;
RND  ;
T    PAW  1024;
```

Custom IO Transfer

```
L    "Bx PID D".PID[1].Out  //0 - 100
T    #CustomDestination //Could be a Profinet, or internal destination
```

Setting prior to BatchXpert 1.9

Prior to version 1.9 of BatchXpert, the Project Engineering tool did not allow for individual selection of IO types per Regulator. The Value was fixed for S7 Compatible modules to 0-27648.

Counter Module (Cnt)

A counter represents a module that counts the pulses of a digital input. This module is used for (for example), counting the quantity of a meter flow through a quantity pulse. This technique is often used for totalizing volumes from flow meters. These send an impulse whenever a specific amount of volume has passed through the flowmeter. By counting the impulse and multiplying them with an impulse value, you can totalize the amount from such flowmeters.

General Principle

If the xSig value becomes TRUE, it will increment the "CVal" of the Counter value. The "PVal" is then calculated by multiplying the "CVal" with the "ImpVal."

Simulation

If a Cnt is simulated, the current internal Process Value can be simulated by the user, and the field signal is completely ignored

Structure

Address	Symbol	Type	Remark
0.0	EAImp	BOOL	enable impulse alarm
0.1	ELLA	BOOL	enable low low alarm
0.2	EHHA	BOOL	enable high high alarm
0.3	xAI	BOOL	alarm from extern
0.4	ResetBlock	BOOL	interlock counter reset
0.5	xSig	BOOL	impulse input
0.6	B30	BOOL	spare
0.7	B31	BOOL	spare
1.0	MLLA	BOOL	low low alarm - alarm if enabled
1.1	MLL	BOOL	low low limit - warning if enabled
1.2	ML	BOOL	low limit
1.3	MSp	BOOL	setpoint
1.4	MH	BOOL	high limit
1.5	MHH	BOOL	high high limit - warning if enabled
1.6	MHHA	BOOL	high high alarm - alarm if enabled
1.7	ImpHM	BOOL	impulse help memory
2.0	GAIQuitt	BOOL	general alarm acknowledge
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	Reset	BOOL	reset counter
2.4	iELLA	BOOL	counting reserve
2.5	iEHHA	BOOL	enable HH alarm
2.6	iELLW	BOOL	enable LL warning
2.7	iEHHW	BOOL	enable HH warning
3.0	GAI	BOOL	general alarm
3.1	GAIS	BOOL	general alarm save
3.2	Warn	BOOL	general warning
3.3	Imp	BOOL	impulse flank
3.4	B04	BOOL	spare
3.5	B05	BOOL	spare
3.6	B06	BOOL	spare
3.7	User	BOOL	memory free for user

4.0	PVal	REAL	process value
8.0	Sp	REAL	setpoint
12.0	LScal	REAL	low scaling
16.0	HScal	REAL	high scaling
20.0	LLAVal	REAL	low low alarm value
24.0	LLVal	REAL	low low value (warning limit)
28.0	LVal	REAL	low value
32.0	HVal	REAL	high value
36.0	HHVal	REAL	high high value (warning limit)
40.0	HHAVal	REAL	high high alarm value
44.0	ADVal	REAL	alarm delay value
48.0	ADSp	REAL	alarm delay setpoint
52.0	ImpVal	REAL	value per impulse
56.0	CVal	DINT	counter value

Commands

Symbol	Remark
EAImp	enable impulse alarm
ELLA	enable low low alarm
EHHA	enable high high alarm
xAI	alarm from extern
ResetBlock	interlock counter reset
xSig	impulse input
User	memory free for user

Status

Symbol	Remark
MLLA	low low alarm - alarm if enabled
MLL	low low limit - warning if enabled
ML	low limit
MSp	setpoint
MH	high limit
MHH	high high limit - warning if enabled
MHHA	high high alarm - alarm if enabled
GAI	general alarm
GAIS	general alarm save
Warn	general warning
Imp	impulse flank
PVal	process value

Parameters

Symbol	Remark
GAIQuitt	general alarm acknowledge
Ign	ignore alarm
Sim	simulation
Reset	reset counter
iELLA	counting reserve
iEHHA	enable HH alarm

iELLW	enable LL warning
iEHHW	enable HH warning
Sp	setpoint
LScal	low scaling
HScal	high scaling
LLAVal	low low alarm value
LLVal	low low value (warning limit)
LVal	low value
HVal	high value
HHVal	high high value (warning limit)
HHAVal	high high alarm value
ADSp	alarm delay setpoint
ImpVal	value per impulse

Faceplate

Win Time / sec:		9999.00	37.46
1	08.01.41 FQS	water mash tun 1	
R	Process Value:	33.00	hl 1 Simulation 0
		Values	Alarm
<input checked="" type="radio"/>	High High Alarm:	14.00	1 <input type="radio"/> 0
<input checked="" type="radio"/>	High High Warning:	12.00	1 <input type="radio"/> 0
<input checked="" type="radio"/>	High Limite:	10.00	
<input checked="" type="radio"/>	Setpoint:	8.00	
<input checked="" type="radio"/>	Low Limite:	6.00	
<input checked="" type="radio"/>	LowLow Warning:	4.00	1 <input type="radio"/> 0
<input checked="" type="radio"/>	LowLow Alarm:	2.00	1 <input type="radio"/> 0
	High Scale:	150.00	
	Low Scale:	0.00	
	Impulse Value	1.00	33
	Impulse Alarm delay:	10.00	0.00
		<input type="radio"/> Alarm 0 Warning 1 Ignore 0	
		Status Info E. Imp.Alarm E. High Alarm E. Low Alarm Extern Alarm Block Reset Impulse Input	

Special Configurations

Mouse Click		
	SET	RESET
Quit Alarm:	1 <input checked="" type="radio"/> 0	
Ignore	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Simulation	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Enable HH Alarm	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Enable HH Warn	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Enable LL Alarm	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Enable LL Warn	1 <input type="radio"/> 0	1 <input type="radio"/> 0
Reset Counter	1 <input type="radio"/> 0	

In addition to the system window for the counter for default parameter settings, there is also a window for mouse settings. This determines what should happen when you click the mouse over the item.

Programming Examples

Signal Transfer

```
U "Act". Act[18]. Out           //Water Valve
U "DIn". Din[233]. Imp          //Positive Digital OneShot or Flank
= "Cnt". Cnt[4].xSig            //External signal for the counter
```

Transfer of Values

```
L "Cnt". Cnt[4]. PVal           //Water Meter
T "Uxx". Para[17]. Val          //Unit Parameter 17
```

Alarm Evaluation

```
U "Act". Act[18]. Out           //Water Valve
S "Cnt". Cnt[4]. EAImp          //Enable Pulse Alarm

U "Cnt". Cnt[4]. GAlS"           //Alarm
S "HoldReq"                      //Maintains Unity
```

Message Module (Msg)

Since the other modules such as actuators, digital inputs, etc. already integrate messages, this message module is very rarely used. It is only used to generate alarms or instructions for the operator that are not related to the periphery.

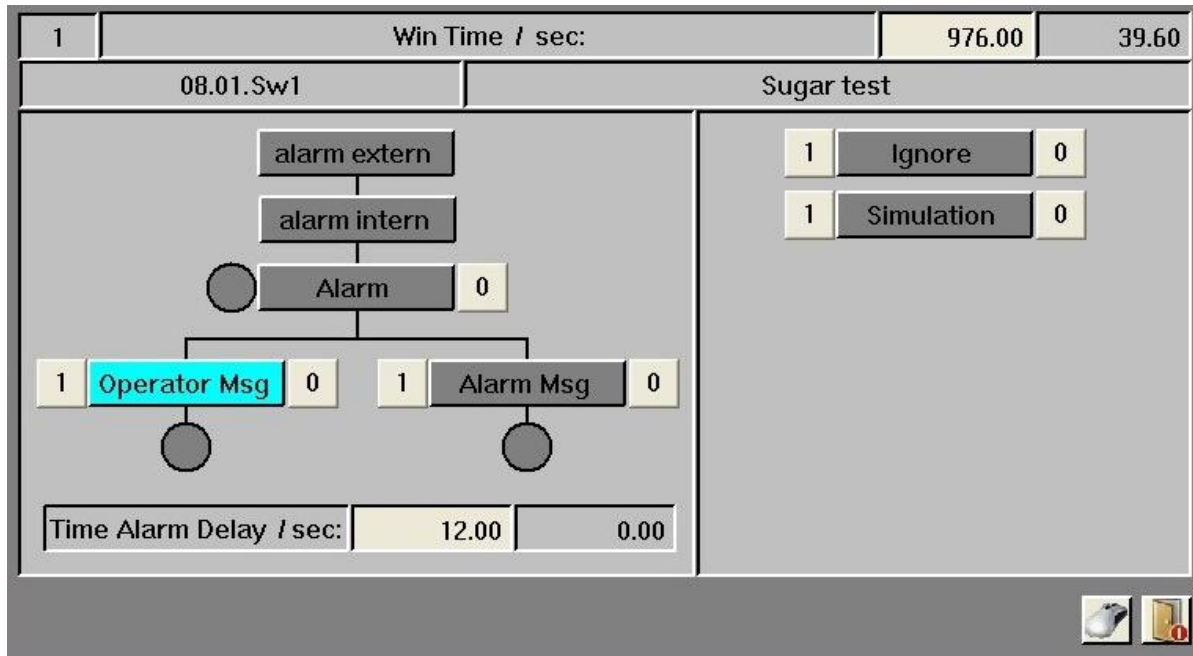
General Principle

- If the external signal becomes TRUE it starts the On-Delay timer and then activates either an OpMsg or a Gal, depending on the Parameters (if it is an Alarm or Operating Message)
- If the external signal becomes FALSE, the Off-Delay timer starts,

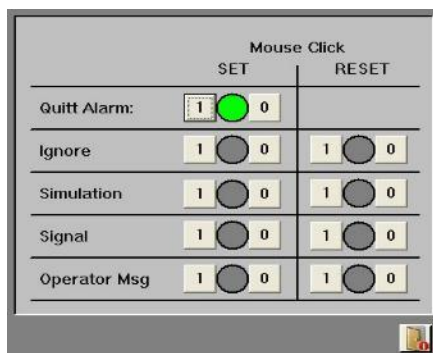
Structure

Address	Symbol	Type	Remark
0.0	B24	BOOL	spare
0.1	B25	BOOL	spare
0.2	B26	BOOL	spare
0.3	B27	BOOL	spare
0.4	xAlarm	BOOL	signal extern for alarm condition
0.5	B29	BOOL	spare
0.6	B30	BOOL	spare
0.7	B31	BOOL	spare
1.0	B16	BOOL	spare
1.1	B17	BOOL	spare
1.2	B18	BOOL	spare
1.3	B19	BOOL	spare
1.4	B20	BOOL	spare
1.5	B21	BOOL	spare
1.6	B22	BOOL	spare
1.7	B23	BOOL	spare
2.0	GAIQuitt	BOOL	general alarm acknowledge
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	OPMsg	BOOL	operator message
2.4	B12	BOOL	spare
2.5	B13	BOOL	spare
2.6	B14	BOOL	spare
2.7	B15	BOOL	spare
3.0	GAI	BOOL	general alarm
3.1	GAIS	BOOL	general alarm save
3.2	OPMsgActive	BOOL	operator message active
3.3	AlarmMsgActive	BOOL	alarm message active
3.4	iAlarm	BOOL	Alarm active intern
3.5	B05	BOOL	spare
3.6	B06	BOOL	spare
3.7	User	BOOL	free for user
4.0	ADVal	REAL	alarm delay current value
8.0	ADSp	REAL	alarm delay setpoint

Faceplate



Special Configurations



In addition to the system window for the counter for default parameter settings, there is also a window for mouse settings. This determines what should happen when you click the mouse over the item.

Programming Examples

Generate Message

```
U "Malzlaster Wartet" //Malta truck is waiting
= "Msg". Msg[12].xAlarm //External signal for the message
```

Alarm Assessment

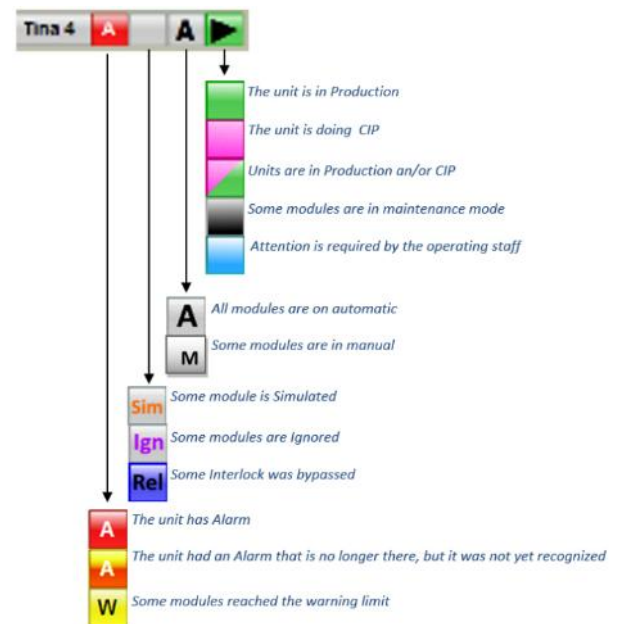
```
U "Msg". Msg[12]. Gals //Active Message
S "SignalLamp" //Visual cue for the operator
```


Alarm Groups

Alarm groups allow You to group certain alarms together into logical groups and then show and collective status for all control modules belonging to this group. This is usually done to group all control modules of a specific production area or a production unit together to show an overall unit or production area status.

BatchXpert allows you to configure unit and control module assignments in its batch configuration tools and then allows you to generate alarm group function blocks that you can download into a PLC to represent your configuration in the database.

Alarm groups are extensively used in the BatchXpert HMI menu structure of the operating stations. In the default menu structure, you can see and hierarchical few of all your production units and areas and an alarm group symbol is shown next to them to represent the current alarm simulation and process status of all models that belong to this group, or subgroups thereof.



Executing the Alarm Groups

The project engineering tool generates a system function for calling all automatically generated alarm groups. BatchXpert includes one alarm group per unit, where each alarm group contains all submodules that are assigned to this unit in the batch configuration.

You should never manually change this file since your manual changes will be overwritten the next time a new alarm configuration system function block is being generated.

Name	Function	Description
Bx Alarm Group System	FC 570	Executing all Automatically Generated Alarm groups Is being called from OB1
Bx Alarm Group User	FC 571	Executing all custom alarm groups Is being called from OB1
Bx Alarm Group D	DB 550	Holds all Alarm groups data

Functions to Add Modules to Alarm Groups

Name	Function	Description
Bx Alarm Group	FC 550	The Main Alarm Group function. Must be called exactly one for each Alarm group
Bx Alarm Group Unit	FC 551	Add a Unit to the alarm group to report its Process status
Bx Alarm Group Act	FC 552	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group DIIn	FC 553	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group AIIn	FC 554	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group PID	FC 555	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status

Bx Alarm Group Cnt	FC 556	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group Msg	FC 557	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group Mat	FC 558	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group FC	FC 559	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group AlGroup	FC 560	Add an Existing Alarm group to an Alarm Group, thus creating a Hierarchy
Bx Alarm Group SideUnit	FC 561	Add a Unit to the alarm group to report its Process status. Side Units do not report the Production process, only if they are in CIP. This should be used for units that may run for a long time but should not mark the Alarm group as “Running a process”. This may be the case for “Weak wort Tanks”, “Trub Tanks” etc.
Bx Alarm Group SideAlGro	FC 562	Add an Existing Alarm group to an Alarm Group, thus creating a Hierarchy Side Alarm Groups do not report on the Production process, only if they are in CIP. This should be used for units that may run for a long time but should not mark the Alarm group as “Running a process”. This may be the case for “Weak wort Tanks”, “Trub Tanks” etc.

Structure

Statuses of an alarm group always reflect a summary of all control modules contained in an alarm group. This means if any one of the control modules assigned to an alarm group has a particular status, the status pit in the alarm group will be set. If and command is set to true, the respective action is applied to all control models assigned to an alarm group.

Address	Symbol	Type	Remark
0.0	SCS	Bool	Command: Initiate a Status Check
0.1	Ign	Bool	ignore alarm
0.2	Sim	Bool	simulation
0.3	Auto	Bool	automatic mode
0.4	SetAuto	Bool	Command: Set to Automatic Mode
0.5	EmRel	Bool	emergency release
0.6	s2	Bool	
0.7	Maint	Bool	maintenance
1.0	GAI	Bool	general alarm
1.1	GAIS	Bool	general alarm save
1.2	SCE	Bool	status check error
1.3	Warn	Bool	A warning is active
1.4	Msg	Bool	An Operating message is active
1.5	ProcRun	Bool	At least one unit, except side units, is running a process
1.6	ProcProd	Bool	At least one unit, except side units, is running production
1.7	ProcCIP	Bool	At least one unit, except side units, is running CIP

Examples

For each alarm group you must call the main “Bx Alarm Group FC550” alarm group function once for each alarm group that you want to create and then must call one of the alarm group functions to add unspecific module to an alarm group.

The default alarm group data block that also contains all the default alarm groups for each units that are generated by the project engineering tool, also contains space to add customized alarm groups.

```
FUNCTION "Bx Alarm Group User" : VOID
TITLE =Custom alarm Groups
VERSION : 0.1

NETWORK
TITLE =Custom Alarm Group

//Calling the Alarm groups main Function here
CALL "Bx Alarm Group"
  AlGroup:="Bx Alarm Group D".AlGroupKS2 //this is our custom Alarm groups

//Trub Tank and Weak Wort Tank, should not report Process
CALL "Bx Alarm Group SideAlGro"
  AlGroupVisu:="Bx Alarm Group D".Unit[15]
  AlGroup      := "Bx Alarm Group D".AlGroupKS2
CALL "Bx Alarm Group SideAlGro"
  AlGroupVisu:="Bx Alarm Group D".Unit[17]
  AlGroup      := "Bx Alarm Group D".AlGroupKS2

//Add Untis Alarm Groups to this Alarm Group
CALL "Bx Alarm Group AlGroup"
  AlGroupVisu:="Bx Alarm Group D".Unit[2]
  AlGroup      := "Bx Alarm Group D".AlGroupKS2
CALL "Bx Alarm Group AlGroup"
  AlGroupVisu:="Bx Alarm Group D".Unit[3]
  AlGroup      := "Bx Alarm Group D".AlGroupKS2

//add other custom Control Modules
CALL "Bx Alarm Group Act"
  ActVisu:="Bx Act Visu".Act0420
  AlGroup:="Bx Alarm Group D".AlGroupKS2

CALL "Bx Alarm Group Act"
  ActVisu:="Bx Act Visu".Act0424
  AlGroup:="Bx Alarm Group D".AlGroupKS2

CALL "Bx Alarm Group AIn"
  AInVisu:="Bx AIn Visu".AnI[63].Status
  AlGroup:="Bx Alarm Group D".AlGroupKS2

CALL "Bx Alarm Group DIn"
  DInVisu:="Bx DIn Visu".SpI0059
  AlGroup:="Bx Alarm Group D".AlGroupKS2]

CALL "Bx Alarm Group DIn"
  DInVisu:="Bx DIn Visu".SpI0060
  AlGroup:="Bx Alarm Group D".AlGroupKS2

CALL "Bx Alarm Group PID"
  PIDVisu:="Bx PID Visu".PID[19].Status
  AlGroup:="Bx Alarm Group D".AlGroupKS2

END FUNCTION
```

Unit Assignments and Automatically generated Alarm Groups

For alarm groups to be generated automatically by the project engineering tool, you must assign control modules to their respective units. You can either assign them by filling out the unit assignment column in the tag list or by using the “Batch configuration” tool ([Generate Alarm Groups](#)). This will create an alarm group system function block that you can compile and download into your PLC.

Whenever you update any unit assignment you must recreate this alarm group function block and redownload the compiled block into your controller.

Alarm group commands

Also implement some simple comments which allow you to control all control modules assigned to this alarm group at the same time. You can for example set all control models of an alarm group into automatic mode, or initiate a status check on them during the start check process phase ([Phases every unit should implement](#)).

Currently the following Commands are implemented:

SCS	Initiate a Status check on all modules
SetAuto	Try to set all control modules to automatic

Software Switch (Switch)

To achieve simple and uniform operation, the Switch module (software switch) is integrated for the operator to generate a signal to the PLC (independent of the other modules), the default switch most used would be the button to "confirm alarm", confirm a manual operation or activate optional Process features. A software switch can also be forced to set or forced to reset in the PLC and is then blocked from operating on the HMI.

Structure

Address	Symbol	Type	Remark
0.0	Set	BOOL	set software switch
0.1	Reset	BOOL	reset software switch
0.2	B26	BOOL	spare
0.3	B27	BOOL	spare
0.4	B28	BOOL	spare
0.5	B29	BOOL	spare
0.6	B30	BOOL	spare
0.7	B31	BOOL	spare
1.0	B16	BOOL	spare
1.1	B17	BOOL	spare
1.2	B18	BOOL	spare
1.3	B19	BOOL	spare
1.4	B20	BOOL	spare
1.5	B21	BOOL	spare
1.6	B22	BOOL	spare
1.7	B23	BOOL	spare
2.0	B08	BOOL	spare
2.1	B09	BOOL	spare
2.2	B10	BOOL	spare
2.3	B11	BOOL	spare
2.4	B12	BOOL	spare
2.5	B13	BOOL	spare
2.6	B14	BOOL	spare
2.7	B15	BOOL	spare
3.0	B00	BOOL	spare
3.1	B01	BOOL	spare
3.2	B02	BOOL	spare
3.3	Sig	BOOL	spare
3.4	B04	BOOL	spare
3.5	B05	BOOL	spare
3.6	B06	BOOL	spare
3.7	User	BOOL	free for user

Programming Examples

Switch reset

```
A "RUN"
S "Switch". Switch[3]. Reset      //Reset the switch to block operations
```

Checking the status of the Switch

```
U "PH"  
U "Switch". Switch[3].Sig           //Confirmation of the operator "Manual Sugar Emptying"  
S "Act". Act[42]. Aco               //Mixer
```

Clock current Switch status, and do not allow the user to operate it

```
U Your condition here  
U "Switch". Switch[3].Sig           //Confirmation of the operator "Manual Sugar Emptying"  
S "Switch". Switch[3].Set  
  
U Your condition here  
UN "Switch". Switch[3].Sig          //Confirmation of the operator "Manual Sugar Emptying"  
S "Switch". Switch[3].Reset
```

Frequency Converters

Nowadays variable speed controllers for pumps and motors are common in industrial settings. Usually these are implemented as “Variable Frequency Drives” or FConv for short. Frequency drives require a speed nominal value in addition to the “Start” signals from a control system. Traditionally these FConv have been connected via an electrical 4-20mA loop, which should be implemented directly by using Analog Output (PID) modules. Nowadays an increasing amount of Frequency drives implement a digital communication option that is usually are connected either by Profibus, or by Profinet and implement the “ProfiDrive” protocol or implement proprietary protocols from the drives manufacturer.

Unfortunately, there is not really an “standard” that all manufacturers follow for the implementation of a communication interface with Frequency drives. There is “Profidrive”, but many manufacturers do not support this protocol and implement their own proprietary protocol.

Since the implementation of these protocols can be vastly different, you must call a function specific to the Frequency drive protocol you are using, for each of the connected frequency drives. This protocol specific function implements the standard BatchXpert FConv module and adapts it to the specific communication protocol you are using.

General Principle of an FConv communication

Even though the communication protocols are different, and not compatible with each other, they follow the general communication principle. They exchange the following kind of data:

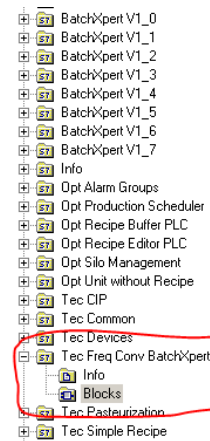
- Status Word
- Control Word
- Alarm Word
- Speed Nominal
- Speed Current Value
- Other Current Values

These bits of information are packed together into “Messages” and exchanged with the frequency drive. The biggest difference between the Protocols is how the Status, Control and Alarm words are encoded and the state model that is implemented for the frequency drive. These are entirely dependent on the protocol in questions, and most of the time are not compatible with other protocols at all.

Communication protocols implemented by default in BatchXpert.

By default, BatchXpert implements the following Protocols, but newer types may be added in the future or on request. You can find the Supported Frequency drives in the corresponding folder of the BatchXpert library:

- [ProfiDrive](#). Mostly used by Drives from “Siemens” or “ABB”, but many other manufacturers support this protocol optionally by configuration in the Frequency drive.
- [Danfoss FC](#). This protocol is only available on frequency drives from “Danfoss”. These also support ProfiDrive, but the default selection is set to “Danfoss FC”. It implements a simpler control scheme than ProfiDrive and thus is easier to use. This protocol is especially used in the FC300, FC200 and FC100 series. It is like ProfiDrive.
- [SEW MoviDrive](#). SEW implements a very minimal communication protocol with their “MoviDrive” series. This also supports connection of up to 4 drives via an “DFE32B” adapter, which acts as a Profinet device and communicates with up to 4 MoviDrives via an integrated serial interface.



Setting parameters on the Frequency Drives

Most Frequency drives require you to set certain parameters to make the communication protocol work. This configuration depends completely on the device manufacturer, and you must consult the manufacturer’s manual to check how and which settings to adjust.

BatchXpert usually mentions the settings for some of the common drives in the Header of the Frequency Converter function blocks, where you should look for more information.

Common Parameters that you usually must adjust are. As stated above, this is entirely dependent on the manufacturer, but usually the settings include the following:

- Select Communication protocol.
- Device Profibus ID or Profinet Name
- Control Source should be set to “Bus” or “Communication.”
- Reference Speed should be set to 0% = 0Hz and 100% = 50Hz (or 60Hz)
- “Free run” or “Inertia” should be set to “Bus” or “Ignored/Deactivated.”
- Current Values should be set to “Current Speed”, “Torque”, “Current” and “Power”, if available

Settings for Sinamics G120 via Profidrive

Antrieb	
Name	-FC-02SICM07
Rolle	Gerät
IP-Adresse	10 . 20 . 30 . 1
Telegramm	Standard Telegramm 1
Steckplatz	2
Anfangsadresse	PZD 1
Länge	2 Wörter
Verlängerung	0 Wörter

Connect to Actuators

Since Frequency Drives usually need to be activated by an Actuator Control module, BatchXpert provides a helper function to connect an Actuator to a Frequency drive and send the “out” signals of the Actuator to the Control signal of the Frequency drive, and the feedback back to the Actuators.

A version for Reversible Frequency drives is also available, which internally manages all the interlocking and feedback generation from and to the Actuators.

Implement your own Frequency Converter Protocol

If you have a frequency drive protocol that is not one of the communication protocols covered by the default implementations, you must implement this protocol in a new function, specific to your frequency drive. This implementation requires a great level of expertise and is not trivial to do. You should use one of the existing functions as the starting point and work your way up from there. Details on the implementation of this communication are not part of this manual, as it is not required for implementing BatchXpert projects.

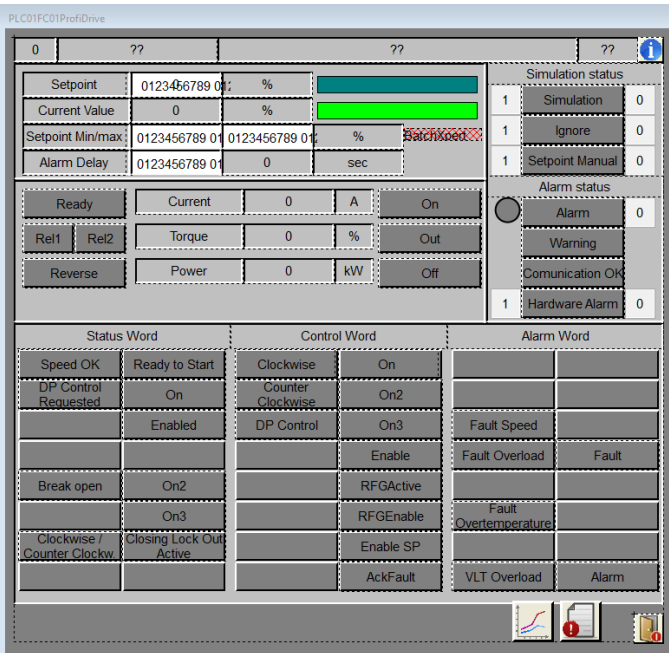
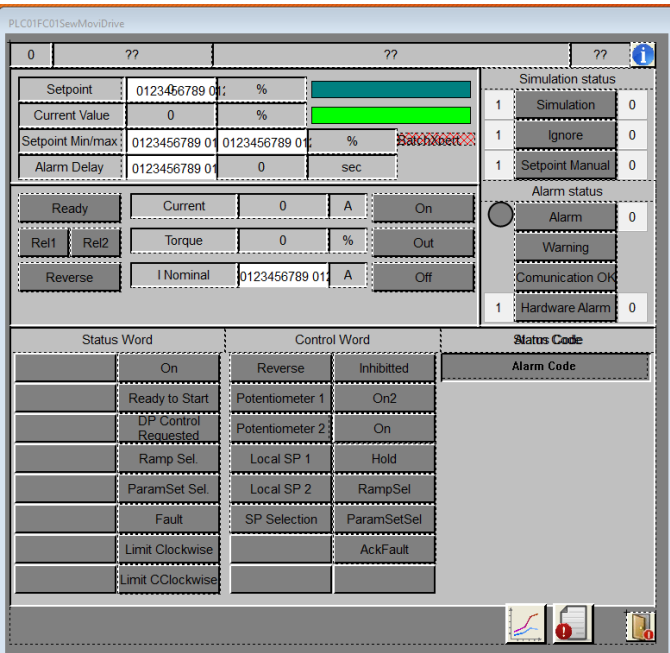
Simulation

If the Frequency drive is simulated, it will never receive a communication error, always return the correct feedback, and return the requested nominal speed as current speed.

Faceplates

Each Frequency drive also includes its own customized faceplate that contains information tailored to the frequency driver protocol you are using. Mostly the difference comes from the different status signals, command signals and alarm message that a driver can send.

This Faceplate allows you to see details about the Status Word, Control Word and the Status/Alarm Code sent to and received from the PLC.



Programming Examples

Frequency drive with single actuator and Fixed Speed

```
TITLE = Act303 - raking machine

U      "Bx DIn D".DIn37.Sig;
U      "Bx DIn D".DIn38.Sig;
U      #VesselSec;
=      "Bx Act D".Act303.Rel2;

//This function connects the Actuator to the FConv and exchanges control signals
//and Feedbacks between the Actuator and the FConv
CALL "Bx FC Actuator" (
    Act      := "Bx Act D".Act303,
    FConv    := "Bx FC D".FConv1);

NETWORK
TITLE =Frequency Converter Raking Machine

//Here the Nominal speed is set. In this example it is a fixed speed coming
//from a Parameter. But it may as well just be the output of a PID

L      "U005 data".Para[24].Sp;
T      "Bx FC D".FConv1.SPext;

//Here we call the function that exchanges data with the Frequency
//drive. In this example it is an SEW Movidrive
CALL "Bx FC Sew MoviDrive" (
    PEW      := 4000,    //This is the Input PEW of the
    PAW      := 4000,    //Hardware Configuration
    Data     := "Bx FC D".FConv1,
    Visu     := "Bx FC Visu D".FConv[1]);
```

Regulating Pump

```
//Normal Actuator Release
U      #VesselSec
UN     "Bx DIn D".DIn120.GAlS
=      "Bx Act D".Act311.Rel2

//The setpoint is derived from the output of a PID Regulator
L      "Bx PID D".PID59.OVal
T      "Bx FC D".FConv8.SPext

CALL "Bx FC Actuator"
    Act := "Bx Act D".Act311
    FConv := "Bx FC D".FConv8

CALL "Bx FC Sew MoviDrive"
    PEW := 4084
    PAW := 4084
    Data := "Bx FC D".FConv8
    Visu := "Bx FC Visu D".FConv[8]
```

Reversible Frequency Drive

```
U      #VesselSec
UN     "Bx DIn D".DIn120.GAlS
=      "Bx Act D".Act311.Rel2

L      "Bx PID D".PID59.OVal
T      "Bx FC D".FConv8.SPext
```

```
//if Act 311 is activated, it will run in "normal" direction
//if Act 312 is activated, it will run in "Reverse" direction
CALL  "Bx FC Actuator Reversabl"
  Act      :="Bx Act D".Act311
  ActReverse:="Bx Act D".Act312
  FConv     :="Bx FC D".FConv8

CALL  "Bx FC Sew MoviDrive"
  PEW :=4084
  PAW :=4084
  Data:="Bx FC D".FConv8
  Visu:="Bx FC Visu D".FConv[8]
```


ProfiDrive Protocol : Status Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0	SpeedOK	BOOL	Setpoint/actual value deviation within tolerance range
9	0.1	PControlRequested	BOOL	The automation system is requested to accept the
10	0.2	FaultSpeed	BOOL	Speed is greater than or equal to the corresponding
11	0.3	FaultOverload	BOOL	Comparison value for current, torque or power has been
12	0.4	HoldingBreakOpen	BOOL	Signal to open and close a motor holding brake.
13	0.5	FaultOvertemperature	BOOL	Alarm, motor overtemperature
14	0.6	CW	BOOL	Motor rotates Clockwise (True) or CounterClockwise (False)
15	0.7	FaultVLTOverload	BOOL	Alarm, converter thermal overload
0	1.0	ReadyToStart	BOOL	TRUE: Ready to Start. FALSE: Not Ready to start motor
1	1.1	On	BOOL	TRUE: Motor is Switched on and no Fault. FALSE: Motor not running or Error
2	1.2	Enabled	BOOL	Motor follows setpoint. See control word bits "On" and "Enable"
3	1.3	Fault	BOOL	The converter has a fault. Acknowledge fault using "AckFault"
4	1.4	On2	BOOL	Confirmation of ON2 Command Signal. Coast down to standstill is not active.
5	1.5	On3	BOOL	Confirmation of ON3 Command Signal. Quick stop is not active.
6	1.6	ClosingLockOutActive	BOOL	It is only possible to switch on the motor after an OFF1
7	1.7	Alarm	BOOL	Motor remains switched on; no acknowledgment is

ProfiDrive Protocol: Control Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0	CW	BOOL	TRUE: Clockwise operation. FALSE: no function. (Specific to Drive)
9	0.1	CCW	BOOL	TRUE: CounterClockwise operation. FALSE: no function. (Specific to Drive)
10	0.2	DPControl	BOOL	TRUE: Profibus Control Active. FALSE: VLT Ignores DP commands
11	0.3	C11	BOOL	Specific to Drive
12	0.4	C12	BOOL	Specific to Drive
13	0.5	C13	BOOL	Specific to Drive
14	0.6	C14	BOOL	Specific to Drive
15	0.7	C15	BOOL	Specific to Drive
0	1.0	On	BOOL	TRUE: Turn On Motor, if "Enabled". FALSE: Ramp down the motor and then switch off
1	1.1	On2	BOOL	TRUE: Motor can be turned on. FALSE: Switch Off motor immediatly with Coast dow
2	1.2	On3	BOOL	TRUE: Motor can be turned on. FALSE: "Quick Stop" Ramp down with time "OFF3"
3	1.3	Enable	BOOL	TRUE: Motor is released. FALSE: Immediatly switch off motor
4	1.4	RFGActive	BOOL	TRUE: Ramp down active (Operation Condition). FALSE: Ramp down deactivated
5	1.5	RFGEnable	BOOL	TRUE: Ramp down enabled (Operation Condition). FALSE: Ramp value is frozen
6	1.6	EnableSP	BOOL	TRUE: Motor Accelerates to Setpoint. FALSE: Ramp down the motor to standstill
7	1.7	AckFault	BOOL	FALSE -> TRUE: Acknowledge fault

ProfiDrive Protocol: Alarm Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0		BOOL	
9	0.1		BOOL	
10	0.2	FaultSpeed	BOOL	Speed is greater than or equal to the corresponding
11	0.3	FaultOverload	BOOL	Comparison value for current, torque or power has been
12	0.4		BOOL	
13	0.5	FaultOvertemperature	BOOL	Alarm, motor overtemperature
14	0.6		BOOL	
15	0.7	FaultVLTOverload	BOOL	Alarm, converter thermal overload
0	1.0		BOOL	
1	1.1		BOOL	
2	1.2		BOOL	
3	1.3	Fault	BOOL	The converter has a fault. Acknowledge fault using "AckFault"
4	1.4		BOOL	
5	1.5		BOOL	
6	1.6		BOOL	
7	1.7	Alarm	BOOL	Motor remains switched on; no acknowledgment is

SEW MoviDrive: Status Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0	StatusOrFault0	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
9	0.1	StatusOrFault1	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
10	0.2	StatusOrFault2	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
11	0.3	StatusOrFault3	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
12	0.4	StatusOrFault4	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
13	0.5	StatusOrFault5	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
14	0.6	StatusOrFault6	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
15	0.7	StatusOrFault7	BOOL	Either Fault code or Status Code denpending if FAULT is TRUE
0	1.0	On	BOOL	TRUE: Motor is Switched on and no Fault. FALSE: Motor not running or Error
1	1.1	ReadyToStart	BOOL	TRUE: Ready to Start. FALSE: Not Ready to start motor
2	1.2	DPControlRequested	BOOL	The automation system is requested to accept the
3	1.3	RampSel	BOOL	Ramp generator Selection
4	1.4	ParamSetSel	BOOL	Parameter set selection
5	1.5	Fault	BOOL	The converter has a fault. Acknowledge fault using "AckFault"
6	1.6	LimitCW	BOOL	
7	1.7	LimitCCW	BOOL	

SEW MoviDrive: Control Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0	Rev	BOOL	TRUE: CounterClockwise operation. FALSE: no function. (Specific to VLT)
9	0.1	Potionmeter1	BOOL	Potionmeter control. Keep at FALSE
10	0.2	Potionmeter2	BOOL	Potionmeter control. Keep at FALSE
11	0.3	LocalSP1	BOOL	Binary coded. 0 = Bus SP, 1,2,3 Local Setpoints
12	0.4	LocalSP2	BOOL	Binary coded. 0 = Bus SP, 1,2,3 Local Setpoints
13	0.5	SPSelection	BOOL	Unclear; Keep at FALSE
14	0.6	C14	BOOL	
15	0.7	C15	BOOL	
0	1.0	Inhibit	BOOL	TRUE: Motor is released. FALSE: Immediatly switch off motor
1	1.1	On2	BOOL	TRUE: Motor can be turned on. FALSE: Switch Off motor immediatly with Coast dow
2	1.2	On	BOOL	TRUE: Turn On Motor, if "Enabled". FALSE: Ramp down the motor and then switch off
3	1.3	Hold	BOOL	TRUE: Hold Active; FALSE: no Hold
4	1.4	RampSel	BOOL	TRUE: Ramp 2; FALSE: Ramp 1
5	1.5	ParamSetSel	BOOL	TRUE: Param Set selection 2; FALSE: Param set selection 1
6	1.6	AckFault	BOOL	FALSE -> TRUE: Acknowledge faults
7	1.7	C8	BOOL	

SEW MoviDrive: Alarm Word

The alarm word of SEW Movidives is an Integer number indicating the current Alarm that is present in the drive. It must be interpreted as an Integer, rather than a Bit Array.

The Status codes are:

- 0: Not ready
- 1: Controller Inhibit
- 2 : No Enable
- 3: Stand still, current Active, No enable
- 4: Enable
- 5: Factory Setting Active

The Alarm codes are dependent on the MovDrive Model and should be looked up in the corresponding Manual.

Danfoss FC Protocol: Status Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0	SpeedReference	BOOL	
9	0.1	BusControl	BOOL	
10	0.2	FreqLimitOK	BOOL	
11	0.3	InOperation	BOOL	
12	0.4	Stopped	BOOL	
13	0.5	VoltageError	BOOL	
14	0.6	TorqueError	BOOL	
15	0.7	TimerError	BOOL	
0	1.0	ControlReady	BOOL	
1	1.1	DriveReady	BOOL	
2	1.2	Enable	BOOL	
3	1.3	Trip	BOOL	
4	1.4	Error	BOOL	
5	1.5	Reserved	BOOL	
6	1.6	TripLock	BOOL	
7	1.7	Warning	BOOL	

Danfoss FC Protocol: Control Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0	Jog1	BOOL	
9	0.1	Ramp1or2	BOOL	
10	0.2	DataValid	BOOL	
11	0.3	Relay01	BOOL	
12	0.4	Relay04	BOOL	
13	0.5	SelectionLSB	BOOL	
14	0.6	SelectionMSB	BOOL	
15	0.7	Reverse	BOOL	
0	1.0	RefValueLsb	BOOL	
1	1.1	RefValueMsb	BOOL	
2	1.2	DcBrake	BOOL	
3	1.3	enable	BOOL	
4	1.4	QuickStopOrRamp	BOOL	
5	1.5	HoldFreqOrRamp	BOOL	
6	1.6	Start	BOOL	
7	1.7	Reset	BOOL	

Danfoss FC Protocol: Alarm Word

HMI Address	PLC Address	Symbol	Type	Remark
8	0.0		BOOL	
9	0.1		BOOL	
10	0.2		BOOL	
11	0.3		BOOL	
12	0.4		BOOL	
13	0.5	VoltageError	BOOL	
14	0.6	TorqueError	BOOL	
15	0.7	TimerError	BOOL	
0	1.0		BOOL	
1	1.1		BOOL	
2	1.2		BOOL	
3	1.3	Trip	BOOL	
4	1.4	Error	BOOL	
5	1.5		BOOL	
6	1.6		BOOL	
7	1.7	Warning	BOOL	

Material Modules

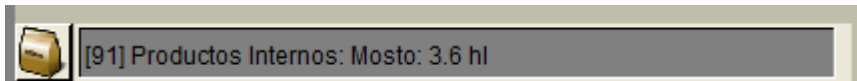
A material module allows you to easily create Material transfers either for Raw materials, produced materials or Transfers between two batches. This module allows you to specify a material that is being transferred, and origin, destination batch and an amount. When the Transfer is marked as “Finished”, a new historical record for this transfer is created.

For material transfers one must consider the following definitions:

- Raw Material Intake: A material is put into the batch, which has no “Origin”. These are usually things like Malt, Sugar, Hops, and other ingredients that go into a Batch. These dosing can either be “Manual”, meaning the dosing amount is not metered, or “Automatic”, meaning that there is some kind of measurement to determine the amount of this material that went in.
- Batch to Batch Transfers: These are the most important and common transfers. These are transfers that connect two batches together to form a relationship between them. An example of this would be the transfer of a “Brewhouse” brew into a Fermenting “Batch”.
- Produced Material: These transfers are not common, and refer to materials that are produced, but somehow extracted from the batch. For example, “Spent Grains”, “Alcohol” from dealcoholizing plants.

In summary the Material module has two objectives:

- Record Material intakes.
- Create relations between two or more batches to form a Batch Trace



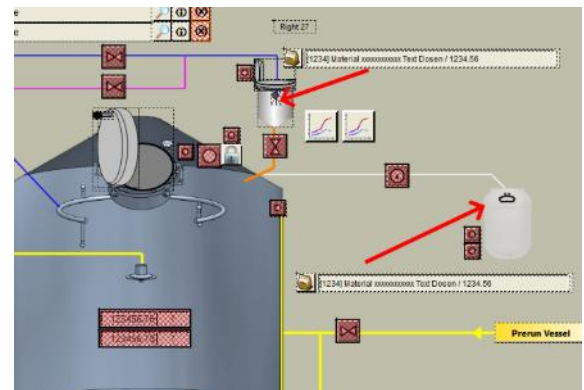
A Material Transfer Module

Conceptually a material module should be used for “each possibility of a Dosing” and “Each possibility of Batch Transfer”. For example, you should have a material module for

- Each Hops dosing Pot
- For each dosing type in the Mash tun for example “Lactic acid”, another one for “Enzymes” et.
- For each dosing in the Wort kettle for example “Honey”
- For each dosing in the Wort kettle for example "Gypsum"

If you have an automatic dosing pump, you should implement a Material Module for each of the Dosing Pumps, or automatic dosing mechanisms.

In Summary, you should represent each manual addition step and each automatic dosing that is implemented in the process, by its own material module.



Configuration of available materials

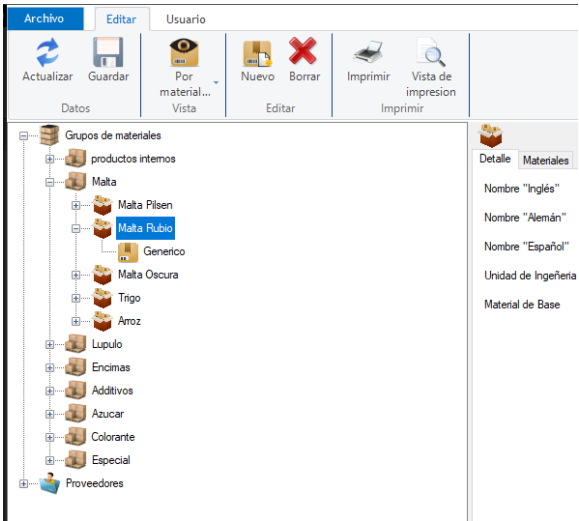
All the configuration of material types and materials is done in the “Material Configurator” application. In this application you can manage your different Material Types and the specific materials in them.

Material Group

A material group is a hierarchical entity to group material types of a specific category together. The main purpose of Material groups is the organization of Material types into categories.

Material Type

A Material is a specific “Product” that can be added into a Batch or is produced by a batch. Material type can be for example: Wheat Malt, Aroma Hops, Honey, Cold Wort etc.



Material

A specific material is a specific batch, lot, or serial number of products of a material type. If an installation receives multiple batches, for example aroma hops, each batch of received aroma hops would be a specific Material. When it comes time to add the aroma Hops to the Brew, the operator can select the specific serial number of hops he is inputting into the batch.

If no specific material exists, you can use the “Generic” material, that exists for all Material types.

Material is a way to be able to track the specific identification of a specific batch of raw materials that was used during production.

Faceplate

PLC01MM01x

BatchXpert Material Management Win Time / sec: 0123456789 0

0	??	??
---	----	----

Material Type: [123] Material Type XXXXXXXXXXXXXXXX ▼

Material: [123] Material XXXXXXXXXXXXXXXX ▼

Transfer Quantity: 0123456789 01 ?? Send Data

Share: 0123456789 01

Transfer time: 0123456789 01 min

	Own	Partner
PrID	0123456789 01	0123456789 01
Start ID	0123456789 01	0123456789 01
End ID	0123456789 01	0123456789 01

1 Raw Material

1 Product

1 Side Product

1 Reset also Quantity 0

1 Reset also Material 0

Alarm 0

Status Warning

1 Operator Request 0

Transfer Active

Transfer Done

Reseted

Material receive

1 Module Active 0

1 Auto 0

1 Simulation 0

1 Ignore 0

Manual and Automatic Mode

Material transfer can be done Manually or in Automatic mode. If you implement a Material module in the process itself, it is usually done “Automatically”. The status is defined as:

- **Automatic:** The Plc defines when a transfer starts and ends. The operator may still enter the quantity, but fundamentally the transfer is linked to the process of the units involved. The quantity may be provided automatically by some sensor or provided manually by the operator. The crucial distinction is that at least either the source or destination PRID is being assigned by the plc “Automatically”.
- **Manual:** The Transfer is not linked to any current process. It is entirely done by the operator, and the Material module functions as a mere method of registering this transfer. Usually this may happen if you have a Manual cellar and after the fact that, for example, and green beer transfer happens, the operator creates an “Manual” batch Transfer that links two batches together. Of course, this still requires there to be batches that the transfer can be linked to, so some process automation must be present anyway.

In 99% of the cases, you will want to use “Automatic” operating mode.

Raw Material, Product and Side product

These settings define how this transfer should be treated. In the plc there is an indicator for “Prid” which always refers to the current Units batch, and the “Partner Prid” to the associated batch. Depending on the setting, the partner Prid will be either “0” for Raw material intakes, or the destination batch where the transfer goes to. Side products are the opposite of Raw material transfers. These also do not have an “Partner Prid” associated, but instead of going into the current batch, they go out. These are materials such as “spent Grains”, “used Filtration Earth”, etc.

Operation Requests

If the dosing is a manual operation, you may generate an “Operator Request”, so that the module will be marked with operator request and requires confirmation. At that moment it is also possible for the operator to select a specific quantity and even select a specific Material to be added into the batch transfer.

General Usage for Raw material intakes

A raw material intake is a transfer that has no “Origin” or “Source” PRID associates with it. This is usually the case for dosing such as “Malt”, “Hops” and other ingredients, which may be entered into a batch either manually (for example Hops), or automatically by a dosing system.

The Module must be configured as “Raw Material Intake”

Manual dosing with Operator Request

```
NETWORK
TITLE =phase 11: Prepare Dosings

X011: SET ;
//user Message
    U    "PA";
    S    "OpReq";           //this will trigger the Operator Request for the Process unit

//Set modules Destination Prid
//Since it is a material going into this batch, we are the destination, and there is no "Source"
//how the PRID and Partner_PrID are considered, depends on the setting of "Raw Material",
//"Product" or "Side Product"
    L    L#0;
    T    "Bx MM D".Mat21.Partner_PrID
    T    "Bx MM D".Mat22.Partner_PrID

    L    "Uxx".U.Prid;
    T    "Bx MM D".Mat21.Prid
```

```

T      "Bx MM D".Mat22.Prid

// activate material modules
UN     "PFCycle";
SPB    fc11;

//if the Dosing is requested in from the recipe, then we request the operator to confirm the Material
//Module
//Mash Tun - Additive 1
L      "Uxx".Para[31].Sp;           //We preset the MM's quantity to the one requested by the recipe
T      "Bx MM D".Mat21.Val;         the operator may still adjust //This, before confirming
L      0.000000e+000;
>R     ;
S      "Bx MM D".Mat21.OPReq;       //this will mark the MM as Operator requested

//Mash Tun - Additive 2
L      "Uxx".Para[32].Sp;
T      "Bx MM D".Mat22.Val;
L      0.000000e+000;
>R     ;
S      "Bx MM D".Mat22.OPReq;
fc11: SET ;

```

Automatic dosing with measurement for dosing quantity

```

NETWORK
TITLE =phase 11: Dosing Lactic Acid

X012: SET

//Material Transfer
U      "PFCycle"
=      "Bx MM D".Mat20.Reset       //we reset the module to prepare it for a new Transfer

U      "PA"
=      "Bx MM D".Mat20.Start       //We start the Transfer, which generates a historical record

U      "Uxx".Para[2].D             //We end the transfer when dosing is finished. This will record
O      "PLCycle"                   //The amount and the dosing time with source and destinations
=      "Bx MM D".Mat20.End

L      "Uxx".Para[2].Val           //We provide the dosing quantity from any measurement we like
T      "Bx MM D".Mat20.Val         //in this example it is done by "Time"

//Dosing Parameters
U      "PA"
S      "Uxx".Para[2].S

UN     "Bx Act D".Act49.On
U      "PA"
=      "Uxx".Para[2].H

UN     "Uxx".Para[2].D
U      "PH"
S      "Bx Act D".Act49.ACo

// phase end condition
U      "Uxx".Para[2].D
U      "Uxx".Para[4].D
=      "PhaseEnd"

SPA    END

```

Usage for Batch-to-Batch transfers

Batch-to-Batch Transfers are like Raw material intakes, with the difference that they have an “Partner PRID” associated to them, that of course corresponds to the Batch where we are transferring into.

Transferring Wort into a Fermenting Cellar

In this example we are transferring wort via a Wort line unit into a selected fermenting cellar. The wort line unit will always have the same process PRID as the wort cooler, as it still belongs to the brewhouse. The fermenting tank has its own PRID, since it mixes different batches into a new batch.

