Engineering
Part 2: Plc Programming

English Version 3.0

Content

About this Manual

BatchXpert is a control system for managing and visualizing batch processes. It supports recipe execution, batch tracking (including material handling), alarm and event management, trend recording, and batch reporting via BatchXpert stations and HMI clients.
The system is modular and can be scaled from single units (for example, CIP or pasteurization skids) to complete production lines. In a typical automation project, BatchXpert integrates three layers—PLC control, SCADA/HMI visualization, and the engineering/database toolchain—that operate together as one system.

This manual is organized into three parts: **Part 1** provides a general overview of BatchXpert, **Part 2** focuses mainly on PLC-related topics and engineering in SIMATIC TIA Portal, and **Part 3** explains how to create the HMI and Operating Station. **Part 4** contains information about additional modules that can be used in the BatchXpert system.

Target Audience of this Manual

This manual is intended for project engineers and automation specialists who implement or maintain BatchXpert projects. It assumes familiarity with batch/process automation concepts and general commissioning practices. This document explains how BatchXpert applies these concepts; it is not an introduction to PLC programming. Working knowledge of Siemens SIMATIC S7 programming and the associated engineering tools (SIMATIC Manager and/or TIA Portal) is required.

AI-assisted editing notice:

This manual was produced with the support of AI tools. AI was used primarily to assist with grammar and spelling corrections and to improve readability. The technical content, procedures, and engineering guidance are based on the author's expertise and project knowledge, not generated by AI.

Version of this Manual

Version 3.0	Reworked completely Separated into multiple Parts

Overview

In the BatchXpert automation architecture, the PLC is the real-time execution layer. It is responsible for deterministic control of the process, meaning it reads field inputs, evaluates logic, and updates outputs cyclically with predictable timing. Batch execution, unit and phase state handling, interlocks, and time-critical sequencing are therefore implemented in the controller, where they remain operational even if an HMI client is disconnected.

BatchXpert standardizes PLC programming by using the PLC Frame and its **control modules** as an abstraction layer over physical I/O. Application logic should interact with devices through module commands and status signals (for example, *Act*, *DIn*, *AIn*, *PID*) instead of directly addressing input/output bits. The physical wiring is mapped to the modules' external (**x**-) signals in the **Transxx** transfer blocks, which provides a consistent place for signal assignment, inversion/selection, and project-specific adaptations while keeping generated and user logic separated.

The PLC also forms the communication endpoint for BatchXpert stations and HMI clients. It exposes standardized data blocks for recipe download and execution, unit status, alarms, events, and diagnostics so that the operating stations can visualize and log the process consistently. In practice, this means the PLC remains the single source of truth for the live process state, while the stations provide supervision, reporting, trending, and operator interaction on top of the controller's deterministic runtime.

Control Modules

BatchXpert application logic should not access physical I/O addresses directly. Instead, the application interacts with the plant via **control modules**, which provide a standardized abstraction layer above the PLC's digital and analog I/O. This approach separates device behavior from sequence logic and ensures that commissioning features such as permissives, on/off delays, simulation, and alarm supervision are implemented consistently across the project. For example, rather than energizing an output bit, the user program commands an *Actuator* module to start a pump or open a valve; the module then evaluates configured releases/interlocks, applies timing rules, supervises feedback, and finally produces a validated output command and status information for the HMI and for sequence logic.

The most frequently used I/O-related control modules are **Actuator (Act)**, **Digital Input (DIn)**, **Analog Input (AIn)**, and **PID Regulator (PID)**, because they map directly to field devices and measurements. Using these modules consistently provides uniform scaling, filtering, alarming, simulation behavior, and faceplate operation across the entire system. **Examples:** an *Act* module can represent a solenoid valve, pump contactor, or indicator lamp; a *DIn* module can represent a limit switch, level switch (LSL/LSH), or emergency-stop feedback contact; an *AIn* module can represent temperature, pressure, flow, or level transmitters; and a *PID* module can regulate a control valve or variable-speed drive to maintain a setpoint (e.g., temperature or flow).

Not all control modules correspond to a physical I/O channel. BatchXpert also includes modules that provide higher-level, reusable functionality required for batch execution and reporting—for example *Material* modules (material tracking and consumption/production accounting), *Counters (Cnt)* (event or quantity counting), *Message (Msg)* (standardized operator/system messages), and *Software Switch (Switch)* or *Special Values (SVal)* modules (logical flags and shared values). These modules primarily exchange data with sequences and the HMI and may influence process logic without directly driving hardware outputs.

Each module exposes a standardized set of **status signals** that describe its operating state, mode, and diagnostic condition. Sequence logic should use these status signals for interlocks, transitions, and troubleshooting instead of raw I/O, because module status remains consistent across simulation/ignore modes and already reflects configured supervision (delays, plausibility checks, and alarm handling). Common status categories include:

- **General alarm** (e.g., **GAl** / **GAlS**): indicates an active alarm condition and/or a latched alarm requiring acknowledgement.
- **Operating mode**: automatic vs. manual, including maintenance/blocked states or external/forced modes where applicable.
- **Primary state**: module-specific state such as **On**/**Off**/**Mov** for actuators, **Sig** for digital inputs, or **PVal**/**Sp** for analog/PID modules.
- **Interlock / permissive state**: indicates whether releases are present and, if not, which interlock group is preventing activation (module-dependent).
- **Diagnostics**: status-check errors (e.g., **SCE**), warnings, limit violations, sensor plausibility faults, or communication/drive faults depending on the module type.

Each module instance is addressed by its module number (array index) within the corresponding module type (for example, *Act[10]*, *DIn[25]*, *AIn[3]*, *PID[2]*). The symbolic names and descriptive comments shown in the engineering tools are derived from the BatchXpert database—typically by importing a Taglist via the Project Engineering tool—and are then propagated into generated PLC blocks and HMI faceplates. **Practical examples:** *Act[10]* might represent "P-101 Transfer Pump", *DIn[25]* could be "Tank 1 LSH (High Level)", *AIn[3]* could be "TIC-201 Temperature", and *PID[2]* might regulate "Steam Valve CV-201" to maintain the temperature setpoint.