```
NETWORK
TITLE =phase 11:  Transfer

X007: SET
// signals to partner
U    "PA"
S    "Bx UnitCom D".U.Master1.FillReq
S    "Bx UnitCom D".U.Master1.FillActive
S    "Bx UnitCom D".U.Slave1.TransReq
S    "Bx UnitCom D".U.Slave1.TransActive

U    "PH"
S    "Bx UnitCom D".U.Master1.FillRel
S    "Bx UnitCom D".U.Slave1.TransRel

U    "PH"
U    "Bx UnitCom D".Master1.OpenTank
S    "Bx UnitCom D".U.Slave1.OpenTank
//Material module
U    "PA"
=    "Bx MM D".Mat1.Start

// transfer prid
L    "PrId"
T    "Bx MM D".Mat1.PrID

SET                                     //we select "Product Transfer" in the plc, so it cannot be changed
R    "Bx MM D".Mat1.Receive             //by the Faceplate. Just so nobody can select incorrect options

L    "Comm KS2 Data".ReceiveData.WCcomm.Val1           //The Wort quantity comes in from an
T    "Bx MM D".Mat1.Val                               //Communication from the wort cooler

L    1.000000e+000
T    "Bx MM D".Mat1.Share           //We have 1 = 100% share of the transfer

L    "Bx UnitCom D".Slave1.PrId           //the Partner PRID is the PrId of the selected Fermenter
T    "Bx MM D".Mat1.Partner_PrID

//phase end condition
U    "Bx UnitCom D".Master1.TransEnd
O    "Bx UnitCom D".Master1.Step0
UN   "PFCycle"
S    "PhaseEnd"
S    "Bx MM D".Mat1.End                 //We set the End of the Material transfer when the
                                         //phase ended

SPA   END
```

Other settings

The “Reset Quantity” and “Reset Material” are options to automatically reset the corresponding value whenever a new transfer starts by setting “Bx MM D”.Matxx.Reset” to TRUE. This forces the operator to input new valid data to be able to confirm the operator’s request.

Process Unit

A "unit" is a production unit, such as the Fermentation tank, Pasteurizers or Filter, called in the above systems and sequencer. Programming these Units is where most of the programming is being done in a typical project. Units contain all the Phases and action and are the consumers of all the control modules. To achieve quick and easy programming, there are some helper functions and Tools provided.

Units are the entities that execute the phases of the steps defined in a Recipe. A unit will carry out the following tasks:

- Copy the Parameters from the current recipe step to the current parameters in the Unit Data block
- Execute the Unit function, with the corresponding Phase, according to the current step
- Register Historical events if any Phase or step has finished.

General Structure of a Unit in the PLC

In the PLC, and Unit is always composed of:

- A Unit data block which holds the Units recipe, status, and some user data.
- A Unit Function block, which includes most of the general unit settings, such as Actuator release, Parameter transfers, and other logic that is not related to recipe execution. This Function block, eventually called FC100, which then called the Unit Function.
- A Unit Function, where all the Units Actions are programmed.

The Units follow a fixed numbering scheme of: Unit DB, FC, and FB = Unit Number + 100. For example:

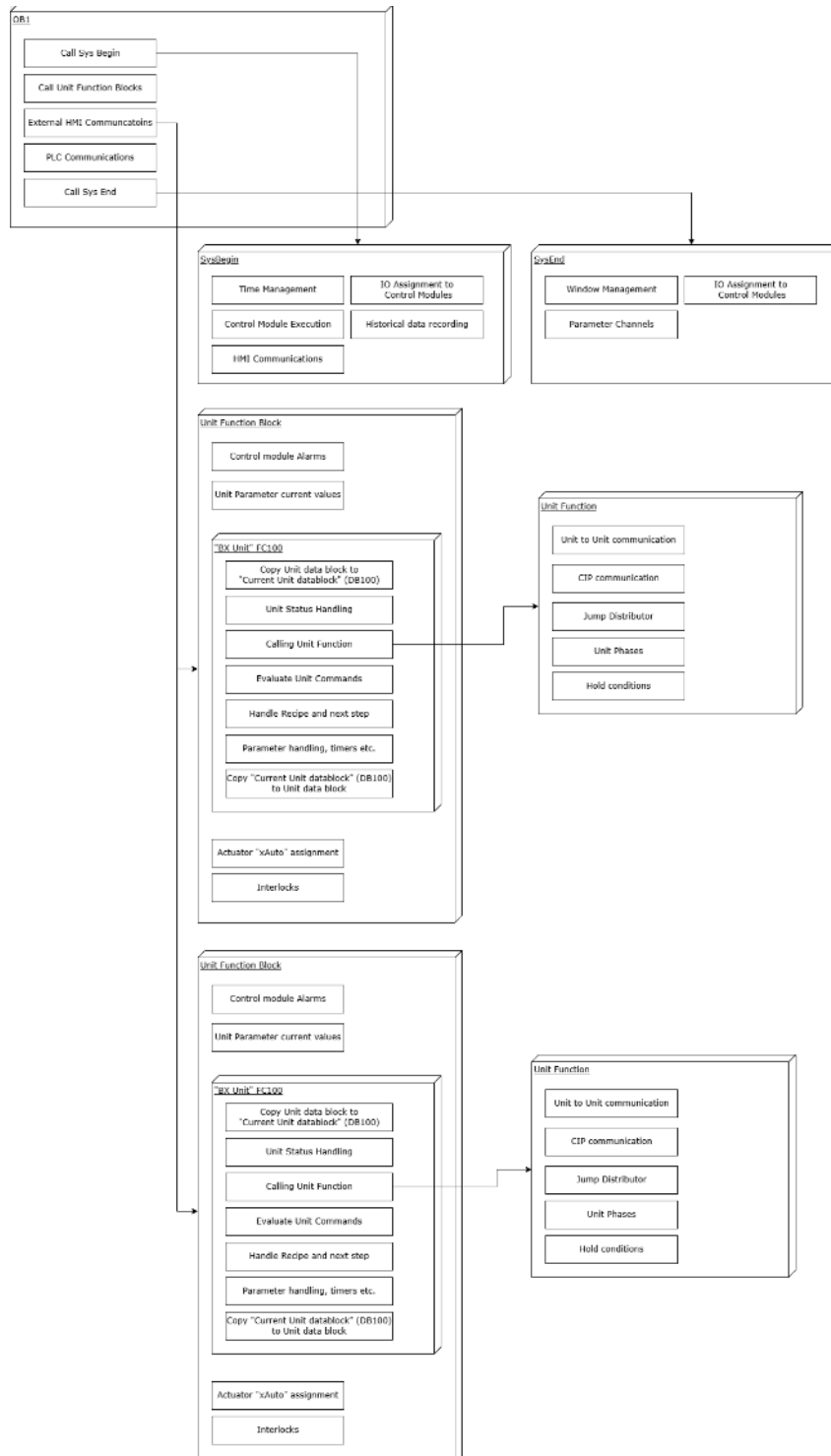
- Unit 1 DB 101, FB 101, FC 101
- Unit 2 DB 102, FB 102, FC 102
- Unit 120 DB 220, FB 220, FC 220

Flowchart

The following diagram represents the execution flow of a unit of one single PLC cycle:

- First starts with the OB1 where the unit's FB is called.
- In the FB, actuator interlocks (releases) are performed, process values and parameters are transferred, and the FC of the same unit is called.
- In the FC, the executions that must be conducted in each step are conducted, counters are reset, etc.

The flowchart in the cycle can be represented as follows.



Unit Function Block

The Unit function block represents the Entry point for the implementation of a Process unit and its functionality. The function block is an FB that should have no Parameter and no Static values, so that it can be called by an “UC” (Unconditional Call) from the OB1 plc cyclic execution Organization block.

The Process Units function block has the following responsibilities, more details in the next

- Transfer Current values to the Units Parameter Modules
- Call FC100, which will be called the Unit Function, where all the phases are implemented.
- Activate Digital input alarms
- Activate Analog input alarms
- Handle Counters, assign pulse values to them, or generate pulses from current flow values
- Set xAuto on the Actuator control modules, so they assume their automatic condition from this unit
- Implement Actuator Release logic
- Any other additional logic, which is not dependent on a specific recipe or phase

Unit Function

The unit function implements all the logic of a Process unit’s phases. These phases are being called by a recipe from the BatchXpert configuration, where you define the order of phases and steps to execute, and what parameters exist in each phase.

The unit function will be executed implicitly from the FC100 call, in your Units Function Block.

You will find more information in the dedicated chapter of this manual.

User Data

Many implementations need to store some user data specific to a unit. This may be the accumulation value of an “Flow Integrator” function, the temporary value of an average calculation etc. For this, each unit includes an “USER” section in its Unit data block. These values can be freely used for any purpose the programmer sees fit.

Address	Symbol	Type	Remark
0.0	b24	BOOL	user bit
0.1	b25	BOOL	user bit
0.2	b26	BOOL	user bit
0.3	b27	BOOL	user bit
0.4	b28	BOOL	user bit
0.5	b29	BOOL	user bit
0.6	b30	BOOL	user bit
0.7	b31	BOOL	user bit
1.0	b16	BOOL	user bit
1.1	b17	BOOL	user bit
1.2	b18	BOOL	user bit
1.3	b19	BOOL	user bit
1.4	b20	BOOL	user bit
1.5	b21	BOOL	user bit
1.6	b22	BOOL	user bit
1.7	b23	BOOL	user bit
2.0	b08	BOOL	user bit
2.1	b09	BOOL	user bit
2.2	b10	BOOL	user bit

2.3	b11	BOOL	user bit
2.4	b12	BOOL	user bit
2.5	b13	BOOL	user bit
2.6	b14	BOOL	user bit
2.7	b15	BOOL	user bit
3.0	b00	BOOL	user bit
3.1	b01	BOOL	user bit
3.2	b02	BOOL	user bit
3.3	b03	BOOL	user bit
3.4	b04	BOOL	user bit
3.5	b05	BOOL	user bit
3.6	b06	BOOL	user bit
3.7	b07	BOOL	user bit
4.0	DINT0	DINT	user long int
8.0	DINT1	DINT	user long int
12.0	DINT2	DINT	user long int
16.0	DINT3	DINT	user long int
20.0	DINT4	DINT	user long int
24.0	DINT5	DINT	user long int
28.0	DINT6	DINT	user long int
32.0	DINT7	DINT	user long int
36.0	DINT8	DINT	user long int
40.0	DINT9	DINT	user long int
44.0	DINT10	DINT	user long int
48.0	DINT11	DINT	user long int
52.0	DINT12	DINT	user long int
56.0	DINT13	DINT	user long int
60.0	DINT14	DINT	user long int
64.0	Val0	REAL	user value
68.0	Val1	REAL	user value
72.0	Val2	REAL	user value
76.0	Val3	REAL	user value
80.0	Val4	REAL	user value
84.0	Val5	REAL	user value
88.0	Val6	REAL	user value
92.0	Val7	REAL	user value
96.0	Val8	REAL	user value
100.0	Val9	REAL	user value
104.0	Val10	REAL	user value
108.0	Val11	REAL	user value
112.0	Val12	REAL	user value
116.0	Val13	REAL	user value
120.0	Val14	REAL	user value
124.0	Val15	REAL	user value

User Data Example

Simple Time Delay counter

```
//Count timer
L UxxD.User.TimerDelayMem          //Accumulated Time
L CycleTimeSec                     //Current time in seconds: 0,012 = 12 msec
+R
T UxxD.User.TimerDelayMem          //Save new sum to the accumulation
```

```
//Check timer
L UxxD.User.TimerDelayMem
L 30.0 //30 seconds
>R
SPBN TiDo
L 0.0
T UxxD.User.TimerDelayMem //Reset Accumulated Time to restart it

//Do something here
TiDo:
```

Detect Silo number has changed

By using User Variables, the transfer of the current silos has a one-cycle delay. Thus, to activate a log entry in a silo it can be swapped, thereby logging the deletion of the previous silo.

UserDint 01: Memory of the next silo number:

```
L "Dxx".User.Dint1DIn1
T "Uxx".Para[13]. Val

L "SiloNo"
L "Dxx".User.Dint1
<> D
U(
L 0
>D
)
S "ProtWrite"

L "SiloNo"
T "Dxx".User.Dint1
```

Unit Data block

A Unit data block which holds the Units recipe, status, and some user data.

Data block overview

Address	Name	Data type	Description
0.0	U	"sBx Unit"	General Unit status, Program Numbers, Prid etc.
80.0	User	P01U004	The User Data Area
208.0	StartOption	"sBx StartOption"	The Start Options that where active for the current Recipe
240.0	Property	ARRAY[1...32] of REAL	The unit properties
368.0	StatusInfo	"sBx UnitStatusInfo"	The Current Unit's Hygienic status
400.0	Para	ARRAY[1...40] of "sBx ParaM"	The parameters of the current recipe's current active phase
1000.0	Recipe		The current Recipe
1000.0	Recipe. Header	sBx RecipeHeader	The header of the current Recipe
10024.0	Recipe.Steps	ARRAY[0...19] of "sBx RecipeStep"	All the steps of the Current Recipe

Data block "Unit Status"

Address	Symbol	Datatype	Comment
0.0	Used	BOOL	spare
0.1	HornOpRequest	BOOL	horn operator request
0.2	OpRequest	BOOL	operator request
0.3	s03	BOOL	spare
0.4	s04	BOOL	spare
0.5	s05	BOOL	spare
0.6	s06	BOOL	spare
0.7	s07	BOOL	spare
1.0	Steril	BOOL	unit sterile
1.1	Clean	BOOL	unit clean
1.2	NotClean	BOOL	unit not clean
1.3	Product1	BOOL	product 1
1.4	Product2	BOOL	product 2
1.5	Product3	BOOL	product 3
1.6	Product4	BOOL	product 4
1.7	ReqCIP	BOOL	Unit must be Cleaned before next Production
2.0	GAlQuitt	BOOL	general alarm acknowledge
2.1	Ign	BOOL	ignore alarm
2.2	Sim	BOOL	simulation
2.3	Run	BOOL	unit in run mode
2.4	Pause	BOOL	unit in pause mode
2.5	Hold	BOOL	unit in hold mode
2.6	EmHold	BOOL	spare - unit in emergency hold
2.7	Maint	BOOL	maintenance
3.0	GAl	BOOL	general alarm

3.1	GAIS	BOOL	general alarm save
3.2	SCE	BOOL	spare - status check error
3.3	Watchdog	BOOL	watchdog alarm
3.4	Step0	BOOL	unit in step 0
3.5	ReadyStart	BOOL	unit ready for start
3.6	Active	BOOL	unit active (not step 0)
3.7	CIPModus	BOOL	unit in CIP modus
4.0	UnitNo	DINT	unit number
8.0	Phase	DINT	phase number
12.0	StepNo	DINT	step number
16.0	Charge	DINT	charge number
20.0	PrId	DINT	PrId
24.0	ProgNo	DINT	program number
28.0	Message	DINT	message text
32.0	THold	REAL	time Unit in hold
36.0	TRun	REAL	time unit in run
40.0	TStepRun	REAL	time step in run
44.0	TRecDownLoad	REAL	time recipe download

Unit Function Block

The Unit Function Block is where you manage and set up most of the logic that must run independently of any Recipe or recipe phase logic. This usually includes things like “Alarm conditions”, “Parameter Transfers” and “Actuator interlocks”.

Eventually this function block must call FC100 with the current unit’s number as input parameter, so that it can eventually call your Units function, with parameters from the currently executing recipe.

The Unit Function block is executed exactly once every plc execution cycle (directly from OB1).

Usual Structure of a Units Function Block

Usually, a Unit Function block will have the following general structure, although this example is a stripped-down example, which should only indicate its functionality.

```
FUNCTION BLOCK "U005 config"
TITLE =Unit 005 Example Unit
BEGIN
NETWORK
TITLE =Init
//Here we preset some Help Variables that we can use later down for creating all
//the Interlocks

//Here we set the Emergency Stop Input that applies.
UN    "Bx DIn D".DIn[7].Gals      //We want the User to confirm the alarm
U     "Bx DIn D".DIn[7].Sig
=     #EmStop

//Here we set vessel specific safety equipment, such as "Manholes", "Key Switches"
//et. This will block Actuators that are Connected directly to the interior of the
//Tank
UN    "Bx DIn D".DIn[8].Gals      //Manhole Switch: We want the User to confirm the alarm
U     "Bx DIn D".DIn[8].Sig
UN    "Bx DIn D".DIn[9].Gals      //Key Switch: We want the User to confirm the alarm
U     "Bx DIn D".DIn[9].Sig
=     #VesselSafe

//We can mark the Actuator as "Safety relevant", so the user cannot activate
//simulation nor Ignore form the Faceplates. The Delay times are also limited
//to a few seconds (so the User can //not input an OFF Delay of 9999 Seconds,
//effectively eliminating the alarm).
Set
S     "Bx DIn D".DIn[8].xSafety   //Manhole Switch
S     "Bx DIn D".DIn[9].xSafety   //Key Switc

NETWORK
TITLE =Parameter transfer
// Here we transfer some Current values form the corresponding Control modules
// to the Parameter modules of this Unit
L     "Bx Cnt D".Cnt3.PVal;       //Totalizer Water Amount
T     "U005 data".Para[15].Val;   //Amount Water

L     "Bx AIn D".AIn14.PVal;      //Level of Unit
T     "U005 data".Para[14].Val;   //Level of Unit

L     "Bx AIn D".AIn19.PVal;      //Temperature of Unit
T     "U005 data".Para[18].Val;   //Temperature of Unit

... //here are more assignments, as needed
```


NETWORK

TITLE =Unit Phase Control

//Here we must call FC100, which in turn eventually will call the Unit function,
//where all the Unit Phases are implemented. It is important to assign the Current
//Units number to the function

```
CALL "Bx Unit" (  
    UnitNo                := 5); //Here we must give the Unit Number
```

NETWORK

TITLE =Analog Inputs

//Here we can activate any analog input related Alarm. Usually this means that we
//activate extreme limites.

//

//Please keep in mind, that you can also activate these alarm limits from the
//Faceplate, so if we set them here in the PLC, we take away the option to adjust
//them from the facplate, since the PLC will always overwrite the value set by
//the faceplate.

//

//This means, that we should only set alarms that are absolutly necessary, or have
//some kind of logic to them, and leave the other alarms to the user to be
//activated or deactivated.

//Permanent Alarms

Set

```
S    "BX AIn D".AIn[1].EHHA    //Volume of Tank, high alarm, to avoid overfilling
```

//Simple Alarm logic.

//If the Actuator output is TRUE, meaning the Actuator tries
//to activate, the Low Low Alarm is activated. Keep in mind that the Alarm delay
//will still apply and give the alarm condition some thime to "normalize"

```
U    "BX Act D".Act[179].Out    //Pump Running
```

```
S    "BX AIn D".AIn[1].ELLA    //Flow Alarm
```

// low water flow

```
U    "Bx Act D".Act130.Out;    //Product Pump
```

```
S    "Bx AIn D".AIn19.ELLA;    //Product Flow Low Alarm
```

// alarm low pressure in pipe

```
UN   "Bx Act D".Act127.Out;
```

```
U    "Bx Act D".Act304.Out;
```

```
UN   "CIPModus";
```

```
S    "Bx AIn D".AIn11.ELLA;
```

NETWORK

TITLE =Calculated values

//Here we have a calculated value that is represented as an Analog Input
//We do not want any scaling to happen, since we already calculated its
//Process value. We avoid this by setting the "NPA" signal

//AIn 60: LT_P_02x1: difference pressure measurement lauter tun

```
L    "Bx AIn D".AIn12.PVal;
```

```
L    "Bx AIn D".AIn11.PVal;
```

```
-R    ;
```

```
T    "Bx AIn D".AIn60.xPVal;
```

```
SET    ;
```

```
S    "Bx AIn D".AIn60.NPA;
```

NETWORK

TITLE =Digital Input Alarm

//Similarly here we activate Digital Alarms

//Simple example

SET

S "Bx DIn D".DIn[2].EA0 //if the Digital input becomes FALSE, it will activate an alarm

//Example for alarms depending on the Process

//if the Wron panel os connected, or the correct one is missing, we activate an
//alarm

U #Production

S "Bx DIn D".DIn[12].EA0 //Panel Connection sensor for Production

S "Bx DIn D".DIn[13].EA1 //Panel Connection sensor for CIP

U #CIP

S "Bx DIn D".DIn[12].EA1 //Panel Connection sensor for Production

S "Bx DIn D".DIn[13].EA0 //Panel Connection sensor for CIP

//Using an "LSL Empty Signal" as dry run Protection for pumps

//If the pump is transferring from the vessel, we activate an Alarm on the LSL

//when it is "Empty" and adjust an Alarm delay of arround 30 seconds.

//This means, if the pump runs for more than 30 seconds, while the vessel is empty

//it will activate an alarm, which in turn can block the pump

U "BX Act D".Act[179].Out //Transfer Pump

U "Bx Act D".Act[123].Out //Outlet Valve of Vessel

S "Bx DIn D".DIn[1].EA1 //Empty signal of Vessel (TRUE = Empty)

//Similarly to the Analog Input alarm, we manage Digital input alarms

SET ; //These are unconditionally active

S "Bx DIn D".DIn37.EA0;

S "Bx DIn D".DIn38.EA0;

S "Bx DIn D".DIn40.EA0;

U "Bx Act D".Act304.Out; //Here the alarm depends on an Actuator

S "Bx DIn D".DIn84.EA0;

//Manhole Alarm

UN "U005 data".U.Step0; //The alarm is not active in "Start position"

S "Bx DIn D".DIn169.EA0;

NETWORK

TITLE =Counters

//Here we have to activate the Conters "Count signal monitoring" if the transfer
//pumps are running and also assign the Counting signal to the Counter module.

//

//We recommend that you configure your Flow meters to deliver a "Normally High"

//signal, which becomes "LOW" when a pulse is transmitted. The reason for this

//ist, that then you can make an "Alarm when FALSE" monitoring on the Counting

//signal, with a delay time of 2 seconds, knowing the pulse will pull the

//inputs low for only a few miliseconds.

//But if the counting input gets disconnected, you will create an alarm.

//

//Most modern Flow sensors allow you to adjust the Counting signal to "Normally

//High". Sometimes the setting is called "NPN" or "PNP", referring to the

//Transistor type that is used.

//if the Pump is running, we activate the Pulse monitoring.

//this montiores that we receive impuleses earlier than the

//Alarm delay on the Conouter module.

U "BX Act D".Act[179].Out //Transfer Pump

```

S      "BX Cnt D".Cnt[5].EAImp

//Here we assign the Counting Digital input to the Counter module
U      "BX DIn D".DIn[97].Imp
S      "BX Cnt D".Cnt[5].xSig

//As mentioned above, if we have a "Normally HIGH" counting signal, we can monitor
//it. If FALSE for more than about 2 seconds, we can safely assume that the Signal
//is damaged and activate an alarm
SET
S      "Bx DIn D".DIn[97].EA0

NETWORK
TITLE =Regulators
//Here we define when a regulator starts acting, and what its nominal and current
//values are. Usually, the Regulator starts acting when an Actuator is activated.
//if there is no physical device, you should create a "Virtual Actuator", that has
//no physical output but nevertheless can be activated by its ".Aco" and has an
//HMI symbol on the Graphics display.
//
//For regulators we basically must assign the following to the Regulator Module:
//--When should it start "Regulating"
//--What is its Setpoint
//--What is its Current Value
//--What is its "Fixed value"

//if the Valve opens, we start regulating and activating alarm monitoring
U      "Bx Act D".Act328.Out;
S      "Bx PID D".PID9.MStrt; //This will start regulating
S      "Bx PID D".PID9.EAL;   //and start the Alarm monitoring

//Transfer the Nominal and current values
L      "Uxx".Para[12].Sp;
T      "Bx PID D".PID9.xSp;

L      "Uxx".Para[12].Val;
T      "Bx PID D".PID9.xPVal;

//The Static Value is used if the Regulator receives the "MStc" command
L      "U001 data".Para[15].SP
T      "Bx PID D".PID[1].Stc

NETWORK
TITLE =Manual/Automatic Handling
//Here we assign the desired automatic state from the Unit onto the Actuator
//modules. Since we are using the "Run" symbols, which always reflects the status
//of the last FC100 call, this must be programmed after the FC100 was called,
//above.

//Actuators
U      "Run";
S      "Bx Act D".Act10.xAuto;
S      "Bx Act D".Act11.xAuto;
S      "Bx Act D".Act42.xAuto;
S      "Bx Act D".Act43.xAuto;

//Here we do the same for Regulators and Frequency Converters
R      "Bx PID D".PID5.MSpExt;
R      "Bx PID D".PID5.MCon;
R      "Bx FC D".FConv1.MSpInt;

NETWORK

```

TITLE =Act Safety Releases

```
//Here we program the logic for the "Security Release" (Rel). The user cannot
//ignore this release, and should include safety equipment, such as Manhole
//switches, Emergency Stops etc.
```

```
//Here Are Actuator Interlocks, which depend on the Emergency Stop, but are not
//Connected directly to the interior of the Tank, so the "Vessel" safety does not
//Apply to them
```

```
U      #EmStop
=      "Bx Act D".Act[27].Rel
=      "Bx Act D".Act[556].Rel
=      "Bx Act D".Act[555].Rel
=      "Bx Act D".Act[432].Rel
=      "Bx Act D".Act[123].Rel
=      "Bx Act D".Act[334].Rel
```

```
//Here Are Actuator Interlocks, which are Connected directly
//to the interior of the Tank, so the "Vessel" safety does
//Apply to them
```

```
U      #EmStop
U      #VesselSafe
=      "Bx Act D".Act[28].Rel
=      "Bx Act D".Act[56].Rel
=      "Bx Act D".Act[55].Rel
=      "Bx Act D".Act[42].Rel
=      "Bx Act D".Act[13].Rel
=      "Bx Act D".Act[34].Rel
```

```
//Actuturs that are not subject to any of the safety interlocks, come here.
//This however usually only applies to "Lamps", "Accoustic indicators" and such,
//And NEVER to Energized equipment such as Valves or Pumps.
//Energized equipment must always be interlocked by at least the Emergency Stop
```

```
SET
=      "Bx Act D".Act[13].Rel
=      "Bx Act D".Act[34].Rel
```

NETWORK

TITLE =Act Releases

For all further releases please see [Actuator Interlocks](#)

Simple Interlocks examples

More Detailed examples can be found here: [Actuator Interlocks](#).

In this example we will only show some examples. We separate it out into dedicated segments because Interlocks usually are quite complex and verbose

```
NETWORK
TITLE = Act45 - dosing pump CaCl2 lauter tun
//This pump may only run when the signal is true and the actuator 125 is off
U      "Bx DIn D".DIn88.Sig;
U      "Bx Act D".Act125.Off;
=      "Bx Act D".Act45.Rel2;

NETWORK
TITLE = Act 112 - valve CIP return lauter tun
//this Valve may only open if the other one is closed

U      "Bx Act D".Act107.Off;
=      "Bx Act D".Act112.Rel2;

NETWORK
TITLE = Act124 inlet valve 1 lauter tun
//the inlet can only open if the tank is not full
U      #VesselSec;
UN     "Bx AIn D".AIn12.MHHA;
=      "Bx Act D".Act124.Rel2;

NETWORK
TITLE = Act127 - valve weak wort to weak wort tank
//There is no Interlock, it is always enabled

SET    ;
=      "Bx Act D".Act127.Rel2;
```

Pump Interlocks

Pumps are complex to correctly interlock. The interlocks are divided into three parts:

- At least one suction way must be open for the pump to not cavitate.
- At least one pressure way must be open, so the pump has somewhere to go
- And the safety equipment must be OK. This usually includes “Dry Run” switches etc.

```
NETWORK
TITLE = Act304 - lautering pump
//Here we preset the signals, so we can later “S” them and avoid Parentheses and
complex AND/OR constructs
Clr
=      #Suction;
=      #Pressure;

//Suction side
//In this example, there is no Suction side valve, the pump is directly connected
//to the Tank
SET    ;
=      #Suction;

//Pressure to PRV
U      "Bx Act D".Act132.On;
U      "Bx Act D".Act143.On;
U      "Bx Act D".Act324.On;
S      #Pressure;

//Circulation
```

```
U    "Bx Act D".Act126.On;  
U    "Bx Act D".Act324.On;  
S    #Pressure;  
  
//Uadl  
U    "Bx Act D".Act127.On;  
U    "Bx Act D".Act324.On;  
S    #Pressure;  
  
//General Release  
U    #Pressure;  
U    #Suction;  
UN   "Bx DIn D".DIn84.GAlS;  
U    #VesselSec;  
=    "Bx Act D".Act304.Rel2;
```

Unit Parameter Module

Before we can talk about implementing Phases logic in the Unit Function, we must talk about Unit Parameter Modules, since they are a vital part of how this works.

BatchXpert allows you to define a maximum of sixteen simultaneous Parameters per Phase, from a total of forty available modules. This means that you have 40 Parameters available, but you can only assign sixteen of them simultaneously to a single phase. Of these 40 Parameter modules, parameter Module Number 1 is always used as an “Watchdog” timer. On this module you can change the time resolution (Seconds, minutes, hours), but it is always internally treated as an “Watchdog” time.

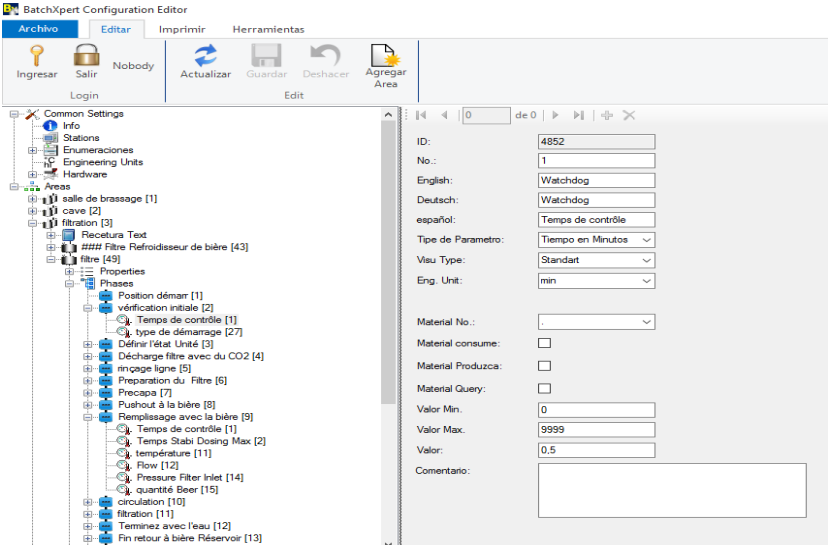


Figure 5 Example of a Parameter module configuration in the Batch Configurator

If you need more than sixteen simultaneous parameters, you should check if there is a possibility to offload some parameters into properties, switches, or other modules. For this you must analyze carefully if the parameters really must be different in each recipe, or if they can be a global parameter. A good example of this is “Empty delay Timers”. Usually these are never changed, and really work as global settings, so they should not be Parameters, but rather be Properties.

All parameters of a step are represented as a parameter line in the “Process Details”, also called “Lupe” or “Magnifier”. In this Process detail window, you will have access to all sixteen simultaneous Parameters that you defined for the current Phase of a Unit.

Parameters have a Type, which defines how they behave, a Nominal and current value, as well as some statuses that the user can use in the Phase logic.

UnitNo	1	WinOpen:	600.00	43.78	
[1] Watchdog / min:		1.00	0.86		
[11] Temperature / °C:		2.00	11.00		
[14] Quantity Water / hl:		100.00	14.00		
[15] Flow Water / hl/h:		20.00	15.00		
[16] Temperature Water / °C:		52.00	22.00		
[21] Rührwerk:		Aus			

Parameter “Done” status.

Every Parameter, independently of its type, has an “D” status, which indicates that the parameter is “Done.” The “Done” state is simply defined as:

$$D \text{ (Done)} = \text{Current Value} \geq \text{Nominal Value}$$

This means, that every time the current value is greater or equal to the Nominal value from the recipe, it is marked as “Done” from the Unit function. This means, that the “D” status can be used as comparison limit, and in the case of timers, will indicate that at least the set amount of time has passed.

Parameter type: Nominal and Current Value

This is the most common case that you will use. It simply represents a nominal value that has a current value assigned to it. It has no special functionality, except for the comparison to create the “Done” status, that every Parameter type uses.

The current value for these values MUST be assigned from the user program in the Unit function block in the [“Parameter transfer”](#) section.

Parameter type: Time as Seconds, Minutes, hours, or days

This timer will automatically increase their Current value by the time elapsed during the last execution. Depending on the time base used, the nominal and current value will represent these base values.

- If the “S” (Start) Command is true, the current value will start counting.
- If the “S” (Start) Command becomes true, the current value will reset to 0.0.
- If the “H” (Hold) Command becomes true, the current value will not count and remain on its current value.
- If the “Res” (Reset) Command becomes true, current value will reset to 0.0.

Since the Current value is automatically increasing and the “Done” status is also compared, the “Done” status effectively functions as you Indication that the “Timer has elapsed”, as long as the Parameter “S” Command stays true.

Since the current value is counting automatically, you cannot write the “Val” of the parameter in the Unit function block, as you would for an “Nominal and Current Value”.

Parameter type: Nominal Value

These work similarly to the “Nominal and Current Value”, except that they do not show the current value in the process details. The transfer of the Current value in the Unit function block is optional.

Also, the “Done” status may, or may not be valid, depending on if you have transferred a valid Current value to the “Val” of the parameter module.

Parameter type: Current Value

These work similarly to the “Nominal and Current Value”, except that they do not show the Nominal value in the Process details. Since there is no way to define a Nominal value from the operator or the recipe, the “Done” status cannot be used with these parameter types.

Parameter type: Enumeration

This value works like the “Nominal Value”, except that it shows an enumeration selection list in the process details, instead of a numeric value.

This is extremely helpful if you have discrete values that may be valid nominal values, instead of a value range. For example, for the selection of an “Agitator Behavior” where:

0 = Off

1 = On Slow

2 = On Fast

3 = On Intervals

Parameter type: Material type and Material

This value works like the “Nominal Value”, except that it shows a selection of a Material Type or a Material itself in the Process details. These parameters usually work in conjunction with the “Material Modules” and are used to indicate material usage in the recipe to the user.

For example, if you have a Beverage Mixer that has separate “Mixing ingredient” steps. You can use this parameter to let the recipe define what ingredient must be dosed into the beverage mixer by the operator. The selected Material Name is then displayed in the Process details.

Visualization Type

The visualization of a Parameter affects the color that is given to the parameter in the recipe and the Process detail dialog. It has no effect on the PLC programming and does not affect any functionality.

Change Setpoint at Run Time

UnitNo	1	WinOpen:	600.00	43.78	
[1] Watchdog / min:			1.00	0.06	
[11] Temperature / °C:			2.00	11.00	
[14] Quantity Water / hl:			100.00	14.00	
[15] Flow Water / hl/h:			20.00	15.00	
[16] Temperature Water / °C:			52.00	22.00	
[21] Rührwerk:			Aus		

Parameter module Nominal and current values are accessible from the Process detail Window, also called “Lupe” or “Magnifier”. If any nominal value is changed, this will generate an entry in the historical data collection system, so that each parameter change is traceable.

Programming Examples

Basic Timer to end a Phase.

```
//The Parameter module 2 is defined as Parameter Type = Time as seconds
U "PA"
S "Uxx".Para[2].S           //Start the Parameter Module

A "DIn". DIn[12].Sig         //LSL
ON "Act".Act[32].Out         //Pump
ON "PH"
S "Uxx".Para[2].H           //Stop counting if the pump does not run

U "Uxx".Para[2].D           //the timer has reached its nominal value
= "PhaseEnd"                //End of step
```

Basic Nominal and Current Value to end a Phase.

```
//The Parameter module 2 is defined as Parameter Type = Nominal and Current Value

//In the Unit function block:
Network
TITLE =Parameter Transfer

    L "AIn". AIn [1].PVal     //Temperature of Tank
    T "Uxx".Para[10].Val

//In the Unit function:
U "PH"
S "Act".Act[32].Aco          //Heating

U "Uxx".Para[10]. D          //The Temperature has reached its nominal value
= "PhaseEnd"                //End of step
```

Structure

Address	Symbol	Type	Remark
0.0	S	BOOL	start parameter module
0.1	H	BOOL	hold parameter module
0.2	Reset	BOOL	reset parameter module
0.3	OK	BOOL	OK
0.4	s04	BOOL	spare
0.5	s05	BOOL	spare
0.6	s06	BOOL	spare
0.7	s07	BOOL	spare
1.0	No	BYTE	Parameter module No
2.0	OnlySp	BOOL	only setpoint
2.1	OnlyVal	BOOL	only value
2.2	Enum	BOOL	enumeration
2.3	TSec	BOOL	time in sec
2.4	TMin	BOOL	time in minutes
2.5	THour	BOOL	time in hours
2.6	TDay	BOOL	time in days
2.7	s27	BOOL	spare
3.0	Endcond	BOOL	phase end condition
3.1	ManuInput	BOOL	manual input required
3.2	AlarmCond	BOOL	alarm condition
3.3	s33	BOOL	spare
3.4	s34	BOOL	spare
3.5	s35	BOOL	spare
3.6	s36	BOOL	spare
3.7	s37	BOOL	spare
4.0	Sp	REAL	setpoint
8.0	Val	REAL	value

Unit Phases

The Unit function is where all the Phases of a unit are implemented. A Unit phase is a discrete processing step in a recipe. It has a definitive beginning and a definitive “End” where the recipe steps forward and activates the next Phase, defined in the recipe. A Phase also comes with a set of up to 16 configured Parameters, and their Nominal values that represent all nominal values of this Phase.

Phases can be combined in a recipe in any order the operator chooses and have no order of execution. The order of execution of the phase depends solely on the recipe and the order in which the passes and steps are defined in it.

Examples for Phases are:

- Heating Up
- Transfer
- Filling
- Mixing
- Homogenization
- Boiling

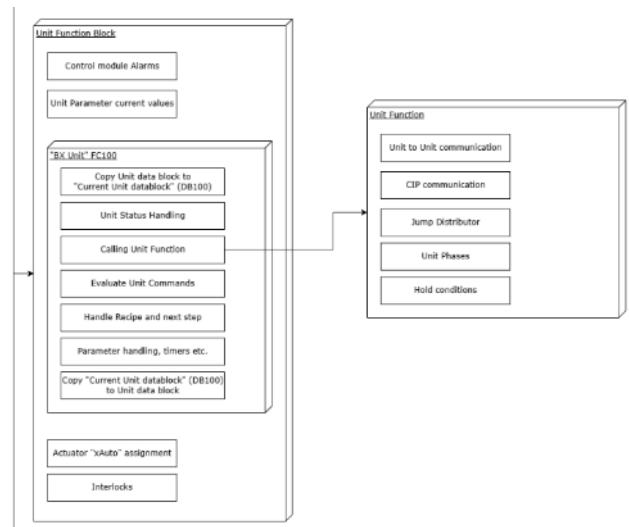
All actions that a Phase must perform, every Actuator must activate, every timer it must start, are programmed in the Phase in this Unit function. Also, the “Phase End condition” is defined here for each of the phases.

However, keep in mind that a phase only “Knows” that it has finished, not what the next step will be, or what the previous steps were. This information is only available in the recipe and should not be of any consideration in the implementation of a phase.

Calling the Unit Function

In your Unit Function block, there is no explicit call to your Unit function, because this function is being called from the FC100 that you call from your Unit Function block. This FC100 call will provide a lot of facilities, status signals and evaluate all the commands that you can send to the Unit from your Unit function, and it will manage recipe loading, check which step should be executed, parameter handling and so on.

But you must keep in mind that your Unit function will be executed by FC100 whenever and more importantly, as often as this function sees fit. In consequence, this Unit function may be called multiple times during a single plc cycle, with different parameters. So, you cannot rely on this Unit function, to be called only once every plc cycle.



When calling “Bx Unit” (FC100), this will perform the following actions:

- Copy the Unit’s Data block to the “Current Unit Data block” (DB100). More on that below.
- Create the unit status signals. See below
- Reset the unit commands.
- Check what Phase must be executed now.
- Process the current steps Parameters, count timers etc.
- Call the Unit Function
- Evaluate unit commands and perform the requested action.
- Check what phase should be executed next.
- If there was a step change, load data for the next step and preset the nominal values and configurations for the parameter modules.
- Send historical data if needed.

The “Current Unit Data block” (DB100)

To make Unit Functions more portable between units and projects, the “Bx Unit” (FC100) copies all statuses of a unit onto the “Current unit data block” (DB100) which must be used when referring to data of the current unit inside the Unit function. After the Unit function was executed, this data is copied back to the unit’s data block.

This means that you must always use DB100 to refer to any data from the inside the unit function or use the provided unit commands.

For example, if you want to start a parameter module, you cannot do it from inside the Unit function:

```
U "PH"  
S "Unit 10 Data".Para[10].S
```

But must:

```
U "PH"  
S "Bx Unit".Para[10].S
```

The “BX Unit” (DB100) will always refer to the currently executing unit of your Unit function. You cannot use the Unit data block itself, as it will get overwritten by the current unit’s data block at the end of your Unit functions execution.

All this makes porting units to other units much easier, as you do not need to make any code changes most of the time. Especially if you are implementing “Class” programming, for example in fermenting cellars, this makes these kinds of applications straightforward and even trivial.

Status Signals

For convenience, there are many signals provided for the Programmer to be used in the Unit Function, that always represent the status of the current unit, where this Unit function belongs to. All these signals can be used through the execution of the Unit function but are ONLY valid inside the Unit function.

If you must refer to these signals outside of the Unit function, or from a different unit, you must access the requested unit's data block directly.

Symbol	Address		Remark
StatusSteril	M	1.0	unit status sterile
StatusClean	M	1.1	unit status clean
StatusNotClean	M	1.2	unit status not clean
StatusProd1	M	1.3	unit status product 1
StatusProd2	M	1.4	unit status product 1
StatusProd3	M	1.5	unit status product 1
StatusProd4	M	1.6	unit status product 1
ReqCIP	M	1.7	unit status request CIP
Ign	M	2.1	ignore alarm
Sim	M	2.2	simulation
Run	M	2.3	unit in run mode
Pause	M	2.4	unit in pause mode
Hold	M	2.5	unit in hold mode
EmHold	M	2.6	spare - unit in emergency hold
Maint	M	2.7	maintenance
GAI	M	3.0	general alarm
GAIS	M	3.1	general alarm save
SCE	M	3.2	status check error
Watchdog	M	3.3	watchdog alarm
Step0	M	3.4	unit in step 0
ReadyStart	M	3.5	unit ready for start
UnitActive	M	3.6	unit active (not step0)
CIPModus	M	3.7	unit in CIP modus for Visu
ShowAlarm	M	4.1	unit status alarm without hold
CIP	M	4.4	unit in CIP modus
PhaseEnd	M	4.7	phase end
PA	M	5.0	phase active
PEH	M	5.1	phase active with emergency hold function
PH	M	5.2	phase active with hold function
PP	M	5.3	phase active with pause function
Start	M	5.4	unit start button
OperatorOK	M	5.5	unit operator OK button
PFCycle	M	5.6	phase first cycle
PLCycle	M	5.7	phase last cycle
StatusInfo	MB	1	unit status info
UnitNo	MB	10	unit number
Phase	MB	11	phase number
StepNo	MB	12	step number
Charge	MD	28	charge number
PrId	MD	32	PrId
ProgNo	MD	36	program number

Commands

For convenience, there are some Commands provided for the Programmer, to be used in the Unit Function, which will always trigger a specific action on the current unit, where this Unit function belongs to. All these commands can be used through the execution of the Unit function but are ONLY valid inside the Unit function.

A command is executed by setting it to TRUE for a single cycle. All commands are automatically reset, so they can never stay TRUE indefinitely.

Symbol	Address		Remark
GAIQuitt	M	2.0	general alarm acknowledge
HoldReq	M	4.0	hold request Stops the current unit and puts it into "Hold"
OpReq	M	4.2	Operator Request Marks the Units as "Operator Request", shows an appropriate indication on the HMI screen and the operation can confirm this action by clicking the confirmation which will trigger the "OperatorOK" status
ProtWrite	M	4.3	phase protocol write If activated, will record the current steps parameters to the Historical protocol, without affecting any parameter module, or terminating the current step
PhaseEnd	M	4.7	phase end Maks the current Recipe Phase as finished. The Unit will look up the next step to activate and proceed to initiate a Step Change
StepNoNew	MB	9	step number new (Jump) By default, this will always point to the Next step. If you want to "jump" to a different step, you must write the step you want to jump to, onto this value, while the "PhaseEnd" is TRUE

Step Change

When a step of a recipe is completed on a unit (PhaseEnd = TRUE), or the operator aborts the current step, the following actions are performed in the same plc cycle without interruption.

- The current step is executed and detects the final condition = 1
- The current step is processed again with identifier "PLCycle" (last cycle)
- Creating an Historical record in the database of an "Automatic Step Completion", or Manual "Step+" or "Step-"
- Check which step must be executed next. Usually, it is the next step in the recipe, but may be different due to "StepNoNew", or the operator uses the "Step+" or "Step-" functionality.
- Recipe values for the next Step are loaded
- Status indicators and parameter modules are updated.
- Parameter that are configured as timers, are reset
- New step is processed with identifier "PFCycle" (first cycle)

Jump Distributor

The Jump distributor is used in the default phase selection implementation.