Memory Reset (Factory Reset)

Warning: A **Memory Reset** (factory reset) erases the complete user memory of the PLC. This includes the application program, BatchXpert runtime data, communication data blocks, and all parameter data stored in the controller. **After a Memory Reset, you must download the PLC project again before the system can run.** For step-by-step recovery instructions, refer to the manual **"PLC Restore Manual"**.
A Memory Reset is generally NOT recommended. Perform it only if explicitly required by the troubleshooting procedure and only after you have a verified backup. Please consider the following consequences:

- *The PLC will run with **no application logic** until a complete download is performed. During this time, **no automatic or manual device operation** (valves, motors, pumps, etc.) is possible.*
- *HMI/SCADA communication will be unavailable because the required **communication channel data blocks** are removed and must be restored by downloading the project.*
- *All module parameters and settings (e.g., controllers, valves, drives) are restored to the **startup values contained in the backup/project**. Any adjustments made after the backup was taken will be lost unless restored separately.*
- *All active batches, units, and sequences are reset to the state captured in the backup. **In-progress production cannot be continued**; processes must be restarted and units may require step alignment (advance to the appropriate step) after commissioning checks.*

Compatible Plc Types

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

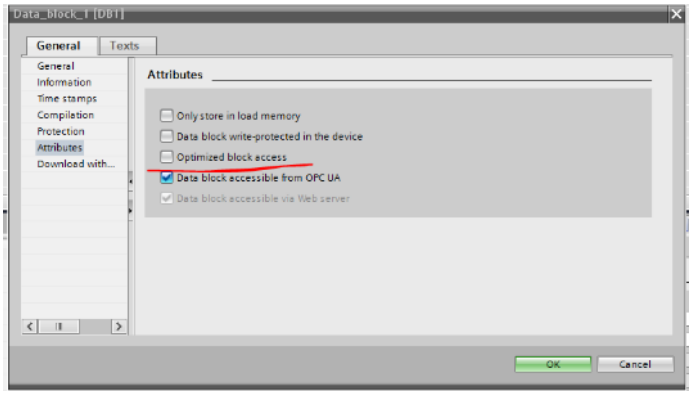
The Projects can be created in “Simatic Manager,” but also “Tia Portal” starting from Version 16 are supported.

Special Considerations for Simatic S7-1500/1200

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.

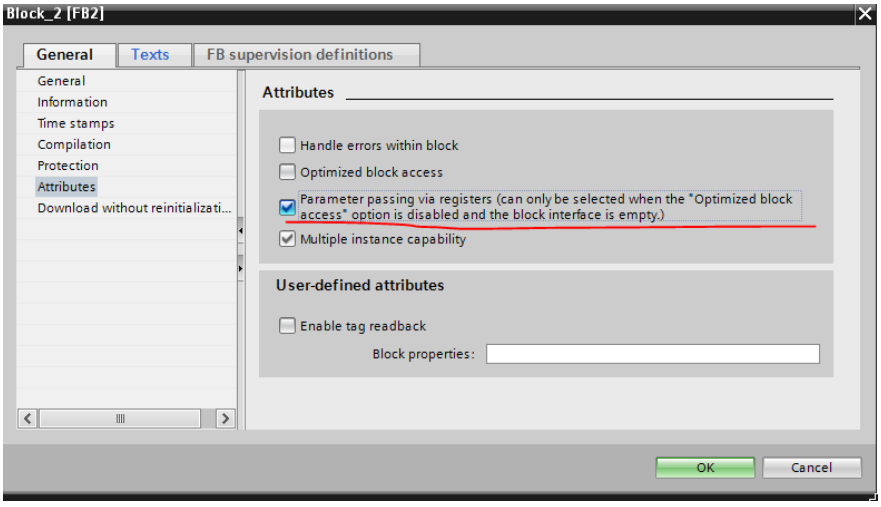
Activate “Parameter Passing via registers” for Unit Function Blocks

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 1 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 :=#UnitDBNc
       RET_VAL   :=#SFC_RetVal
       DB_LENGTH :=#SFC_Length
       ATTRIB    :=#SFC_WriteProt
    SPBI  nodb      Output: Byte
    L     DINT#0
```

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.

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.

Connecting to a S7-1500 or S7-1200 PLC

The SIMATIC S7-1200 and S7-1500 PLC families require specific security and communication settings when establishing data exchange with the BatchXpert system. These requirements differ from the legacy S7-300/400 platform and must be considered during commissioning and troubleshooting.

Starting around 2023 (depending on CPU firmware and TIA Portal version), Siemens introduced additional protection mechanisms such as encrypted communication and certificate-based trust for PG/PC and HMI access. BatchXpert communicates using the classic S7 protocol (S7 connections) for reading and writing data blocks. If the controller is configured to allow only secured communication, these classic connections may be blocked unless the appropriate options are enabled.

If you cannot establish stable communication with a PLC, verify the settings below and then consult the **Diagnosing PLC communication problems** section for additional checks and typical error patterns.

Ping will still work

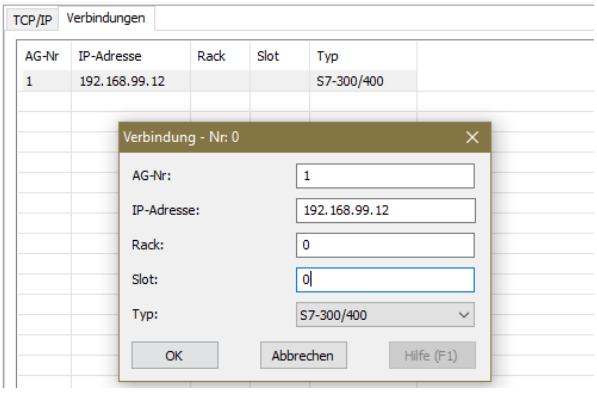
The S7-1200/1500 platform includes more advanced access control than the S7-300/400 series. As a result, an ICMP ping can succeed even when the PLC is rejecting application-level communication. In other words, ping confirms only basic IP reachability (network routing and addressability) — it does **not** confirm that S7 read/write services are permitted.

A common symptom of restricted access is that a connection appears to establish briefly, but subsequent reads/writes of the BatchXpert data blocks fail. This can occur when the PLC access level is set too low (for example, when **Full access for HMI applications** is not enabled) or when only secure communication is allowed.