```
L      "Phase"
SPL    X000 //For invalid phase number we jump to "X000"
SPA    X000 //Phase 0, is invalid
SPA    X001 //if Phase = 1
SPA    X002 //if Phase = 2
SPA    X003 //if Phase = 3
SPA    X000 //if Phase = 4, but it does not exist, we use "X000" as padding
SPA    X000 //if Phase = 5, but it does not exist, we use "X000" as padding
SPA    X006 //if Phase = 6
SPA    X007 //if Phase = 7
SPA    X008 //if Phase = 8
X000: SET
SPA    END
```

The Jump distributor load the currently requested Phase number and then jumps to the Jump mark indicated by the Phase number, with the first jump mark being "0".

WARNING! The SPL instruction will NOT! Jump to the Jump mark with the correct number but will count from top to bottom starting at 0 and jump to the resulting jump. Make sure you put "SPA X000" instructions in there as padding

Phases every unit should implement.

Every unit should implement a set of "default" phases that make the execution of recipes much easier. These phases are implemented in a way so that they can be copied without changes between Units.

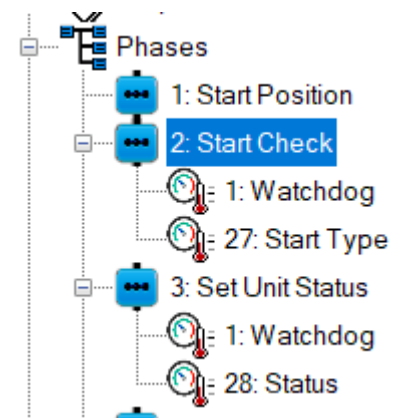
Start Position. This is a system phase that always must exist and cannot have any parameters. This is the step that a unit Executes if no recipe is started.

```
NETWORK
TITLE =phase 01:  start position

X001: SET    ;

//this is a typical start request from a Master unit
U      "Bx UnitCom D".Master1.Start;
U      "Bx UnitCom D".Master1.Run;
S      "PhaseEnd";

SPA    END;
```



Start check. This step executes a check of all control modules that belong to this unit and only advances if all modules are automatic and without error.

```
NETWORK
TITLE =phase 02:  start check

X002: SET    ;
// material modules
U      "PFCycle";
S      "Bx MM D".Mat20.Reset;

//Possible automatic Draining
U      "Bx Act D".Act109.Off;
U      "Bx Act D".Act112.Off;
U      "Bx Act D".Act114.Off;
U      #AnalogLSL;
```

```

UN    #LSL;
U     "PH";
S     "Bx Act D".Act107.ACo;
S     "Bx Act D".Act108.ACo;

//Start Check by sending a command to the Alarm group of this unit
U     "PA";
S     "Bx DIn D".DIn75.SCS1;
S     "Bx Alarm Group D".Unit[4].SCS;

//If this Alarm group does not have a Start Check error, we go forward
UN    "Bx Alarm Group D".Unit[4].SCE;
UN    "PFCycle";
=     "PhaseEnd";

SPA   END;

```

Unit Status. This allows you to set the Unit's Hygienic status when you finish a CIP. The "Status" parameter allows you to define the step you want the unit's status set to.

```

NETWORK
TITLE =phase 03:  set unit status

X003: SET    ;
L     "Uxx".Para[28].Sp;
L     0.000000e+000;
<=R   ;
SPB    sest;                                //if a Status is selected, set it to the Unit
L     "Uxx".Para[28].Sp;
RND    ;
T     "StatusInfo";
UN     "PFCycle";
S     "PhaseEnd";
sest: SET    ;

SPA    END;

```

Stop Unit if a Control module is on error or Watchdog time has elapsed.

Depending on the needs of the process you are implementing, you may want to add an "HoldRequest" when any alarm of the unit or its control modules gets activated. Of course, this depends on the situation, and you may want to adjust this condition to fit your needs. This should be implemented after all phases, at the bottom of the Unit Function.

```

NETWORK
TITLE =End of Phases

END:   SET    ;

NETWORK
TITLE =unit hold

U     "Watchdog";                                //if we have an Elapsed Watchdog alarm.
O     "Bx Alarm Group D".Unit[4].GAl; //if any Control module has an error.
UN    "Step0";                                //Ignore if we are in Start position
S     "HoldReq";                                //Stop the unit and put it into Hold

```


Querying the Unit's current Recipe

From the "Current Unit Data block" (DB100) you only have access to the current steps parameter declarations and nominal values. If you want to query future recipe steps, you can use one of the provided system functions, that allow you to query the recipe steps that are currently scheduled for execution in the recipe.

Please refer to the ["Common Help functions"](#) for more information.

Examples

Heating up to set Temperature.

```
NETWORK
TITLE =phase 07:  heat up

X007: SET    ;
      CALL "Tec Heat Control 2 Point" (
        CurrTemperature      := "Uxx".Para[11].Val,
        SPTemperature        := "Uxx".Para[11].Sp,
        PreTemperature       := "Uxx".Para[12].Sp,
        SPPulseTime          := "Uxx".Property[3],
        SPPauseTime          := "Uxx".Property[4],
        HeatRequest          := #HeatReq,
        TimerMemory          := #User.Heating_HV);

// actuators
U    "PH";
S    #Agitator;

// start Lauter Tun
U    #User.SugarTested;
S    "Bx UnitCom D".U.Slavel.Start;

// phase end contition
L    "Uxx".Para[11].Val;
L    2.500000e-001;
+R    ;                                //If the Current value is close to the Nominal
L    "Uxx".Para[11].Sp;
>R    ;
O    "Uxx".Para[11].D;                //or the temperature is already exceeded
=    "PhaseEnd";

SPA   END;
```

Wait until Timer is done.

```
NETWORK
TITLE =phase 08:  rest

X008: SET    ;

      CALL "Tec Heat Control 2 Point" (
        CurrTemperature      := "Uxx".Para[11].Val,
        SPTemperature        := "Uxx".Para[11].Sp,
        PreTemperature       := "Uxx".Para[12].Sp,
        SPPulseTime          := "Uxx".Property[3],
        SPPauseTime          := "Uxx".Property[4],
        HeatRequest          := #HeatReq,
        TimerMemory          := #User.Heating_HV);

// actuators
U    "PH";
S    #Agitator;

// phase end condition
U    "PA";
S    "Uxx".Para[2].S;

U    "Uxx".Para[2].D;
=    "PhaseEnd";

SPA   END;
```

Transfer until empty.

```
//Special Function in Primary Processing
U "PFCycle"
S "Cnt". Cnt[2]. Reset           Counter Reset

//Actuator Control
U "PH"
S "Act". Act[12]. Aco           Activates the actuator in automatic mode
S "Act". Act[13]. Aco
S "Act". Act[14]. Aco
S "Act". Act[18]. Aco

/End the step
U "DIn". DIn[17]. Sig           //LSL
= "PhaseEnd"                   //End of step
```

Operator request and Step on when operator Confirms.

```
//Request operator by putting the Unit into "Request Operator"
U "PH"
S "OpReq"

U "OperatorOK" //if the operator clicks on the unit, he confirms the operator request
S "PhaseEnd"

//Hold unit if we are waiting too long, or an error happened
U "Act". Act[12]. Gals
O "Cnt". Cnt[2]. Gals
O "Watchdog"
S "HoldReq"
```

Dosing an Additive with a Pump

```
NETWORK
TITLE =phase 10: Dosing Additive

X010: SET ;
//Heating
CALL "Tec Heat Control 2 Point" (
    CurrTemperature      := "Uxx".Para[11].Val,
    SPTemperature        := "Uxx".Para[11].Sp,
    PreTemperature       := "Uxx".Para[12].Sp,
    SPPulseTime          := "Uxx".Property[3],
    SPPauseTime          := "Uxx".Property[4],
    HeatRequest          := #HeatReq,
    TimerMemory          := #User.Heating_HV);

// actuators
U "PH";
S #Agitator;
S "Bx Act D".Act104.ACo;

U "Uxx".Para[2].D;           //After the Initial delay timer has elapsed.
U "PH";
S "Bx Act D".Act105.ACo;     //Request Dosing Pump

// phase end condition
U "PA";
S "Uxx".Para[2].S;           //Start Dosing initial delay timer.

U "Uxx".Para[2].D;           //After the initial delay
U "PA";
S "Uxx".Para[3].S;           //We start the Dosing timer and leave it started.

UN "Bx Act D".Act104.On;     //if the Agitator does not run, for some reason.
O "Uxx".Para[2].D;
U "PA";
= "Uxx".Para[2].H;           //We put the Initial Delay timer to "Hold", so it pauses

UN "Bx Act D".Act105.On;     //if the dosing pump does not run, for some reason.
U "PA";
= "Uxx".Para[3].H;           //We put the dosing timer to "Hold", so it pauses

U "Uxx".Para[3].D;           //if the dosing time es reached
```

```

U      "Uxx".Para[3].S;
=      "PhaseEnd";           //We finish the step

SPA    END;

```

Prepare dosing and Handle Material modules.

```

NETWORK
TITLE =phase 11:  Prepare Dosings

X011: SET  ;
//user Message
U      "PA";
S      "OpReq";

//Heating
L      0.000000e+000;
T      "Uxx".Para[3].Sp;

CALL "Tec Heat Control 2 Point" (
    CurrTemperature      := "Uxx".Para[11].Val,
    SPTemperature        := "Uxx".Para[11].Sp,
    PreTemperature       := "Uxx".Para[12].Sp,
    SPPulseTime          := "Uxx".Property[3],
    SPPauseTime          := "Uxx".Property[4],
    HeatRequest           := #HeatReq,
    TimerMemory          := #User.Heating_HV);

// activate material moduls
UN     "PFCycle";
SPB    fc11;

//if the Dosing is requested in from the recipe, then we request the operator to confirm the Material
//Module
L      "Uxx".Para[31].Sp;
T      "Bx MM D".Mat21.Val; //Mash Tun - Additive 1
L      0.000000e+000;
>R     ;
S      "Bx MM D".Mat21.OPReq; //Mash Tun - Additive 1

L      "Uxx".Para[32].Sp;
T      "Bx MM D".Mat22.Val; //Mash Tun - Additive 2
L      0.000000e+000;
>R     ;
S      "Bx MM D".Mat22.OPReq; //Mash Tun - Additive 2

L      "Uxx".Para[33].Sp;
T      "Bx MM D".Mat23.Val; //Mash Tun - Additive 3
L      0.000000e+000;
>R     ;
S      "Bx MM D".Mat23.OPReq; //Mash Tun - Additive 3
fc11: SET  ;

```

Unit Status Signals: “Phase Active”

As mentioned before, the “BatchXpert” framework provides many status signals that you can and should use when implementing your unit phases inside your unit function code. All the following Signals are only valid inside a Unit FC and apply to all units always to the one where the Unit FC belongs to.

These signals are meant as “tools” that you can use to make implementing Unit Phases as easy as possible.

The most important Status signals are:

Symbol	Address	Remark
PhaseEnd	M 4.7	phase end
PA	M 5.0	phase active
PEH	M 5.1	phase active with emergency hold function
PH	M 5.2	phase active with hold function
PP	M 5.3	phase active with pause function
PFCycle	M 5.6	phase first cycle
PLCycle	M 5.7	phase last cycle

Phase Active

In “BatchXpert” the “PA” signal is basically always true and is never false during the execution of your Unit phases in your function code. Different systems may implement a function where this phase active signal goes to false when “PhaseEnd” gets signaled. **“BatchXpert” Does not put phase active to false when the phase end signal gets triggered.**

This sometimes has implications if you are setting memory variables inside your face logics, which would then be “Frozen” if your step is not executing anymore.

Phase Last Cycle

For this the framework provides the “PLCycle” and “PFCycle”. When phase end gets triggered, your faces will get executed one last time with the “PLCycle” signal to true to signal that this is the last phase cycle of your steps in this step. If you want signals to get reset when you exit faces you should use the “PLCycle” signal.

Phase First Cycle

The first cycle signal is basically the opposite and gets through the first time a new step or phase is run. You usually want to use this signal if you have some initializations to do on your Phase, such as resetting counters, resetting signals or doing some initial calculations.

Phase Hold and Phase Pause

The Phase hold and Phase pause signals are basically the same as the phase active signal but are only true if your unit is not in hold or not in pause respectively.

This signal is usually used to turn on actuators which usually should turn off when you put your unit into hold. If you put your unit into hold the face hold signal will get “False”, and thus deactivating all automatic controls of all your actuators.

Prefer “BatchXpert Unit-to-Unit” communication

To avoid this problem entirely we recommend you use our [Unit-to-Unit Communication](#), which is integrated into the BatchXpert framework. This unit-to-unit communication framework implements proper “Command

semantics” and resets all their commands cyclically, avoiding this problem altogether. BatchXpert is designed to use this communication mechanism to send process signals between units and is thus the preferred method for communication between multiple units.

Examples

```
NETWORK
TITLE =phase 07:  heat up

X007: SET    ;

// actuators
// These actuators will turn "Off" when your Unit will be set to "HOLD"
U    "PH";
S    "Bx Act D".Act[123].Aco;
S    "Bx Act D".Act[124].Aco;
S    "Bx Act D".Act[125].Aco;

// These actuators will be "ON" regardless of Run/Pause/Hold of your unit
U    "PA";
S    "Bx Act D".Act[133].Aco;
S    "Bx Act D".Act[134].Aco;
S    "Bx Act D".Act[135].Aco;

// These actuators will be "ON" only if your unit is in "Run"
U    "PP";
S    "Bx Act D".Act[143].Aco;
S    "Bx Act D".Act[144].Aco;
S    "Bx Act D".Act[145].Aco;

// start Lauter Tun
U    #User.SugarTested;
S    "Bx UnitCom D".U.Slavel.Start;

// phase end contition
L    "Uxx".Para[11].Val;
L    2.500000e-001;
+R    ;                                //If the Current value is close to the Nominal
L    "Uxx".Para[11].Sp;
>R    ;
O    "Uxx".Para[11].D;                //or the temperature is already exceeded
=    "PhaseEnd";

SPA    END;
```

What not to do

As stated above you should not set memory signals that are not explicitly marked as commands inside your Phase logics, since they get frozen in the last state the PHASE was active

```
NETWORK
TITLE =phase 08:  rest

X008: SET    ;

//The "Uxx D".User.b1 will be TRUE even after the phase is not active anymore, since PA is never FALSE
U    "PA";
S    "Uxx D".User.b1;

// actuators
U    "PH";
S    #Agitator;

// phase end condition
U    "PA";
S    "Uxx".Para[2].S;

U    "Uxx".Para[2].D;
=    "PhaseEnd";
SPA  END;
```

Do this instead

```
NETWORK
TITLE =phase 08:  rest

X008: SET    ;

//The "Uxx D".User.b1 will be FALSE the phase is not active anymore, since PA is never FALSE, but Phase
//Last Cycle will be true before leaving the Phase and thus resetting the signal
U    "PA";
UN   "PLCycle"
S    "Uxx D".User.b1;

// actuators
U    "PH";
S    #Agitator;

// phase end condition
U    "PA";
S    "Uxx".Para[2].S;

U    "Uxx".Para[2].D;
=    "PhaseEnd";
SPA  END;
```

Implement "Botec" behavior

In Contrast to the "BatchXpert" system, the "Botec" system from Krones put the "Phase Active" (SOPA) signals to false during the last cycle. To implement this same functionality in "BatchXpert", you can put the following Code on top of your "Function Code" of your Unit Phases.

Put this in your First network of your Unit FC.

```
//This will set the "Phase Active" and "Phase Active hold" functions to False during the last Cycle of
//your Phase, same as Botec Systems
U    "PA";
U    "PLCycle"
R    "PA";

U    "PH";
U    "PLCycle"
R    "PH";

U    "PP";
U    "PLCycle"
R    "PP";
```

Subphases (Phases inside Unit Phases)

BatchXpert in and of itself does not support Subphases, which are Phases that are running inside of Phases. These are most used in phases such as “rinsing”, where you could have sub-phases for each of the different rinsing intervals, such as spray bowl, lateral inlet, lower inlet etcetera. Since BatchXpert does not support subphases, there's no configure option in your batch configuration, and you must implement this sub-phase logic yourself in the PLC.

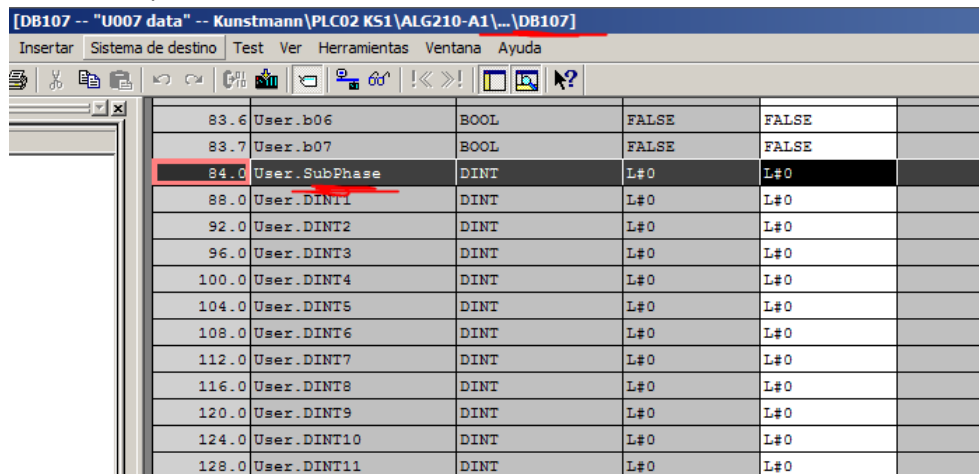
To implement these, you basically must program a jump distributor inside of your existing processing phase. However, keep in mind that these sub-phases will not be managed by BatchXpert and do not generate any patch events and do not have dedicated statuses. You can, however, trigger a user batch event when you want an event to be registered in the PLC. You can do this for example every time you change one of your sub-phases, you can register and batch event manually.

To memorize the current executing sub phase of your unit phase, you will need one of your “User.Dint”, which are found inside of your unit data block.

Sub-Phase memory

In order to know which current Sub-Phase is active in a Unit-Phase, we have to reserve a memory in our Unit Data-block, which holds the current “sub-Phase”. You usually put this memory in the “UnitDb.User” section, as shown below.

In the Example below we use the first “Dint”, “Dint0” in the “User” section of the unit data-block of Unit 7, DB107.



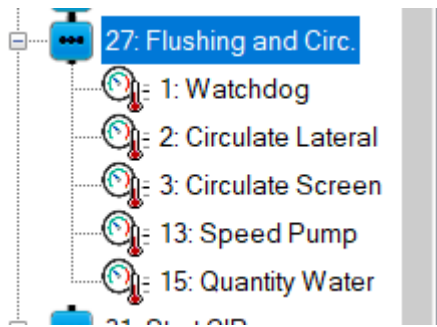
[DB107 -- "U007 data" -- Kunstmann\PLC02 KS1\ALG210-A1\...\DB107]					
Insertar Sistema de destino Test Ver Herramientas Ventana Ayuda					
83.6	User.b06	BOOL	FALSE	FALSE	
83.7	User.b07	BOOL	FALSE	FALSE	
84.0	User.SubPhase	DINT	L#0	L#0	
88.0	User.DINT1	DINT	L#0	L#0	
92.0	User.DINT2	DINT	L#0	L#0	
96.0	User.DINT3	DINT	L#0	L#0	
100.0	User.DINT4	DINT	L#0	L#0	
104.0	User.DINT5	DINT	L#0	L#0	
108.0	User.DINT6	DINT	L#0	L#0	
112.0	User.DINT7	DINT	L#0	L#0	
116.0	User.DINT8	DINT	L#0	L#0	
120.0	User.DINT9	DINT	L#0	L#0	
124.0	User.DINT10	DINT	L#0	L#0	
128.0	User.DINT11	DINT	L#0	L#0	

Sub-Phase Parameters

Since BatchXpert does not natively support “Sub-Phases”, you have to add all subphase parameters to the Unit phase itself, and do not reuse sub phase parameters, but instead use one parameter for each sub phase dependent parameter.

Simple Sub-phase implementation that does not repeat.

In this example we implement a phase that internally runs through different sub-phases, and on the last sub-phase it terminates the whole phase. Additional comments for description and explanation are maced with blue.



```
NETWORK
TITLE =phase 27:  Rinsing and Circulate

//Rinsing Amount
//Here we do some standard phase logic
U    "PA"
=    "Uxx".Para[15].S

UN   "Bx Act D".Act136.On
=    "Uxx".Para[15].H

// phase end condition
//Since we want to end the Unit Phase when the last Sub-Phase ends,
//we do NOT signal the "PhaseEnd" here, but rather in the last sub-Phase
CLR
=    "PhaseEnd"

//Here we put our sub-phase logic
//
//Subphase Handling

//here we preset our "SubPhase" memory to point to the
//first subphase when whe enter the Phase.
//this ensures that the sub-phases always start at the
//first sub-phase
U    "PFCycle"
SPBN x027
L    1                                //Subphase 1
T    #User.SubPhase
x027: SET

//here we build an "Jump distributor" similar to an
//Unit phase. Sepending on our "#User.SubPhase"
//we jump to the section of code where our logic lives
L    #User.SubPhase
SPL  e027
SPA  e027
SPA  a027
SPA  b027
SPA  c027
e027: L    1                                //Subphase 1
T    #User.SubPhase
SPA  END

//Subphase 1 Filling with water
a027: SET
U    "PH"
S    "Bx Act D".Act136.ACo

//instead of ending, we write the next subphase number to our memory
//this will activate the next phase, the next plc cycle
U    "Uxx".Para[15].D
SPBN END
L    2
T    #User.SubPhase
SPA  END
```

```

//Subphase 2 Circulate via lateral inlet
b027: SET
  U      "PA"
  =      "Uxx".Para[2].S

  UN     "Bx Act D".Act62.On
  =      "Uxx".Para[2].H

  U      "PH"
  S      "Bx Act D".Act139.ACo
  S      "Bx Act D".Act141.ACo
  S      "Bx Act D".Act62.ACo
  S      "Bx Act D".Act132.ACo

//instead of ending, we write the next subphase number to our memory
//this will activate the next phase, the next plc cycle
  U      "Uxx".Para[2].D
  SPBN   END
  L      3
  T      #User.SubPhase
  SPA    END

//Subphase 3 Circulate via screen
c027: SET
  U      "PA"
  =      "Uxx".Para[3].S

  UN     "Bx Act D".Act62.On
  =      "Uxx".Para[3].H

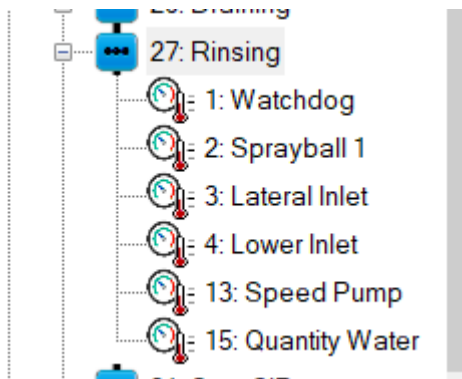
  U      "PH"
  S      "Bx Act D".Act132.ACo
  S      "Bx Act D".Act141.ACo
  S      "Bx Act D".Act62.ACo
  S      "Bx Act D".Act173.ACo

//Since we want to end the Unit Phase when the last Sub-Phase ends,
//we signal the "PhaseEnd" here
  U      "Uxx".Para[3].D
  UN     "PFCycle"           //Jaime
  =      "PhaseEnd"
  SPA    END

```

Sub-phase implementation that does repeat its subphases until the Unit Phase ends

In this example we implement a phase that internally runs through different sub-phases and starts again on the first one, when the last one has ended. It does this until the Unit Phase itself has ended. Additional comments for description and explanation are maced with blue.



```

NETWORK
TITLE =phase 27:  Rinsing

//Rinsing Amount
//Here we do some standard phase logic, in this case our Rinsing Counter
U    "PA"
=    "Uxx".Para[15].S

UN   "Bx Act D".Act136.On
=    "Uxx".Para[15].H

// phase end condition
//If we decide to end the phase, we signal the Phase by "PhaseEnd", just as
//on any other Unit Phase
U    "Uxx".Para[15].D //rinsing Ammount reached
=    "PhaseEnd"

//Here we put our sub-phase logic
//
//Subphase Handling

//here we preset our "SubPhase" memory to point to the
//first subphase when whe enter the Phase.
//this ensures that the sub-phases always start at the
//first sub-phase
U    "PFCycle"
SPBN x027
L    1 //Subphase 1
T    #User.SubPhase
x027: SET

//here we build an "Jump distributor" similar to an
//Unit phase. Depending on our "#User.SubPhase"
//we jump to the section of code where our logic lives
L    #User.SubPhase
SPL  e027
SPA  e027
SPA  a027
SPA  b027
SPA  c027
e027: L    1 //Subphase 1
T    #User.SubPhase
SPA  END

//Subphase 1 Sprayball 1
a027: SET
U    "PA"
=    "Uxx".Para[2].S

U    "PH"
S    "Bx Act D".Act136.ACo

//instead of ending, we write the next subphase number to our memory

```

```
//this will activate the next phase, the next plc cycle
```

```
U      "Uxx".Para[2].D
SPBN   END
L      2
T      #User.SubPhase
SPA    END
```

```
//Subphase 2 Lateral Inlet
```

```
b027: SET
```

```
U      "PA"
=      "Uxx".Para[3].S
```

```
UN     "Bx Act D".Act62.On
=      "Uxx".Para[3].H
```

```
U      "PH"
S      "Bx Act D".Act139.ACo
S      "Bx Act D".Act141.ACo
S      "Bx Act D".Act62.ACo
S      "Bx Act D".Act132.ACo
```

```
//instead of ending, we write the next subphase number to our memory
```

```
//this will activate the next phase, the next plc cycle
```

```
U      "Uxx".Para[3].D
SPBN   END
L      3
T      #User.SubPhase
SPA    END
```

```
//Subphase 3 Lower Inlet
```

```
c027: SET
```

```
U      "PA"
=      "Uxx".Para[4].S
```

```
UN     "Bx Act D".Act62.On
=      "Uxx".Para[4].H
```

```
U      "PH"
S      "Bx Act D".Act132.ACo
S      "Bx Act D".Act141.ACo
S      "Bx Act D".Act62.ACo
S      "Bx Act D".Act173.ACo
```

```
//Since we want the sub-phases to loop, we go back to the first sub-Phase, instead of the next one
```

```
U      "Uxx".Para[4].D
SPBN   END
L      1 //Back to first sub phase
T      #User.SubPhase
SPA    END

SPA    END
```

Recommendations when implementing a Phase

There are some recommendations you should consider and follow when implementing a Phase in a Unit. These considerations are not unique to BatchXpert but are worth mentioning at this point, so you can make more reliable applications.

Phases should be independent from each other.

You should never have implicit dependencies between two Phases of a Unit. For example, consider that we have an “Filling” phase, where we are setting an “Product in Tank” bit somewhere in the plc. When we reach the “Emptying to Drain” step, we reset the “Product in Tank”. And let us say that this “Product in Tank” bit will block you from filling it up again.

Now consider that the operator can rearrange and even remove steps and phases from the recipe as he chooses. Now the operator does not want to “Empty to Drain” anymore to optimize product losses and removes the phase from the recipe. This now means, that there is no possibility anymore for the “Product in Tank” to get reset.

Do not depend on a Specific Phase “being there”

Remember, the operator can remove any step he wishes. For example, consider that you have two Units that transfer from one to the other.

Unit 1 enters in “Transfer” which causes Unit 2 to enter “Filling”. Unit 2 will end “Filling” when Unit 1 sends the “TransferDone” signal. Unit 1 sends this “TransferDone” signal in the “Rinsing to Drain” step at the end of the recipe.

What happens if the operator decides to remove the “Rinsing to Drain” step? In that case, Unit 1 will never send “TransferDone” to Unit 2, which will never leave “Filling”.

Rather than depend on the “Rinsing to Drain” phase to set the “TransferDone”, you should send an “TransferActive” signal in all phases where a Transfer is in progress. If this “TransferActive” becomes FALSE, Unit 2 can finish “Filling”. This way you are not dependent on any step.

Do not confuse this solution with the “Transfer” and “Filling” steps, which are a little exception to this rule.

Do not make Phases dependent on previous Phases.

Since phases may be ordered in any order the operator wishes, or not even be in the recipe at all, you should not assume that a specific Phase has been executed when a Phase executes.

Make Discrete phases, not Super “do it all” phases.

Usually, it is better to model your process in smaller discrete steps, instead of having an “Active” step, that just does everything. Of course there are exceptions to this, but you should strive for Phases that do one thing, and one thing only.

Consider Error Conditions.

When implementing Phase logic, you should always consider the possibility of some error happening. Your Phases should always react in a predictable and save manner to errors generated by control modules.

A phase should always expect critical control modules alarms, and should thus always move to a save configuration if an error happens.

Most of the time it is sufficient to “Hold” a phase if a Control module alarm activates, since this usually deactivates all Automatic controls of your Actuators.

However, sometimes you will have to implement special logic to take these failure conditions into account.

Confirm Dosing whenever possible.

Sometimes you need to measure ingredients into your process. You should always try to measure and confirm the dosage, and NEVER assume that a product enters your Batch, only because you activated an Actuator. After all, an Actuator may fail and not open properly.

If you do not have totalizer, flow sensor or similar, you must use “Time” to calculate the amount of dosage. In that case you should only be able to count when all valves and pumps are confirmed running.

```
//Activate the time permanently, so it does not get reset
U "PA"
S "UxxD".Para[10].S

//But only allow counting if the Valve is confirmed open
UN "Act". Act[13]. ON
S "UxxD".Para[10].P
```

Always Initialize and De-initialize your Phases

If you are implementing phases that use “Help” bits, especially in the “UxxD”.User section, you must be aware what value these variables may have when entering the step. To ensure a consistent Execution of your phase, you should always initialize your signals and values when entering the phase (PFCycle) and when exiting the Phase (PLCycle).

```
//Preset Help signal when starting phase and also when exiting
U "PFCycle"
O "PLCycle"
R "UxxD".User.HelpBitLevelReached

//Tank is full
U "Bx DInD".Din[124].Sig //Full signal
R "UxxD".User.HelpBitLevelReached

//Tank is empty
U "Bx DInD".Din[123].Sig //Empty signal
S "UxxD".User.HelpBitLevelReached

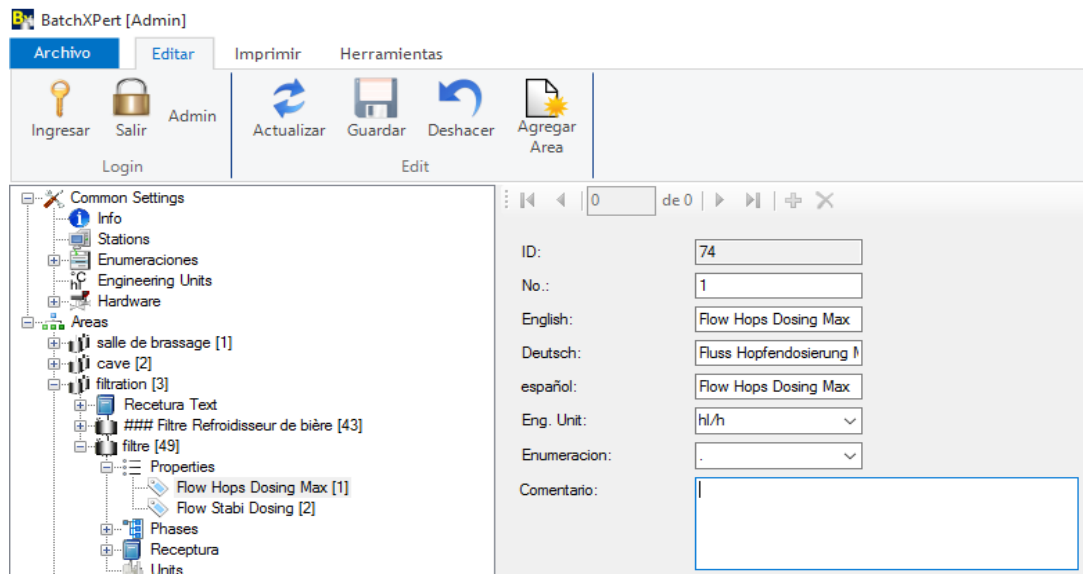
//Refilling of Tank
U "UxxD".User.HelpBitLevelReached
S "Act". Act[13]. Aco
```

Unit Properties

Each unit can have properties, which are values that are “static”, and do not change depending on the executing recipe. These “Properties” are meant for configuration of the equipment, for example “Lauter tun diameter”, “Overflow Pipe Height”, “Pushout volume” and such. Parameters that are equipment dependent, but not recipe dependent.

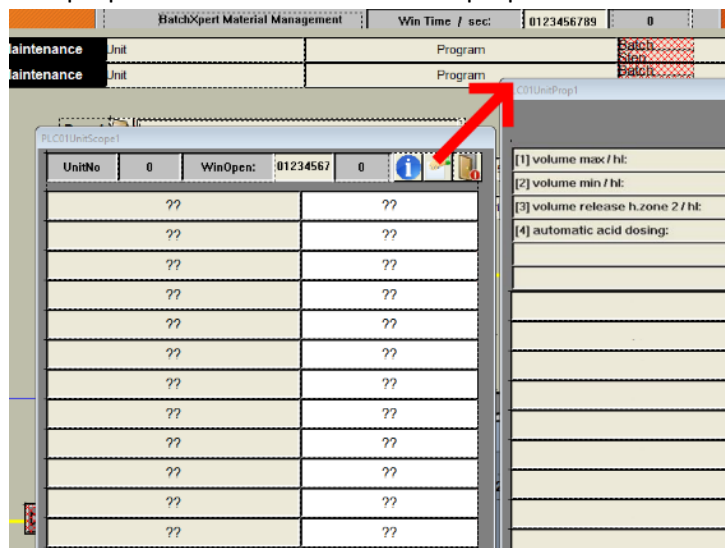
Configuration

The configuration of unit properties is done at the class level in the configuration editor, similarly to the parameters of a Phase, but since they do not belong to a phase, the configuration is done one hierarchical level above.



Faceplate

Unit properties are available from the properties window that is available from the “Process detail windows.



Programming Examples

Calculate push out amount from properties.

```
L "Uxx".Para[11].Val //Pushout volume offset from Recipe
L "Uxx".Property[2]  //Pushout volume from Properties as fixed value
+R
T "UnitCom". U.Val1  //Total pushout needed
```


Unit Status

Each unit in BatchXpert automatically incorporates Hygienic state management. It can have the following statuses:

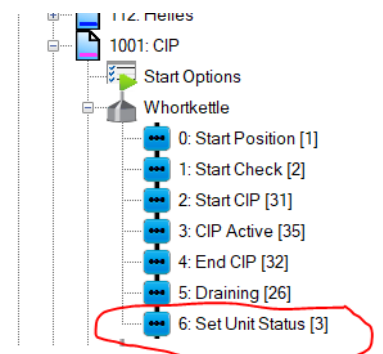
- Without CIP
- CIP elapsed.
- With CIP

Each of the hygienic status will automatically downgrade after set amounts of time. Additionally, there is also the status “Request CIP” where you can input a set amount of production that can run through a unit until it “Requests CIP”, which means that it must realize CIP before continuing.

Implementation in a Unit

The Unit itself detects when it is running production or a CIP recipe. It does this by monitoring the for changes in the Hygienic state. We recommend that you always put the “Unit Status” step into every unit, so you have a way to set the Hygienic state from a recipe when an CIP has finished.

On every CIP recipe you should put the following steps at the end



Faceplate

UnitNo	1234	WinOpen:	01234567	1234
Process Status				
Unit Active				
CIP active				
Startposition				
Alarm Status				
alarm active				
Watchdog Alarm				
Emergency Hold				
Maintenance				
Message Status				
Startposition Error				
Operator Request				
Hygienic Status				
Steril				
Clean				
not Clean				
Request CIP				
Product Status				
Product 1				
Product 2				
Product 3				
Product 4				
Time/Cycles without CIP				
Cycles without CIP maximum:		0123456789 0		
Cycles without CIP actual:		1234		
Time maximum for steril:		0123456789 0		
Time maximum for clean:		0123456789 0		
Time without CIP:		1234		
Active Time:		1234		

Starting a unit with an existing or new Batch

For any units to execute and recipe, you must start corresponding unit with an existing Batch or Start a new Batch. New Batches are usually started by the operator via the Graphical user interface.

Prid

Each batch is identified by a unique "Process Identifier" or "Prid". Each process that has the same PRID belong to the same Batch. The PRID is unique between all processes, even between different PLCs. The BatchXpert PLC frame includes functionality to ensure uniqueness of the Prid for up to 8 PLCs connected to the system.

Manually starting a New Batch

To start a recipe or program on a Unit, you must click your mouse on the recipe name box in the drive symbol.



This opens the recipe start window, where you can choose the recipe, you want to start. For a prescription to be initiated, the unit must be in the "Automatic" state.

This will assign a new PRID to the selected unit and thus start a new Batch.

Starting a unit from the PLC

BatchXpert provides multiple Helper functions to start a specific unit from the PLC. This usually happens if a previous unit wants to start the next unit to continue the process in this new unit.

The starting logic of a Unit is usually done in the "Start Position" of the unit that is going to be started. Usually, the Unit that tries to start the unit, will send some "signal" which in turn calls the appropriate functions in its start position.

Of course, it is also possible to implement any other custom logic to initiate new or existing batches.

At its most basic level, you have to supply a Program number, a PRID and then set the PhaseEnd signal in a Unit, to initiate a new Unit start. There are multiple different functions to either initiate a new batch or continuing an existing batch.

```
NETWORK
TITLE =phase 01:  start position

X001: SET    ;

    L    #ProgNo      //Try to start the requested recipe.
    T    "Uxx".U.ProgNo

    L    #Prid
    T    "Uxx".U.PrId  //trigger an existing Prid a new recipe being downloaded

    SET
    S    "PhaseEnd"

    SPA    END;
```

If no existing Prid is provided, a new Prid is automatically assigned and a new batch is initiated, since a new PRID is being created.

```
NETWORK
TITLE =phase 01:  start position

X001: SET    ;

    L      #ProgNo      //Try to start the requested recipe.
    T      "Uxx".U.ProgNo

    //No prid is being assigned, so a new Prid will be generated and used

SET
S      "PhaseEnd"

SPA    END;
```

What happens after starting a unit

When a new Program number is being written to the "Uxx".U.ProgNo" variable, a new Recipe download is being triggered by the Unit from the currently active recipe master. This recipe download may take a few seconds, which means that a Unit may not leave the start position as soon as "PhaseEnd" is set, but rather when the recipe is requested, downloaded and validated by the Unit.

Keep in mind that this process may take a few seconds.

Starting a unit with a new Batch from the PLC

There are system functions to facilitate the startup of new batches or existing batches by any logic you wish to include.

```
NETWORK
TITLE =phase 01:  start position

X001: SET
    CALL  "Bx StartNewBatch"
    Start := "Custom logic Signal"
    ProgNo := "Program number you wish to start"
```

Starting a unit with an existing Batch from the PLC

There are system functions to facilitate the startup of new batches or existing batches by any logic you wish to include.

```
NETWORK
TITLE =phase 01:  start position

X001: SET
    CALL  "Bx StartExistingBatch"
    Start := "custom start signal"
    ProgNo := "The program number you wish to start"
    Prid  := "The PRID of the existing Process"
```

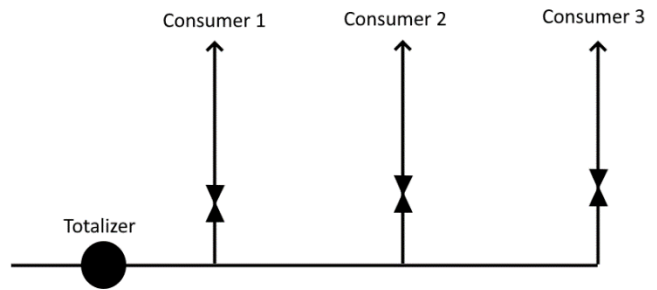
If you are using integrated Unit-To-unit communication, then you can use the following helper method

```
NETWORK
TITLE =phase 01:  start position

X001: SET
    CALL  "Bx StartBatchFromMaster"
        Release:=TRUE
        Master := "Bx UnitCom D".Master1
```

Shared process Resources

Sometimes you will encounter processes that share certain equipment but cannot use said equipment at the same time. In Process technology this may often happen with water supply lines. Here often you can see that you have one totalizer in a water supply line, and multiple consumers. If we want to accurately totalize the amount of water going to each consumer, we must make sure that only one consumer is using the totalizer at any given moment. Otherwise, we cannot know how much water went to the first consumer and how much water to the second, as the water will never distribute itself evenly to all consumers.



Mutexes

Mutexes are an algorithm in programming to protect a resource, in this case the totalizer, from simultaneous usage. They ensure “Mutual Exclusivity”, hence their name. The concept of Mutexes is, that the resource (in our case the totalizer) has an “owner” who is allowed to use it, and only when the current owner releases the resource, the next consumer can claim it and use it if he successfully allocated this resource.

Mutexes may also have “Priorities”, which translates very well to a process environment, since there may also be consumers that have higher priority than others.

This means Mutexes are a perfect way, in fact the ideal way, to manage this problem.

Programming Examples

BatchXpert comes with two default implementations of Mutexes. A simple one, and one with support for Priorities. Each Mutex requires some private data to manage access to its resources. This data should be kept in a data block that you should create for this purpose.

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	WaterMixer1	STRUCT		Mutex with priority
+0.0	CurrID	DINT	L#0	
+4.0	CurrPriority	BYTE	B#16#0	
+5.0	CycleCnt	BYTE	B#16#0	
+6.0	MinTimer	REAL	0.000000e+000	
=10.0		END_STRUCT		
+10.0	WaterMixer2	DINT	L#0	Simple Mutes
=14.0		END_STRUCT		

Simple Mutex to protect a shared process resource.

```
//Each requesting user must provide an "ID" to identify each of the requests and be
//able to assign the resource to one of them. The ID must be unique between all
//the requests. In this example we simply use the "Unit Number" as ID

//request from Mashtun
"U004 MT data".user.ReqWater
= #Request

CALL "Mutex"
Alloc          :=#Request
ID             :=4           //Unit Number of Consumer as unique ID
AllocationOK   :=#AllocationOK //The Resource can be used
AllocationFailure:=#AllocationFailure
ProcessResource :="Process Mutex D".WaterMixer2

U #Request           //if this consumer is requesting
U #AllocationOK      //and has acquired (allocated) the Mutex
S "Bx Act D".Act22.Aco //it may use the Water valve

//request from Lauter tun
"U005 LT data".user.ReqWater
= #Request

CALL "Mutex"
Alloc          :=#Request
ID             :=5           //Unit Number of Consumer as unique ID
AllocationOK   :=#AllocationOK
AllocationFailure:=#AllocationFailure
ProcessResource :="Process Mutex D".WaterMixer2

U #Request
U #AllocationOK
S "Bx Act D".Act22.Aco

//request from Wortkettle
"U006 WK data".user.ReqWater
= #Request

CALL "Mutex"
Alloc          :=#Request
ID             :=6           //Unit Number of Consumer as unique ID
AllocationOK   :=#AllocationOK
AllocationFailure:=#AllocationFailure
ProcessResource :="Process Mutex D".WaterMixer2

U #Request
U #AllocationOK
S "Bx Act D".Act22.Aco
```

Advanced Mutex with priority

```
//Each requesting user must provide an "ID" to identify each of the requests and be
//able to assign the resource to one of them. The ID must be unique between all
//the requests. In this example we simply use the "Unit Number" as ID
//
//Additionally, the Priority can be specified. The higher the number, the higher
//the Priority. Higher priority requests can interrupt lower priority ones.
//
//In this example, the Mash tun has a higher priority than the others, thus will
//always acquire the Mutex first, and even "steal" the mutex from requests that
//already acquired the mutex.

//request from Mashtun
  "U004 MT data".user.ReqWater
  = #Request

  CALL  "Mutex Ex"
  Alloc      :=#Request
  ID          :=4                //Unit Number of Consumer as unique ID
  Priority     :=B#16#2          //The higher number is higher priority
  AllocationOK :=#AllocationOK
  AllocationFailure:=#AllocationFailure
  ProcessResource := "Process Mutex D".WaterMixer1

  U #Request          //if this consumer is requesting
  U #AllocationOK      //and has acquired (allocated) the Mutex
  S "Bx Act D".Act22.Aco //it may use the Water valve

//request from Lauter tun
  "U005 LT data".user.ReqWater
  = #Request

  CALL  "Mutex Ex"
  Alloc      :=#Request
  ID          :=5                //Unit Number of Consumer as unique ID
  Priority     :=B#16#1          //The higher number is higher priority
  AllocationOK :=#AllocationOK
  AllocationFailure:=#AllocationFailure
  ProcessResource := "Process Mutex D".WaterMixer1
  U #Request
  U #AllocationOK
  S "Bx Act D".Act22.Aco

//request from Wortkettle
  "U006 WK data".user.ReqWater
  = #Request

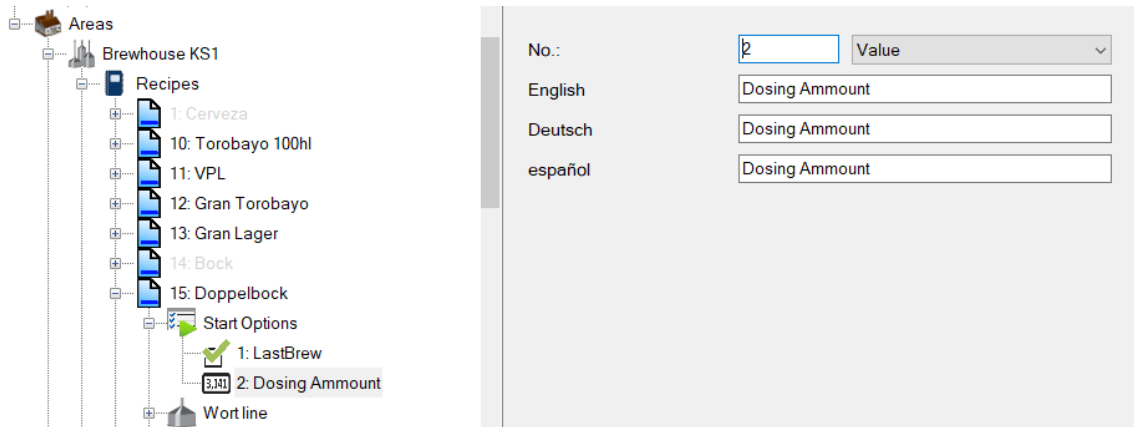
  CALL  "Mutex Ex"
  Alloc      :=#Request
  ID          :=6                //Unit Number of Consumer as unique ID
  Priority     :=B#16#1          //The higher number is higher priority
  AllocationOK :=#AllocationOK
  AllocationFailure:=#AllocationFailure
  ProcessResource := "Process Mutex D".WaterMixer1

  U #Request
  U #AllocationOK
  S "Bx Act D".Act22.Aco
```

Recipe Options

When you start a recipe from the Production scheduler, you can define recipe options for each Recipe individually. Recipe options represent settings that the operator wants to do for a single Batch only, but not for all batches running a particular recipe. This for example applies to situations where you have an option called “Last Brew” where you can mark the “Last brew” of the week, so you can take certain actions in the plc, like for example not recovering weak wort, or waiting with spent grains removal until you can add Trub to the lauter tun.

The Recipe options are configured for each recipe in the Bach configuration as follows.



Start Option Value

In the PLC they get downloaded as Recipe options into the recipe and are available in the structure “ReipeOptions” in the unit data block.

Address	Symbol	Type	Remark
0.0	b24	BOOL	start option bit
0.1	b25	BOOL	start option bit
0.2	b26	BOOL	start option bit
0.3	b27	BOOL	start option bit
0.4	b28	BOOL	start option bit
0.5	b29	BOOL	start option bit
0.6	b30	BOOL	start option bit
0.7	b31	BOOL	start option bit
1.0	b16	BOOL	start option bit
1.1	b17	BOOL	start option bit
1.2	b18	BOOL	start option bit
1.3	b19	BOOL	start option bit
1.4	b20	BOOL	start option bit
1.5	b21	BOOL	start option bit
1.6	b22	BOOL	start option bit
1.7	b23	BOOL	start option bit
2.0	b08	BOOL	start option bit
2.1	b09	BOOL	start option bit
2.2	b10	BOOL	start option bit
2.3	b11	BOOL	start option bit
2.4	b12	BOOL	start option bit
2.5	b13	BOOL	start option bit
2.6	b14	BOOL	start option bit

2.7	b15	BOOL	start option bit
3.0	b00	BOOL	start option bit
3.1	b01	BOOL	start option bit
3.2	b02	BOOL	start option bit
3.3	b03	BOOL	start option bit
3.4	b04	BOOL	start option bit
3.5	b05	BOOL	start option bit
3.6	b06	BOOL	start option bit
3.7	b07	BOOL	start option bit
4.0	Val1	REAL	start option value
8.0	Val2	REAL	start option value
12.0	Val3	REAL	start option value
16.0	Val4	REAL	start option value
20.0	Val5	REAL	start option value
24.0	Val6	REAL	start option value
28.0	Val7	REAL	start option value

Programming Example

Using a start option bit in the Unit Function. You should read the option bits and then make temporary variables out of them at the beginning of the unit function.

```

U "Uxx". StartOption.b01
= #WeakWort           //option 1 = with weak word

U "Uxx". StartOption.b02
= #Trub               //option 2 = With trub dosage

```

Run and Hold timers.

BatchXpert provides timer counter that give you time in seconds how long the Unit already has been in Hold or in Run, or how long the current step is already executing. This is especially useful in situation where you want to keep certain actuators running when a unit is “Holding”, in other words, with these counters you can easily implement an “Holding” transitional state when going from “Run” into “Hold.”

Step Time in "RUN"

Address in Unit DB	Symbol	Datatype	Comment
32.0	THold	REAL	time Unit in hold Time in seconds, the unit is in “Hold”
36.0	TRun	REAL	time unit in run Time in seconds, the unit is in “Run”
40.0	TStepRun	REAL	time step in run Time in seconds, the current Phase is executing
44.0	TRecDownload	REAL	time recipe Download. Time in Seconds, since the recipe was downloaded

Programming Example

The following example shows a simple two stepped shutdown. If the unit is switched to stop, either by an “HoldReq” or by the operator, the actuators 12 and 13 are turned off immediately. Actuator 18 keeps running for 10 seconds and Actuator 19 keeps running for 15 seconds.

```
U "PH"
S "Act". Act[12]. Aco
S "Act". Act[13]. Aco

L "Uxx". U.THold
L 1.000000e+001           //10 seconds
<R
S "Act". Act[18]. Aco     //keep running for 10 seconds when in hold

L "Uxx". U.THold
L 1.500000e+001           //15 seconds
<R
S "Act". Act[19]. Aco     //Keep running for 15 seconds when in hold
```

Event recording

One of the main functions of BatchXpert is its ability to record historical events from batches, but also events and operations that were executed on any Control module of the system. It must be noted that all historical events are created by the PLC, not the operating stations, so they do not depend on any operating station. This way HMI touch panels, which do not execute VisXpert SCADA, can operate modules and since the plc creates all historical events, these are still recorded.

What is being recorded for Batches?

Every Batch creates a batch event when one of the following events occur.

- Status Changed, wither switched to Run, pause or Hold, by either the Operator or internally in the PLC.
- The step was finished automatically.
- The step was aborted by Step+ or Step- from the operator.
- The PLC triggered an event by sending the “ProtWrite M4.3” from a Unit function.

Every time an Event is created, the status of all 16 parameters is being sent to the operating stations master’s so they can record all the data into the database.

This creates an “Event Timeline” of all events that happened in relation to a single batch and allows the “Batch Report Viewer” to visualize this information so the operator can reproduce what happened in a single batch.

What is being recorded for Control Modules?

All control modules will create an “Manual Operation” event whenever one of the following events happens.

- A Control modules Status such as “Simulation”, “Ignore” or “Auto” or any other status is changed.
- A control modules Parameters such as “On Dealy”, “Off Delay” or “Alarm Delay” or any other parameter is changed.
- A control module’s alarm is confirmed.

Every time a control module observes a status or parameter change, it will record this change with the following data is recorded:

- Event Type
- Old Value and New Value
- Control Module Type
- Control Module Number
- Operating Station on which the operation happened.
- User who executed the operation

Master to Operating station communication channels

All recording is being done in the plc by the BatchXpert system. Every time and module or unit registers an event, it will be enlisted into the communication buffers for each of the connected “BatchXpert” operating stations configured as Masters. The buffers are independent to each BatchXpert master and designed to be “First in First out”. The BatchXpert operating station master’s will periodically read their own buffer status and upload events if they exist. The buffers in the PLC can typically hold about 1000 events, which depending on your process and size of the implementation is good for about a few days’ worth of data.

If an BatchXpert Master is not able to empty the buffer in time, for example it is turned off, a buffer overflow happens, and the oldest events get overwritten. However, this will only affect the Master that was turned off, as each master has its own independent buffer.

Each BatchXpert Master station has its own Communication channel that includes its own independent Event buffer in the PLC. So that faults of one operating station can never affect other operating stations.

How is the historical data being stored?

All historical data is uploaded from the event buffers of each of the BatchXpert masters through the Masters communication channel. It is then sent to the SQL server, where the raw data is resolved, and the resolved usable data is stored in the appropriate tables.

The SQL server will resolve data in real time when it receives a new event from the PLC. This data resolution and recording usually takes less than a second, which means that recorded data is available in the database in near real time on the SQL server. It should be noted that the timestamp of the event is put on the event in the plc when the events happened, not when the SQL server observed the event.

What Time resolution do the events have?

The events have a maximum Time resolution of 1 second.

Batch Configuration Audit trail.

Whenever some value in the Configuration database is changed either from the Batch Configuration application or by other means, an historical record is created that stores what was changed, and what the original value was. These recordings extend to configuration changes, Recipe changes, recipe parameter changes and changes in the configuration of control modules.

The “Batch Configuration Auditing Trail” is doing all of this. This system is described in depth in the manual “Manual BatchXpert Database Audit trail”.

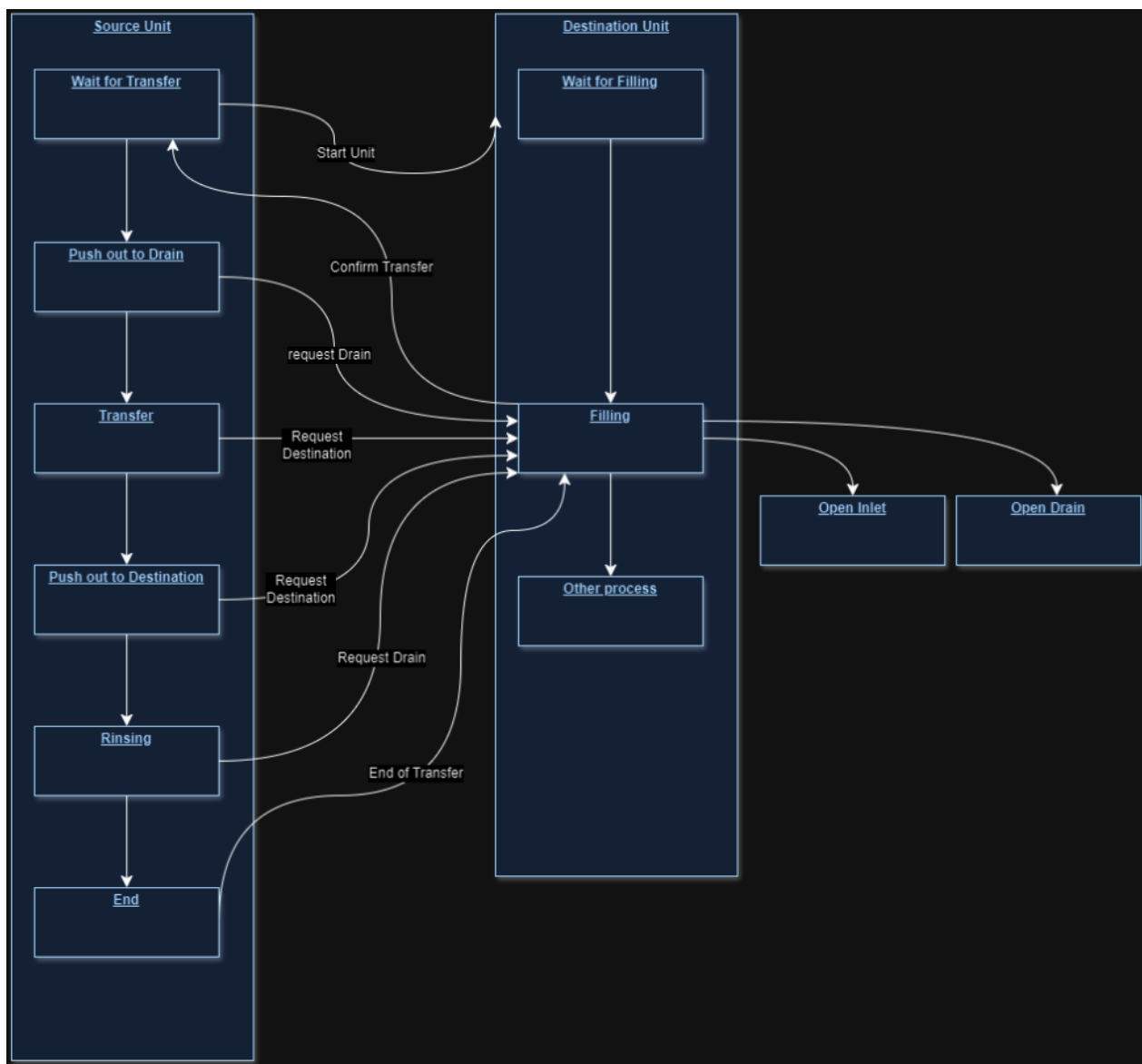
Unit-to-Unit Communication

One of the most challenging parts of process automation is the synchronization between multiple units that participate in a process cell. This usually involves the sending and coordination of signals sent between the participating units, so they can coordinate their phases, activate actuators appropriately and terminate their phases in a coordinated manner.

BatchXpert provides an elaborate mechanism to implement easily and safely these kinds of communication channels in your plc application.

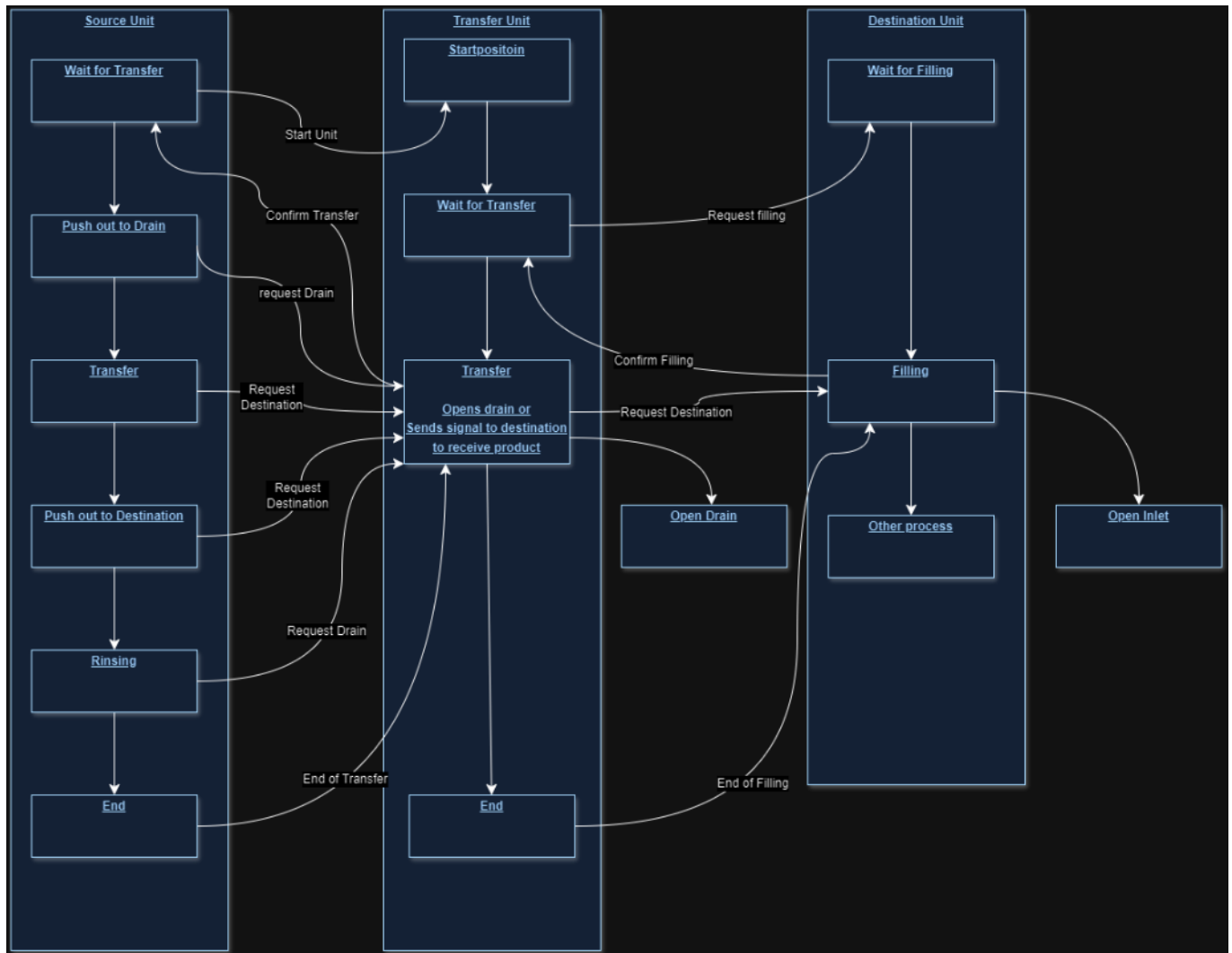
Simple Process Example

A common example of these communication channels between units is product transfer. This usually involves an origin tank and a destination that receives the product. Each of the two parts is represented by its own Process Unit.



More Complex Example

However, for more advanced communication scenarios that involve more than two units, the communication is not as straight forward anymore and requires the application of programming principles to be able to write applications that are readable and safe.



Dynamic communication partners

For some processes it may additionally be necessary to be able to communicate with multiple partners, sometimes at the same time.

An example of this would be any transfer into a “Tank farm”, such as wort cooling into a fermenting cellar. In this example the transferring Unit, the wort cooler, needs to communicate with any of its destination units, the fermenting tanks.

Interfaces

The solution to these problems is “Interfaces”. These define a precise set of signals that each unit must implement and since every unit must implement these signals in the same way as any other unit, they are interchangeable. This means that each unit can consume and produce the same interface.

Interfaces should be as generic as possible and never contain any Unit specific information. This is usually reflected by the naming of the signal names. These must be as generic as possible and never reference any unit specific details. If an interface references unit details, it is not interchangeable with other units anymore, that may not implement this specific detail.

You should never have signals such as “X1105V32_Aco” in any interface because this signal references a specific actuator of one specific unit and is not generic and not interchangeable with other units anymore.

BatchXpert provides a default implementation that should be used for all unit-to-unit communication and is optimized for “Process transfers” that cover all types of transfers with optional push outs. The interfaces are generic on purpose and should not be changed to fit a particular unit, since this would defeat the purpose of interfaces.

Process Unit Allocation

Another consideration that is included in the BatchXpert implementation is the concept of “Unit Allocation”. This means that two units that are communicating are “occupied” by each other, so that no other unit can utilize their communication channel. This is like “Mutex” explained above but additionally serves the purpose of ensuring that a unit communication cannot be interrupted from another unit, requesting the same communication channel. Once a communication channel has been established, both units are occupied by each other, and any further attempt to establish a communication channel from other units will fail.

This ensure that no data is ever overwritten, or two units compete for signals. This avoids “Race conditions” and mixing of different Batches.

Naming Convention

BatchXpert uses the “Master/Slave” nomenclature for its implementation of “Unit-to-Unit” communications. An “Master” is defined as the unit that establishes the communication link and chooses its “Slave”. The “Slave” is the recipient of an “Master’s” communication request. The Slave will automatically accept the request if another does not yet occupy it “Master”.

In summary:

- Master = the one that establishes the Communication Link
- Slave = the target of the communication Link

All data that a Unit sends to its communication partner is inside the “UnitCom.U.Master1” structure and all data that is coming from all slaves and masters is inside the “UnitCom.Master1” structure. The sending data is indicated by “.U.”.

An example:

- UnitCom.U.Master1.xxx Current Unit Signals sent to Master 1
- UnitCom.Master1.xxx Signals received from Master 1
- UnitCom.U.Slave1.xxx Current Unit Signals sent to Slave 1
- UnitCom.Slave1.xxx Signals received from Slave 1
- For Master 2 to 4, or slave 2 to 4, the same applies.

Example of Naming Convention

If we have an “Mash tun” and a “Lauter tun”, and the mash tun wants to transfer into the lauter tun, the mash tun would be the master that establishes its connection by defining to which “Slave” it wants to communicate.

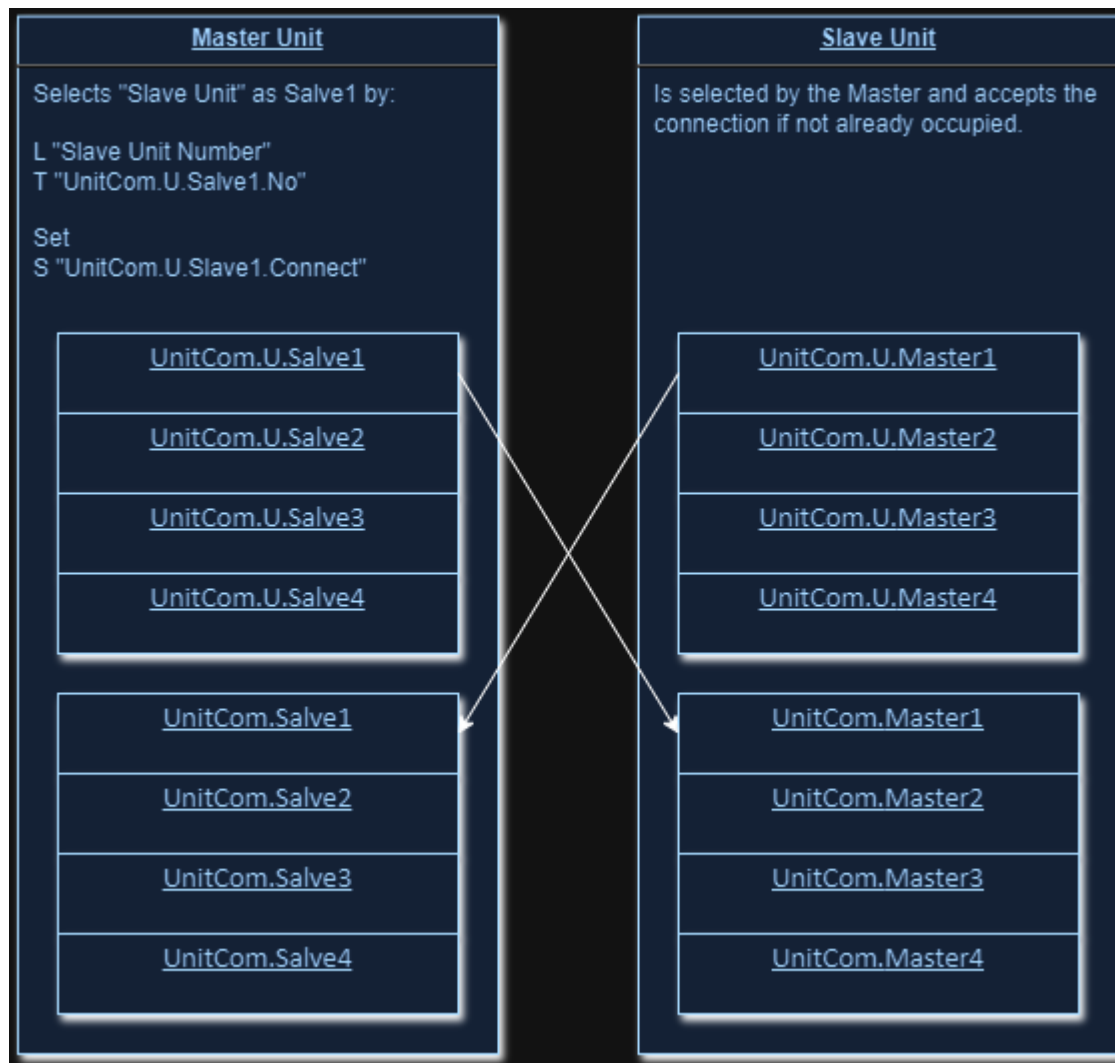
- Mash tun = Master
- Lauter tun = Slave

The mash tun selects the lauter tun as Slave 1, so that its data is sent to the first communication channel.

- If the Mash tun wants to send a signal to the lauter tun, it would do so in “UnitCom.U.Slave1.xxx”
- If the Mash tun wants to read a signal from the lauter tun, it would need to use “UnitCom.Slave1.xxx”
- If the Lauter tun wants to send a signal to the mash tun, it uses “UnitCom.U.Master1.xxx”
- If the Lauter tun wants to read a signal from the mash tun, it uses “UnitCom.Master1.xxx”

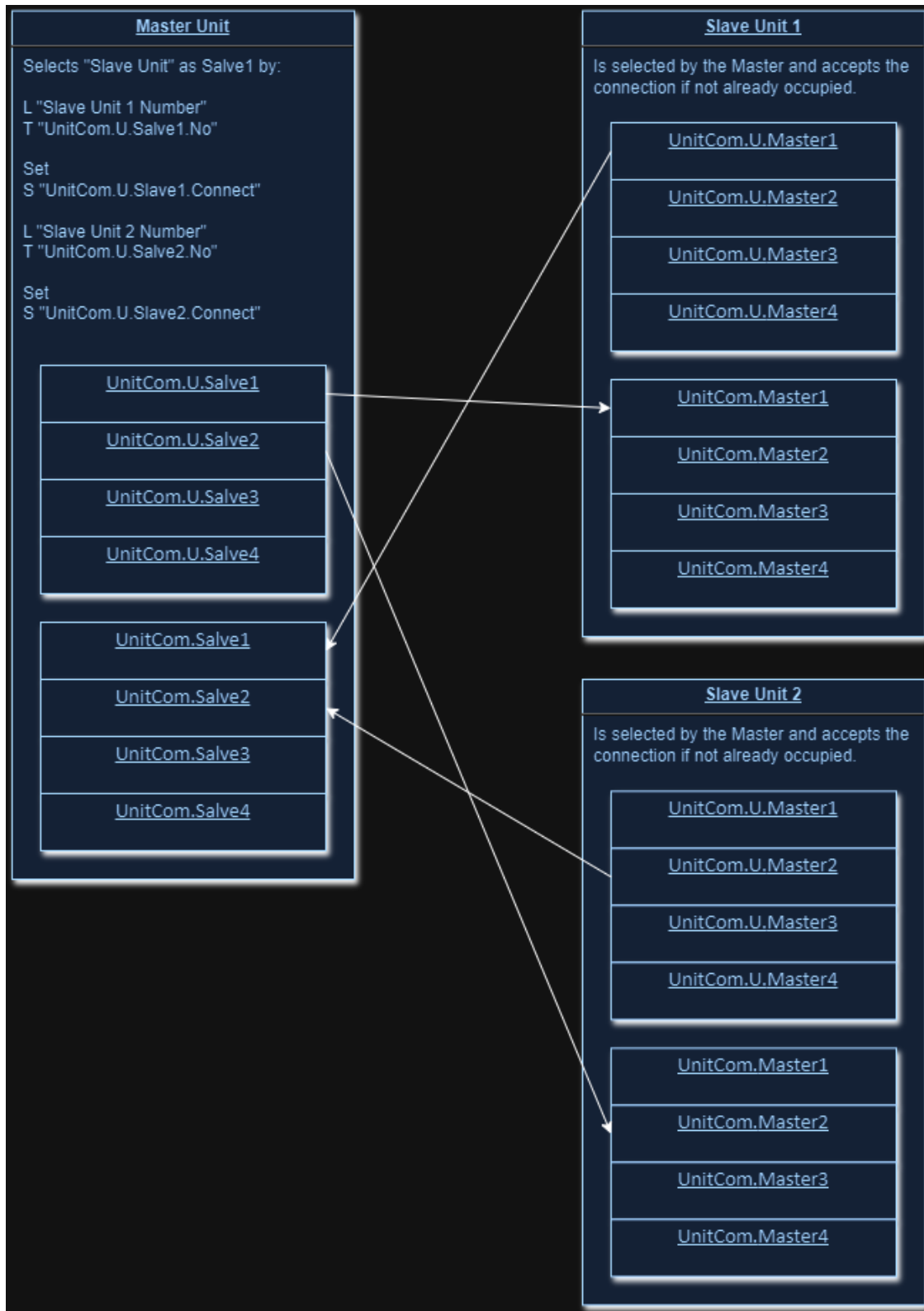
Data Transfer diagram with between two Units

This diagram illustrates this data transfer between the two communication partners in a simple “Master to Slave” communication channel.



Data Transfer diagram with between three Units

This diagram illustrates the data transfer between the three communication partners. Where one master communicates with two slaves at the same time. This may for example be the case for the "Wort cooler," "Wort line" and "Yeast dosing Line".



Data layout

The data is laid out in the communication data block as follows:

Address	Symbol	Type	Comment
This is the “U” area, meaning the data that is sent from the unit to the Slaves/Maters			
0.0	U.General	STRUCT	This data is written automatically, and holds data form the current unit.
4.0	U.PriId	DINT	
8.0	U.Charge	DINT	
12.0	U.ProgNo	DINT	
16.0	U.Master1	UnitComPartnerOwn	The data signals sent to the Respective Master and slaves. It holds the signals defined in the default interface that is exchanged between each master and slave
24.0	U.Master2	UnitComPartnerOwn	
32.0	U.Master3	UnitComPartnerOwn	
40.0	U.Master4	UnitComPartnerOwn	
48.0	U.Slave1	UnitComPartnerOwn	
56.0	U.Slave2	UnitComPartnerOwn	
64.0	U.Slave3	UnitComPartnerOwn	
72.0	U.Slave4	UnitComPartnerOwn	
80.0	U.Val1	REAL	This is a list of arbitrary values that are being sent to each master and slave. These are intended to contain user specific data, such as flow rates, volumes etc.
84.0	U.Val2	REAL	
88.0	U.Val3	REAL	
92.0	U.Val4	REAL	
96.0	U.Val5	REAL	
100.0	U.Val6	REAL	
104.0	U.Val7	REAL	
108.0	U.Val8	REAL	
This is the area where a unit receives the data that it gets from its Masters/Slaves			
200.0	Master1	UnitComPartner	These are the interface signals that are received from the current unit from its masters and slaves
300.0	Master2	UnitComPartner	
400.0	Master3	UnitComPartner	
500.0	Master4	UnitComPartner	
600.0	Slave1	UnitComPartner	
700.0	Slave2	UnitComPartner	
800.0	Slave3	UnitComPartner	
900.0	Slave4	UnitComPartner	

The default signals defined in the Interface

Symbol	Data type	Description
TransReq	BOOL	transfer request Indicates to the communication partner that the unit wants to do a Transfer. This is usually set in the “Wait for Transfer” phases. It does not mean that a Transfer is already running, although it may be active during the actual transfer as well.
TransActive	BOOL	transfer active Indicates to the communication partner that the Transfer is taking place, and the transferring unit is in the “Transfer” step.
TransEnd	BOOL	transfer end Indicates to the communication partner that the transfer should finish. It should not be used as a signal to send the receiving unit into its next step, because of the arguments made in Do not depend on a Specific Phase “being there”
TransRel	BOOL	transfer release control Indicates to the communication partner that the other unit may activate its actuators. It is usually associated with the units Run/Hold Status and with Feedback of valves.
FillReq	BOOL	filling requested Indicates to the communication partner that the units wants to be Filled. This is usually set in the “Wait for Filling” phases. It does not mean that a Filling is already running, although it may be active during the actual transfer as well.
FillActive	BOOL	filling active Indicates to the communication partner that the Filling is taking place, and the transferring unit is in the “Filling” step.
FillEnd	BOOL	filling end Indicates to the communication partner that the Filling should finish. It should not be used as a signal to send the receiving unit into its next step, because of the arguments made in Do not depend on a Specific Phase “being there”
FillRel	BOOL	filling release control Indicates to the communication partner that the other unit may activate its actuators. It is usually associated with the units Run/Hold Status and with Feedback of valves.
Connect	BOOL	Connected Indicates to the Communication system that a connection to the unit indicated in “No” should be initiated. This actually allocates the partner unit and transfers the data. For Slaves it indicates if a master is connected.
Start	BOOL	start partner unit Indicates to the communication partner that they should start up a recipe. This may be the recipe of the communication partner or a new process entirely. This depends on your implementation in the Unit Function.
OpenTank	BOOL	open tank / vessel Open the Destination is requests
OpenDrain	BOOL	open drain Opening the Drain is requested
Water	BOOL	Water Water is currently arriving on the communication partner, or being sent to it
Product	BOOL	Product is currently arriving on the communication partner, or being sent to it
s16	BOOL	
PrIdChange	BOOL	PrId change for partner The partner should execute a PrId change
FunctionNo	INT	special function number

		This is a User defined “Function” code to trigger user defined action on either the master or the slave, that then can be executed. You can interpret this either as an integer value or as individual “bit” signals.
No	DINT	unit number master. This is the Unit number of the communication partner. For masters this is the Unit number that should be connected, and for slaves indicates the connected Master’s unit number.

Custom signals

Custom signals should be sent through the “FuncionNo” integer. This way you can either send a number to trigger action on the communication partner or treat this 16-bit integer as 16 individual bits.

For example, one could define the following table:

0 = no action

1 = This is the first Brew in the Tank

4 = This is the last Brew in the Tank

8 = Some other signal

This way you can easily send some specific signals between the communication partners.

On the sending unit, let us say the slave sends this to its master. On the Slave side you would write:

```
L      4                                //This is the last Brew in the Tank
T      "UnitCom".U.Master1.FuncionNo    //Send to Master
```

And on the Master side you would

```
L      "UnitCom".Slave1.FuncionNo        //Receive from Slave
L      4                                //This is the last Brew in the Tank
==I
... Do stuff
```

Programming Examples

Connect with Slave

Communication can be established with a slave, if the slave's unit number is transferred, and the "Connect" signal can be established. For most situation, the slave's unit number can be transferred statically, as this does not change.

The slave unit number can also be dynamically assigned for example from a preselection system such as “Wort cooler” selecting the Fermenting tank and similar.

```
U "Step0"  
S "UnitCom". U.Slave1.Connect      //Initiate Communication  
  
L 13                               //Partner Unit Number  
T "UnitCom".U.Slave1.No
```

Simple transfer between two units Master side

```
//This code is usually done above the "Jump distributor"  
L 5 //The unit number of the slave  
T "UnitCom".U.Slave1.No  
  
//Then some phases would be implemented like that  
Network  
Title = Wait for Transfer  
  
X10: set  
  
//Send communication Signals  
U "PA"  
S "UnitCom".U.Slave1.Connect //Establish the Communication channel  
S "UnitCom".U.Slave1.Start //Start up the Unit  
S "UnitCom".U.Slave1.TransReq //We want to Transfer  
  
//Receive Signals  
U "UnitCom".Slave1.FillActive //if the Slave has started and entered filling  
S "Phase End"  
  
SPA COMM  
  
Network  
Title = Transfer  
  
X11: set  
  
//Send communication Signals  
U "PA"  
S "UnitCom".U.Slave1.Connect //Establish the Communication channel  
S "UnitCom".U.Slave1.TransReq //We want to Transfer  
S "UnitCom".U.Slave1.TransActive //We are actively in transfer now  
  
U "PH"  
S "UnitCom".U.Slave1.TransRel  
  
//Actuators  
U "UnitCom".Slave1.FillActive  
U "UnitCom".Slave1.FillRel  
U "PH"  
S "ActD".Act123.Aco  
S "ActD".Act124.Aco  
S "ActD".Act125.Aco  
  
//Step End  
U LSL //We got empty  
S "Phase End"  
  
SPA COMM  
  
Network  
Title = Rinsing
```

```

X12:    set

//Reset Water Counter
U      "PFCycle"
S      "CntD".Cnt123.Reset

L      "CntD".Cnt123.PVal
T      "Uxx".Para[11].Val

//Send communication Signals
U      "PA"
S      "UnitCom".U.Slave1.Connect    //Establish the Communication channel
S      "UnitCom".U.Slave1.TransReq  //We want to Transfer
S      "UnitCom".U.Slave1.TransActive //We are actively in transfer now

U      "PH"
S      "UnitCom".U.Slave1.TransRel

//Actuators
U      "UnitCom".Slave1.FillActive
U      "UnitCom".Slave1.FillRel
U      "PH"
S      "ActD".Act126.Aco
S      "ActD".Act124.Aco
S      "ActD".Act125.Aco

//Receive Signals
U      "Uxx".Para[11].D              //Water amount
UN     "PFCycle"
S      "Phase End"

SPA    COMM

... More unit Phases

```

Simple transfer between two units Slave side

```

Network
Title = Start Position

X1:    set

//This function will request a new recipe if the Master requests
//the Salve to be started
CALL   "Bx StartBatchFromMaster"
      Release:="Bx UnitCom D".Master1.Start
      Master := "Bx UnitCom D".Master1

SPA    COMM

Network
Title = Wait for Filling

X2:    set

//Send communication Signals
U      "PA"
S      "UnitCom".U.Master1.FillReq  //We want to be filled

//Step End
U      "UnitCom".Master1.TransReq  //The Master wants to transfer
S      "Phase End"

SPA    COMM

Network
Title = Filling

X3:    set

```

```

//Send communication Signals
U      "PA"
S      "UnitCom".U.Master1.FillReq
S      "UnitCom".U.Master1.FillActive

U      "PH"
S      "UnitCom".U.Master1.FillRel

//Actuators
U      "UnitCom". Master1.TransRel
U      "UnitCom". Master1.TransActive
U      "PH"
S      "ActD".Act12.Aco
S      "ActD".Act13.Aco

//Step End
//We end when no transfer is active anymore
Un     "UnitCom". Master1.TransReq
Un     "UnitCom". Master1.TransActive
S      "Phase End"

SPA    COMM

```

... More unit Phases

Some Common Help functions

Starting your unit with a Program

//All these functions should be called in the "Start Position" phase

```
CALL "Bx StartBatchFromMaster"
  Release:="Bx UnitCom D".Master1.Start
  Master := "Bx UnitCom D".Master1

CALL "Bx StartNewBatch"
  Start := #SomeSignal
  ProgNo:=123 //The Program number to start

CALL "Bx StartExistingBatch"
  Start := #SomeSignal
  ProgNo:=123 //The Program number to start
  Prid := #PridFromSomewhere //The Program number to start
```

Finding the next occurrence of a Phase in the recipe and getting a parameter from it

```
//First, we try to find the next occurrence of phase 8 in the current recipe
L    "Uxx".U.StepNo           //We start searching at the current step
T    #TempInt

CALL "Bx Find SetpNo from Rec"
  StartStepNo:=#TempInt
  PhaseNo      :=8              //try to find Phase number 8
  Recipe       :=DB101.Recipe
  StepNo       :=#FoundStepNumber //if it fails it will be "-1"

//after we found one, we get "Parameter 4" from it
CALL "Bx Get Param from Rec"
  StepNo := #FoundStepNumber
  ParamNo:=4              //We want Parameter 4
  Recipe :=DB101.Recipe
  SP      := #Phase8_Param4_SP //if it fails, it will be "0.0"
  Status  := #Phase8_Param4_Status //if it fails, it will be "0" which is invalid
```

Timer help functions

//If you need some arbitrary delay times you can use the Timer functions
//All of these need an DINT emory from any data block you have available
//There are functions to create "One Shot" cycles
//Variable Pulse and pause timers, as well as "On Delays"

```
CALL "Timer Impulse"
  ImpulseWidth:=T#1S500MS
  Release      :=TRUE
  ImpulseEdge  :=#Impulse
  TimeMemory   := "Uxx".User.DINT1

CALL "Timer Pulse/Pause"
  PulseWidth:=T#1S
  BreakWidth:=T#500MS
  Release    :=TRUE
  PulseOut   :=#PulseOut
  TimeMemory:= "Uxx".User.DINT2

CALL "Timer TOn"
  Start:=#SomeCondition
  SP    :=T#10S
  Done  :=#TimerDone
  ACC   := "Uxx".User.DINT3
```


Flow Integrators

```
//These functions allow you to "Integrate" or "Totalize" a current flow to a volume. There
//are two versions of this. One generates an "Amount" as a Totalized number, and the other
//one generates impulses that can be used to connect to Counter control modules
```

```
CALL "Tec Flow2Impulse"
    FlowVal      := "Ain D".Ain123.PVal //Flow in hl/h
    TimeRatio    := 3600.0 //Seconds in Timebase of flow. It is . per hour, so 3600 sec
    ImpulseValue := "Cnt D".Cnt123.ImpVal //We use what the Counter already defined
    Impulse      := "Cnt D".Cnt123.xSig
    ValCount     := "DB101".User.Reall //a Helper value for counting

CALL "Tec Flow2Amount"
    FlowVal := "Ain D".Ain123.PVal //Flow in hl/h
    TimeRatio := 3600.0 //Seconds in Timebase of flow. It is . per hour, so 3600 sec
    Start     := true
    Pause     := false
    Reset     := false
    ValCount  := "DB101".User.Real2 //The totalized ammount
```

Value to Bit array (Value to Enumeration)

```
//This funcoin is helpful to easier handle Enumerations in sequences. It will take in an
//value n, and then set the n-th bit in a bit array.
```

```
VAR_TEMP
    EnumAiration : STRUCT
        No:      BOOL;
        YES:     BOOL;
        Impulse:  BOOL;
    END_STRUCT ;
    TempInt: INT;
    TempBool: BOOL;
END_VAR
BEGIN

NETWORK
TITLE= Enumeation handling

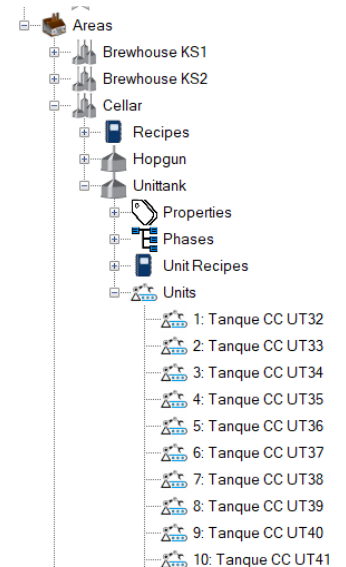
    L "Uxx D".Para[10].SP
    RND
    T #TempInt

    CALL "Conv Number2Enumeration"
        Structure:=#EnumAiration
        Value     :=#TempInt
        Error     :=#TempBool

//if the Value is
//0 it will set "#EnumAiration.No"
//1 it will set "#EnumAiration.Yes"
//2 it will set "#EnumAiration.Impulse"
```

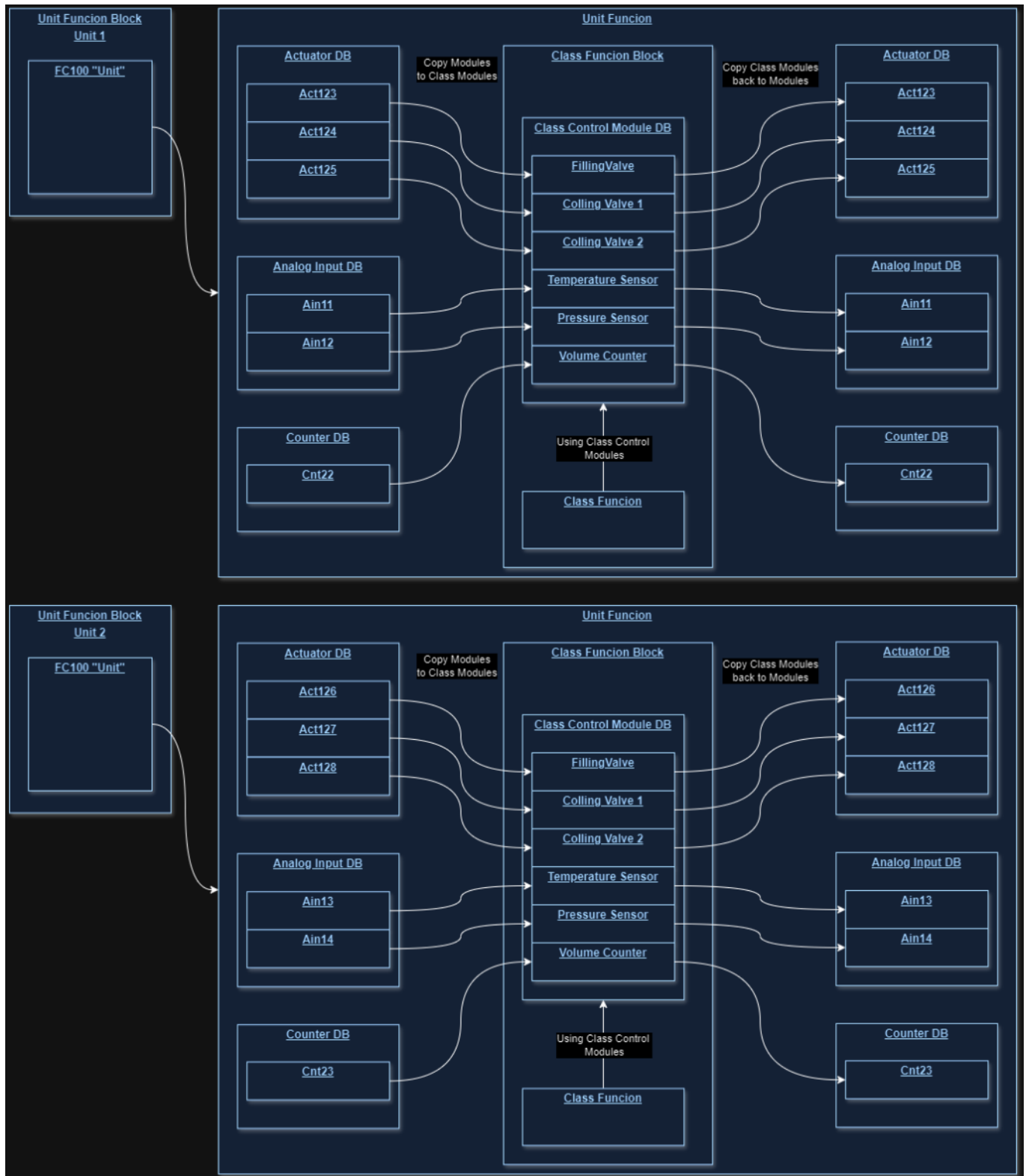
“Class” Programming

For applications where you must implement a lot of units that have identical functionality, such as in an “Tank Farm”, you can use the BatchXpert “Class Programming” paradigm. In the batch Configuration you can create multiple units blow a single Class, which means that all units will share all phases, parameters, and recipes of the class. This is done by simply assigning the same class to all the units belonging to this class.



PLC Fow diagram

In your plc Application this is implemented by the following logic:



The Class Data block

You should create one item for each control module that your class is going to use. You can either make one data block that contains all control modules, or one data block for each module type. It is recommended to use the following naming scheme.

Address	Name	Type	Initial value
0.0		STRUCT	
+0.0	CoolingCono	"sBx Act"	
+40.0	CoolingBottom	"sBx Act"	
+80.0	CoolingTop	"sBx Act"	
+120.0	InOutlet	"sBx Act"	
=160.0		END_STRUCT	

Address	Name	Type	
0.0		STRUCT	
+0.0	TempBottom	"sBx AIn"	
+92.0	TempTop	"sBx AIn"	
=184.0		END_STRUCT	

DB1311	Class UT Act D
DB1312	Class UT DIIn D
DB1313	Class UT AIIn D
DB1314	Class UT Cnt D
DB1315	Class UT PID D
DB1316	Class UT MM D
DB1317	Class UT Switch D
DB1318	Class UT Valv D
DB1321	Class DT Act D
DB1322	Class DT DIIn D
DB1323	Class DT AIIn D
DB1324	Class DT Cnt D
DB1325	Class DT PID D
DB1326	Class DT MM D
DB1327	Class DT Switch D
DB1328	Class DT Valv D

The Unit Function block.

Since you want to write all the logic in the "Class function block", you should not put any logic in the "Unit Function block". Apart from calling "FC100 Unit", you must call no other functions.

```
FUNCTION_BLOCK "U001 UT32 config"
TITLE =Unit 001 Configuration

VAR_TEMP
  EmStop : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Unit Phase Control

//Eventually the FC "Bx Unit" will call the Unit FC belonging to this unit (here FC 110) will be called.
//Refer to this Function for further information about class concept programming
  CALL "Bx Unit" (
    INO      := 1);

END_FUNCTION_BLOCK
```

The Unit Function

The main purpose of the Unit function in a class programming scheme is to copy all the control modules onto the “Class Control Module” data block(s) and then call the “Class Unit Function Block”.

```
FUNCTION "U001 UT32 phases" : VOID
TITLE =Unit xxx Class UT phases
VERSION : 0.1

VAR_TEMP
  dummybool : BOOL ;
  RetValInt : INT ;
END_VAR
BEGIN
NETWORK
TITLE =Supply Class Control modules

//Acts
  CALL "BLKMOV" (
    SRCBLK          := "Bx Act D".Act1,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT Act D".CoolingCono);
  CALL "BLKMOV" (
    SRCBLK          := "Bx Act D".Act2,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT Act D".CoolingBottom);
  CALL "BLKMOV" (
    SRCBLK          := "Bx Act D".Act3,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT Act D".CoolingTop);

//DIn
  CALL "BLKMOV" (
    SRCBLK          := "Bx DIn D".DIn1,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT DIn D".LSL);

//AIn
  CALL "BLKMOV" (
    SRCBLK          := "Bx AIn D".AIn1,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT AIn D".TempBottom);
  CALL "BLKMOV" (
    SRCBLK          := "Bx AIn D".AIn2,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT AIn D".TempTop);

//Cnt
  CALL "BLKMOV" (
    SRCBLK          := "Bx Cnt D".Cnt1,
    RET_VAL         := #RetValInt,
    DSTBLK          := "Class UT Cnt D".Content);

//PID
//no modules existing in  this class

//Msg
//no modules existing in this class

//Switch
//no modules existing in  this class

//ValX
//no modules existing in  this class

//FU
//no modules existing in  this class
```

```

NETWORK
TITLE =Call Class Structure

    UC    "Class UT Config";

NETWORK
TITLE =Desupply from Classes

//Acts
    CALL "BLKMOV" (
        SRCBLK      := "Class UT Act D".CoolingCono,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx Act D".Act1);
    CALL "BLKMOV" (
        SRCBLK      := "Class UT Act D".CoolingBottom,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx Act D".Act2);
    CALL "BLKMOV" (
        SRCBLK      := "Class UT Act D".CoolingTop,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx Act D".Act3);

//DIn
    CALL "BLKMOV" (
        SRCBLK      := "Class UT DIn D".LSL,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx DIn D".DIn1);

//AIn
    CALL "BLKMOV" (
        SRCBLK      := "Class UT AIn D".TempBottom,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx AIn D".AIn1);
    CALL "BLKMOV" (
        SRCBLK      := "Class UT AIn D".TempTop,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx AIn D".AIn2);

//Cnt
    CALL "BLKMOV" (
        SRCBLK      := "Class UT Cnt D".Content,
        RET_VAL      := #RetValInt,
        DSTBLK      := "Bx Cnt D".Cnt1);

//PID
//no modules existing in  this class

//Msg
//no modules existing in  this class

//Switch
//no modules existing in  this class

//ValX
//no modules existing in  this class

//FU
//no modules existing in  this class

END_FUNCTION

```

The Class Function Block

This is where you would program all your logic that is normally done in the “Unit Function Block”, but instead of referring to the Control modules directly, you refer to the “Class control module” that have just been updated with data from the corresponding control modules by the “Unit function”.