Therefore, do not use ping as a definitive communication test. Always validate connectivity by performing an online connection in the engineering tool and/or by verifying that an HMI/BatchXpert client can read and write the required non-optimized data blocks without access errors.

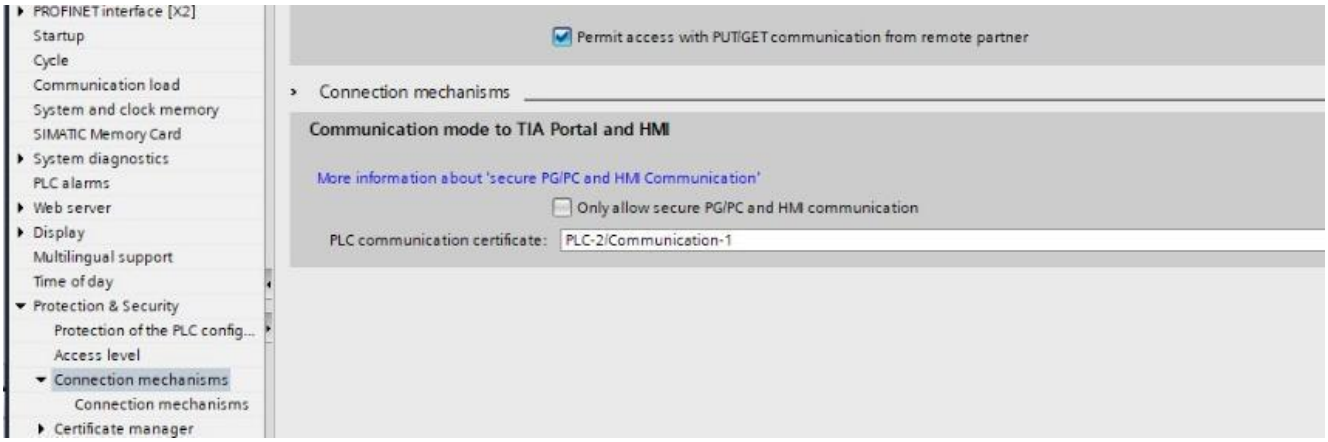
Rack and slot should both be 0

Classic S7 connections require a rack/slot addressing concept to identify the target CPU. On S7-300/400 systems this typically points to the CPU slot in the rack. For S7-1200 and S7-1500 controllers, the CPU is addressed differently and the rack/slot parameters are not used in the same way. **For BatchXpert/VisXpert S7 communication to S7-1200/1500, set both Rack and Slot to 0.** Using any other value can cause the PLC to reject the connection or ignore read/write requests.
Note: In the VisXpert driver, the selected connection type label (e.g., “S7-300/400”) does not affect communication to S7-1200/1500 as long as the IP settings are correct and Rack/Slot are set to 0.



Allow “Get/Put”

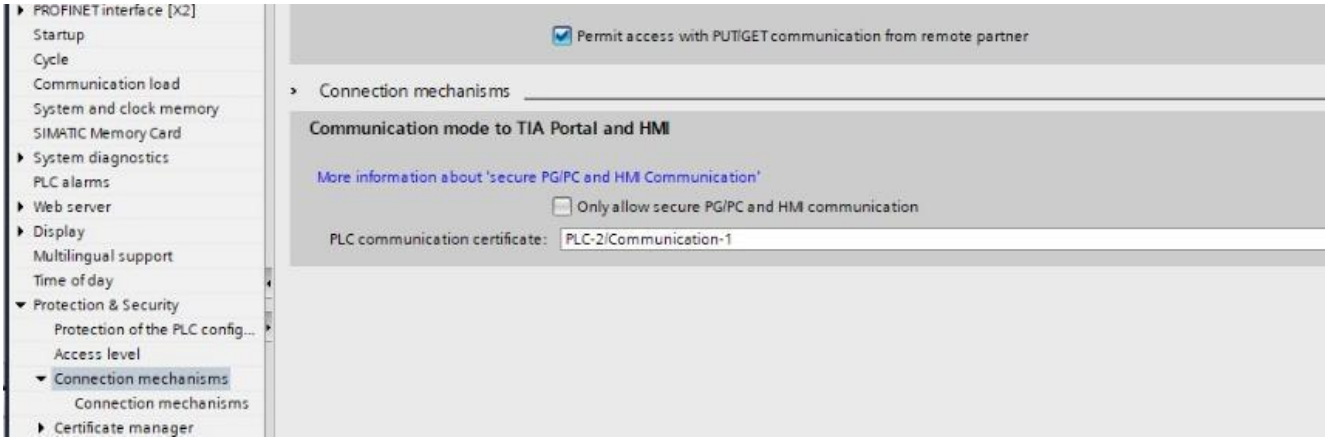
The S7-1500 by default are blocking “Get/Put” connection, which is what S7-connections use for communication. This can impact your ability to communicate with your plc via HMI devices and other PLC via any S7-Connection. We recommend allowing these connections at all S7-1500 plc’s.



Deactivate “Only allow secure PG/PC and HMI communication

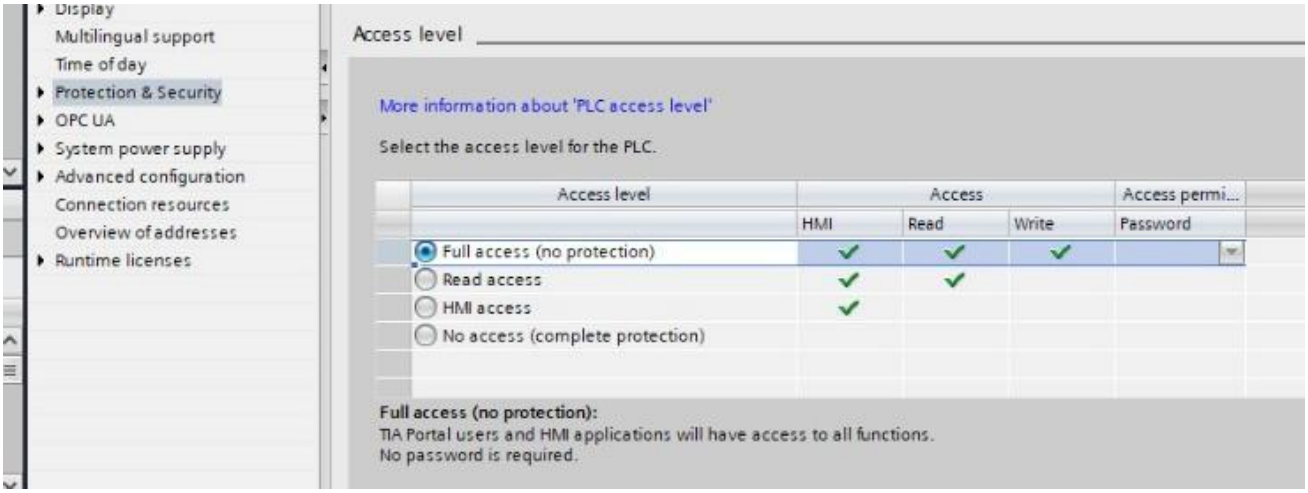
Since firmware version 3.0 of the S7-1500 controllers, the option to only allow “Secure” communication to HMI systems is active by default. This settings means that you MUST exchange certificates between the HMI and the PLC to communicate with each other. This effectively deactivates the “Traditional”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