```
FUNCTION_BLOCK "Class UT Config"
TITLE =Class Uni Tank Configuration
//Info: Starting from this point the Class is entered, and this the global
//Control module Data blocks ("Bx Act D", "Bx DIn D"... ) are NOT available! they
//cannot be accessed from inside a class (this FB and all the Functions it
//calls)!
//
//Instead, the Class depending on control module data blocks ("Class Example 1  DIn
//D", "Class Example 1  Act D"... ) must be used!
//
//for more information, please refer to the documentation.

VAR_TEMP
  EmStop : BOOL ;
  RelTempSensor : BOOL ;
  TempZone1 : BOOL ;
  TempInt : INT ;
  TempDint : DINT ;
  TempReal : REAL ;
  Temperature : REAL ;
  EnumTempSetpoint : STRUCT
    Auto : BOOL ;
    Top : BOOL ;
    Cone : BOOL ;
  END_STRUCT ;
END_VAR
BEGIN
NETWORK
TITLE =Init

      SET      ;
      =      #EmStop;

NETWORK
TITLE =Analog Input Alarm

//Activate Alarms according to level
      SET      ;
      S      "Class UT AIn D".TempBottom.ELLA; // level sensor Cone
      S      "Class UT AIn D".TempBottom.iEHWA;
      S      "Class UT AIn D".TempBottom.iELLW;
      S      "Class UT AIn D".TempTop.ELLA;
      S      "Class UT AIn D".TempTop.iEHWA;

      S      "Class UT AIn D".TempTop.iELLW;

NETWORK
TITLE =Digital Input Alarm

NETWORK
TITLE =Counters

NETWORK
TITLE =Regulators

NETWORK
TITLE =Parameter Transfer

// tank level
      L      "Class UT Cnt D".Content.PVal;
      T      "Uxx".Para[13].Val;

//Extract
```

```

L      "Class UT Valx D".CurrentExtract;
T      "Uxx".Para[10].Val;

//Temperatures
//Are done in the Class Phases function because they depend on selections from the
//Process details
NETWORK
TITLE =Unit Phase Control

      CALL "Class UT phases" ;

NETWORK
TITLE =Manual/Automatic Handling

      U      "Run";
      S      "Class UT Act D".CoolingCono.xAuto;
      S      "Class UT Act D".CoolingBottom.xAuto;
      S      "Class UT Act D".CoolingTop.xAuto;
      S      "Class UT Act D".InOutlet.xAuto;

      R      "Class UT PID D".Temp.MSpInt;
      R      "Class UT PID D".Temp.MOutMan;

NETWORK
TITLE =Actuator Emergency Interlocks

      U      #EmStop;
      =      "Class UT Act D".CoolingCono.Rel;
      =      "Class UT Act D".CoolingBottom.Rel;
      =      "Class UT Act D".CoolingTop.Rel;

NETWORK
TITLE =Cooling Valves

//do not activate when empty or one of the zones is very cold
      U      "Class UT AIn D".TempBottom.MLLA;
      U      "Class UT AIn D".TempTop.MLLA;
      UN     "Class UT DIn D".LSL.Sig;
      =      "Class UT Act D".CoolingCono.Rel2;
      =      "Class UT Act D".CoolingBottom.Rel2;
      =      "Class UT Act D".CoolingTop.Rel2;

NETWORK
TITLE =In/outlet

      SET      ;
      =      "Class UT Act D".InOutlet.Rel2;

END FUNCTION BLOCK

```


The Class Function

```
FUNCTION "Class UT phases" : VOID
TITLE =Class UT phases
VERSION : 0.1

NETWORK
TITLE =Init

//Current Tank Number
L      "UnitNo";
T      #TempDint;

CALL "Class UT from UnitNo" (
    UnitNo           := #TempDint,
    TankNo           := #User.TankNo);

NETWORK
TITLE =phase number evaluation

L      "Phase";
SPL    X000;
SPA    X000;
//Production Phases-----
SPA    X001; //Start position
SPA    X002; //Start Check
SPA    X003; //Set Status
SPA    X004; //Wait Filling Wort
SPA    X005; //Filling Wort
... More Phases here

//Invalid Phase-----
X000: SET      ; //Invalid or not yet programmed phase number
SPA    END;

NETWORK
TITLE =phase 01: start position

X001: SET      ;
//Start from wort cooler
U      "StatusSteril";
O      "StatusClean";
S      "Bx UnitCom D".U.Master1.FillReq;

U      "Bx UnitCom D".U.Master1.FillReq;
U      "Bx UnitCom D".Master1.Start;
SPBN   Rel;

CALL "Bx StartNewBatch" (
    Start           := "Bx UnitCom D".Master1.Start,
    ProgNo          := "Bx UnitCom D".Master1.ProgNo);

L      "Bx UnitCom D".Master1.Charge;
T      "Uxx".U.Charge;
Rel:   SET      ;

NETWORK
TITLE =phase 02: start check

X002: SET      ;
//Set Status Check
U      "PA";
S      "Class UT Act D".CoolingCono.SCS;

//Phase end
UN     "Class UT Act D".CoolingCono.SCE;
UN     "PFCycle";
=      "PhaseEnd";
```

```

        SPA    END;

NETWORK
TITLE =phase 3:  set unit status

X003: SET    ;

// check status is "selected"
    L        "Uxx".Para[28].Sp;
    L        0.000000e+000;
    <=R      ;
    S        "OpReq";

// phase end condition
    L        "Uxx".Para[28].Sp;
    RND      ;
    L        "StatusInfo";
    ==I      ;
    UN       "PFCycle";
    UN       "OpReq";
    S        "PhaseEnd";

    U        "OpReq";
    SPB      END;
// set unit status
    L        "Uxx".Para[28].Sp;
    RND      ;
    T        "StatusInfo";

    SPA      END;

NETWORK
TITLE =phase 4:  Wait Filling wort

X004: SET    ;
// signal to partner
    U        "PA";
    S        "Bx UnitCom D".U.Master1.FillReq;

// phase end condition
    U        "Bx UnitCom D".Master1.TransReq;
    U        "Bx UnitCom D".Master1.Run;
    UN       "PFCycle";
    S        "PhaseEnd";

    U        "PhaseEnd";
    U        "Run";
    S        "Class UT Cnt D".Content.Reset;

    SPA      END;

NETWORK
TITLE =phase 5:  Filling wort

X005: SET    ;

// amount brews
    UN       "Bx UnitCom D".Master1.TransEnd;
    O        #User.EndFilling_HF;
    SPB      cntb;

    L        "Uxx".Para[29].Val;
    L        1.000000e+000;
    +R       ;
    T        "Uxx".Para[29].Val;
    L        "Uxx".Para[29].Sp;
    <R       ;
    S        "ProtWrite";
cntb: SET    ;

```

```

// alarms
U    "Bx UnitCom D".Master1.TransActive;
U    "PA";
S    "Class UT DIn D".LSL.EA1; //if the wort is not reaching the tank
S    "Class UT Cnt D".Content.EHHA;

// signals to partner
U    "PA";
S    "Bx UnitCom D".U.Master1.FillReq;
S    "Bx UnitCom D".U.Master1.FillActive;

UN   "Class UT Cnt D".Content.GAlS;
U    "PH";
S    "Bx UnitCom D".U.Master1.FillRel;

//Indicate last Brew to fill
L    0;
T    "Bx UnitCom D".U.Master1.FunctionNo;

L    "Uxx".Para[29].Sp;
L    "Uxx".Para[29].Val;
-R   ;
L    1.000000e+000;
==R  ; //when only
SPBN UltC;
L    1;
T    "Bx UnitCom D".U.Master1.FunctionNo;
UltC: SET ;

// cooling
U    "PH";
S    #CoolingACO;

// phase end condition
U    "Uxx".Para[29].D;
UN   "Bx UnitCom D".Master1.TransReq;
UN   "Bx UnitCom D".Master1.TransActive;
UN   "PFCycle";
S    "PhaseEnd";

// tank volume
U    "Bx UnitCom D".Master1.OpenTank;
U    #FillingCountSignal;
S    "Class UT Cnt D".Content.xSig;
L    1.000000e+000;
T    "Class UT Cnt D".Content.ImpVal;
SPA  END;

... More phase processing logic

END_FUNCTION

```

The Unit No to Class Number converters

Many times, you will have to convert your Unit Number to the Class entity Number, for example Uni-Tank Number, and back. This should be implemented by two Conversion functions. These conversion functions should always exist and be used.

Class Entity Number to Unit Number

```
FUNCTION "Class UT to UnitNo" : VOID
TITLE =Convert Syrup Tank No to Unit No
//Company:      MLogics
//Dependencies:  None
//Class:        Class "UT"
//Comment:      This function converts an input UT Tank no to its Unit No for
//              further processing. If the input Tank number cannot be resolved
//              to a unit number (if it is a valid tank number), then this
//              function returns dez -1.

NETWORK
TITLE =Evaluate input Syrup Tank No

    L      #TankNo;
    SPL    Err;
    SPA    Err;

//UT270
    SPA    T002; //UT 1
    SPA    T002;
    SPA    T002;
    SPA    T002;
... Skipped for brevity

    SPA    T001;
    SPA    T001;
    SPA    T004; //UT41
    SPA    T001;

//Invalid tank number
Err:  L      L#-1;
      T      #UnitNo;
      CLR    ;
      SAVE   ;
      BEA    ;

NETWORK
TITLE =Calculate Tank number

T001: L      #TankNo;
      L      31;
      -D     ; //UT101 = Unit 11
      T      #UnitNo;
      SET    ;
      SAVE   ;
      BEA    ;

NETWORK
TITLE =UT270

T002: L      #TankNo;
      L      60;
      +D     ; //UT101 = Unit 11
      T      #UnitNo;
      SET    ;
      SAVE   ;
      BEA    ;

NETWORK
TITLE =UT41

T004: L      85; //UT101 = Unit 11
      T      #UnitNo;
      SET    ;
      SAVE   ;
      BEA    ;

NETWORK
TITLE =UT549

T005: L      #TankNo;
      L      62;
      +D     ;
      T      #UnitNo;
      SET    ;
      SAVE   ;
      BEA    ;

END FUNCTION
```

UnitNumber to Class Entity Number

```
FUNCTION "Class UT from UnitNo" : VOID
TITLE =
//Company:      MLogics
//Dependencies:  None
//Class:         Class "UT"
//Comment:       This function converts an input Unit No to its UT Tank
//              number for further processing. If the input Unit number can not
//              be resolved to a Tank number (if it is a valid Unit number),
//              then this function returns dez -1.
AUTHOR : MLogics
FAMILY : UT900
NAME : UT_Unit

NETWORK
TITLE =Evaluate input Syrup Tank No

    L      #UnitNo;
    SPL    Err;
    SPA    Err;
    SPA    T001; //Unit 1
    SPA    T001;
... Skipped for brevity
    SPA    Err;
    SPA    Err;
    SPA    Err;
    SPA    Err;
    SPA    Err;
    SPA    Err;
    SPA    Err;
    SPA    Err;
    SPA    T002; //Unit 61
    SPA    T002;
    SPA    T002;
    SPA    T002;
... Skipped for brevity
    SPA    T005;
    SPA    T005;

//Invalid tank number
Err: L      L#-1;
    T      #TankNo;
    CLR    ;
    SAVE   ;
    BEA    ;

NETWORK
TITLE =UT900

T001: L      #UnitNo;
    L      L#31;
    +D     ; //Unit 11 = UT32
    T      #TankNo;
    SET    ;
    SAVE   ;
    BEA    ;

NETWORK
TITLE =UT36

T003: L      36;
    T      #TankNo;
    SET    ;
    SAVE   ;
    BEA    ;

NETWORK
TITLE =UT41

T004: L      41;
    T      #TankNo;
    SET    ;
    SAVE   ;
    BEA    ;

NETWORK
TITLE =UT540

T005: L      #UnitNo; //Unit86 = UT24
    L      62;
    -D     ;
    T      #TankNo;
    SET    ;
    SAVE   ;
    BEA    ;

END_FUNCTION
```

Programar y gestionar lotes

Archivo

Horario de lote

Sistema en ejecución

Actualizar pedidos sistema

Guardar pedidos

Nuevos pedidos

Eliminar pedidos

Bloquear pedidos

Liberar pedidos

+1 hora

-1 hora

+15 minutos

-15 minutos

Editar Hora de inicio

+1 hora

-1 hora

+15 minutos

-15 minutos

Sala de Cocimiento KS1

27/05/2024 06:18:32 p. m.: 1 Bock (14)

27/05/2024 09:18:32 p. m.: 2 Bock (14)

28/05/2024 12:18:32 a. m.: 3 Bock (14)

28/05/2024 03:18:32 a. m.: 4 Bock (14)

28/05/2024 06:18:32 a. m.: 5 Bock (14)

Detalles del lote programado

Vista Escala de tiempo

Receta

Bock

...

Transporte de malta

▼

Información de lote

Número de lote

2

▼

Inicio

27-05-2024

▼

21:18:32

▼

☑

Hora de inicio relativa al lote anterior

Scheduled

Opciones de lote

57 Mensajes

🔍 Detalles

Created 5 new Batches

The planned time for the production rate is passed on to a deferred batch.

+	ST	Info
+	ST	Opt Alarm Groups
-	ST	Opt Production Scheduler
		<div> <div>Info</div> <div>Bausteine</div> </div>
+	ST	Opt Recipe Buffer PLC
+	ST	Opt Recipe Editor PLC

Object name	Symbolic name	Created in language	Size in the work me...	Type
FC3	Bx ProdSchedule	STL	1042	Function
FC581	Bx ProdSchedule2	STL	596	Function
DB3	Bx ProdSchedule D	DB	488	Data Block
DB551	Bx ProdSchedule2 D	DB	5646	Data Block
UDT281	sBx ProdSchedul	STL	---	Data Type
SFC24	TEST_DB	STL	---	System function

```
//this network must be inserted into OB1
Network
Title= Production Schedule

CALL "Bx ProdSchedule2"
```

PLC-to-PLC communications

BatchXpert does include optional Templates on how we recommend implementing communications between PLC multiple PLCs. If the provided examples do not fit your specific needs, you can implement your own communication mechanisms.

The Provided Example allows you to establish connections to all S7-Connection Compatible PLC's, even if they do not run with BatchXpert. You can, for example, establish communications with individual Machine control systems, such as, pasteurizers, or other BatchXpert PLCs.

S7 Communications

S7-connections are a proprietary communication protocol from Siemens which is implemented by most Simatic compatible PLC, for example from Siemens or Vipa. It is supported by all couldn't and past Simatic PLC series. The communication protocol allows you to read and write arbitrary data from any S7- communication compatible CPU.

S7 communications define on server and client model where clients will actively establish connections to server and then request data from the server which in turn replies. This means that only clients will actively establish connections to servers, but servers will never actively establish connections and only ever accept connection requests from clients.

S7 connections only must be configured on the client side. The server usually does not know which or how many clients are connected and requests data from it. This is an advantage since this usually means that you do not have to configure the communication on both PLCs only on the PLC that will act as communication client, Meaning the one actively establishing the connection to the server. However, this also means that servers will have no control over which data will be read or written from and to the PLC by clients.

S7 Communications processors

most of the modern S7-1500 PLC series implement server and client functionality on the CPU. However not all communication processors, especially on the previous 300 and 400 series, will implement client functionality. All the Ethernet capable communication processors and CPUs will at least implement the S7-communication server functionality, But not necessarily also the client functionality.

Most notably the CP 343-1 communication processors do not implement S7-connection client functionality, and only implement S7-connection server functionality, since their intended use is to be used to connect to Ethernet based HMI systems.

NOTE! Please check the data sheet for your specific communication processor or CPU to support your required communication features. If your communication processor or CPU does not support client functionality, it cannot actively establish connections to any other PLC system, and only serve as communication partner if partner PLC access active client and reads and writes data from and to the PLC.

S7 Communications Settings

The S7-Connection require the Slot and Rack communication parameters, which should be defined according to which PLC you are trying to communicate with

S7-300 series	Rack = 2, Slot = 0
S7-400 Series	Rack = 3 (usually), Slot = 4. But both parameters depend on your specific configuration
S7-1500 Series	Rack = 0, Slot = 0
S7-1200 Series	Rack = 0, Slot = 0

Get-Get Communication

To improve readability of PLC programs, you should always implement and get to get communication on all involved PLCs. This means that each PLC will only ever read data from the other PLCs and never write data to any other PLC.

This way data will only ever get read from any PLC in their communication system, and will never be written by outside sources, such as communication channels.

However, it may be necessary for you to implement and get put communication in the case your partner PLC does not support S7-Communication Client functionality. In this case your only option is to read and write data from your partner PLC.

Note if both of your PLC's only support server functionality and do not support client functionality you cannot establish communication between the two PLC's. At least one of the PLCs must support client communication.

Configure Communication Channels

The communication method is being configured in Simatic manager or in Tia-portal, and basically allows you to define your communication partner, it's IP address and connection settings, and your local ID of your connection. This local ID is important since this is your identifier that you must use in your application for the communication data blocks so these data blocks can reference the connection that they should use.

The screenshot displays the 'Properties - S7 connection' dialog box, which is used for configuring S7 communication between two PLCs. The dialog is divided into two tabs: 'General' and 'Status Information'. The 'General' tab is currently selected and shows the following settings:

- Local Connection End Point:** Includes checkboxes for 'Configured dynamic connection' (unchecked), 'Configured at one end' (checked), 'Establish an active connection' (checked), and 'Send operating mode messages' (unchecked).
- Block Parameters:** Shows 'Local ID (Hex): 2' and 'W#16#2'.
- Connection Path:** Includes fields for 'End Point' (PLC01 KS2/ALG211-K03), 'Interface' (ALG211-K03, ALG211-K03(R0/S2)), 'Subnet' (EthernetKunstmann [Industrial Ethernet]), and 'Address' (192.168.99.12).

Below the dialog box, a table lists the active connections:

Local ID	Partner ID	Partner	Type	Active connection p	Subnet
	KS1		S7 connection	Yes	EthernetKunstma...
	Ferment		S7 connection	Yes	EthernetKunstma...

Implementing in the PLC

Function Block example with a single Segment

```
FUNCTION_BLOCK "Comm xxx"

BEGIN
NETWORK
TITLE =Trigger
//Here we trigger the communication every one second.
//then it will run through each of the configured Segments
    U    #Enabled;
    U    M    881.0;
    =    #Trigger;

    L    0;
    T    #Fill;

    L    W#16#2;
    T    #ConnID;
NETWORK
TITLE =Segment 1 Example to GET data from an PLC
//here we define which data we are loading from the Other PLC
    L    #Segment;
    L    0;
    ==I    ;
    SPBN    Seg1;
    CALL #Job1 (
        REQ                := #Trigger,
        ID                  := #ConnID,
        ADDR_1              := P#DB89.DBX 1200.0 BYTE 120,
        RD_1                := #ReceiveData.WCcomm);

//if the Job failed, we reset the received data to all Zeros, so we never get
//Frozen data
    U    #Job1.ERROR;
    SPBN    err1;
    L    #Job1.STATUS;
    CALL "FILL" (
        BVAL                := #Fill,
        RET_VAL              := #Fill,
        BLK                  := #ReceiveData);
err1: SET    ;

    U    #Job1.ERROR;
    O    #Job1.NDR;
    SPBN    Seg1;
    L    0;
    T    #Segment;
Seg1: SET    ;

NETWORK
TITLE =Status Signals
//Here we evaluate an Error, and set the communication to OK or error
    U    #Job1.ERROR;
    S    #Error;
    R    #ConnectionOK;

    U    #Job1.NDR;
    R    #Error;
    S    #ConnectionOK;

    UN    #Enabled;
    R    #Error;
    R    #ConnectionOK;
END FUNCTION_BLOCK
```

Function Block example with “Chunking” of data in two segments

```
FUNCTION_BLOCK "Comm xxx"

BEGIN
NETWORK
TITLE =Trigger
//Here we trigger the communication every one second.
//then it will run through each of the configured Segments
    U      #Enabled;
    U      M      881.0;
    =      #Trigger;

    L      0;
    T      #Fill;

    L      W#16#2;
    T      #ConnID;
NETWORK
TITLE =Segment 1 Example to GET data from an PLC
//here we define which data we are loading from the Other PLC
//this is the first data segment we load
    L      #Segment;
    L      0;
    ==I    ;
    SPBN   Seg1;
    CALL   #Job1 (
        REQ      := #Trigger,
        ID       := #ConnID,
        ADDR_1   := P#DB89.DBX 1200.0 BYTE 120,
        RD_1     := #ReceiveData.WCcomm);

//if the Job failed, we reset the received data to all Zeros, so we never get
//Frozen data
    U      #Job1.ERROR;
    SPBN   err1;
    L      #Job1.STATUS;
    CALL   "FILL" (
        BVAL     := #Fill,
        RET_VAL   := #Fill,
        BLK      := #ReceiveData);
err1: SET ;

    U      #Job1.ERROR;
    O      #Job1.NDR;
    SPBN   Seg1;
    L      1;
    T      #Segment;
Seg1: SET ;

NETWORK
TITLE =Segment 2 Example to GET data from an PLC
//here we define which data we are loading from the Other PLC
//this is the second data segment we load
//Note: We still use the same "Job1" and the same connection ID,
//as for the first segment
    L      #Segment;
    L      1;
    ==I    ;
    SPBN   Seg2;
    CALL   #Job1 (
        REQ      := #Trigger,
        ID       := #ConnID,
        ADDR_1   := P#DB99.DBX 1200.0 BYTE 120,
        RD_1     := #ReceiveData.WCcomm2);

//if the Job failed, we reset the received data to all Zeros, so we never get
//Frozen data
    U      #Job1.ERROR;
```

```

    SPBN  err2;
    L      #Job1.STATUS;
    CALL  "FILL" (
        BVAL      := #Fill,
        RET_VAL    := #Fill,
        BLK        := #ReceiveData);
err2: SET  ;

    U      #Job1.ERROR;
    O      #Job1.NDR;
    SPBN  Seg2;
    L      0;
    T      #Segment;
Seg2: SET  ;
NETWORK
TITLE =Status Signals
//Here we evaluate an Error, and set the communication to OK or error
    U      #Job1.ERROR;
    S      #Error;
    R      #ConnectionOK;

    U      #Job1.NDR;
    R      #Error;
    S      #ConnectionOK;

    UN     #Enabled;
    R      #Error;
    R      #ConnectionOK;
END_FUNCTION_BLOCK

```

Callsite in the OB1 Organization Block

```

ORGANIZATION_BLOCK "CYCL_EXC"
BEGIN
NETWORK
TITLE =Communications

    CALL "Comm xxx" , "Comm KS1 Data" (
        Enabled      := TRUE);

END_ORGANIZATION_BLOCK

```

Unit Communication Channels

The mentioned above, this communication method allows you to read arbitrary data from your partner PLC, however if you are communicating with the patch expert system you can read units communication interface and then use it in your PLC as if it would be any other communication interface on the local PLC.

This allows you to use the same communication interface that inter unit communications use also between different PLC. For more information on how to use inter unit communications please refer to [Unit-to-Unit Communication](#).

Data length for Get and Put connections

The amount of data that you can request with each GET call is limited by several factors, and may vary depending on the Local PLC type, the remote PLC type and the Communication processors involved. The amount of data may be between 100 bytes up to 400 bytes of total data, depending on these factors. Especially for S7-1200 PLC's the amount of data is very limited.

In your experience, the data length limits are approximately according to the following table for each "GET" request. These are just values based on our experience, and should only be used as guidance, rather than documented limits:

If either the Local or Remote PLC is an S7-1200	120 bytes
If either the local or remote PLC is an S7-300	200 bytes
If the local AND remote PLCs are S7-400	350 to 400 bytes
If an “Lean” CP is used	200 bytes

In conclusion, usually the limit is around 200 bytes per “Get” request.

To find the maximum amount of data, we recommend doing empirical tests, to find this limit. Calculating it reliably, is nearly impossible, due to the lack of documentation from the Hardware vendors.

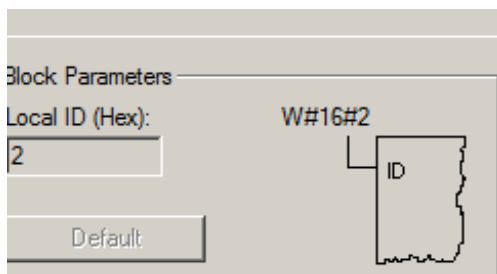
If more data needs to be exchanged between PLCs, than a single “GET” call can support, we recommend you use an “Chunking” or “Segmentation” Mechanism by extending the corresponding network in the communication function accordingly. See above for an example. You can extend the function to as many segments as you need. You must keep in mind however, that each segment requires time to execute, meaning, the more segments you have, the slower your communication will get.

Since “Chunking” or “Segmentation” of your communication data may result in inconsistent data, you may have to employ Double-Buffering or other mechanisms to obtain the required Data consistency level.

Special Considerations for S7-1500 Series

By default, S7- 1500 series PLCs will block get and put requests from PLCs. To enable this functionality, you must activate the setting in your hardware configuration of the PLC ([Allow “Get/Put”](#)).

Special Considerations for Vipa CPU's



Due to limitations of the S7.protocol implementation on Vipa communication processors, the “Local ID” cannot be freely chosen on Vipa PLCs. The communication ID for each connection that is adjusted on the communication dialogue must always be below the maximum number of supported S7-Connections (usually 16). If you are just connection identifiers for example of 101 to implement some naming scheme for connection ID's, these connections will never be able to establish connection to any other PLC you have to use communication IDs between one and the maximum number of supported S7 connections by the PLC.

Usually this means that you can only use the connection IDs between 1 and 16.

Extending of Control Module Data blocks

When adding new control modules to the PLC, special care must be taken since the new control module starter has to be transferred to the appropriate data blocks in your controller. Adding new control modules can be differentiated between two basic cases.

Using a spare control module

At the beginning of a project, you create all your control modules data blocks. These data blocks by default include spare control modules that already exist in the controller but are not actually used by your application. If the control module starter is already present in the data in the data block in the PLC, you can simply open the data block in Simatic manager or tier portal and change its comment to reflect your new control module. A download of deep control module data block is not required in this case.

If enough spares are present in the data block, you can simply rename these pairs and use them straight away.

Extending the data block: Considerations

Sometimes it is necessary to extend the length of the existing control module startup block, to make space for the new control modules. This usually happens if larger portions of existing applications are extended and many new control modules must be added to the existing ones.

In that case an extension of the existing control module data blocks is necessary, which also means that a download of this new data block into the controller is also required. You must keep in mind that a data block download of Simatic controllers implies that all the data block will be overwritten by the ones stored in your project. This means that all control module settings, such as simulation status, alarm delays, alarm levels, will be overwritten by the settings in your project that you are going to download.

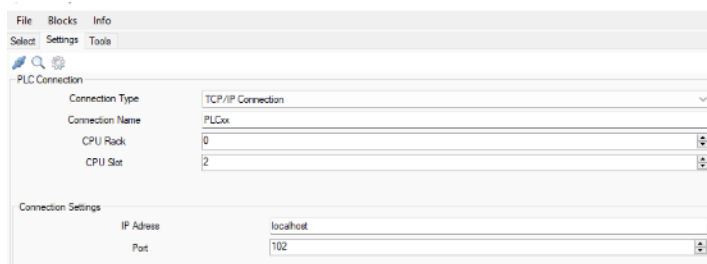
Extending the data block: S7-Backup

MLogics provides tools to mitigate this problem. The S7-backup tool allows you to upload the current data of a data block, download your new extended data block, and then subsequently downloading the original current data of the control modules that existed before downloading your new data block. This allows you to save the current data before downloading the new data block and restore the current data of all control modules after downloading your new extended data block.

S7-backup is available as a separate installer, which can be downloaded by the BatchXpert installation center, or directly from our documentation homepage.

Extending the data block: How to upload and download

Input Connection parameters of your PLC

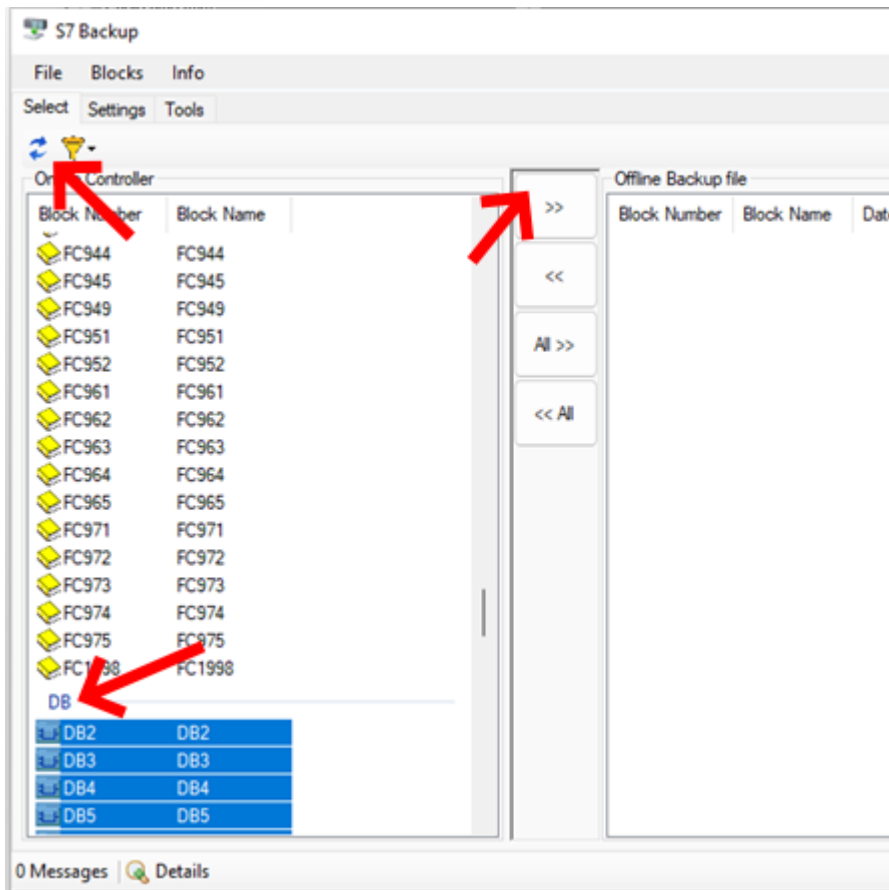


PLC Connection	
Connection Type	TCP/IP Connection
Connection Name	PLC0x
CPU Rack	0
CPU Slot	2

Connection Settings	
IP Address	localhost
Port	102

Discover all blocks and upload at least the data blocks

You can of course also upload all the blocks containing the controller, so you can already obtain an online backup of your current PLC. This upload can then be saved to a backup file, which can be reviewed or even restored if necessary.

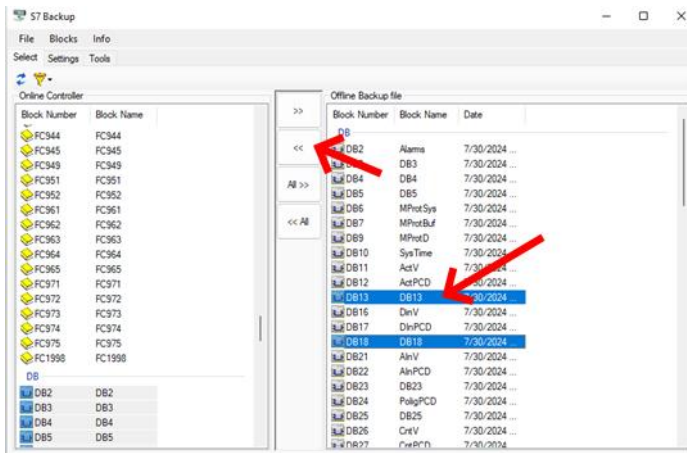


By clicking the upload button, the upload dialog appears, and the download of the selected blocks is executed

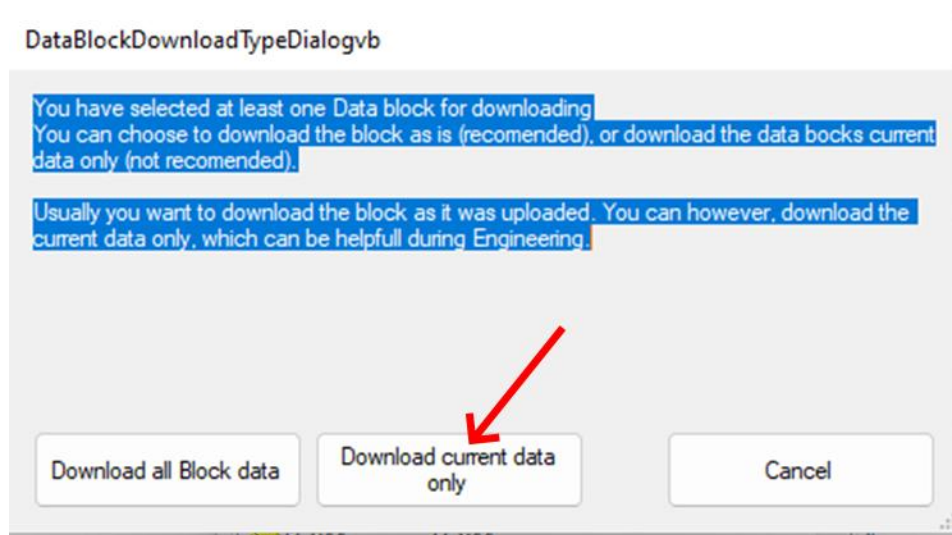


Now you have saved the current data of the selected data blocks to your offline file, which means that you can **now download your new extended data blocks** and temporarily overrides the current data of the control modules.

After your download has finished you can **select the relevant data blocks from your saved file and click the download button.**



After confirming the downloads, you will be asked if you want to download the full data block, or only the current data that you saved previously. It is very important to **select the "download current data only" option**. This will only download the current data of all control modules that existed before you downloaded your new extended control modules doctor block. The "Download all block data" Would restore your controller startup block to the same as it was when you took your backup, which means that your newly extended data block would be reset to its old size.



Overview of HMIs

The BatchXpert system primarily uses the VisXpert SCADA system for its HMI screens, which is integrated with BatchXpert. BatchXpert cannot function without at least one full VisXpert station available. All HMI field panels should be seen as an “auxiliary operating station,” more than a fully-fledged station. Only BatchXpert operating stations using the VisXpert system support the full functionality of BatchXpert, such as Recipe editing, Reports and others.

For this reason, the use of dedicated “traditional” Hmi touch devices such as Simatic Panels, are discouraged in favor of “Touch Panel PC’s”. These Panel PCs are running the same “VisXpert” SCADA system as all other Operating stations, and thus use the same project without modification, thus eliminating the need to maintain multiple HMI and SCADA systems synchronized.

Screen Resolutions

BatchXpert includes libraries for the following SCADA and HMI systems and resolutions. New Resolutions can be added, but the Templates must be adjusted manually to fit the screen size. VisXpert supports full screen scaling, enabling the user of different Screen resolution.



- **VisXpert** Visualization
 - 1920 x 1080 (Full HD)
 - 1600 X 900 (HD)
 - 1366 x 768 (Notebook)

As stated above, the use of traditional touch panels is discouraged in favor of Panel PCs. Thus, the following list of HMI devices is supported, but should not be used for new projects.

- **Movicon** Display for Touch Screens
 - 1024 x 768" (10")
 - 800 x 480 (7")
- **WinCC Flexible** for Simatic Touch Screens (without TIA Portal)
 - 800 x 480
- **WinCC Basic and WinCC Comfort** for Simatic Touch Screens (with TIA Portal)
 - 800 x 480
 - 1280 x 800

Adjust Screen Resolution

In VisXpert, the screen resolution is chosen based on the “BaseProcessWindow,” which comes with versions for different screen resolutions. These different windows have adjusted layout. To choose an “BaseWindow” and a corresponding Screen resolution, you must adjust the default base window in the “Settings” variables in the “VisXpert” variable editor. By default, the screen resolution is set to “Full HD” (1920 x 1080).

Since VisXpert Version 10, the HMI system supports scaling the Hmi screens to fit the current screen resolution. To take advantage of this, you must enable the corresponding setting in the Graphics editor of VisXpert.

This way the recommended solution is to create all process screens for a native “Full HD” resolution and then scale them down for stations that have lower resolutions. However, this scaling can result in suboptimal graphics, which can only be avoided by designing the Graphics display to the desired resolution.

For the other HMI Project templates, there are projects with different resolution prepared in the specific folders.

Multiple Monitors

Multiple monitors are supported by the VisXpert system by using the “MultiMon” VisXpert module. For details, please refer to the “Manual BatchXpert Multi Monitor Support” manual.

"Visu Extern" System (Touch Screens)

The BatchXpert system implements a sophisticated HMI customization mechanism that allows you to implement custom HMI systems with custom status words and even customized data exchange. This communication channel is referred to as "Visu Extern" and allows you to fully customize your communication mechanism with your customized HMI solution. You can implement multiple different "Visu Extern" communication channels on the same PLC, thus supporting multiple HMI mechanisms at the same time. This mechanism is generally used for simpler touch panels that do not run the full VisXpert SCADA system, for example WinCC Comfort or MoViCon Touch panels.

Since this communication mechanism builds upon the internal HMI mechanism of batch expert, all historic data recording for manual interventions is still fully supported, even for external HMI systems.

Nowadays the use of "Visu Extern" for "Touch HMI Systems" is generally discouraged in favor of touch panel PCs running on the VisXpert SCADA System.

External HMI

This communication channel is commonly used for WinCC and MoViCon based touch panels come on but also text panels or even legacy "Mosaik" systems. The BatchXpert system separates between two types of Visualization (HMI)

- **Internal View:** The main system, integrated into the BatchXpert System. Only the "VisXpert" SCADA system is available.
- **External View:** All other HMIs. For example, Siemens WinCC Flexible, Progea Movicon, or other types of touch screens.

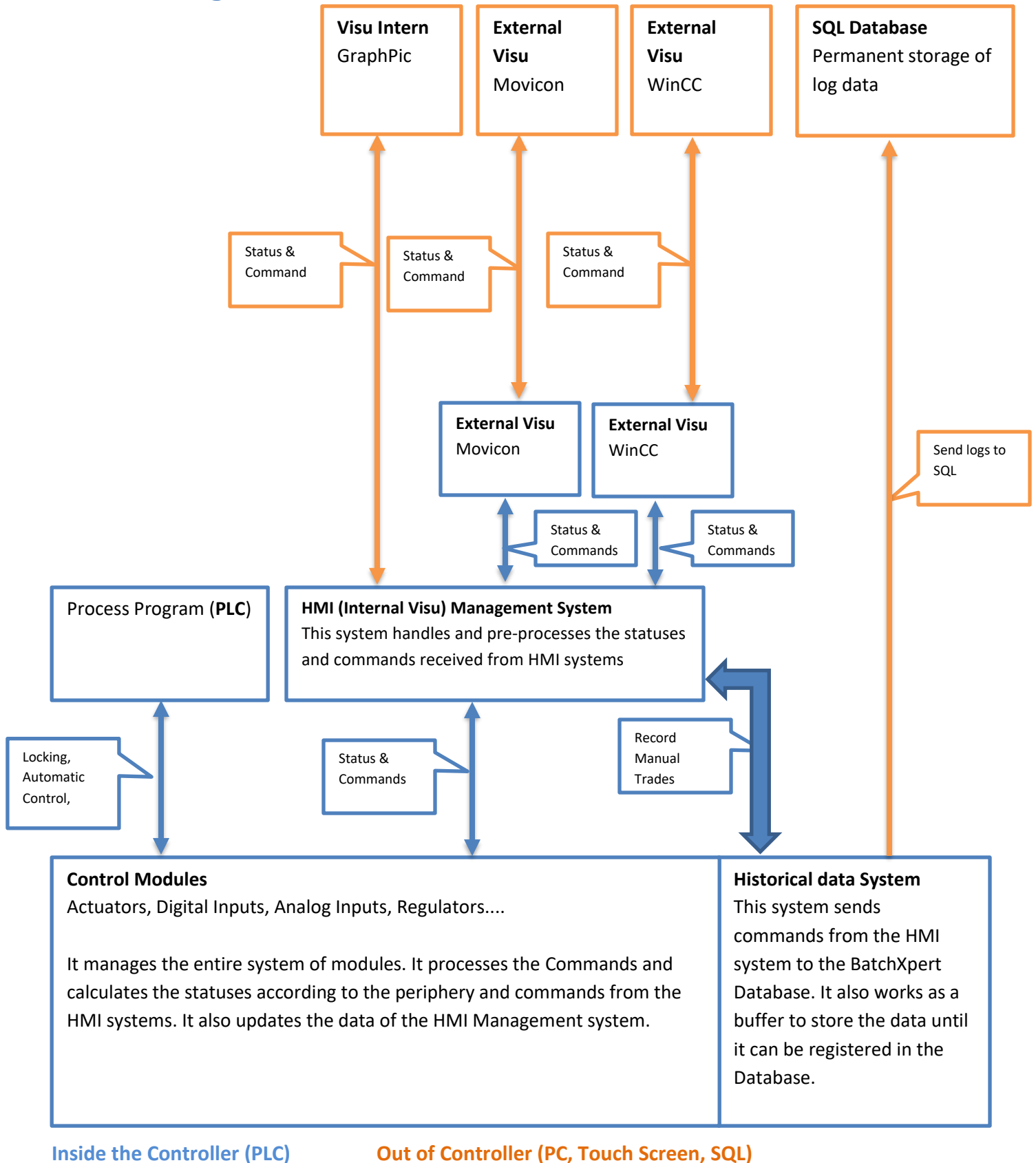
Regardless of the type of Visualization (Internal or External), the system manages and records all the actions performed to some control module, in such a way that there are records in the system's databases. This means that there are also Records on Manual Operations not only of the Internal View, but also of all the External Views added to the system.

Fundamentally, these "External HMI" blocks depend on your symbol library for the Hmi systems that you use. BatchXpert by default comes with libraries for "WinCC Flexible," "WinCC Comfort" and Movicon. Since WinCC and Movicon have different requirements for the data exchange, they both come with their own "External Visu" block, that can be used.

Disadvantages of using HMI Panels

If you are using external HMI systems, you must manage each HMI system separately, which generally means that you must implement your HMI screens in all your HMI systems. This is the reason why this concept is generally discouraged in favor of VisXpert SCADA, since this allows you to manage one single SCADA project for all your touch panel applications and even operating stations.

Functional Diagram



Inside the Controller (PLC)

Out of Controller (PC, Touch Screen, SQL)

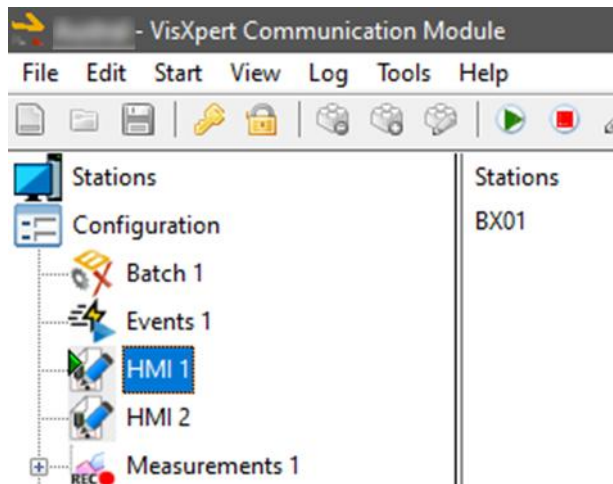
Process Graphics with VisXpert

VisXpert is the basis for all the HMI graphics of BatchXpert. You can find manuals in the MLogics Knowledgebase with more details. In this manual we only want to touch the BatchXpert specific parts of creating a process screen for your application and tries to give you an overview of the general functionality and how you can achieve common use cases with this expert graphics editor.

Here are some resources you should read first, or be already familiar with:

- Mlogics Knowledge base: <https://docu.mlogics-automation.com/>
- VisXpert Graphics Editor: <https://docu.mlogics-automation.com/overview-1/visualization/visualization-editor/create-a-process-image/>

Communication Module



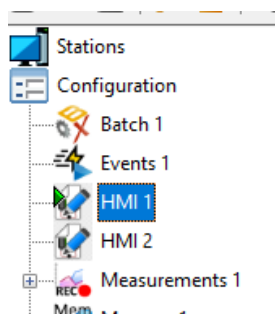
This expert is a modular SCADA system that implements all its functionality through the implementation of different modules. Each module serves a different purpose, but all modules are managed by previous expert communication module.

The communication module is also responsible for managing all the different variables, also called Tags, of your BatchXpert application. This means that the communication module is the application that manages all the data of internal variables, and the data from external OPC-servers or PLCs.

Each of the modules has a runtime and a design time editor, which allows you to adjust your configuration of your module and then execute your configuration during runtime. An example of such a module is the HMI module or the trend recording module.

VisXpert Also includes a rich programming interface, in which you can implement custom modules in C#, C++ or Delphi. You can find more information in our knowledge base, or in the installation directory of this expert, where you can find programming examples for the “.net” platform.

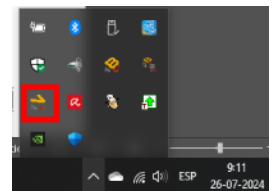
HMI Graphics Editor



The Graphics editor is available in the VisXpert Communication module in the “Configuration section”. It can only be activated if you have sufficient user access rights.

To open the configuration section of VisXpert, you may have to bring the “communication module” window into the foreground. Usually, this window will be minimized when starting the HDMI application and is not visible to the user. When the communication module is minimized, it shows an icon in the windows “SysTray”, which is the icon area

left to where the windows clock is located. If you open this “SysTray”, you can bring the communication model to the foreground by clicking on the VisXpert icon.

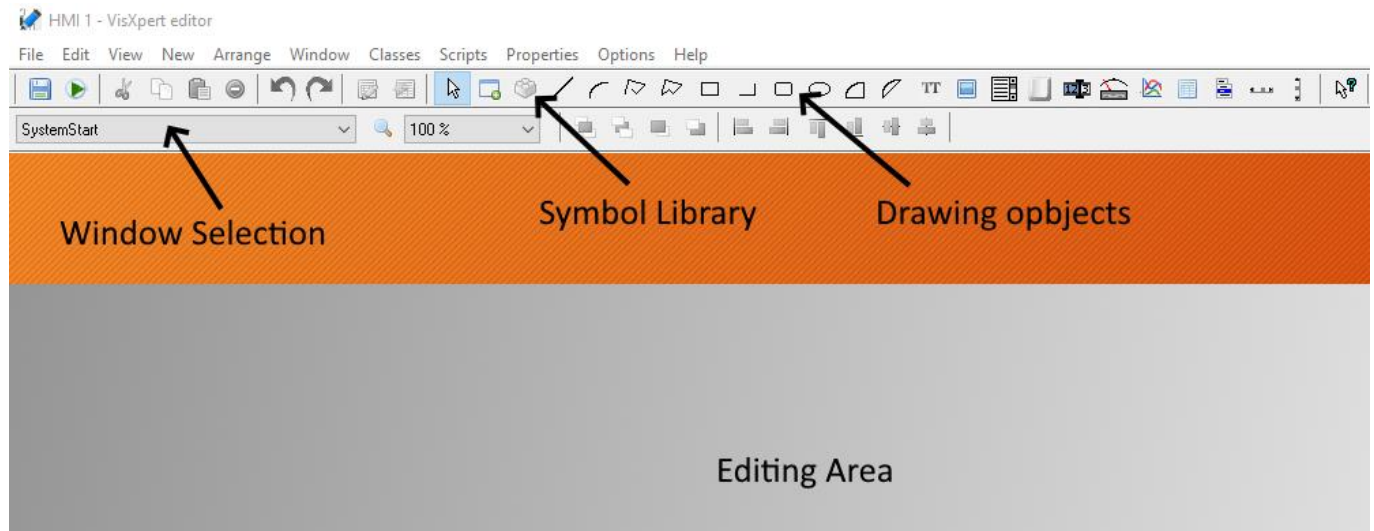


When opening the configuration three in this expert communication module, you might encounter multiple HMI applications. The reason for this is that BatchXpert supports multiple monitors, where each monitor corresponds to one HMI application. Usually you want to use the “multimon” this expert module to manage HMI applications. This module will synchronize the different HDMI application between each of the modules come up which means you should only ever edit and modify an HMI application called “HMI 1”. All other HDMI applications will be synchronized with this HMI application.

If you have sufficient user rights, you can double click on the HMI application to start the graphics editor.

The Graphics editor

The Graphics editor features a drop-down selection where you select the window that you are currently editing.



You select the window that you want to edit from the window selection dropdown menu and can then select, edit or add new drawing objects or symbols to the process screens.

All graphical screens and face plates are implemented in this graphics editor, which also means that they can be edited to fit your needs. However, some advanced dialogues, that heavily depend on data from BatchXpert database, are implemented outside of the graphics editor as in separate application to provide a better user experience.

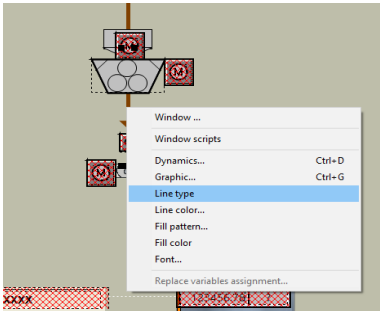
Images and Bitmaps

Images and bitmaps are an essential part of all Process images, and the reason the HMI can create modern looking process screens. BatchXpert includes a library for process screens as part of the “BatchXpert SDK” in the directory: “C:\Program Files (x86)\BatchXpert SDK\Visu\Images”

VisXpert supports a variety of image types, which includes “PNG” including transparency. **Since PNG files are the only image type that supports true transparency, we recommend that you use these image files format.** 32-bit BMP files are also supported (8 bit per color channel and 8 bits for Alpha), which also support transparency, but not all image processing applications support these image formats, which makes it harder to edit.

The image library still includes a lot of images as bmp files with static-colored backgrounds, for which we recommend converting them to PNG files with true transparency before using them.

Graphics and Dynamics properties



The graphics editor allows you to modify basic object parameters by right clicking on the respective item. The context menu gives you access to the most basic appearance properties such as line type, line color or fill colors.

This graphic properties are static properties and only define the basic appearance of all the drawing objects. To animate these properties, you must use “Dynamics”.

The "graphic" menu option on the context menu gives you access to element specific graphical options such as text alignment tool tips value formats and such. Most notably you can adjust many text properties like text layout text rotation and such for text objects and change the graphics file used for image object.

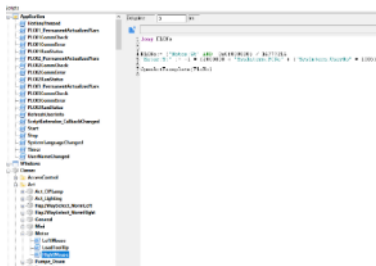
Dynamics

Dynamics is the term used in this expert for the concept of linking variables to graphic objects to animate them, change colors, change text or changed their appearance.

The “dynamics” Menu option on the context menu allows you to adjust dynamic animations for a different graphical property of these drawing objects. This allows you for example to change the background color depending on the value of a variable, animate the text property of a graphics object to reflect the current value of and variable, or even execute scripts and other actions when being clicked.

You can only change graphical properties for simple objects, not for symbols from the library. The graphical properties of symbols, called “Objects”, must be changed in the “Object” editor that is also available inside the graphics editor.

Scripts



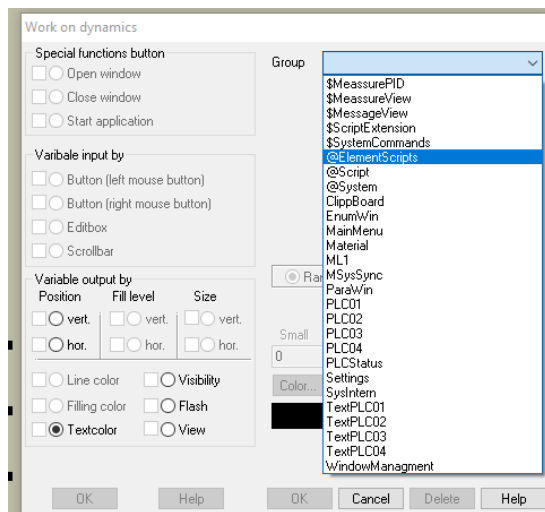
VisXpert supports a powerful scripting language. This scripting language is extensively used in the HMI symbol library of BatchXpert.

You can find mor information here: <https://docu.mlogics-automation.com/overview-1/visualization/visualization-script/>

The Scripting language uses an “pascal” like syntax and includes the following features:

- High level, imperative language
- Database access using queries for reading and modifying data
- Manipulate Graphics properties of Drawing objects
- Access to variables of the PLC
- File access
- String manipulation, trigonometry, and other functionality

Element Scripts



The graphics editor allows you to create powerful customized animations based on the scripting language. With this scripting language you can implement most of your animation requirements that cannot be met by the normal dynamics functionality.

To use element scripts, you must open the dynamics dialog select the property that you want to animate (in the example provided it is the text color), and then select “@ElementScripts” from the groups drop down menu.

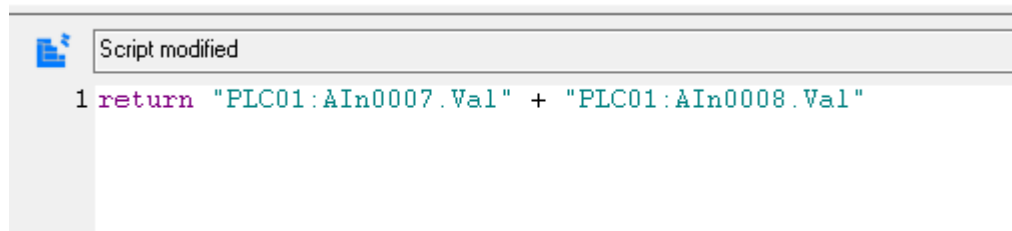
This allows you to write a customized script which calculates the value for the property that you want to animate come up based on the result of your script. This script can access any variables,

functions, databases, or any file you need.

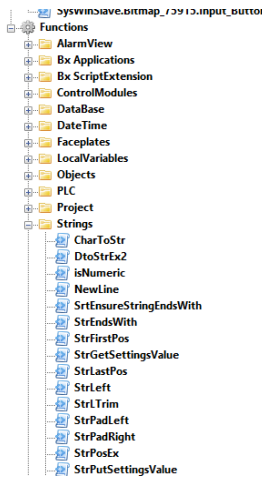
If you click on the “Switch to Script dialog” button, the full scripting interface will open, and you can write your script in the usual scripting dialog.

Usually, you use this functionality to animate drawing object properties based on sometimes complex calculations, for example: showing different colors depending on complex ranges of values of multiple analog inputs. Or showing customized texts for different values or value ranges.

Example you can see that do you result is being calculated by taking two multiple analog input values which are both different variables, doing a calculation on them and returning the result to be shown on the HMI.



Script Functions



the graphics editor allows you to implement commonly used scripts into script functions. These Script functions can then be used in all other scripts, such as element scripts, object scripts and even in other script functions. By default, the BatchXpert application comes with a vast library of script functions that cover multiple use cases and extend the functionality of the scripting language itself.

It implements an advanced string processing library that enhances the string processing capabilities of your HMI applications period. You can also implement your own functions to realize project specific functionality and centralized commonly used code in one single script instead of copying and pasting scripting code into multiple objects. All provided script functions include and header with a detailed description about what the script function does and how to use it.

We strongly recommend you implement common scripts into script functions, so that you can manage them in one single place but still have them available in all other scripts. We also recommend you include a script header with a detailed description about your script function for documentation.

BatchXpert Process Screens

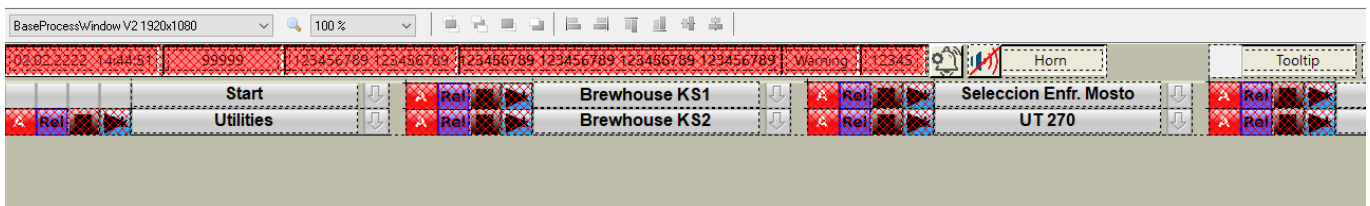


When the BatchXpert HMI is started, first it will open the Window “StartPage”, which is the loading screen of BatchXpert. After loading finished, it will open the “BaseProcessWindow V2 1920x1080” window.

This window forms the basis on top of which all the process windows will be opened. This “BaseProcessWindow V2 1920x1080” always stays open in the background, so the menu bar and all system components will always be visible.

The Base Process Window will hold all controls and drawing objects that are always visible, such as the alarm bar, the Main menu bar, and other controls to control “Expert Mode”, “tooltips” and the current Time. This base window should never be closed during the runtime of your HMI application, as it also runs some scripts in the background that provide additional functionality for the whole application

The Menu bar is also an essential part of this base window which is implemented by an “VisXpert object” that must be configured from scripts. More details later. The Main menu provides access to all your process areas by dispatching to the corresponding “overview screens” and allowing direct access to all individual windows through the drop-down menu for each area. More information can be found in the [“The HMI Main Menu”](#) chapter.

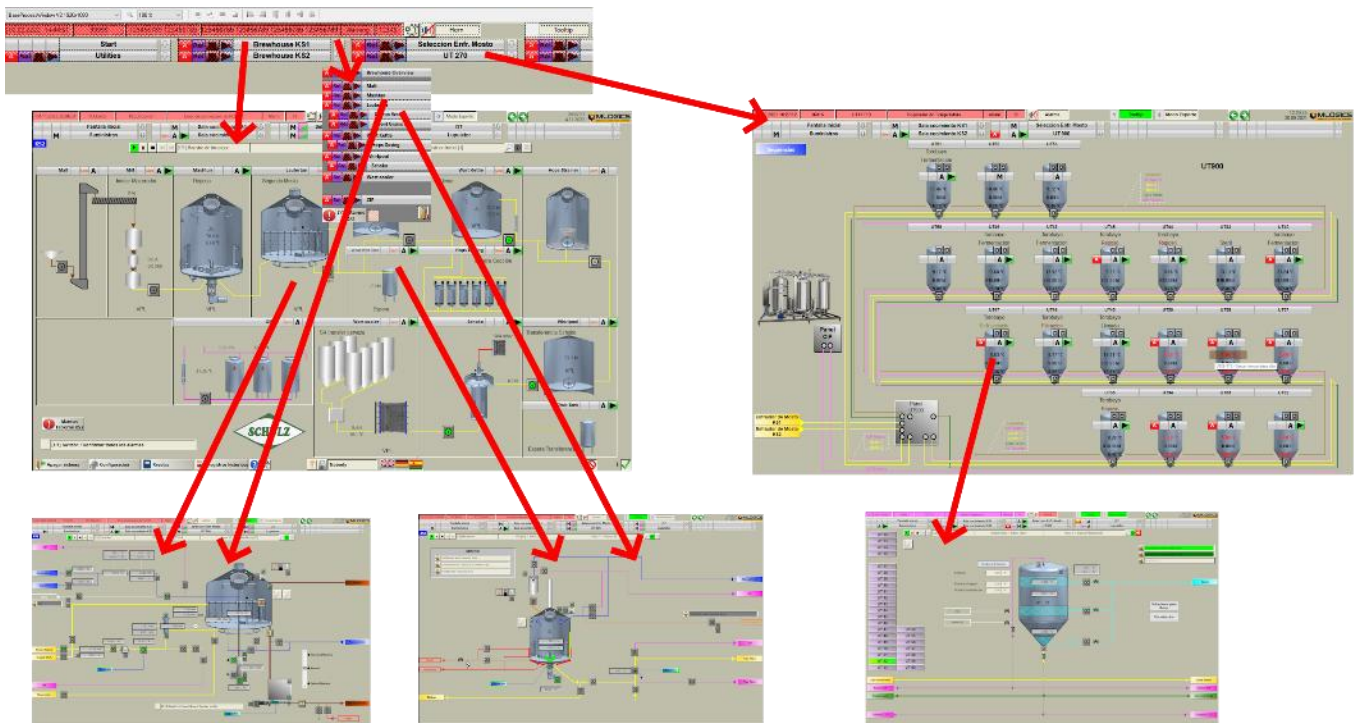


Process Screen Hierarchy

The Main Menu visible on the “BaseProcessWindow V2 1920x1080” Window, is the essential menu that allows the user to dispatch to all process screens. The “Main Button” dispatches to the Process overview, and the drop-down menu gives access to all individual process images of each area.

Each menu area is reserved for a single “Processing Area” such as “Utilities”, “Brewhouse”, “Fermenting”, “Maturation”, etc. The first menu item is reserved for the “Start” menu, which provides access to all BatchXpert applications and serves as main menu for BatchXpert itself.

All process areas are organized multi-level process image hierarchy. On the top level there is his start page which shows information about the connection status and alarm status of all process areas, below that you can access an overview picture for each of the process areas that are available from the main menu, and below that are process images with details for each processing unit. The main menu allows you to access these different process window hierarchies by using drop down menus that allow you to open do you required unit detail process windows directly from the main menu.



Drawing Process Screens

Position and Size			
left	0	width	1920
top	85	height	995

position of process windows should be as shown in the image.

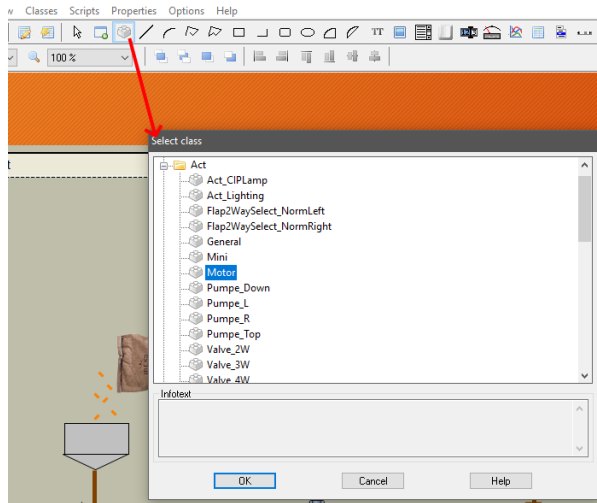
Since the process screens are opened “on top” of the “Base Process Screen” they must have a smaller size and position offset to the “base process window”. By default, the size and

Water
Malt and Grain
Steam and Heating agents
Wort, Beer or main Product
CIP general
CO2, Air and gases
Glycol and cooling agents
Acid
Caustic
Regulation Dependency
Amoniak

The Process Screens can be created by applying Screen layouts, coloring etc. as the client wishes. BatchXpert provides an “Standard” for coloring of process lines, and an image library for implementation in the process screens, which are available in the “Library” process Screen in the HMI template.

All Control modules are represented by “Objects” which encapsulate all functionality of a module and can be easily configured by assigning variables to them, more on their configuration below.

The Symbol Library



BatchXpert implements its symbol library by using “VisXpert Objects”. These objects are self-contained drawing objects that contain dynamics, scripts, and all graphics components of a control module.

They are also fully integrated into the BatchXpert HMI system and incorporate features like Opening Faceplates, Sending commands to the PLC, etc.

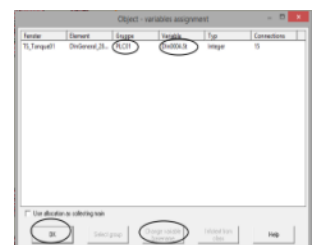
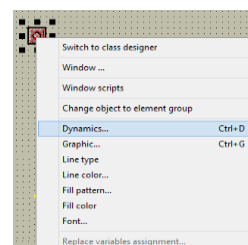
When you add a Symbol to the Process graphics, you must specify its data connection. For example, for actuators, you would have to specify on which Actuator Variable will operate.

Configuring BatchXpert Symbols

In the project images the objects have a PLC variable assignment, right-clicking on the object will open a menu where we will select “Dynamics” where you must select the group and the variable, example, group: PLC01, Variable: DIn0004.

To edit the existing variable, click on the variable and click on “change variable base name”.

Once all the image configurations are finished, save, and start the simulation to verify that everything is correct.

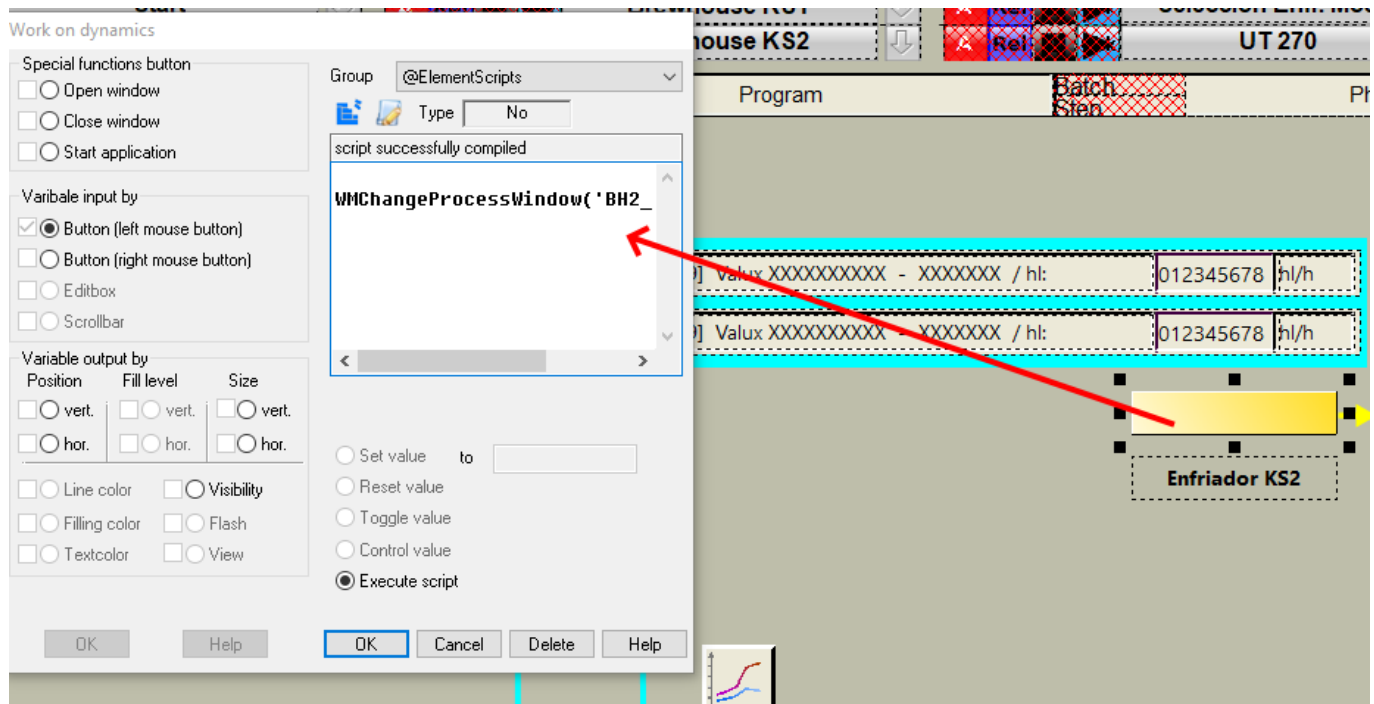


Change Process Windows

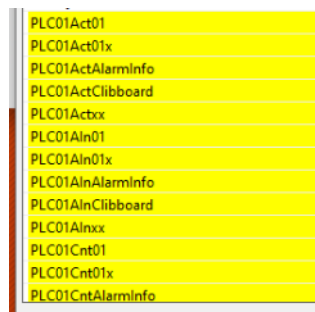
To create buttons that switch the visible process window, you should create a script action on a button or image and call the “WMChangeProcessWindow(string windowName)” function from the provided scripting library.