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



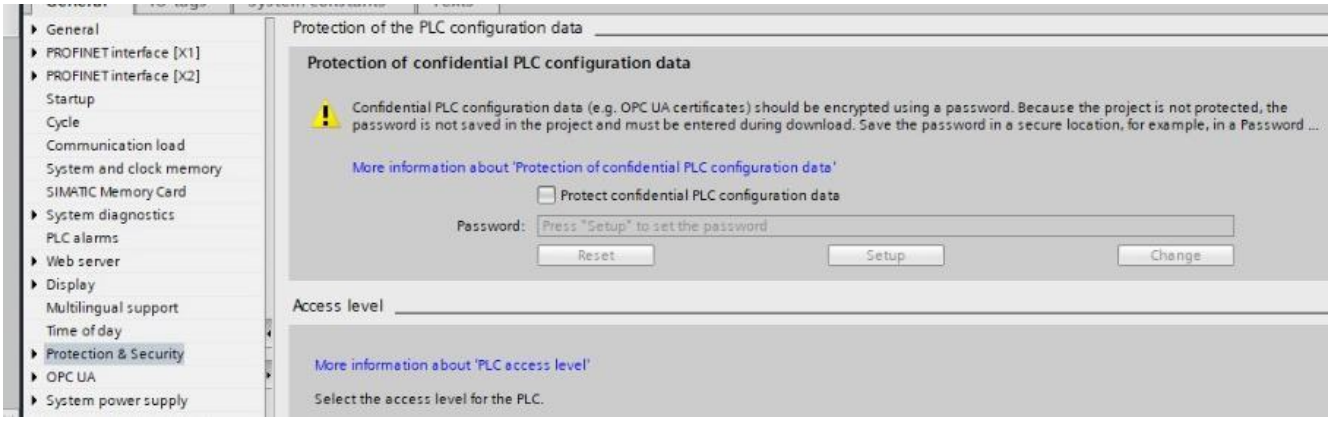
Set “Full Access” to HMI applications

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.



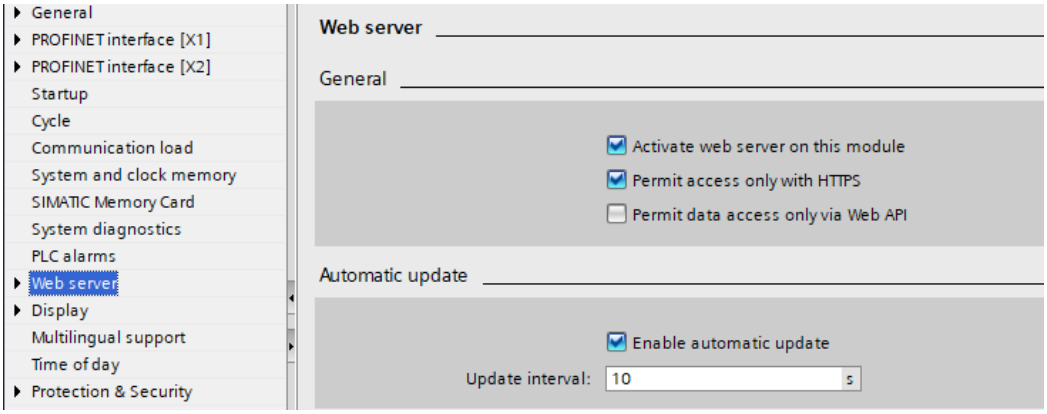
Deactivate “Protect of confidential PLC configuration data

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”

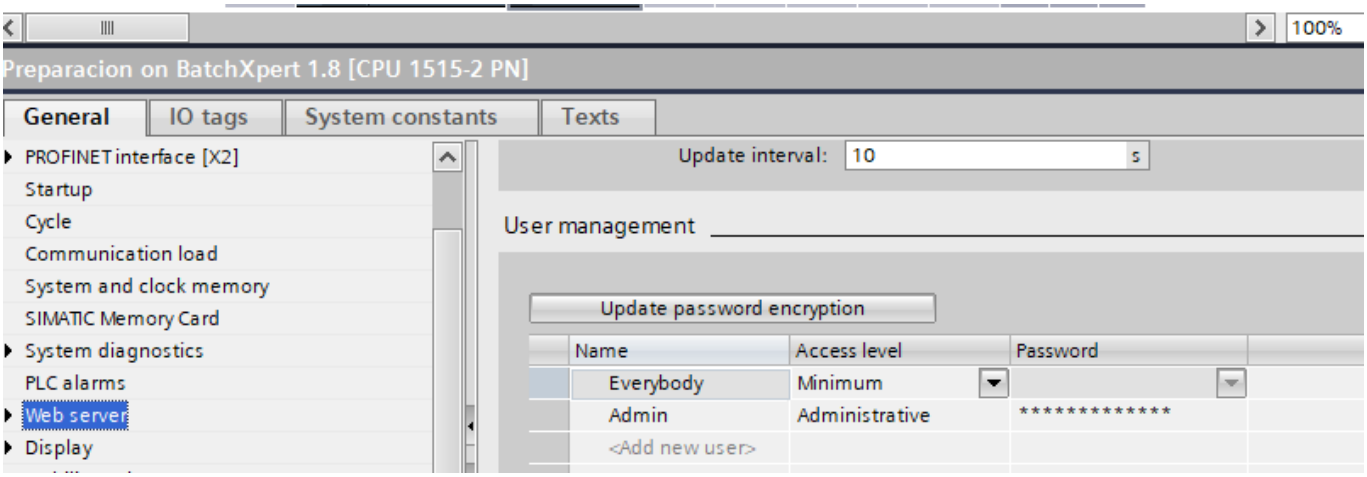


Activate Web Server

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.



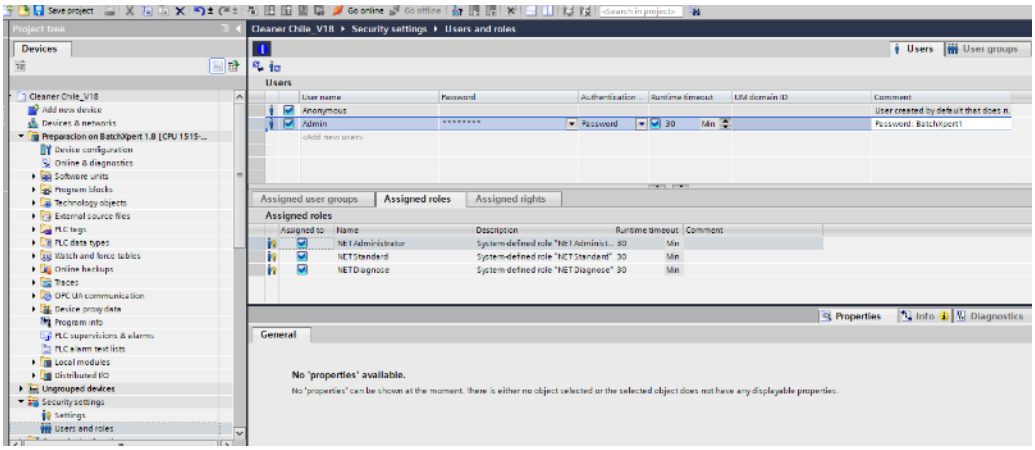
You should always create an "Admin" user with the default password "BatchXpert1"



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"



Simulation of PLC

During engineering you will typically simulate the PLC application to test sequences, validate interlocks, and debug process logic before commissioning on site. The recommended simulation approach depends on the SIMATIC controller family you are targeting. Where available, a real PLC is always the most faithful test environment; however, for most projects a software-based PLC simulation is more practical and enables faster iteration.

PLCSim

Siemens provides a PLC simulation tool named **SIMATIC PLCSIM** for the S7-300/400/1200/1500 families. It is important to distinguish this product from **PLCSIM Advanced**, which is a separate solution with different capabilities and licensing (described later in this chapter).

Standard ("classic") PLCSIM does **not** provide true Ethernet/TCP/IP connectivity. Although TIA Portal may show an IP address for the simulated CPU, engineering communication is handled internally by the Siemens toolchain (for example via local inter-process communication such as named pipes). As a result, third-party HMI/SCADA systems cannot connect to classic PLCSIM over the network, even if they are installed on the same computer.

In practice, only Siemens components that use the integrated SIMATIC driver stack (e.g., WinCC within the same engineering environment) can exchange data with classic PLCSIM. Because BatchXpert/VisXpert communicates via standard S7 connections over TCP/IP, it cannot connect to classic PLCSIM. Therefore, **classic PLCSIM is not suitable for simulating BatchXpert PLC applications**.

Important: Use **PLCSIM Advanced** (or a real PLC) when you need network-capable simulation for BatchXpert.

S7-300/400 series

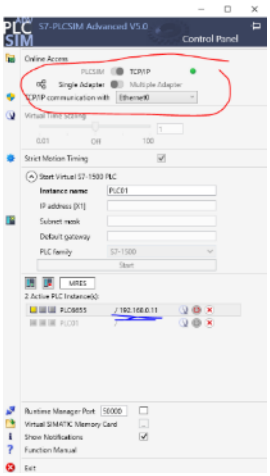
For S7-300/400 (and compatible) projects, several third-party “soft PLC” solutions are available that simulate a controller on a PC and provide TCP/IP communication. These tools can be used to test BatchXpert projects without physical hardware. Most solutions emulate **one PLC instance per computer**; if you need to simulate multiple PLCs concurrently, plan for multiple PCs or virtual machines.

- 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 SIMATIC S7-1500 PLC projects, we recommend **SIMATIC PLCSIM Advanced**, which is a different product from classic PLCSIM and is designed for network-capable controller simulation. PLCSIM Advanced can emulate an S7-1500 CPU with full TCP/IP connectivity, allowing BatchXpert/VisXpert and other external clients to connect using standard S7 communication. Because each simulated CPU can be assigned its own IP address, it is also possible to run multiple PLC instances on a single engineering PC (subject to the licensing and performance limits of the host system).

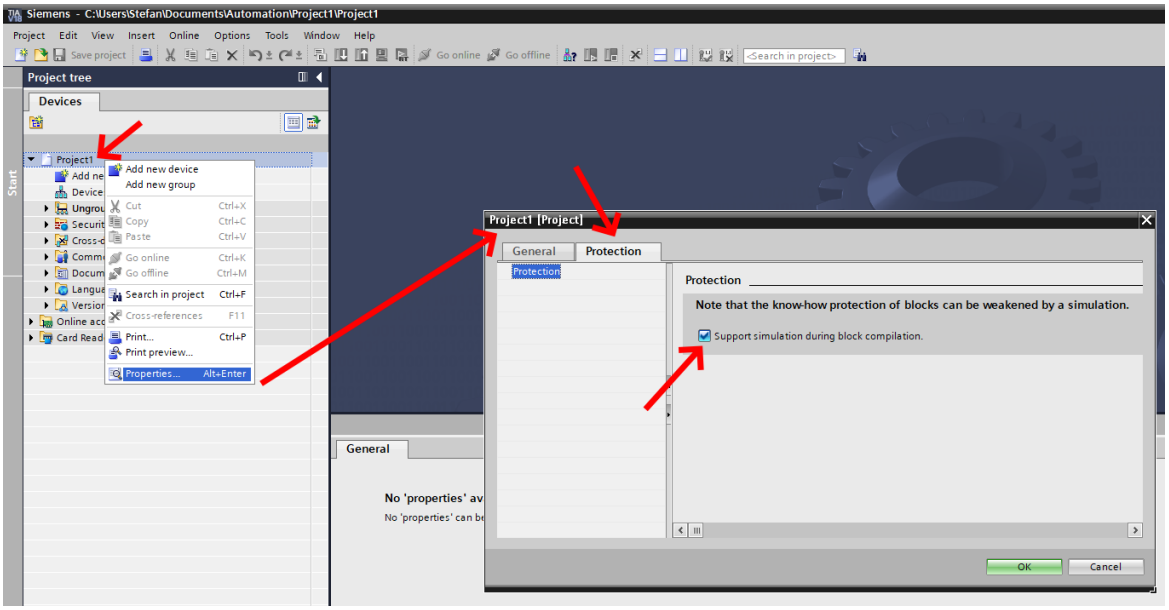


To allow external clients to reach PLCSIM Advanced over a network interface, set the **Online access** interface in the Windows/Siemens “Set PG/PC Interface” (or TIA Portal access settings) to **TCP/IP** instead of **PLCSIM**. If this setting remains on PLCSIM, the simulation is accessible only through the local Siemens interface and behaves like classic PLCSIM (i.e., without usable network connectivity for third-party applications).