You should avoid using the open and close window functionality of buttons or other objects, because the window manager script functions provide additional functionality for the HMI system, such as a previous image buffer, and special handling for face plates and modal dialogues. BatchXpert provides multiple window manager script functions that you can use for different purposes throughout your project.

The buttons on your process screens that allow the user to change process screens are usually composed of and a bitmap of one of the prepared button colors with the appropriate text overlaid on top of this bitmap. The change process windows script is then programmed in the button left mouse click event of the bitmap, as seen in the image below.



Remove unused “Windows”



If you copy a project from a different project, you should remove all screens that are not actively in use. This ensures that you do not have references to variables that you are not using anymore and reduce the size of the HMI Project file.

Especially you should remove all “PLC0x” windows, where x corresponds to an PLC number that you are not using. All these windows can easily be imported again from the HMI library as described in chapter [Importing the PLC dependent Windows from your library](#)

Process Graphics Style Guide

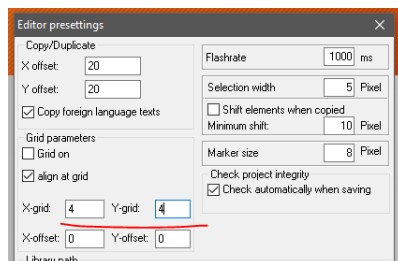
The following chapter we are going to present our recommendations for styling process graphics in the BatchXpert System. Of course, you can adapt this guides to fit your company and client preferences, the following are only guides after all.

High Performance HMI Handbook

We recommend that you read the “High Performance HMI Handbook” by “Bill Hollifield”, which contains a lot of great advice and guidance for creating HMI interfaces for all sort of Processes, vehicles and Machines. BatchXpert adopts some aspects of these guidelines but does interpret others more loosely. The Color scheme used by BatchXpert uses many more colors than is recommended. However, you should adhere to the following general rules:

- Do not use “Flashing” or “Blinking” animations, except for exceptionally critical indications such as things that may damage personal or equipment. Normal Process alarms do not fall into this category and should not be made “Blinking”. In practice, BatchXpert never uses blinking animations in any of its defaults Objects and screens.
- Red is reserved for “Alarms”, and should never be used in any process screen, except for Alarms
- Flow diagrams should reflect the logical flow, and not be an exact representation of the P&ID. You should omit non necessary symbols such as, manual valves, check valves, filters etc., wherever not relevant.
- Avoid Moving Animations. You should not use animations that “move” objects, such as “Rotating Agitators” or such.

Preferred Editor Settings



All BatchXpert Objects are designed to be 36x36 pixels in height and width, which makes the “Centerlines” align exactly to grid points of 4 x 4. This way you can position all your objects precisely in the middle of lines.

In the Graphics editor you can open “Options-> Editor” you can set these grid size and activate “Align at Grid”, so that objects automatically align to the nearest multiple of your Grid settings. We recommend the settings on the left.

Pipes

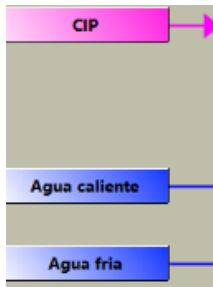
	Water
	Malt and Grain
	Steam and Heating agents
	Wort, Beer or main Product
	CIP general
	CO2, Air and gases
	Glycol and cooling agents
	Acid
	Caustic
	Regulation Dependency
	Amoniak

The following process colors are recommendations, but not mandatory to be used for your process graphics. Of course, this is only a guideline and you can adapt any coloring scheme you wish for in your projects.

You should differentiate between Primary product flow and secondary flow by using different line thicknesses. Usually, the primary product flow has a line thickness of 3, and all secondary product flow have a line thickness of 1 pixel. Secondary product lines should never interrupt primary product lines, neither horizontally nor vertically. Primary Product lines should clearly denote the main product flow.

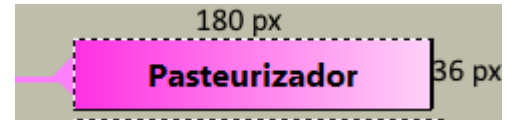
You should connect pipes in 90° angles with each other where possible and avoid using arcs or bends to “simulate Pipe bends”. Use simple 90° angels to create easy to follow diagrams, that are similar but simpler than P&ID drawings.

Process interconnections (Window Jumps)



Connections to different process screens, meaning pipes that come and go from/to different parts of the process that are represented on other Process screens, are represented by an “Page Jump”, that is represented by an “Color graded Rectangle” in the same color as the pipe. Usually, these page jump indicators should have an accompanying “Arrowhead” to indicate flow direction.

These page jump indicators are implemented as Bitmaps, which are available in multiple colors to fit the piping that is indicated. The size of these page jumps should be 180 x 36 pixels. The bitmap itself should then hold the Script logic to “[Change Process Windows](#)”.



Equipment representations

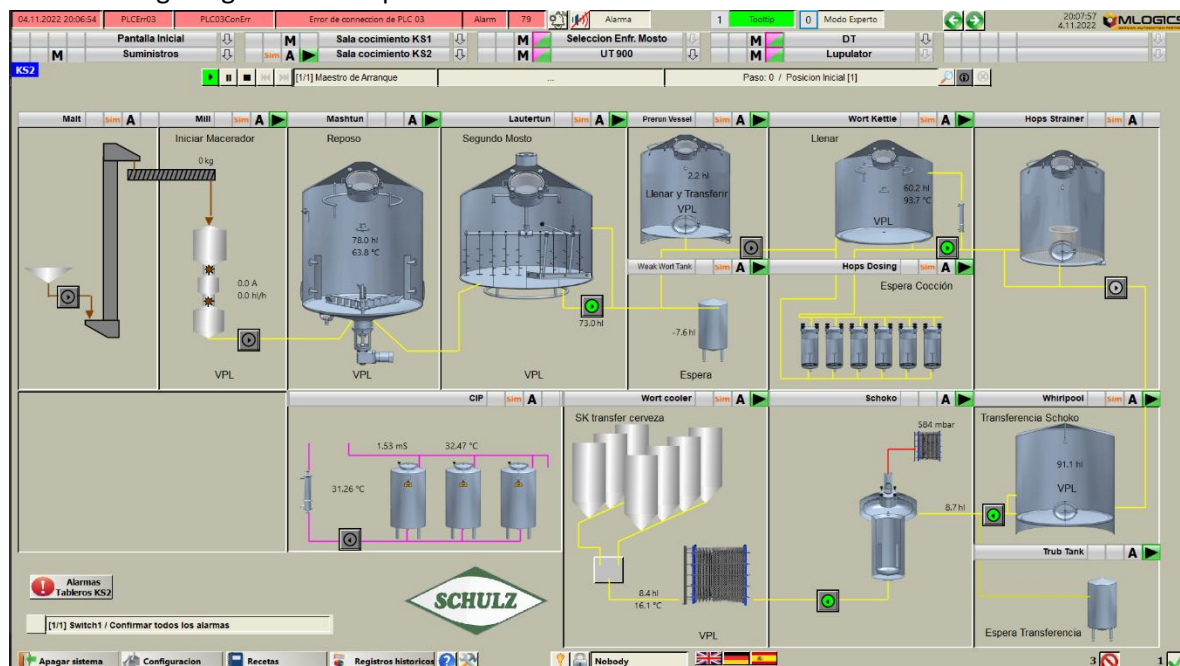
To represent different equipment types such as Lauter tun, Mash tun or Heat exchangers, you should use representative bitmaps or PNG images. These images Should ideally be in muted pastel colors or ideally in grayscale colors, as to not distract the user’s attention away from important information.

Overview Screens

Each processing area should have one overview image, which represents the overall process layout of this area, with as little details as possible, while maintaining the most important control modules. This is usually done by representing a small pictogram off the production vessel, and only showing vague schematics for production lines and only representing the most important control modules, such as the main product pump or main product valves, and omitting all CIP, water and other supply lines.

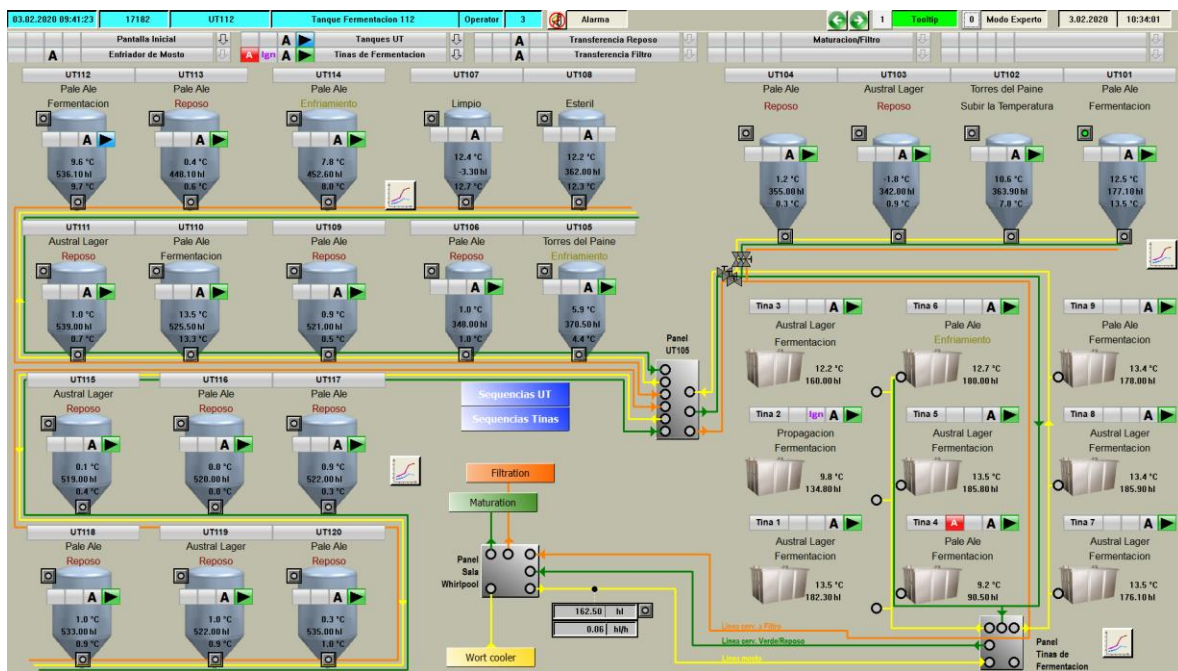
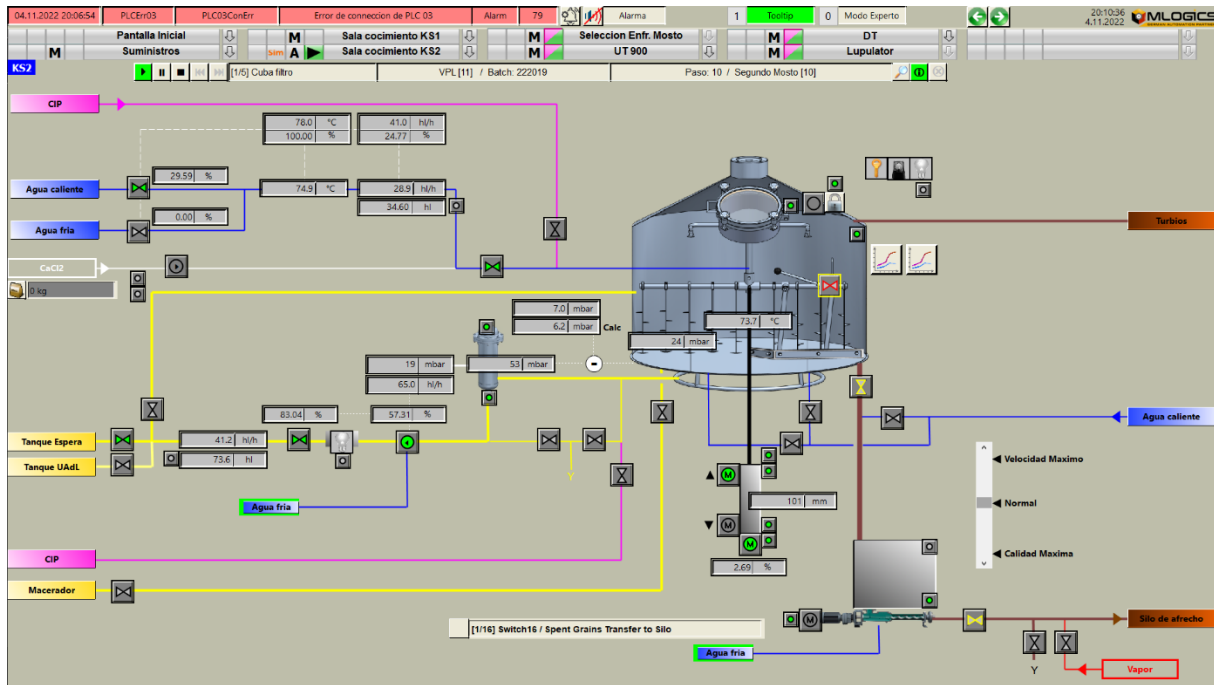
The idea of such an overview image is to give an quick overview and easy to read information to the user. This way the user can easily and rapidly understand the overall status of the whole processing area. All relevant processing steps should also visualize the current operating procedure and recipe, as the rapidly allow user to determine what each equipment is doing in the process.

The following image is an example of an overview of a brewhouse:



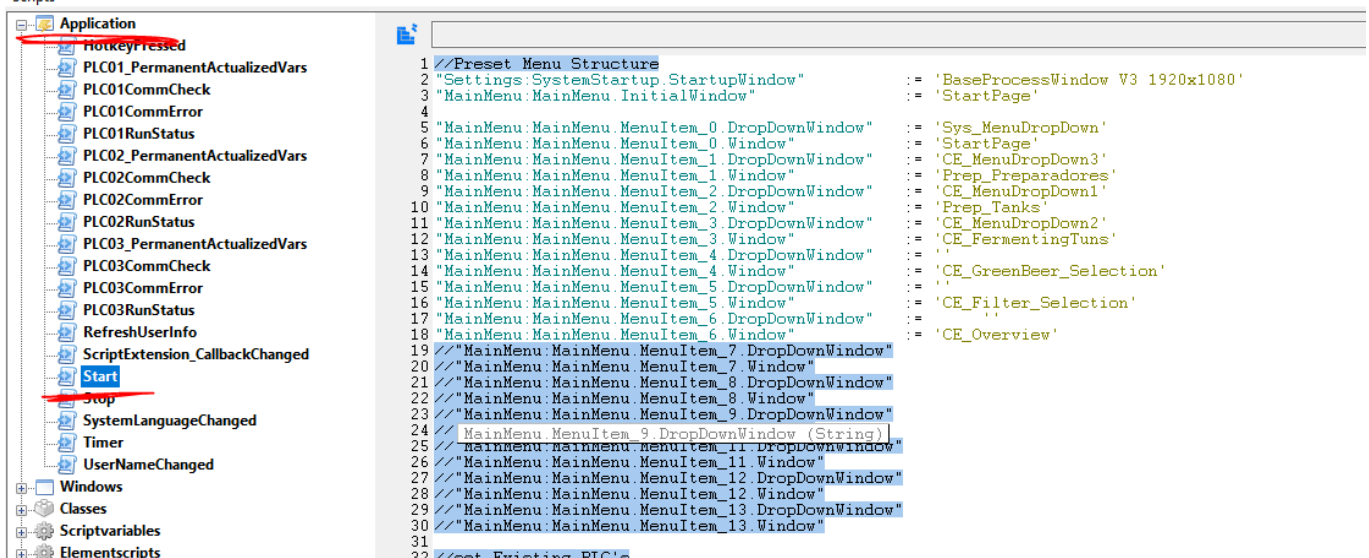
Example screens

The following screens are example screens that represent typical processing graphics of BatchXpert. Of course, you can deviate from this style guide to fit your company's and the clients requirements.



Application Start Script

The HMI Application of your Project includes an “Application. Start” script, that runs as soon as the HMI runtime is started up. This script is used to configure and set up multiple global variables such as the Main menu and optionally synchronize the Operating stations time with the PLC time.



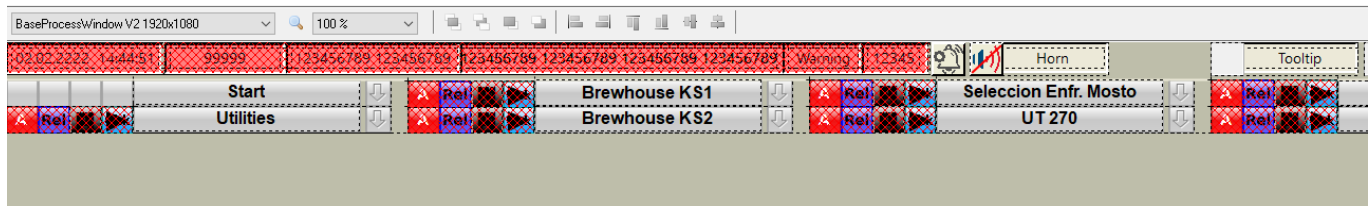
The Application Start script will perform the following actions:

- Configure the Main Menu
- Optionally set the PLC time
- Set the Application Startup language for the HMI according to the project settings
- Optionally block access to the operating system
- Reset PLC system alarms, that may have been “stuck”

You can easily add your custom startup logic here. You can find a lot of comments in the Application start.

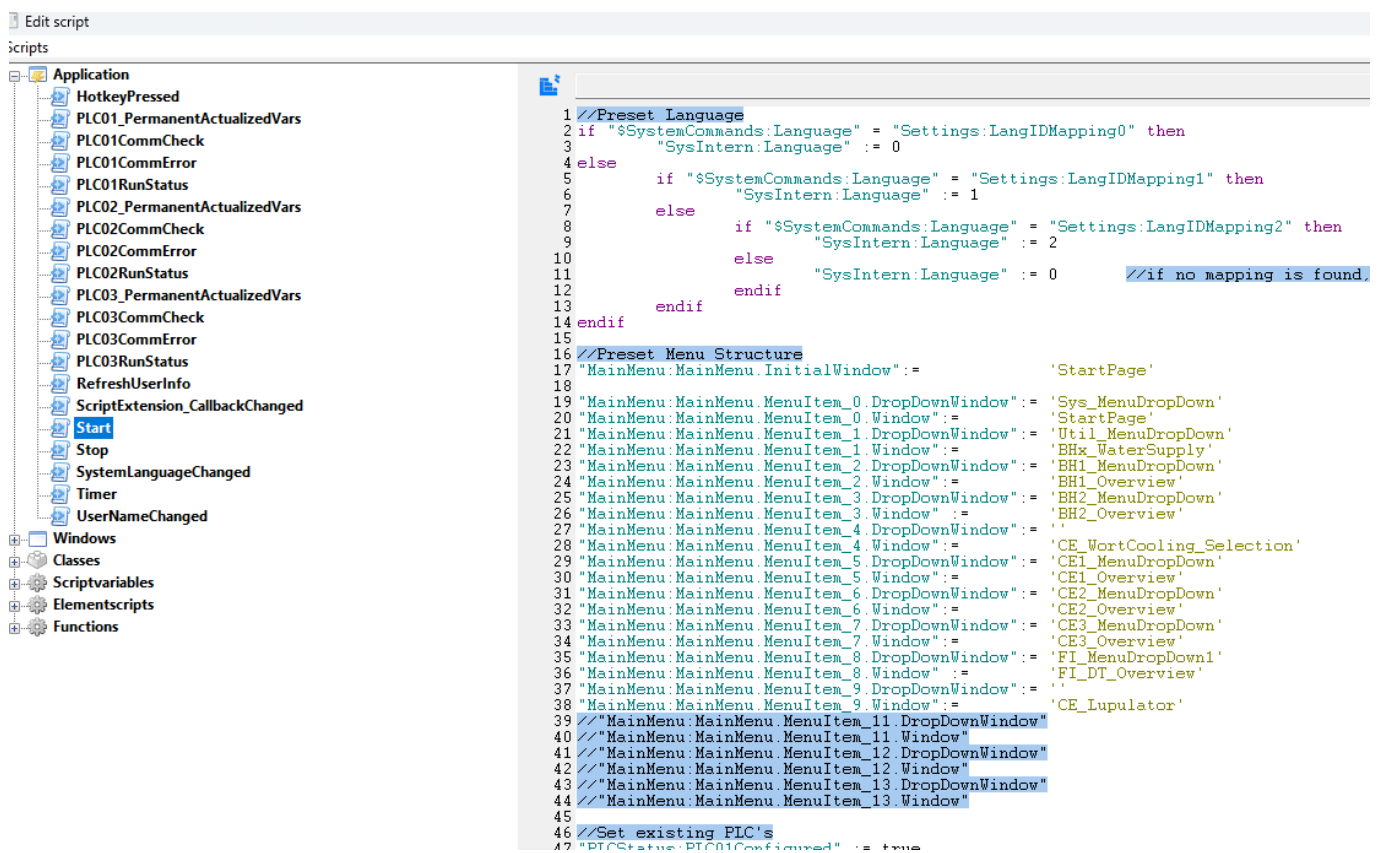
The HMI Main Menu

The HMI Main Menu is located on the “BaseProcessWindow V2 1920x1080” or “BaseProcessWindow V3 1920x1080”. This menu allows the user to quickly and efficiently move to different HMI screens of the plant. The main menu is implemented as a library object, and it's located on the base window of all process screens. The main menu has an area where you put an alarm group, a main button and an optional dropdown button. Usually there is only one single menu per plant, which means that the menu stays the same for all process screens.



Configuration of these Buttons

The menu is a library object, and all its buttons' functionalities are configured through the variables assigned to the library object. The Variables are usually contained in Group “MainMenu”. You can define an “Main Window” and an “Drop Down” window, which opens when clicking on the small “arrow” on the right side. The configurations of the windows that should be linked to the main menu is usually done in the “Application.Start” script.



Main button Text and Translations

The text of each main button is configured directly in the HMI editor on the “BaseProcessWindow. The Main button Text is a simple text on top of the “Main Menu” library object. This means that the normal Translation mechanism of VisXpert applies to these texts as well.

Alarm Group

Each button usually also includes an Alarm Group, that shows the process of the underlying Area with colors and pictograms. These Alarm groups are also normal library objects and can be configured the same way as ordinary objects, by assigning a Variable name and Group to them.

Drop Down Windows

Drop down windows are small windows that pop up and show all underlying process pictures that can be navigated to. The drop-down window in and of itself is an ordinary window and can be configured like any other window and assigned to the main menu from the application scripts. The Drop-down button includes a special functionality to always move the Drop-down window to where the Drop-down button was clicked, to appear as a drop-down window. This also means that the configured location of the drop-down windows is not required and will always be overridden when a drop-down button is clicked. This also means that one drop down window can be used in multiple drop-down buttons.

Configure communication with the PLC

All variables that are being exchanged with a connected BatchXpert PLC, are managed by the “PLC Data” module, which is accessible from the “VisXpert communication module”. Double clicking on this module, you will give access to the “VisXpert variable editor”, which allows you to change the configuration of all configured variables in your project. It allows you to add new variables, or change the configuration of existing variables, and the communication settings for all connected PLCs.

Each PLC is represented by its own group which follows the PLCxx naming scheme. If you want to change the settings for PLC01, you must select the corresponding group from the group list and then edit its communication parameters.

For specific Settings required in a S7-1500 and S7-1200 PLC, please refer to [Connecting to an S7-1500 and S7-1200 PLC](#).

Memory Variables

The VisXpert variable editor also manages internal memory variables that are not connected to any external data source such as OPC servers or PLC's. This means that you can also add custom memory variables if you should need them in your application. To add custom memory variables, you can add a new group and select the memory module for its data driver.

Memory variables only exists on the local station and are not synchronized between all operating stations. This means that memory variables always have and local scope and should be treated as local to the operating station. Each operating station will have individual values on these memory variables, since they are never synchronized between operating stations. Should you need custom variables that are shared between operating stations, you should share them using and data area in your PLC.

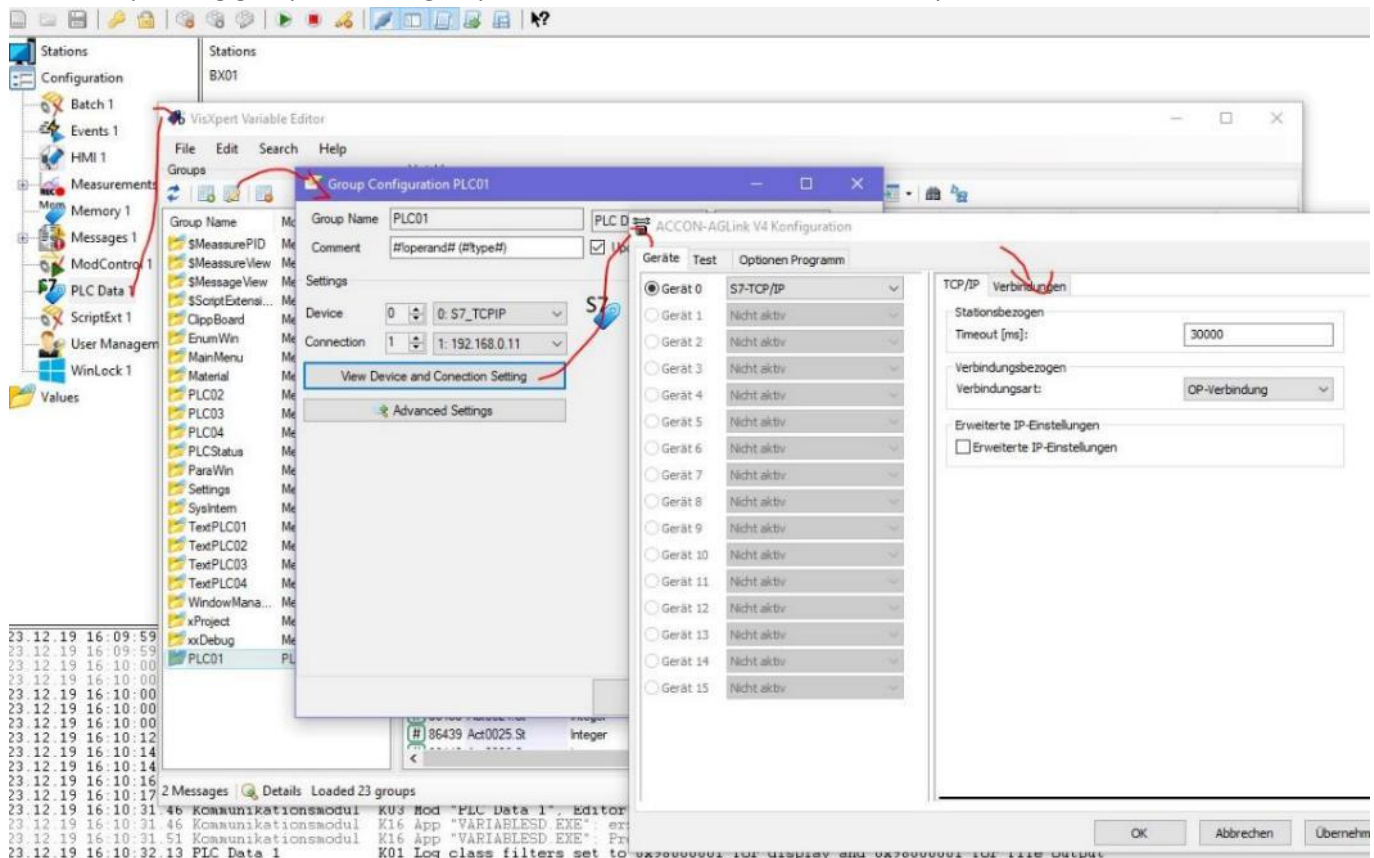
Custom Variables in the PLC

If you want to add custom variables that are communicated with any of the PLCs of your BatchXpert system, you can do so by adding your custom variables and their address configuration to one of the existing PLCxx variable groups. You must input the data address of your custom variables in the standard Simatic addressing format.

Group	Variable	Address	Variable	Address	Variable	Address	Variable	Address
0/1	84590 WinActPC1Open	Boolean	DB 5.DBX 5.0 BOOL TB 1	BOOL	1	cyclic	readwri	
0/1	84589 WinAlnPC1Open	Boolean	DB 5.DBX 5.2 BOOL TB 1	BOOL	1	cyclic	readwri	
#	94539 WinAlarmAreas	Integer	DB 5.DBX 100 DINT TG 4	DINT	4	cyclic	readwri	
0/1	84591 WinCntPC1Open	Boolean	DB 5.DBX 5.5 BOOL TB 1	BOOL	1	cyclic	readwri	
0/1	84592 WinDInPC1Open	Boolean	DB 5.DBX 5.1 BOOL TB 1	BOOL	1	cyclic	readwri	
0/1	84593 WinMessagPC1Open	Boolean	DB 5.DBX 5.4 BOOL TB 1	BOOL	1	cyclic	readwri	

S7 Communications Settings

The S7-Connection require the Slot and Rack communication god parameters, which should be defined according to which PLC you are trying to communicate with. If you want to change the settings for PLC01, you must select the corresponding group from the group list and then edit its communication parameters.



AG-Number: The AG-number, or PLC number, refers to and running number of your PC connection. This should be the number of you PLC as configured in your “Batch Configuration”. for PLC01 it should be 1, for PLC02 it should be 2 and so on.

AG-Nr	IP-Adresse	Rack	Slot	Typ
1	192.168.0.11	0	2	S7-300/400

IP Address: The IP address should be the address of the specific network adapter used by your PLC to connect to the project expert station. This is usually the IP address of your internal CPU, or of your external communication processor, it’s configured in your hardware configuration of your PLC program.

Rack and Slot: Rack and slot off your PLC depends on your specific hardware configuration, but usually follows this table:

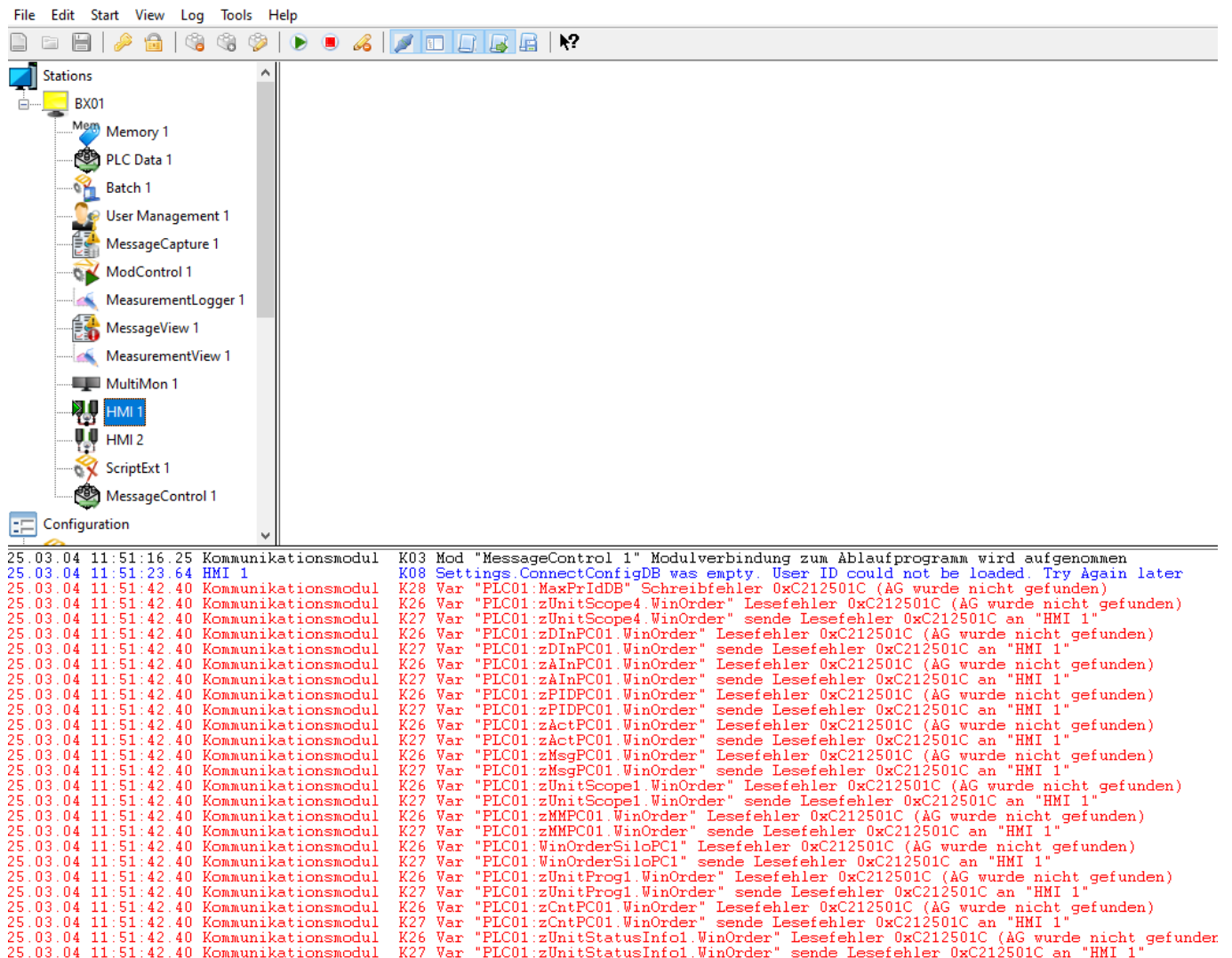
S7-300 series	Rack = 2, Slot = 0
S7-400 Series	Rack = 3 (usually), Slot = 4. But both parameters depend on your specific configuration
S7-1500 Series	Rack = 0, Slot = 0
S7-1200 Series	Rack = 0, Slot = 0

Type: The type can usually be left at 300/400 series, but it is recommended to select the correct PLC type you are connecting to.

Diagnosing PLC communication problems

BatchXpert uses the VisXpert driver infrastructure for data exchange between the HMI application and the process controllers. If a VisXpert driver encounters any problem while trying to connect exchanging data with PLC, it will write an appropriate error message into the VisXpert Logging window.

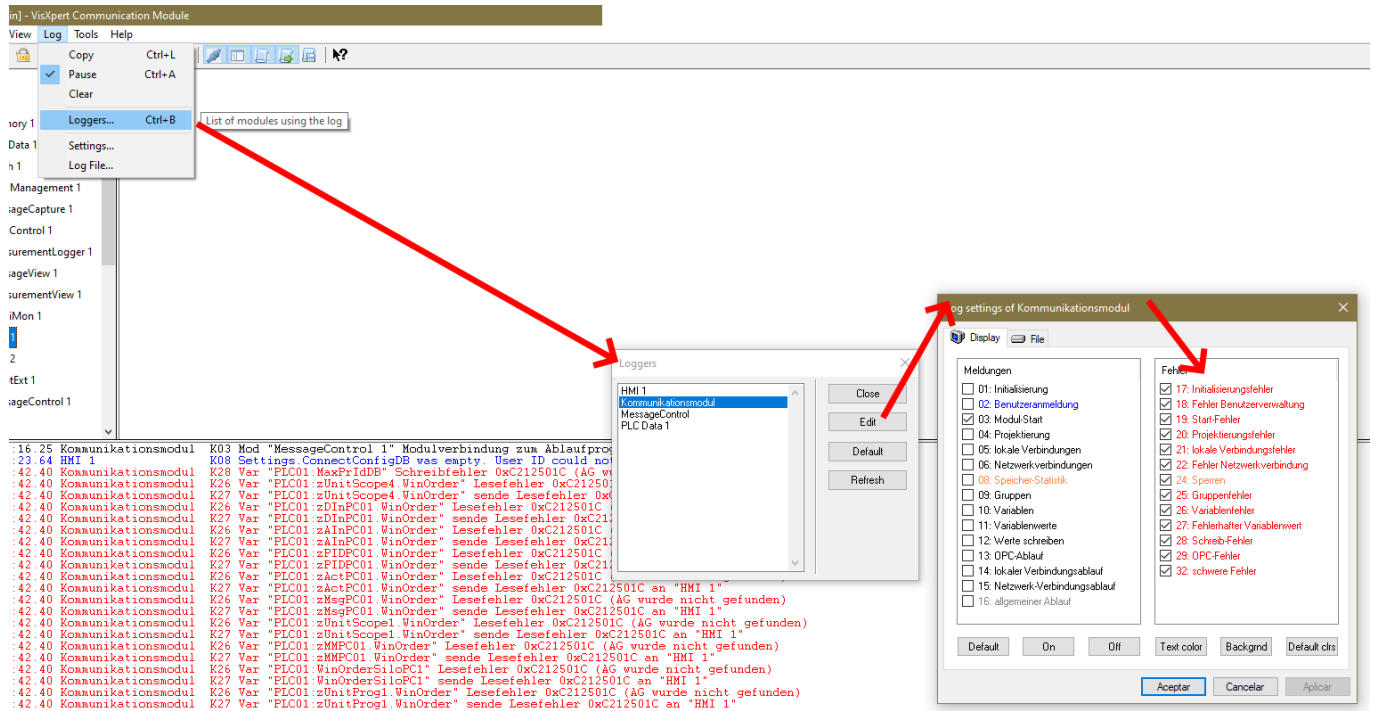
This logging window is part of the VisXpert communication module and should be your first point of reference when you encounter communication errors with the controller.



In the communication module main window, you can see all error messages in the lower part of the application.

Adjusting Log Level

VisXpert Allows you to set different logging levels to show and hide specific error messages in the log window. You can customize these log levels from the main menu of the communication module as shown in the picture below.



Typical Error Messages

All error messages are dependent on the driver used, but for S7 controllers you can observe the following error messages. The error code returned from the controller can be seen between the parentheses of the error message of the variable that failed to connect.

- **“AG Wurde nicht gefunden”**: The PLC could not be connected. Something blocks communication; the IP address might be wrong or the PLC is blocking communication.
- **“DB existiert nicht”**: The data block you are trying to read or write from/to is not loaded in the PLC. Please check your PLC program and our driver variable configuration.
- **“Access denied”**: the PLC actively rejected the communication request. Maybe there are some settings that are not correctly set, a Password was set in the PLC or some firewall is rejecting connections.
- **“Unspezifized Error while initializing the driver”**: this error can happen if there is an error in your VisXpert. You should reinstall VisXpert with the “Reinstall Driver” setting from the installer.

Endianness

One important characteristic of Simatic 7 controllers is that these types of controllers use an CPU with Big endian architecture. This contrasts with most computer systems which are based on the X86 CPU architecture, and which uses little endian.

The Endianness of CPU architecture defines how integers are stored in memory and thus also defines how the memory of these integers is laid out. This becomes important when you are transferring data between two different CPU architectures with different endianness, such as when you are reading data from a Simatic 7 CPU on a SCADA operating station. You must keep the endianness in mind since the data layout of status words from control modules individualization data blocks has to be adapted accordingly on the SCADA systems.

In addition to this you must keep in mind that schematic PLCs implement a byte-oriented memory model whereas modern SCADA systems implement a word or double word-oriented memory model. For example, what would be Byte 1.2 would become bit 18 on the SCADA system, since the endianness is going to be corrected by the corresponding IO communication driver.

Batch expert uses double integers (int32) as their status words for all control modules, which means that all statuses from the PLC must be converted by the following table on and Scada station:

SCADA bit number	Address in PLC
24	0.0
25	0.1
26	0.2
27	0.3
28	0.4
29	0.5
30	0.6
31	0.7
16	1.0
17	1.1
18	1.2
19	1.3
20	1.4
21	1.5
22	1.6
23	1.7
8	2.0
9	2.1
10	2.2
11	2.3
12	2.4
13	2.5
14	2.6
15	2.7
0	3.0
1	3.1
2	3.2
3	3.3
4	3.4
5	3.5
6	3.6
7	3.7

HMI Tag names

In the following description we will refer to the default batch expert take naming scheme as implemented in the VisXpert SCADA system. These tag names define symbolic names for all data that must be communicated between the PLC and the HMI system. You can find a detailed list attached to this manual.

The provided variable list defines the following naming schemes:

Control Module Tags

All variables related to control models are composed of three letter control module short name and four-digit control module number, and dot-separated a post fix denominating its data. For Example:

AIn0001.Sp
AIn0001.St
AIn0001.Val

The Ain denominates an Analog Input, the 0001 denominates Analog Input 1, and the “.Sp”, “.St” and “.Val” the actual datapoint of the module. Other Examples are:

Act1273.St Status of Actuator 1273
Cnt0188.Val Current Value of Counter 188
DIn0327.St Status of Digital Input 327
...

Control Module Post fixes

Post fix	Data Type	Description
.St	Dint (Int32)	Current Status of the Module. This is a “Bitfield” where each bit represents an individual Status of a Control Module. Please check the appropriate section for information about the individual bits of this status.
.Sp	Real	Setpoint. The current Nominal Value for the Control Module. Only applies to Regulators (PID) and Frequency Converters (FConv)
.Val	Real	Current Value. The current value of an analog Measurement, counter, or frequency drive
.Out	Real	Output. The current Analog Output between 0 a 100% of the module. Only applies to Regulators (PID) and FConv

Parameter Channel variables

For easier identification of variables related to parameter channels, all parameter channel variables start with a lowercase “z”, followed by the control module’s short name and the parameter channel number which is PC01.

- zActPC01. Data of Parameter channel 1 for Actuators
- zDInPC01. Data of Parameter channel 1 for Digital Inputs
- zAInPC01. Data of Parameter channel 1 for Analog Inputs

Status Tag Definitions of Control modules

As mentioned in the Endianness section of this manual the addresses from the PLC must be converted to appropriate addresses for the HMI system. For example, the very first address bit 0.0 in the PLC, becomes bit 24 on the SCADA system after the driver uploaded the data and converted it by its endianness. Each individual bit of the control module status has the following meaning:

Actuator

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	ACo	BOOL	automatic control
25	0.1	ExCo	BOOL	extern control
26	0.2	SCS	BOOL	status check start
27	0.3	xFBa1	BOOL	feedback 1
28	0.4	xFBa2	BOOL	feedback 2
29	0.5	Rel	BOOL	release
30	0.6	Rel2	BOOL	release 2
31	0.7	xAuto	BOOL	extern automatic
16	1.0	ACoHM	BOOL	automatic control help memory
17	1.1	ExCoHM	BOOL	extern control help memory
18	1.2	FBaOn	BOOL	feedback ON intern
19	1.3	FBaOff	BOOL	feedback OFF intern
20	1.4	FBaChange	BOOL	change extern feedback (0 FBa1=OFF FBa2=ON / 1 FBa1=ON FBa2=OFF)
21	1.5	FBa1Active	BOOL	feedback 1 active
22	1.6	FBa2Active	BOOL	feedback 2 active
23	1.7	xAutoHM	BOOL	extern automatic old
8	2.0	GAIQuitt	BOOL	general alarm acknowledge
9	2.1	Ign	BOOL	ignore alarm
10	2.2	Sim	BOOL	simulation
11	2.3	Auto	BOOL	automatic mode
12	2.4	MCo	BOOL	manual control
13	2.5	EmRel	BOOL	emergency release
14	2.6	InterlockGAI	BOOL	interlock by alarm
15	2.7	Maint	BOOL	maintenance
0	3.0	GAI	BOOL	general alarm
1	3.1	GAIS	BOOL	general alarm save
2	3.2	SCE	BOOL	status check error
3	3.3	Mov	BOOL	actuator is moving for visu
4	3.4	On	BOOL	actuator is ON
5	3.5	Off	BOOL	actuator is OFF
6	3.6	Out	BOOL	output
7	3.7	User	BOOL	free for user program

Digital Input

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	EA0	BOOL	enable alarm by 0-signal
25	0.1	EA1	BOOL	enable alarm by 1-signal
26	0.2	SCS0	BOOL	status check alarm by 0-signal
27	0.3	SCS1	BOOL	status check alarm by 1-signal
28	0.4	xSig	BOOL	signal extern
29	0.5	B29	BOOL	spare
30	0.6	B30	BOOL	spare
31	0.7	B31	BOOL	spare
16	1.0	AIHM	BOOL	help memory for alarm
17	1.1	ImpHM	BOOL	help memory for impulse
18	1.2	xSigHM	BOOL	signal extern help memory
19	1.3	B19	BOOL	spare
20	1.4	B20	BOOL	spare
21	1.5	B21	BOOL	spare
22	1.6	B22	BOOL	spare
23	1.7	B23	BOOL	spare
8	2.0	GAIQuitt	BOOL	general alarm acknowledge
9	2.1	Ign	BOOL	ignore alarm
10	2.2	Sim	BOOL	simulation
11	2.3	iEA0	BOOL	intern alarm by 0
12	2.4	iEA1	BOOL	intern alarm by 1
13	2.5	ImpProt	BOOL	write impulse flank to protocol
14	2.6	ImpNegProt	BOOL	write negative impulse flank to protocol
15	2.7	Switch	BOOL	convert as switch output
0	3.0	GAI	BOOL	general alarm
1	3.1	GAIS	BOOL	general alarm save
2	3.2	SCE	BOOL	status check error
3	3.3	Sig	BOOL	signal state
4	3.4	Imp	BOOL	impulse flank
5	3.5	ImpNeg	BOOL	negative impulse flank
6	3.6	B06	BOOL	spare
7	3.7	User	BOOL	free for user

Analog Input

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	ELLA	BOOL	enable low low alarm
25	0.1	EHHA	BOOL	enable high high alarm
26	0.2	xAl	BOOL	alarm from extern
27	0.3	NPA	BOOL	no periphery adaption
28	0.4	B28	BOOL	spare
29	0.5	B29	BOOL	spare
30	0.6	B30	BOOL	spare
31	0.7	B31	BOOL	spare
16	1.0	MLLA	BOOL	low low alarm - alarm if enabled
17	1.1	MLL	BOOL	low low limit - warning if enabled
18	1.2	ML	BOOL	low limit
19	1.3	MSp	BOOL	setpoint
20	1.4	MH	BOOL	high limit
21	1.5	MHH	BOOL	high high limit - warning if enabled
22	1.6	MHHA	BOOL	high high alarm - alarm if enabled
23	1.7	MHWA	BOOL	alarm from hardware
8	2.0	GAIQuitt	BOOL	general alarm acknowledge
9	2.1	Ign	BOOL	ignore alarm
10	2.2	Sim	BOOL	simulation
11	2.3	iEHWA	BOOL	enable hardware alarm
12	2.4	iELLA	BOOL	enable LL alarm
13	2.5	iEHHA	BOOL	enable HH alarm
14	2.6	iELLW	BOOL	enable LL warning
15	2.7	iEHHW	BOOL	enable HH warning
0	3.0	GAI	BOOL	general alarm
1	3.1	GAIS	BOOL	general alarm save
2	3.2	Warn	BOOL	general warning
3	3.3	Filter1	BOOL	filter 1 on (75%)
4	3.4	Filter2	BOOL	filter 2 on (88%)
5	3.5	Filter3	BOOL	filter 3 on (94%)
6	3.6	ManuInp	BOOL	manual input (no periphery)
7	3.7	User	BOOL	memory free for user

Regulator

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	EAI	BOOL	enable alarm
25	0.1	SCS	BOOL	status check start
26	0.2	MStC	BOOL	static output value
27	0.3	MStrt	BOOL	starting value
28	0.4	MOVMin	BOOL	output value min.
29	0.5	MOVMax	BOOL	output value max.
30	0.6	OVOOn	BOOL	output value on
31	0.7	B31	BOOL	spare
16	1.0	B16	BOOL	spare
17	1.1	B17	BOOL	spare
18	1.2	B18	BOOL	spare
19	1.3	B19	BOOL	spare
20	1.4	AIHM	BOOL	help memory for alarm
21	1.5	AHystHM	BOOL	help memory outside hysteresis
22	1.6	StrtHM	BOOL	help memory starting value active
23	1.7	Warn	BOOL	warning
8	2.0	GAIQuitt	BOOL	general alarm acknowledge
9	2.1	Ign	BOOL	ignore alarm
10	2.2	Sim	BOOL	simulation
11	2.3	MCOOn	BOOL	mode controller on (0=off)
12	2.4	MSpExt	BOOL	mode setpoint extern (0=intern)
13	2.5	DisOut	BOOL	disable output periphery (0=enable)
14	2.6	EW	BOOL	enable warning
15	2.7	B15	BOOL	spare
0	3.0	GAI	BOOL	general alarm
1	3.1	GAIS	BOOL	general alarm save
2	3.2	SCE	BOOL	status check error
3	3.3	Filter1	BOOL	filter 1 on (75%)
4	3.4	Filter2	BOOL	filter 2 on (88%)
5	3.5	Filter3	BOOL	filter 3 on (94%)
6	3.6	CA	BOOL	control acting (1 = inverse)
7	3.7	User	BOOL	memory free for user

Counter

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	EAImp	BOOL	enable impulse alarm
25	0.1	ELLA	BOOL	enable low low alarm
26	0.2	EHHA	BOOL	enable high high alarm
27	0.3	xAI	BOOL	alarm from extern
28	0.4	ResetBlock	BOOL	interlock counter reset
29	0.5	xSig	BOOL	impulse input
30	0.6	B30	BOOL	spare
31	0.7	B31	BOOL	spare
16	1.0	MLLA	BOOL	low low alarm - alarm if enabled
17	1.1	MLL	BOOL	low low limit - warning if enabled
18	1.2	ML	BOOL	low limit
19	1.3	MSp	BOOL	setpoint
20	1.4	MH	BOOL	high limit
21	1.5	MHH	BOOL	high high limit - warning if enabled
22	1.6	MHHA	BOOL	high high alarm - alarm if enabled
23	1.7	ImpHM	BOOL	impulse help memory
8	2.0	GAIQuitt	BOOL	general alarm acknowledge
9	2.1	Ign	BOOL	ignore alarm
10	2.2	Sim	BOOL	simulation
11	2.3	Reset	BOOL	reset counter
12	2.4	iELLA	BOOL	counting reserve
13	2.5	iEHHA	BOOL	enable HH alarm
14	2.6	iELLW	BOOL	enable LL warning
15	2.7	iEHHW	BOOL	enable HH warning
0	3.0	GAI	BOOL	general alarm
1	3.1	GAIS	BOOL	general alarm save
2	3.2	Warn	BOOL	general warning
3	3.3	Imp	BOOL	impulse flank
4	3.4	B04	BOOL	spare
5	3.5	B05	BOOL	spare
6	3.6	B06	BOOL	spare
7	3.7	User	BOOL	memory free for user

Message

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	B24	BOOL	spare
25	0.1	B25	BOOL	spare
26	0.2	B26	BOOL	spare
27	0.3	B27	BOOL	spare
28	0.4	xAlarm	BOOL	signal extern for alarm condition
29	0.5	B29	BOOL	spare
30	0.6	B30	BOOL	spare
31	0.7	B31	BOOL	spare
16	1.0	B16	BOOL	spare
17	1.1	B17	BOOL	spare
18	1.2	B18	BOOL	spare
19	1.3	B19	BOOL	spare
20	1.4	B20	BOOL	spare
21	1.5	B21	BOOL	spare
22	1.6	B22	BOOL	spare
23	1.7	B23	BOOL	spare
8	2.0	GAIQuitt	BOOL	general alarm acknowledge
9	2.1	Ign	BOOL	ignore alarm
10	2.2	Sim	BOOL	simulation
11	2.3	OPMsg	BOOL	operator message
12	2.4	B12	BOOL	spare
13	2.5	B13	BOOL	spare
14	2.6	B14	BOOL	spare
15	2.7	B15	BOOL	spare
0	3.0	GAI	BOOL	general alarm
1	3.1	GAIS	BOOL	general alarm save
2	3.2	OPMsgActive	BOOL	operator message active
3	3.3	AlarmMsgActive	BOOL	alarm message active
4	3.4	iAlarm	BOOL	alarm active intern
5	3.5	B05	BOOL	spare
6	3.6	B06	BOOL	spare
7	3.7	User	BOOL	free for user

Switch

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	Set	BOOL	set software switch
25	0.1	Reset	BOOL	reset software switch
26	0.2	B26	BOOL	spare
27	0.3	B27	BOOL	spare
28	0.4	B28	BOOL	spare
29	0.5	B29	BOOL	spare
30	0.6	B30	BOOL	spare
31	0.7	B31	BOOL	spare
16	1.0	B16	BOOL	spare
17	1.1	B17	BOOL	spare
18	1.2	B18	BOOL	spare
19	1.3	B19	BOOL	spare
20	1.4	B20	BOOL	spare
21	1.5	B21	BOOL	spare
22	1.6	B22	BOOL	spare
23	1.7	B23	BOOL	spare
8	2.0	B08	BOOL	spare
9	2.1	B09	BOOL	spare
10	2.2	B10	BOOL	spare
11	2.3	B11	BOOL	spare
12	2.4	B12	BOOL	spare
13	2.5	B13	BOOL	spare
14	2.6	B14	BOOL	spare
15	2.7	B15	BOOL	spare
0	3.0	B00	BOOL	spare
1	3.1	B01	BOOL	spare
2	3.2	B02	BOOL	spare
3	3.3	Sig	BOOL	spare
4	3.4	B04	BOOL	spare
5	3.5	B05	BOOL	spare
6	3.6	B06	BOOL	spare
7	3.7	User	BOOL	free for user

Alarm Groups

HMI Address	PLC Address	Symbol	Type	Remark
24	0.0	B24	BOOL	set software switch
25	0.1	Ign	BOOL	reset software switch
26	0.2	Sim	BOOL	spare
27	0.3	Auto	BOOL	spare
28	0.4	B28	BOOL	spare
29	0.5	EmRel	BOOL	spare
30	0.6	B30	BOOL	spare
31	0.7	Maint	BOOL	spare
16	1.0	Gal	BOOL	spare
17	1.1	Gals	BOOL	spare
18	1.2	SCE	BOOL	spare
19	1.3	Warn	BOOL	spare
20	1.4	Msg	BOOL	spare
21	1.5	ProcRun	BOOL	spare
22	1.6	ProcProd	BOOL	spare
23	1.7	ProcCIP	BOOL	spare
8	2.0	B08	BOOL	spare
9	2.1	B09	BOOL	spare
10	2.2	B10	BOOL	spare
11	2.3	B11	BOOL	spare
12	2.4	B12	BOOL	spare
13	2.5	B13	BOOL	spare
14	2.6	B14	BOOL	spare
15	2.7	B15	BOOL	spare
0	3.0	B00	BOOL	spare
1	3.1	B01	BOOL	spare
2	3.2	B02	BOOL	spare
3	3.3	Sig	BOOL	spare
4	3.4	B04	BOOL	spare
5	3.5	B05	BOOL	spare
6	3.6	B06	BOOL	spare
7	3.7	User	BOOL	free for user

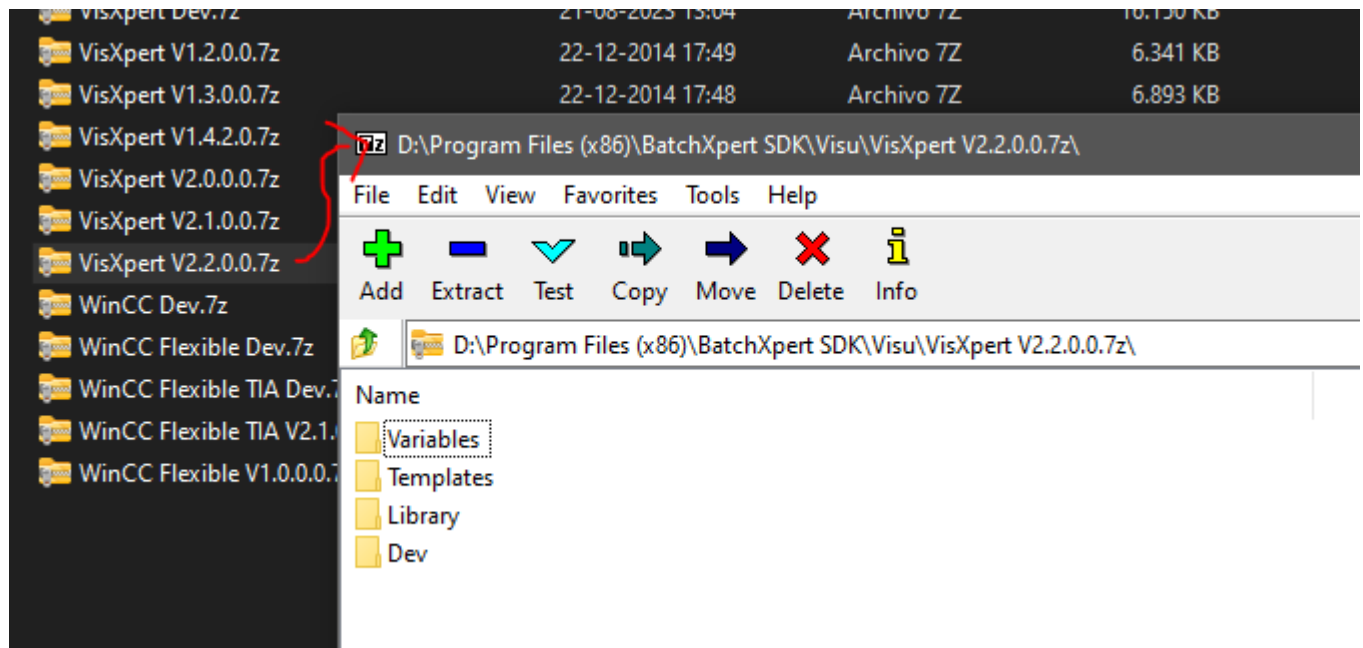
The HMI Library

When installing the BatchXpert SDK, this also installs multiple HMI libraries into your BatchXpert SDK installation directory:

“D:\Program Files (x86)\BatchXpert SDK\Visu”

This directory contains an image library, with many images that you can implement into your process screens. You can find images for different machines, brewing equipment and other type of symbols that are helpful to make appealing process screens.

VisXpert archive contains all templates, variable tables, process symbol library and a Project Template for you to use in your project. In these archives you can find all necessary data to update or create new HMI applications.



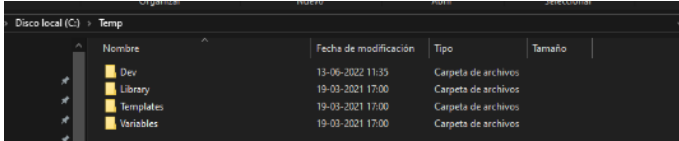
To update existing HMI applications to newer BatchXpert HMI versions, you should consult the appropriate Manual, that explains in more detail how to go about updating an existing HMI application from a newer VisXpert library

If you want to update your existing HMI application to the newest library version, you should refer to the dedicated manual called “Manual BatchXpert Updating VisXpert HMI” for this, which describes the update procedure in more detail.

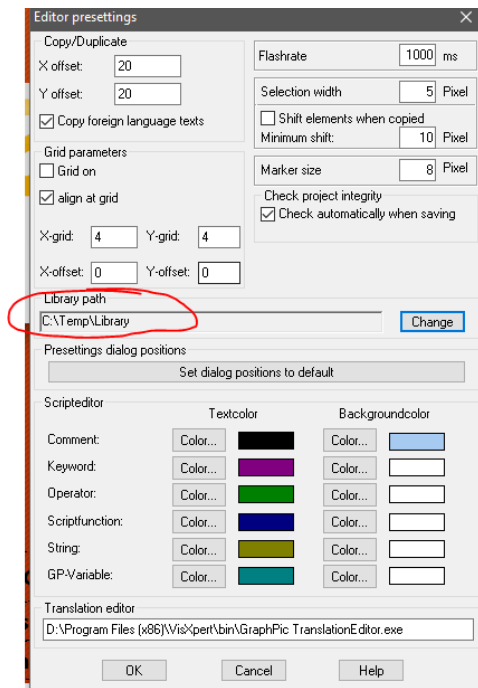
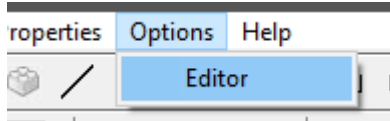
Working with the HMI Library

When working with an HMI library, you want to 1st extract the content of this HMI version that you want to use into a working directory, for example: "C:\Temp", and then point do you VisXpert library Path to the newly created library objects. You can do this in the editor settings of your graphics editor. After setting up the library directory, all Classes, Windows and script objects can be updated from this library.

Extract to Working directory, in this case C:\Temp

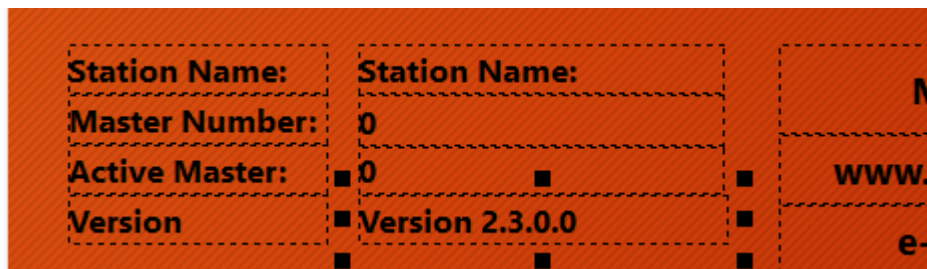
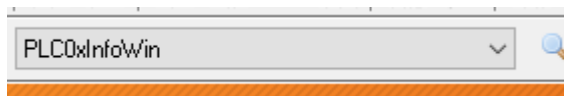


Point Graphics Editor Library path to this Working Directory



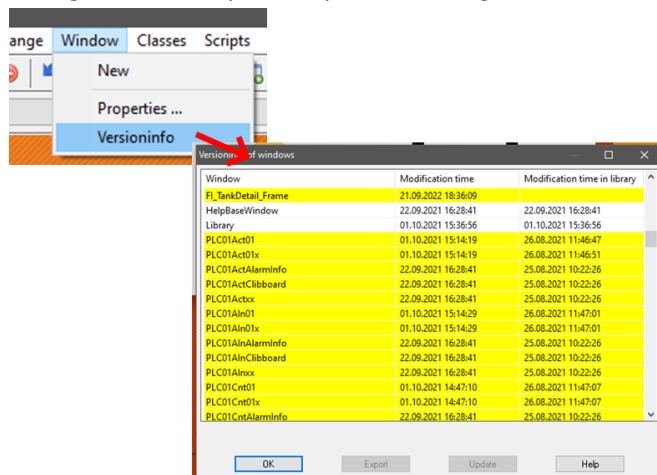
Checking your HMI Version

The current HMI version that you are using in your project is always available in the “**PLC0xInfoWin**” HMI screen, that is available in your graphics editor and during runtime. For more detailed information about which library objects and windows are currently in use, you can use the version control integrated into your graphics editor.



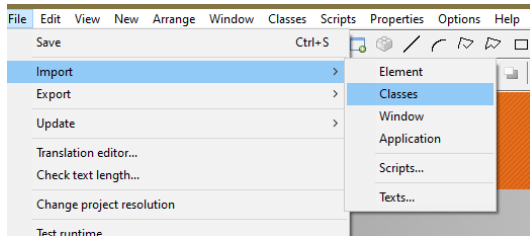
Checking Version of Individual Classes and Windows

Windows and Class objects have individual version numbers that can be checked against your extracted library by using the “Window->version info” or “Classes->version info” option from the main menu. This option opens on dialog that allows you to update existing classes.

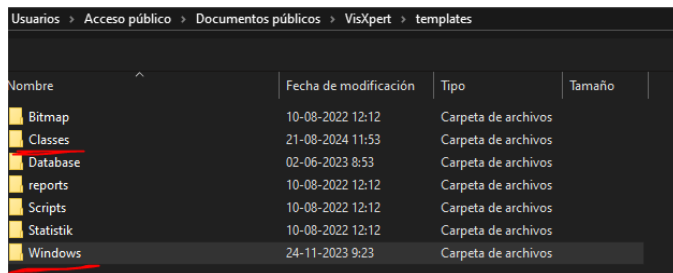


Importing Windows and Classes

The version info option only allows you to check objects that are already present in your project against the version that exist of this object in the library. If you want to add objects from the library that do not currently exist in your project, you must use the input option from the main menu. From this option you can select the window or class respectively and insert them into your project to be used. After importing the objects, you can use the version info dialog to check their version against the library version.

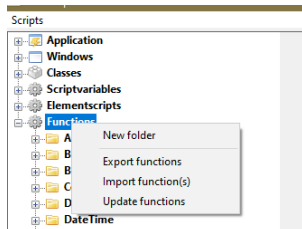


All objects that are part of the library exist in the appropriate subfolder inside your library directory. You just have to select the appropriate object and Insert it into your project.



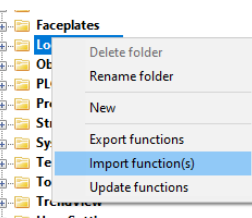
Updating/Importing HMI Scripts

Updating scripts works like updating windows and classes, but from the scripts dialog.



If you choose the “update functions” option, all functions that are currently contained in your project will be updated with the version from the selected library. However sometimes new functions are added to the library which will not be updated by the “update functions” option.

For this you can use the “Import Function(s)” option and select the script file that you want to import. Since you can group different script functions into directories, all import script files are also contained in subdirectories in

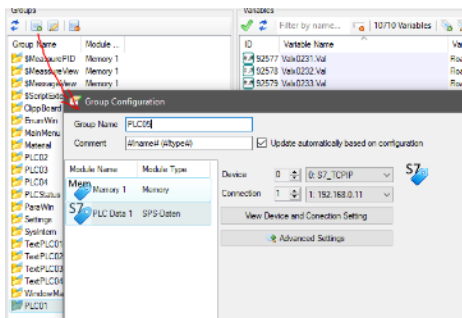


the library directory. We recommend that you order all these functions in the same subdirectories as they appear in the library directory. The easiest way to do this, is by right clicking directly on the folder containing the script functions that you want to import, and not right clicking from the main “Functions” three item but rather from the sub directory below this menu option. This allows you to import only functions belonging to a specific directory.

Adding a new PLC to the HMI

If you want to add a new PLC to your BatchXpert application, you need to add the following components from your corresponding expert archive, found in your HMI library, to your HMI application.

Adding a new PLC Group

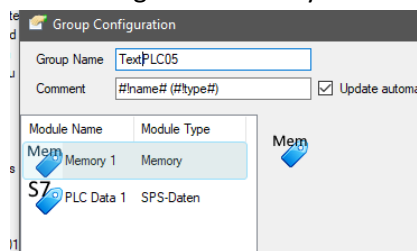


The first action that you should take is that you add a new PLC group that corresponds to the PLC that you want to add to the System. You do this by opening the variable editor, and removing the existing PLCxx variable group if it already exists as a memory driver. This is because the default BatchXpert project includes PLC02, PLC03 and PLC04 already added as memory variables to the project. But since you want to add a real PLC you must change these variables to the appropriate PLC driver.

After removing the existing PLC group, you can add a new PLC group and continue with configuration of your communications settings for this PLC.

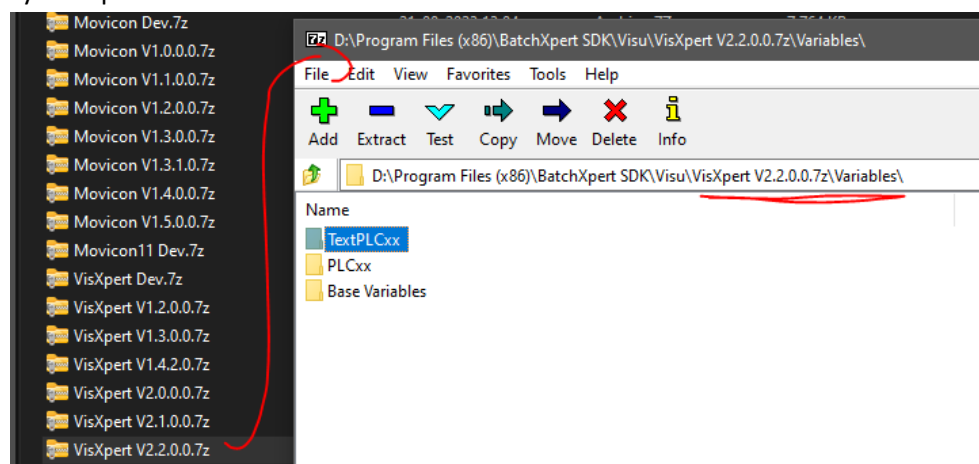
Adding new PLCxx Text Group

BatchXpert also requires some memory variables for each PLC, which are all configured in the TextPLCxx group that belongs to a memory driver.

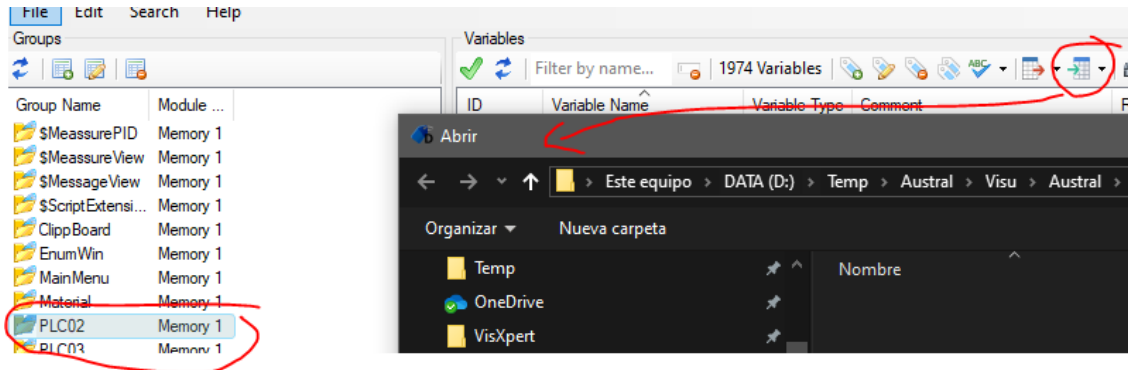


Importing the PLC Variables

After creating a new PLC group, you can now continue to import all variable tables from your HMI library into this newly created variable group. All variables of the "TextPLCxx" directory must be imported into the appropriate "TextPLCxx" memory variable group, and all variables contained in the "PLCxx" and "Base Variables" directories must be imported into the "PLCxx" variable groups. This imports all variables related to parameter channels, system parameters and all available control models.

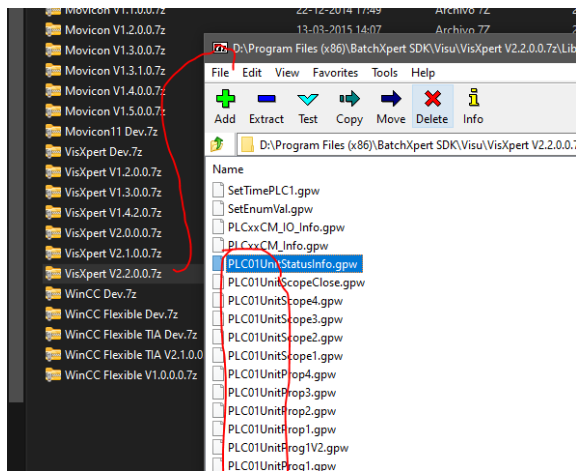


By selecting the appropriate PLC group, you can open the Import dialog from the menu bar on the top and then select the appropriate variable list that you want to import into each of the groups.



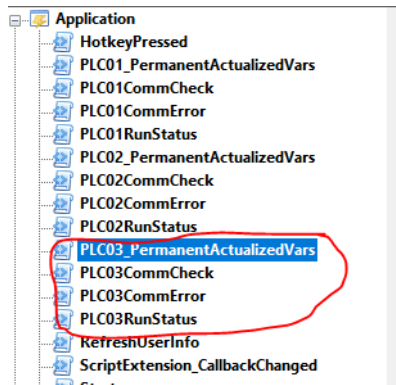
Importing the PLC dependent Windows from your library

After importing all the variables, you only must import all the face plates and PC specific windows for your HMI application. What is windows can also be found in your HMI library and all windows starting with PLCxx must be imported into your HMI application.



Currently there is an defect in VisXpert, which does not allow you to import more than about 10 windows at the same time. To circumvent this please import all windows in batches of at most about 10 windows, and repeat the process until all windows are imported.

Adding Global Monitoring Scripts



Each PLC requires some commonly used global application scripts, that monitor the communication and handle some global variables for each PLC. After adding all the windows and variables you also must add these scripts to the application. You can import them from the project library or copy them from an existing PLC and adjust where it is needed.

PLCxx_PermanentActualizedVars

This script is used to maintain references to certain variables that should always be communicated with the PLC even if no window is actively using these variables. This means the variables that are used in these scripts are

used so that they get activated by the HMI application and communication gets established with the controllers, even if these variables are not even used in any of the HMI Windows. This is required by some functionality of BatchXpert, so that some Variables remain an steady connection to the PLC.

PLCxxCommCheck

This script handles the a live signal from the respective master station to the PLC. PLC sends a life signal that has to be confirmed by each operating station, where each operating station is identified by its master number. This script confirms this life signal, depending on the master number of the local station, to the PLC.

PLCxxCommError

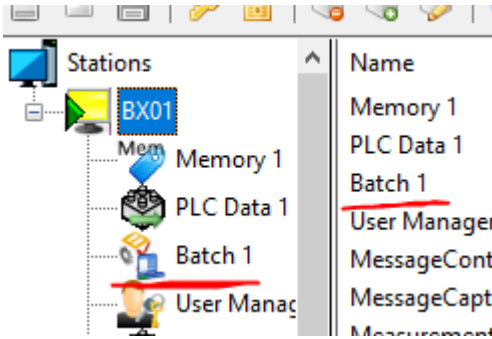
If the PLC driver detects an error with the communication to the PLC it sets the appropriate error status in the ""PLCxx:@ConnectStatus"" variable. This script reacts to changes to this variable and activates the appropriate alarm and the alarm system and also updates the PLC's internal status.

PLCxxRunStatus

this script is used to detect if on PLC is actually executing the user application. It monitors the PLC system time and whenever it changes it knows that the user program is actually executing. If this counter is not changed in more than 5 seconds it activates the appropriate alarm message to tell the user that it's logic controller maybe in stop mode.

This script also detects if the PLC exists by checking if the timer actually exists.

The Batch Module



BatchXpert adds an “Batch Module” to its VisXpert project structure. This module is responsible for generating trend recording data, trend view xml files, and allowing messages for all control modules that are configured in your project.

The Batch Module is started when the project is started and goes through the configuration of all your control modules and creates new trend recording data, Trend few XML configuration files and allow messages in the case that new modules have been added to BatchXpert.

This means that this module always maintains trend view, trend recording and Alarm configuration in sync with your control module configuration of BatchXpert. This synchronization process is started when Module is started, which means that you should restart your project, which in turn restarts the Batch Module, When you have made changes to any of your control modules.

Show Trends with VisXpert

Opening a Trend window from the HMI

To add a Trend View Button to any of the process graphics, you first need to add a button or Bitmap which will act as a button for calling the desired Trend view. Usually you will use the “TrendButton.bmp” as a default trend view invocation object. You can find a preconfigured “Trend button” in the “Library” window.

In the Trend button object, you must add an “Click” event, that executes an “Element Script”, which in turn calls one of the “ShowTrendView” functions. BatchXpert provides multiple script functions that you can put into any button or object you need to open the trend view. There are functions to show only the trend view of an specific Analog input, for all Analog values of a unit, or for custom trend views.

To assign the Script to the Trend button, you must open the “Dynamic” dialog by double clicking the button in the graphics editor or by pressing “Ctrl+d”. This opens the “Dynamic Dialog” which allows you to define the Script that will be executed when a left click happens.

In this script you can call one of the multiple “ShowTrendView” functions to show the desired trend data. You can also define custom trend views, which you can show by simply passing in the name of the “.xml” filename of the trend view data. For custom Trend views, the name of the view that you pass into “ShowTrendView” corresponds to the filename of the xml configuration file of the View. This configuration file defines the trend curves and their configuration in the trend viewer.

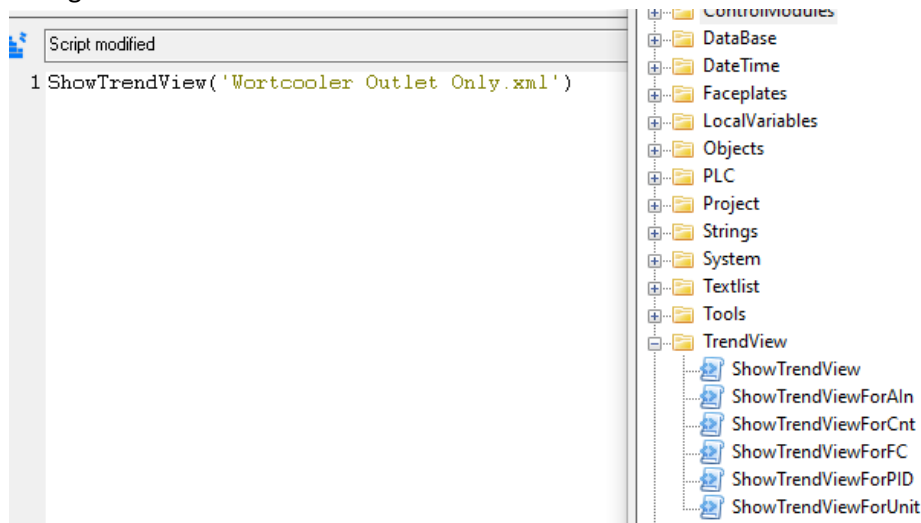


Figure 8 Show Trend Script for an custom trend view

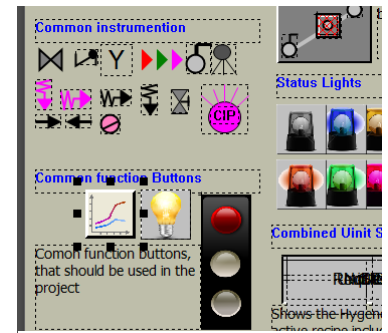


Figure 7 Trend Button in the "Library"

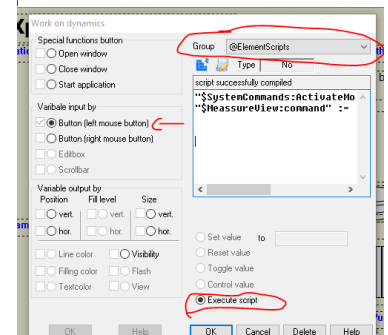


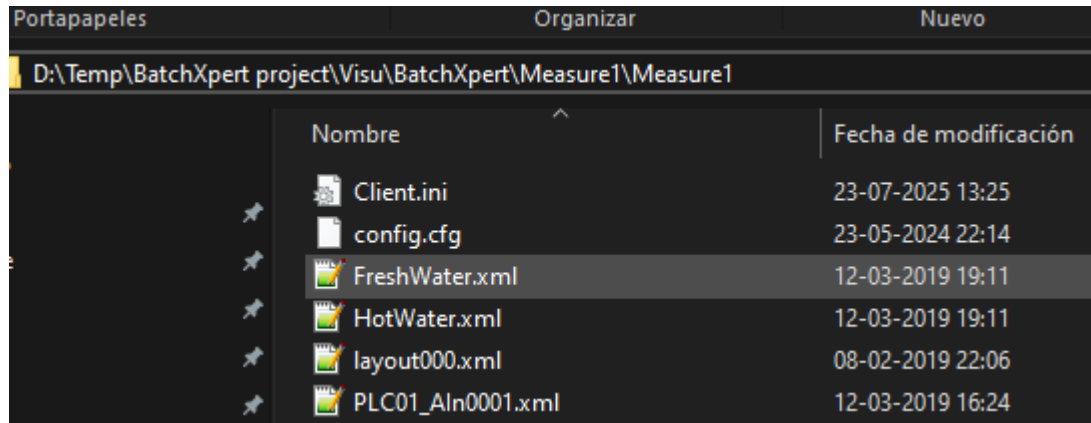
Figure 6 "Dynamic" Dialog of an Trend button

Trend View XML files

The Trend View XML files are configuration files that are used by the “Measurement Viewer” application to configure the trend curves and their configuration to be used. These files store the trend lines that are part of this trend view, their Min and Maximum values, and other values.

These XML files are in the Project Directory:

<BatchXpert Project Directory>\Visu\BatchXpert\Measure1\Measure1\

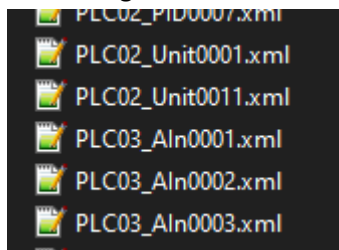


Auto Generated Trend Views

The “Batch Module” in VisXpert automatically generates default Trend views for many object, that can directly be used by calling the appropriate “ShowTrendView” functions from the buttons script. It automatically generates an “Trend View” xml files. It automatically generates files for each analog control module, such as “Analog inputs”, “Counters” or “Regulators”.

For each Unit that is configured in BatchXpert, it also generates an “Unit Trend View”, that contains all analog values that correspond to this Unit.

The auto generated Trend views will follow the following naming convention.

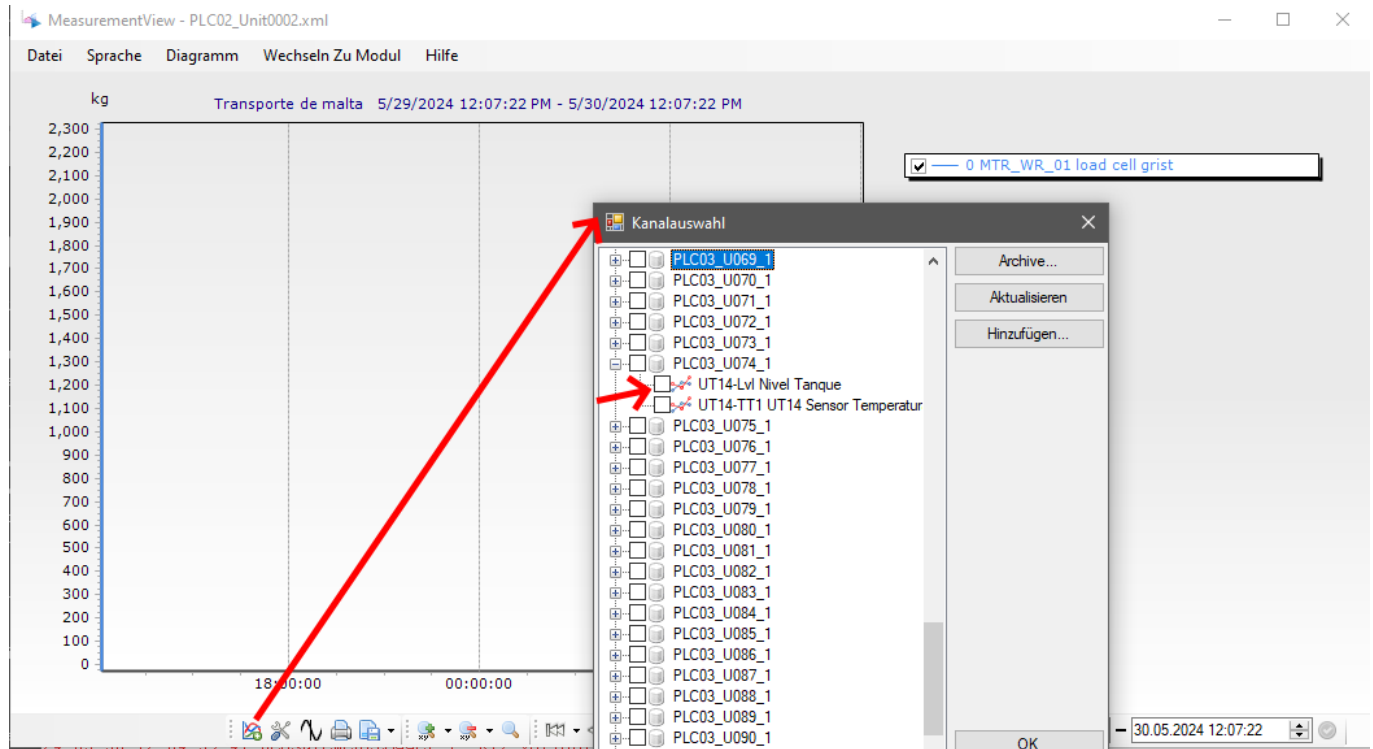


These Auto generated Trend views will be regenerated each time the Project is restarted, since a Project Restart also restarts the “Batch Module”, which in turn generates these trend views when starting up.

Custom Trend views

To add custom or additional trend lines to a trend view, you can do so directly from the “Measurement Viewer” that is being opened when clicking on a trend view button in the HMI screen. From there you can add trend lines from each of the existing archives. After adding the file, you can just save the configuration. The name of the xml Trend view file that you choose, is the one that you will have to use when calling “ShowTrendView” from your Scripts.

By default, each Process unit will have its own trend file.



Record custom Trend Value

Trend recordings are managed by the VisXpert SCADA system. BatchXpert integrates with VisXpert and automatically generates Trend configuration files for all its Control modules that have analog values. This means that usually you do not need to modify the trend configuration at all.

You can, however, add custom values into the trend configuration or change the configuration of automatically generated values. The configuration is automatically generated when the system starts up.

Supported Trend Values

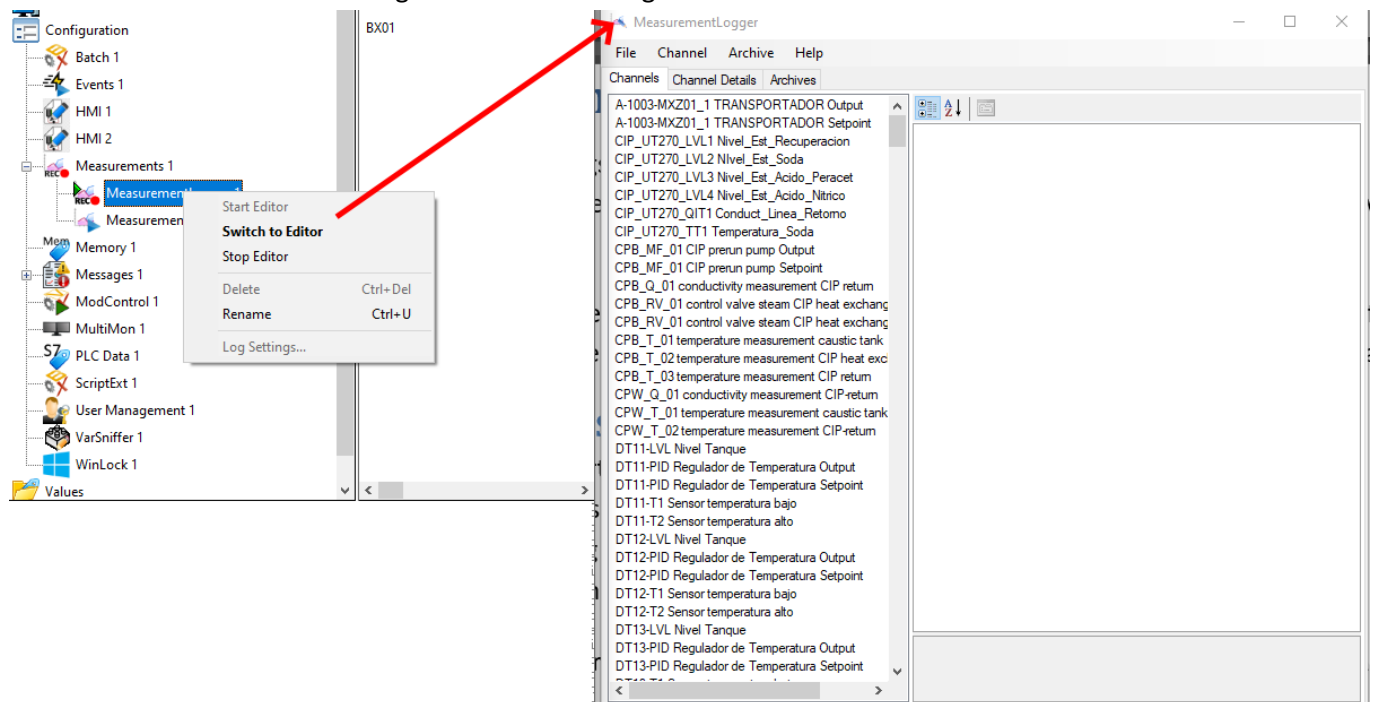
VisXpert supports the following values to be recorded as trends

- Integers
- Floating Point
- Boolean

It does however **not** support recording of variable length data such as “Strings” or “Arrays”

Adding a custom trend to be recorded

The trend recording configuration is done with the “Measurement Recording” editor. This editor allows you to add new “Channels” to the configuration file and assign a trend archive to it.



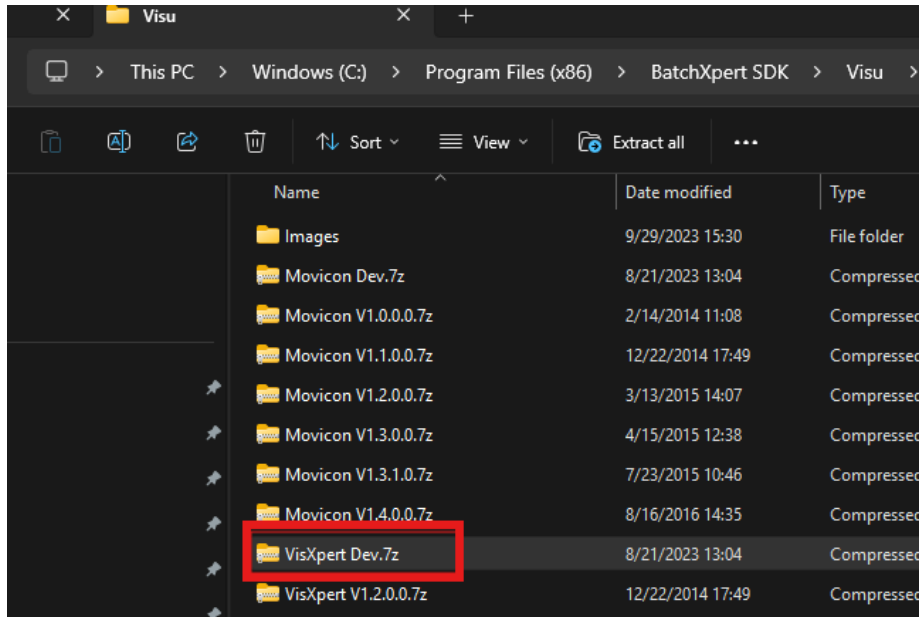
After adding the custom value, you can add them to any of you Trend View configurations for diplay

Adding Batch Number, Step number and Phase

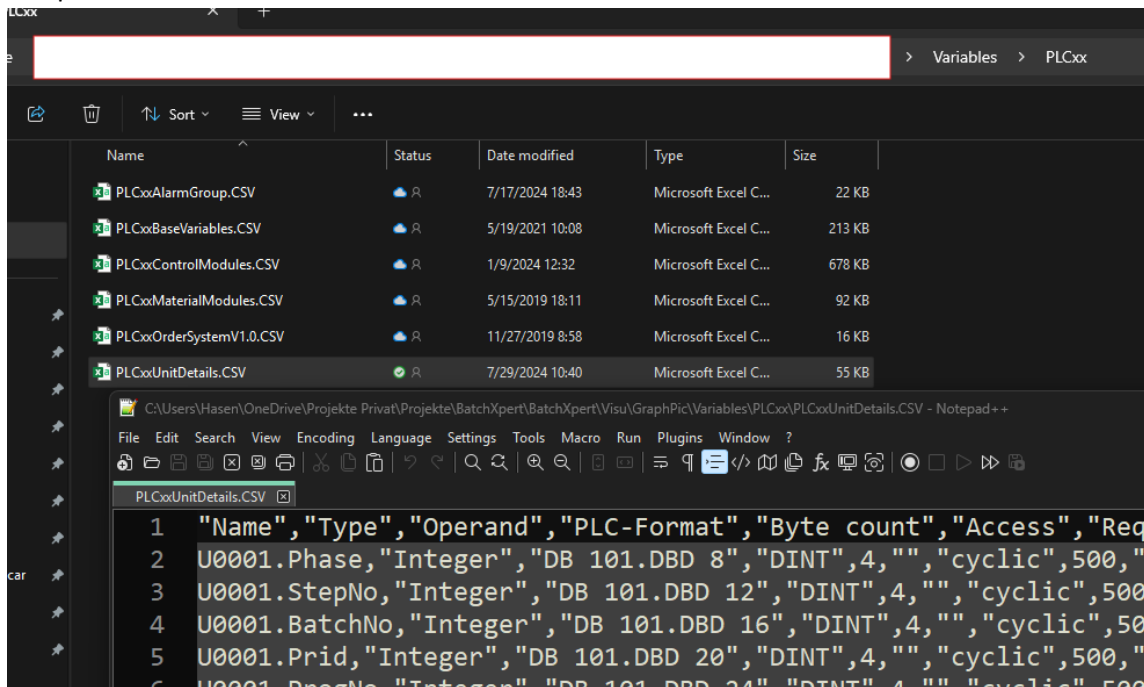
For historical reasons BatchXpert does not have dedicated variables for batch number step number or Phase numbers. However, you can manually add these variables to your variable configuration by importing the corresponding variable list into your “PLCxx” (where xx is your plc number) variable Group in the “SPS Daten” communication Driver.

Locate The Import Tag list

the import table is located inside your BatchXpert SDK directory, usually located in “C:\Program Files (x86)\BatchXpert SDK\Visu”. Inside the corresponding VisXpert archive.

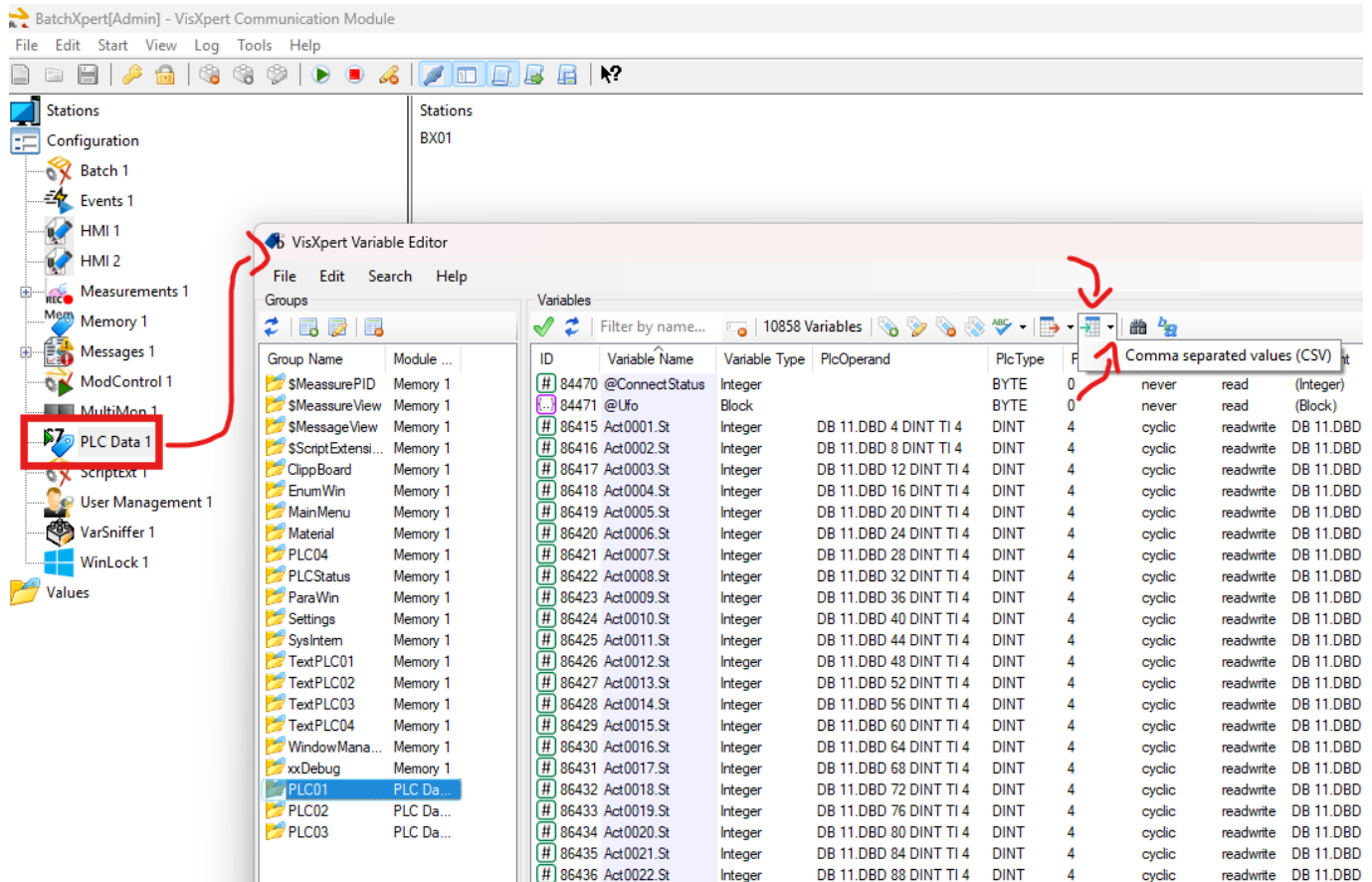


The variable list is called “PLCxxUnitDetails.CSV” a can be found inside the “Variables\PlcXX” directory, inside your VisXpert Archive mentioned above.

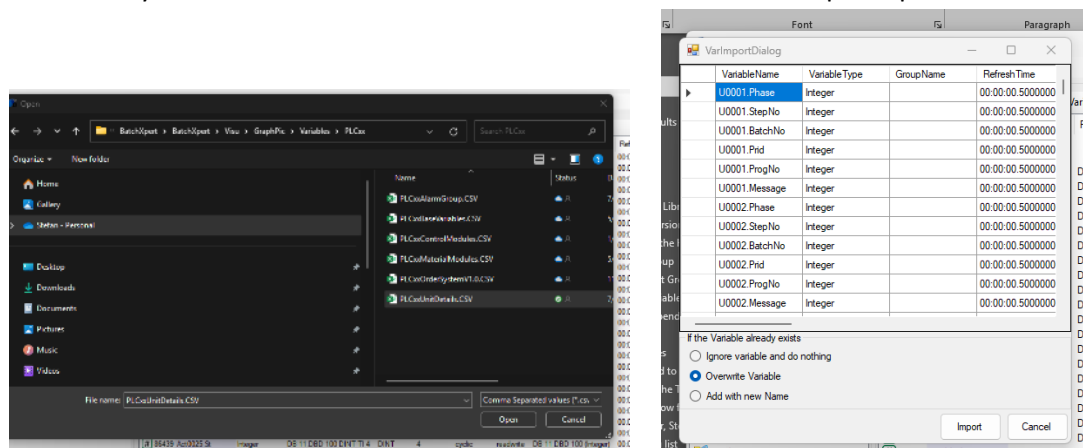


Importing Variables

These variables can be imported by using the variable editor of the “SPS Daten” module from the Communication Module.



From here you can select the file above and select “Overwrite” as import option.



Adding Batch Number to Trends

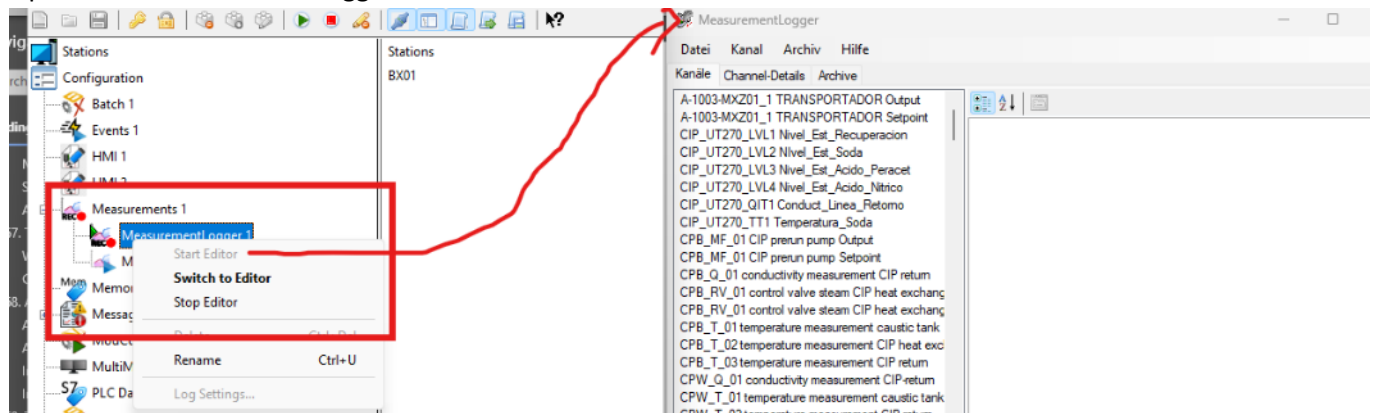
Sometimes it may be helpful to add the batch number of the currently running batch off on units to the Trend Graphs. Since the batch number in BatchXpert is an integer not a string, this can be easily achieved. By default, this batch number is not part of the trend recording but can be manually added easily by following the procedure above.

If your project does not include variables for the Batch number, you must manually import these variables from the current HMI project, in your BatchXpert SDK. Please see [Adding Batch Number, Step number and Phase](#) for more details.

Keep in Mind that VisXpert does NOT support Strings or Block variables to be added to the Trend configuration. This does not affect any standard BatchXpert variables, since these are all Integers or Real variables.

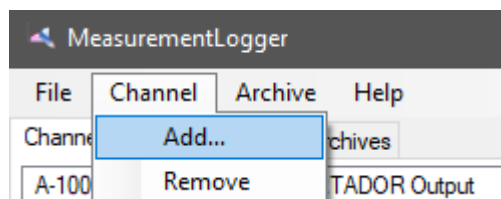
Add a custom Trend Channel to your Trend configuration

Open the “Measurement Logger” editor from the Communications Module.



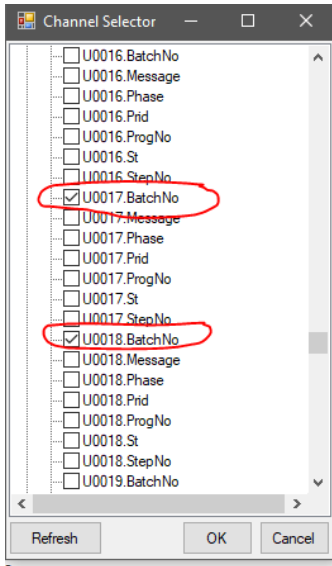
Select “Channel-Add”

This allows you to add custom variables to be added to the Measurement logging, to be recorded and shown in trends.



Select the “BatchNo” of the units that you want to record.

Select the BatchNumber, Step Number or any other variables you need.



Now you can use them in your Trend Views

PLC backup

Backup for S7-300 and S7-400 series

BatchXpert includes the “S7 Backup” utility for creating backups of your PLC data blocks. “S7 Backup” is a small tool for creating **Manual and Automated Backups** of all your online controller data of your **“Simatic S7” compatible controllers**. This tool connects to the specified Controllers and uploads all available online data and saves them in Backup files. These files can be archived and restored when required.

The Tool supports the following Cpu:

- Siemens S7-300
- Siemens S7-400
- Vipa Speed 7, 100V, 200V and 300S series
- Vipa Speed 7 Slio series

The application can be downloaded by using the “BatchXpert installation Center” or from [“S7 Backup – MLogics Documentation”](#)

Backup for S7-1200 and S7-1500 series

The “S7 Backup” utility unfortunately does not support the newer Simatic PLC series. For these series of PLC's, there is the “S7 Backup 1500” utility. This utility serves the same purpose as the old “S7 Backup” utility and can make backups of your online datablocks.

However this new utility requires you to [Activate Web Server](#) on your PLC. Without the Web Server this utility does not work.

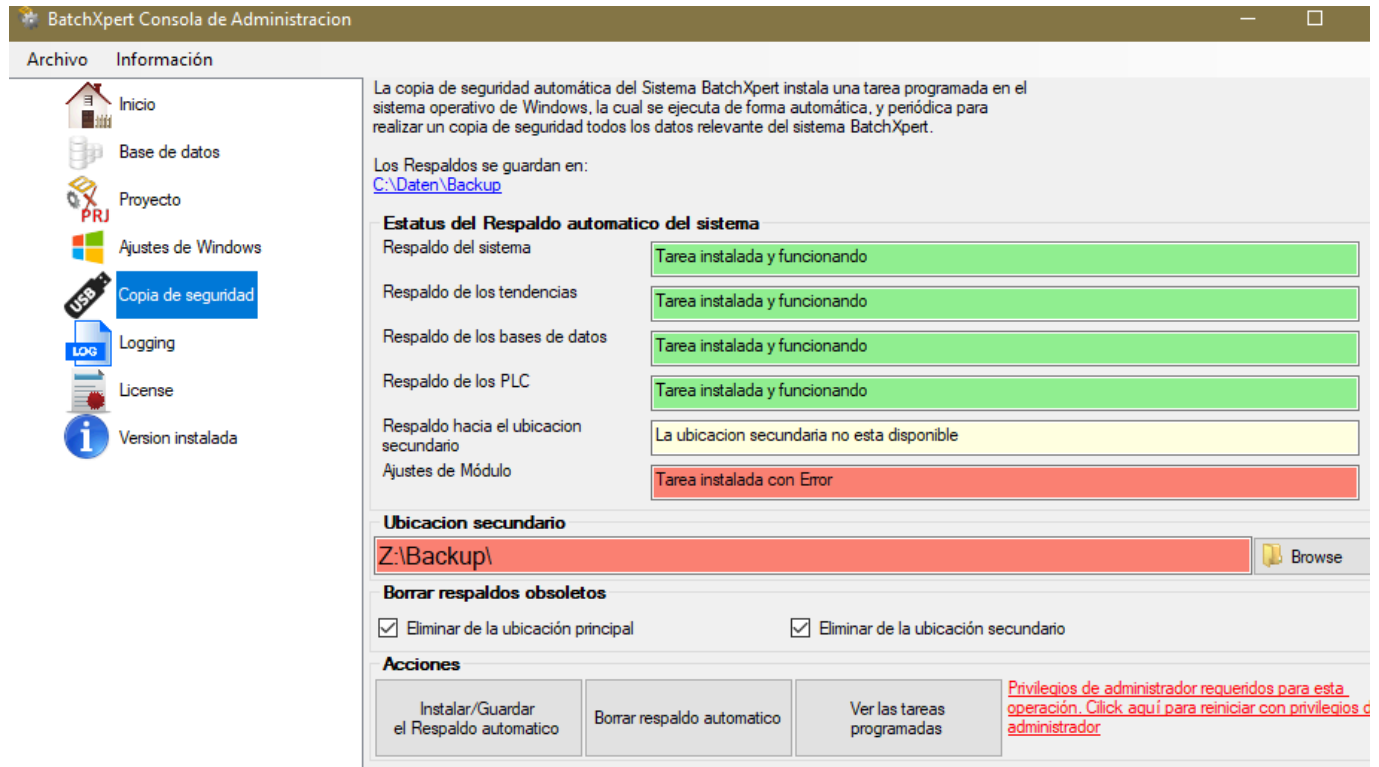
The Tool supports the following Cpu:

- Siemens S7-1200
- Siemens S7-1500

The application can be downloaded by using the “BatchXpert installation Center” or from [“S7 Backup 1500 – MLogics Documentation”](#)

Operating Station Backup

the BatchXpert system incorporates a "project management tool", which allows you to run backups of the entire system. This application is part of BatchXpert and can be launched either manually or by installing the automatic backup tasks from the "BatchXpert Management Console". You can find more information about this tool in the "Manual BatchXpert System Backup" manual.



The "BatchXpert Management Console" Allows you to install automatic backup tasks, that will run on set times and Backup different parts of your project. These automatic backups are implemented as scheduled tasks in the Windows operating system and can be adjusted manually to the needs off your installation.

By default it backups the full system every three months, trending data also every three months, the database every month and PLC every week. If you're in a different schedules you can just open the scheduled task and adjust them as needed.

There's also a scheduled task that allows you to copy the backups to a secondary backup location, which usually is a pen drive or a network share where you can offload your backups. It is recommended that you always offload your backups from the local drives of the computer and even replicate them to all other PCs to create multiple redundancy of your backups.

To avoid filling up the system with backups there's also utility to remove old and obsolete backups. This utility not simply removes by age but applies some heuristics and some smarts, so that it always maintains at least 3 backup archives, and only deletes the oldest ones.

Recommended Settings for Touch Panel use

BatchXpert can be used in conjunction with “Touch Panel PC’s” that are typically installed in electrical panels and act as “Field HMI” for operators to be used. BatchXpert is generally optimized to be used with “traditional” computers, and operated by mouse and keyboard, however it can easily be adapted for “Touch Panel use”.

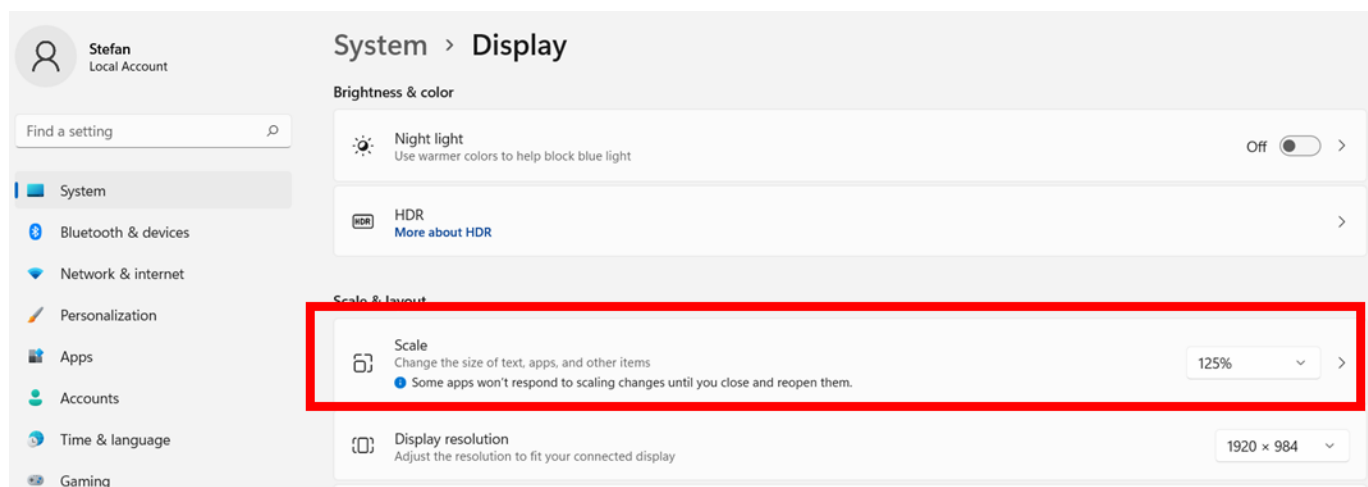


Generally, we recommend that you adjust the following settings in Windows and take the following precautions when engineering an HMI application with BatchXpert.

Set Display Scaling to 125%

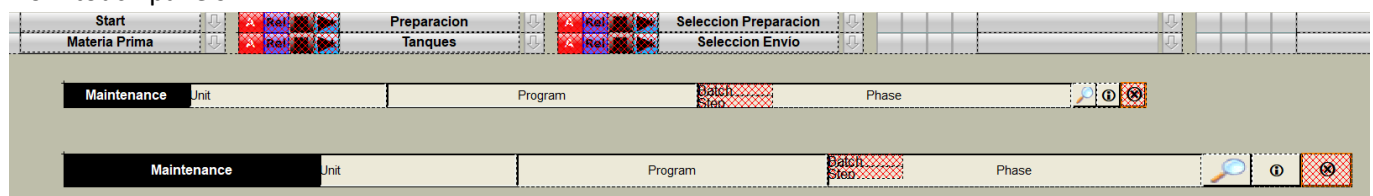
Since touch operation generally requires having bigger buttons, to be able to be reliably being clicked on by touching the screen. In windows you can set the “Display Scale” to 125%, which increases all text, buttons and other UI elements by 25%. This essentially increases all UI elements of windows, and BatchXpert applications to be able to easily click on them by touching the screen.

NOTE: BatchXpert HMI application will not be scaled automatically by increasing the display scale of the operating system. This means that you must adapt some UI elements that you have put in your SCADA screens.



Use bigger Unit symbols

BatchXpert HMI application will not be scaled automatically by increasing the display scale of the operating system. This means that you must adapt some UI elements that you have put in your SCADA screens. The best option to do this is to use the “Unit_ Generic _Big” library object, instead of the regular “Unit_ Generic” library object in your HMI screen. This library object provides bigger patterns to make it easier to be operated reliably from touch panels.

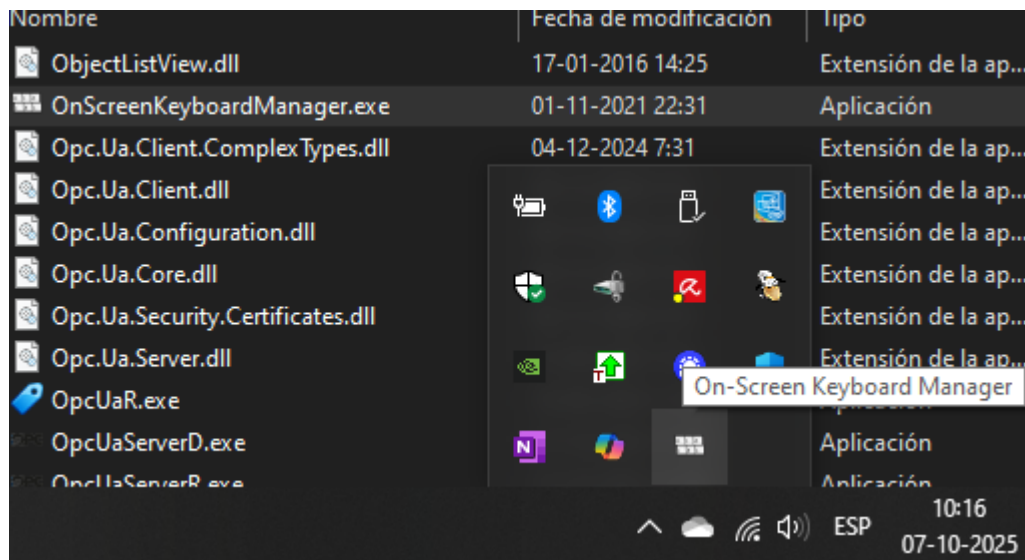


OnScreenKeyboardManager

When using the operating system, you want the on-screen keyboard to be shown whenever you enter any of the input boxes of any UI element the operating system. However, tests have shown that this functionality does not always work reliably when using touch enabled monitors. For this reason, the SCADA system used by BatchXpert, Includes a small application that monitors system events and shows the on-screen keyboard whenever an input box is focused.

This small application is automatically started when BatchXpert is started and a touch monitor is detected. This allows you to use the operating system via the on-screen keyboard. When this application is running a small icon is shown in the taskbar of windows indicating that this application is available and launching the on-screen keyboard whenever an input box is active.

This application is started automatically and no settings or manual adjustments need to be made.



System Hardening and Operating system Security

What is built-in administrator account?

In the Windows operating system, the built-in administrator account -- the first account created when the OS was installed -- has the highest permissions of any profile on the computer system. That means the built-in administrator account has elevated administrator privileges to do anything on the system without requiring confirmation.

Windows OS built-in administrator account explained

In Windows systems, the built-in administrator account is like the "root" or "superuser" accounts in other operating systems. It was originally intended to facilitate system setup and disaster recovery. It can also be used to run programs and apps before a user account is created.

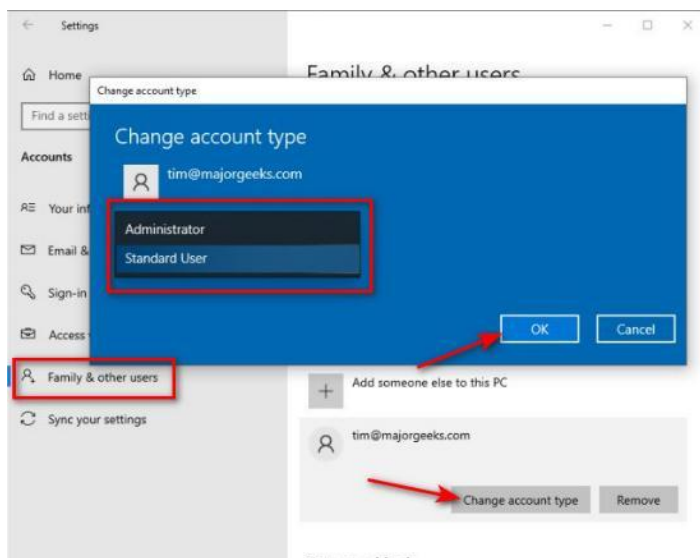
The built-in administrator account is useful for troubleshooting deep system-level issues but must be used sparingly. Even when it is enabled cautiously, it's good practice to immediately disable the account once troubleshooting is complete.

User an “non-Administrator” account for your Operators

When setting up your BatchXpert station, you should always install at least two windows user accounts. One for administrators, which does have administrative privileges, but is never used for operating purposes. The second one should be a non-administrator account, which is used by all operators to execute BatchXpert.

BatchXpert is specifically designed to be run in non-administrative environments and does not require administrative privileges for functioning. The user account that you set up for your operators, with which they will use BatchXpert, should always be a “Windows User” account, and never an “Windows Administrator” account. This helps you to minimize the attack surface of your operating stations, and prevent normal operators changing system level settings, such as IP addresses, or the system clock.

For more information about how to set up your operating stations, you should read the installation manual “Manual BatchXpert Installation in Windows”, found in your installation directory, or on our knowledge base www.Docu.MLogics-Automation.com.

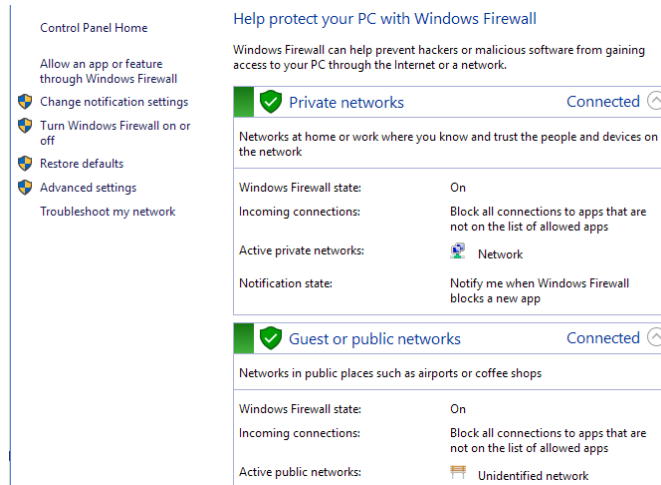


Windows Firewall

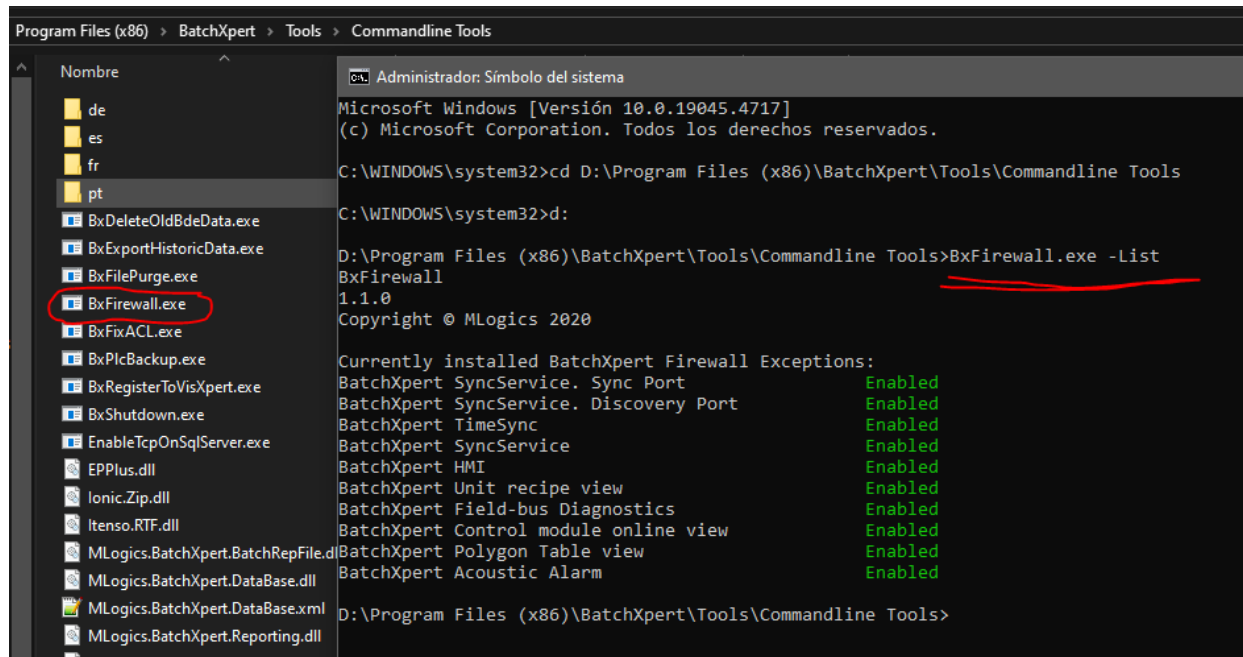
BatchXpert is specifically designed to work with the integrated windows firewall. This means that you should always operate your operating stations with the firewall enabled, and there should never be any reason to disable the windows integrated firewall.

In the command line tools sub directory of your installation directory of BatchXpert, you can find and utility that can help you to configure your firewall in the case you should need to configure the firewall. However this tool is already run during installation, which means that your system should be already configured after running the BatchXpert installer.

Windows Firewall settings



Firewall Utility of BatchXpert, to check and set firewall settings



- AntiVir
- Windows Defender
- Avast

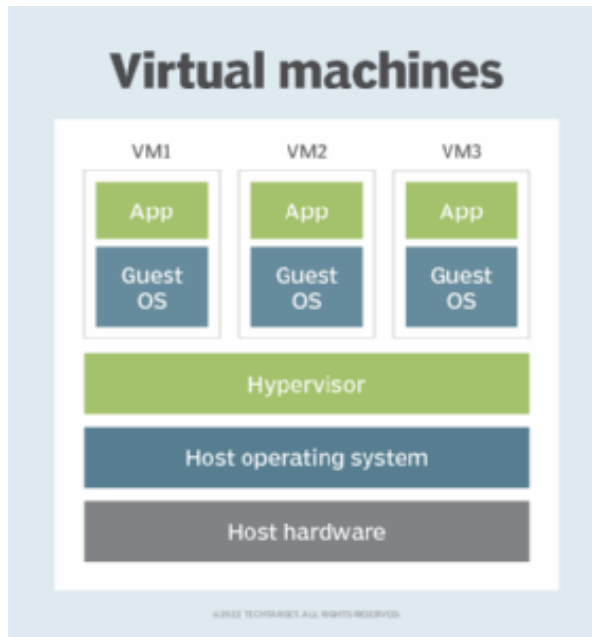
Endpoint Protection



There are multiple vendors for Endpoint protection software, from which we recommend:

- 
- MLOGICS**
GERMAN AUTOMATION PARTNER

Virtualization



BatchXpert is not specifically designed to be run in virtualization environments, such as VMware or Proxmox, however it **can** be installed and be used in virtualized environments. However, **we discourage the use of virtualization environments for process control systems**, because this reduces the redundancy of your distributed control system. BatchXpert is designed to run on multiple independent operating stations, where each operating station can execute all process control tasks independently from all other stations, resulting in and redundancy in the whole system.

Reduced Redundancy in virtualized environments

If you run these Operating Stations in and virtualized environment, you essentially run all operating stations on the

same physical hardware, which reintroduces and single point of failure (the server where the virtual machines are running on) into your process control system.

To run BatchXpert in a virtualized environment you should change some default settings in the project specifically for trend recording and similar. You can contact MLogics to review options and the necessary adjustments to your project application.

Recommended Software

For testing and pick up purposes, it may be advantageous to use virtualized operating stations. From our experience we can recommend products from “VMWare”, which are generally compatible with the BatchXpert control system.

Licensing in Virtualized environments

Licensing does not change for virtualized environments, which means that each virtual machine is treated as a separate operating station which requires its own independent license to be operated. This means that you must have a valid license for each operating station that you intend to run as a virtual machine. Since we discourage the use of virtualized environments, we do not have specific licensing options for virtualized environments.