Note that the IP address of a PLCSIM Advanced instance is **virtual** and independent of the IP address of your computer. It is the address you assign when creating or starting the simulated CPU. Ensure that your engineering PC (or virtual network adapter used by PLCSIM Advanced) is configured so that the chosen simulated subnet is reachable from the BatchXpert/VisXpert station that will connect to it.

In addition, you must enable simulation support in the TIA Portal project so that blocks can be compiled and downloaded to PLCSIM Advanced. Open the **project** properties (select the topmost node in the project tree), then under **Protection** activate **Support simulation during block compilation**. Without this option, the download may be rejected or the simulated CPU may not execute the compiled blocks as expected.

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 form the Simatic library, such as “Continuous regulator” blocks and communication blocks, are generally not compatible.

NetToPlcSim

NetToPlcSim is a third-party utility that can provide **limited TCP/IP connectivity** to **classic SIMATIC PLCSIM** by bridging network S7 requests to the local PLCSIM instance. In some small, single-PLC test setups this may allow an external client (for example, a lightweight HMI test) to read and write selected data blocks even though classic PLCSIM does not natively support network communication.

Limitations and risks: NetToPlcSim does not emulate a full network-capable PLC. In particular, it typically does **not** support PLC-to-PLC communication scenarios (S7 communication between two simulated controllers) and may show inconsistent behavior under load (timeouts, reconnect loops, unexpected read/write errors). Because of these limitations and the potential for hard-to-diagnose simulation artifacts, **this solution is not recommended** for BatchXpert projects. Prefer **PLCSIM Advanced** or a real PLC for reliable, deterministic testing.

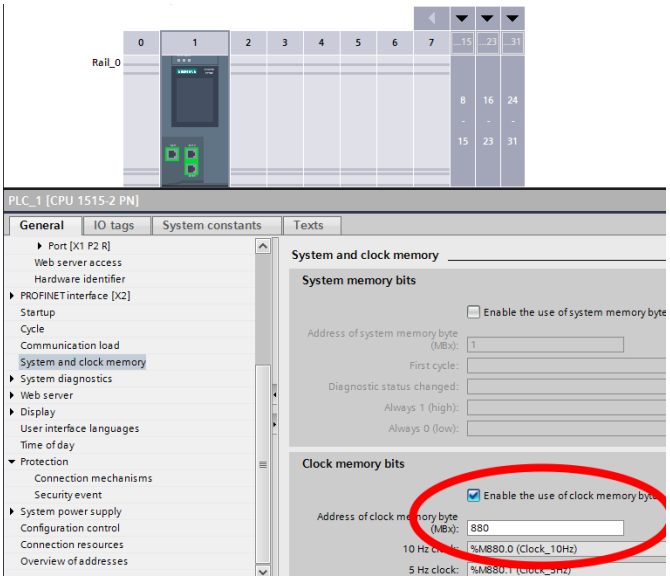
NetToPlcSim can be downloaded here: www.nettoplcsim.sourceforge.net/index.html

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.

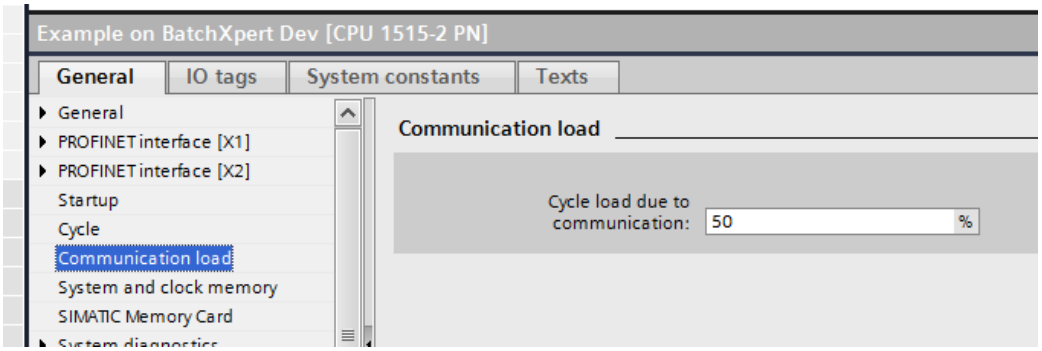
S7-1500 series

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.

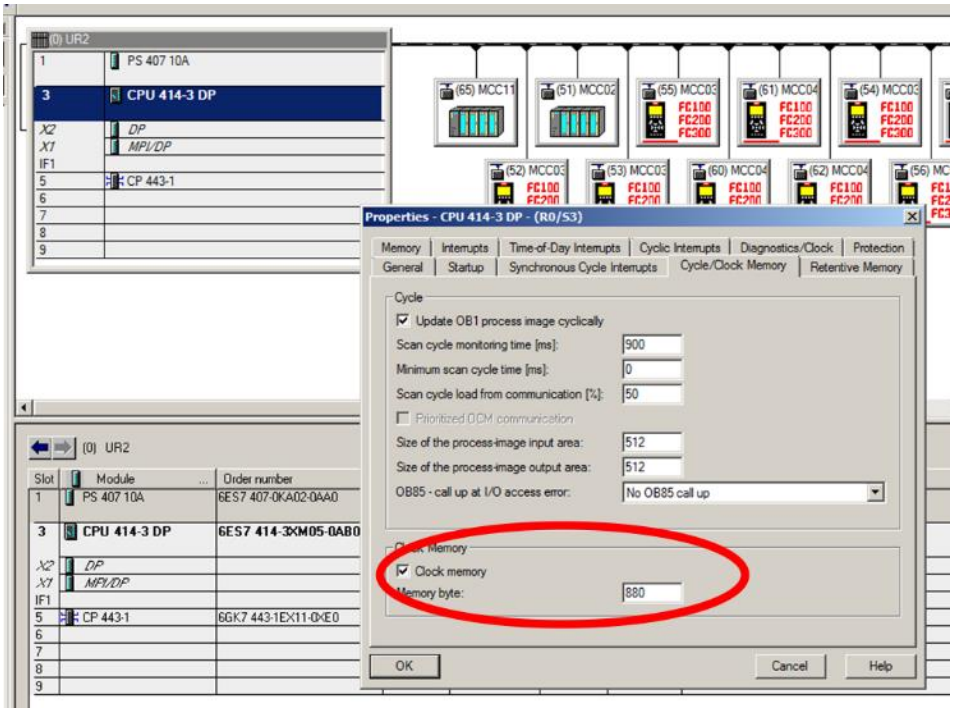
Connection Settings

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

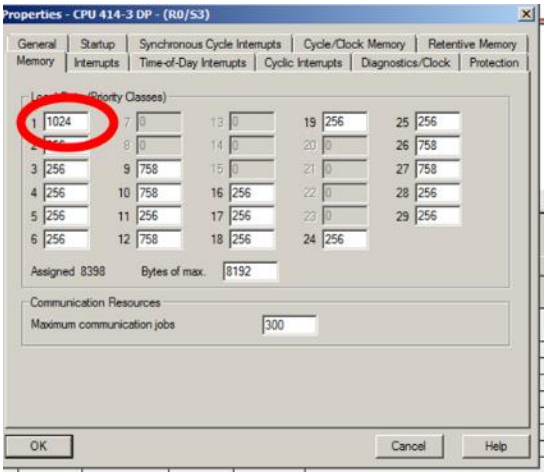
Please review Connecting to a S7-1500 or S7-1200 PLC for more information.

S7-300/400 series, and compatible

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

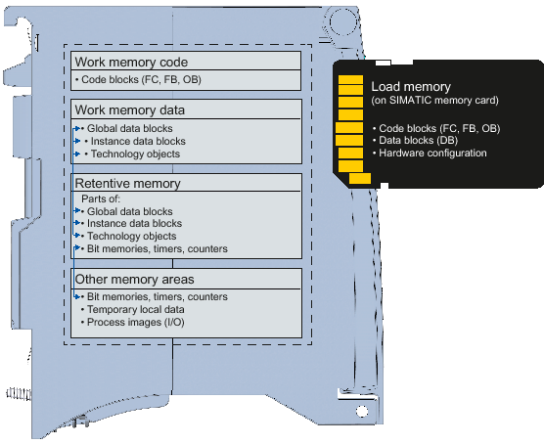


The “Local Data” should also be adjusted to at least 1024 for Priority Class 1, on PLC’s where this can be adjusted. BatchXpert can run with less local memory, but generally this amount of memory is required.



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

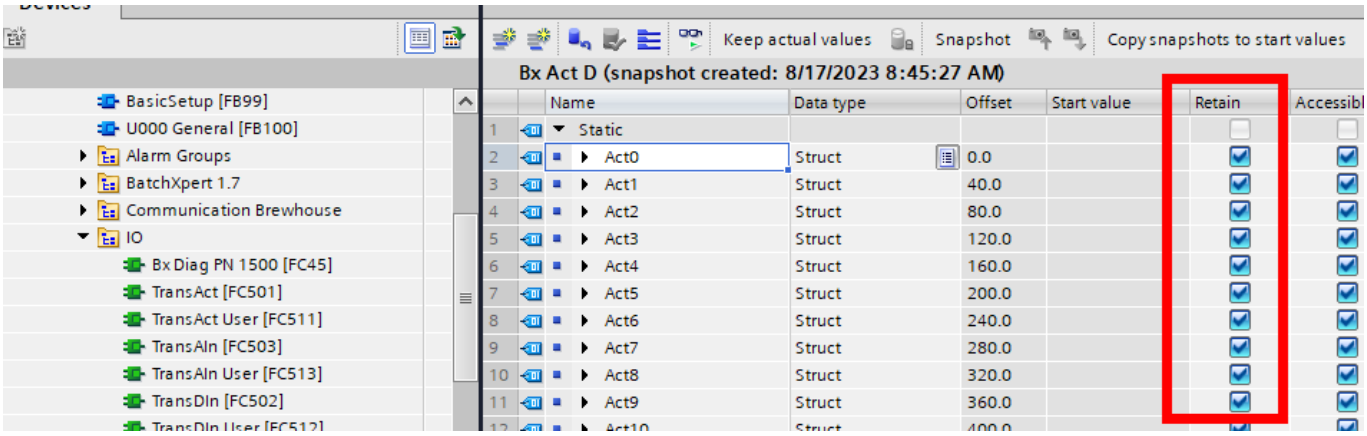
The S7-1500 and S7-1200 series have comparatively little retentive memory integrated into their CPU's. This PC series also has a higher memory footprint overall and does consume generally more memory and also retentive memory than the previous S7-300 and S7-400 series of PLCs.

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.



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 byte
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.

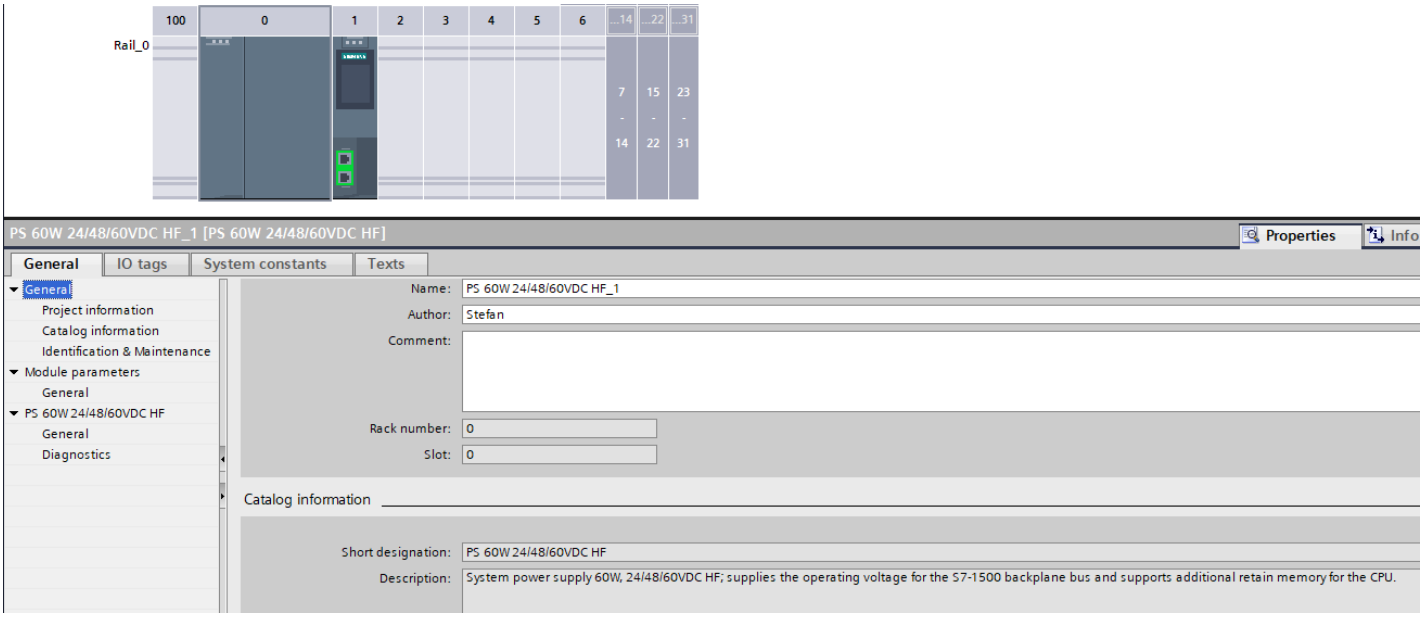


Ilustración 2 the Battery backed power supply in your Hardware configuration

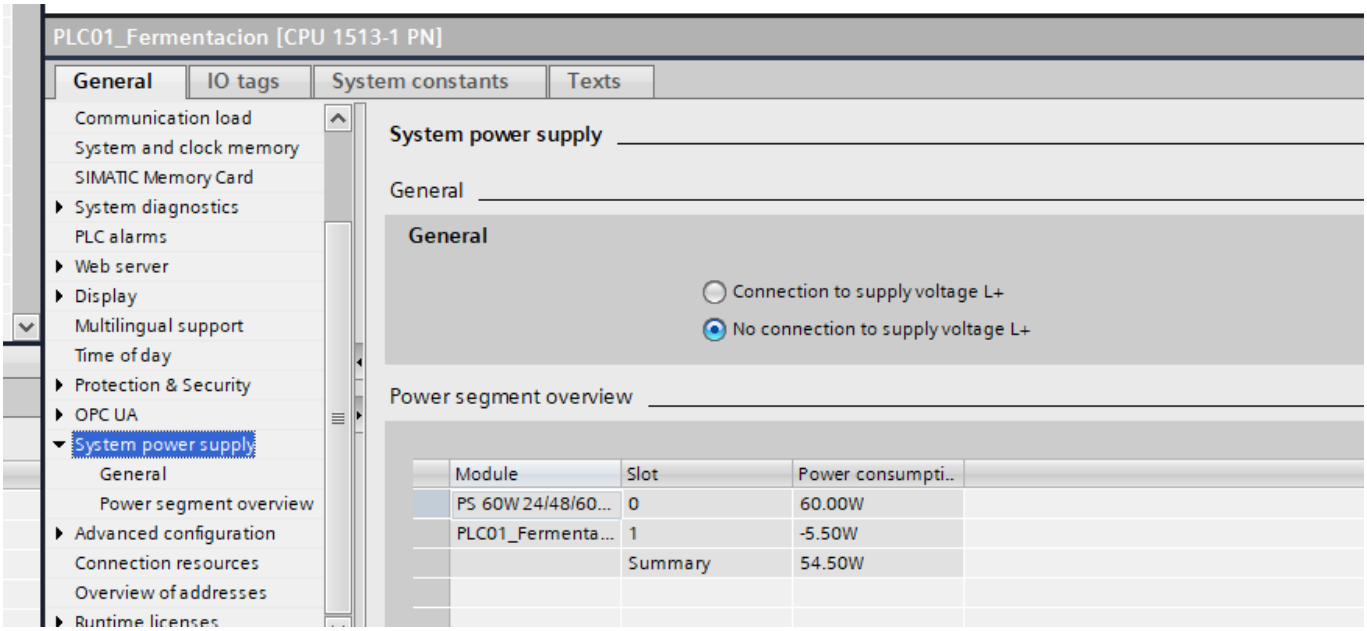


Ilustración 3 You MUST select "No connection to supply voltage L+" to enable full memory retention

Objects	Load memory	Code work-memory	Data work-memory	Retain memory	Mc
	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	

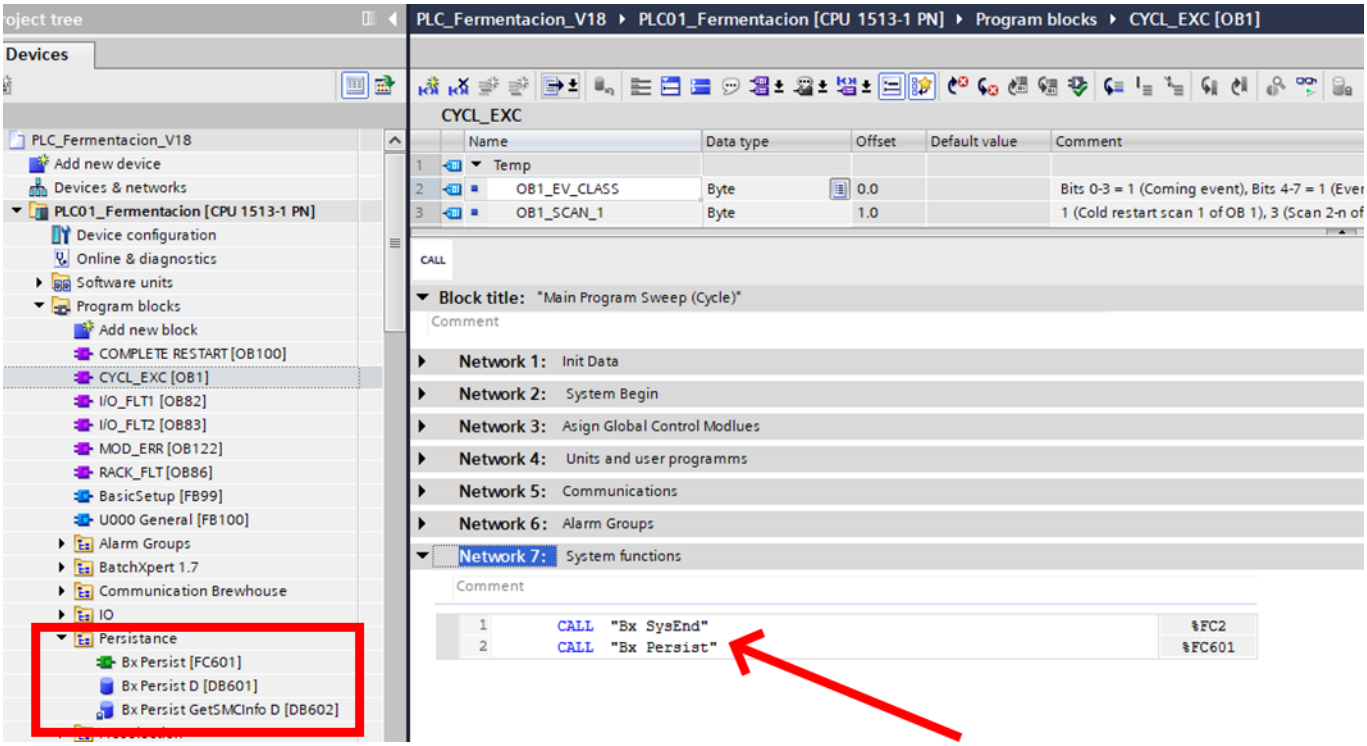
Ilustración 4 All memory is now "Remanent"

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.



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.

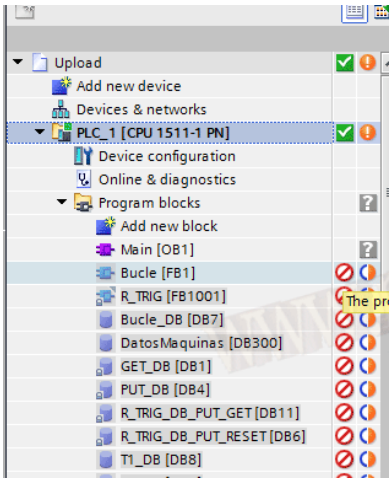
The memory card “write” conditions are as follows:

- 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.

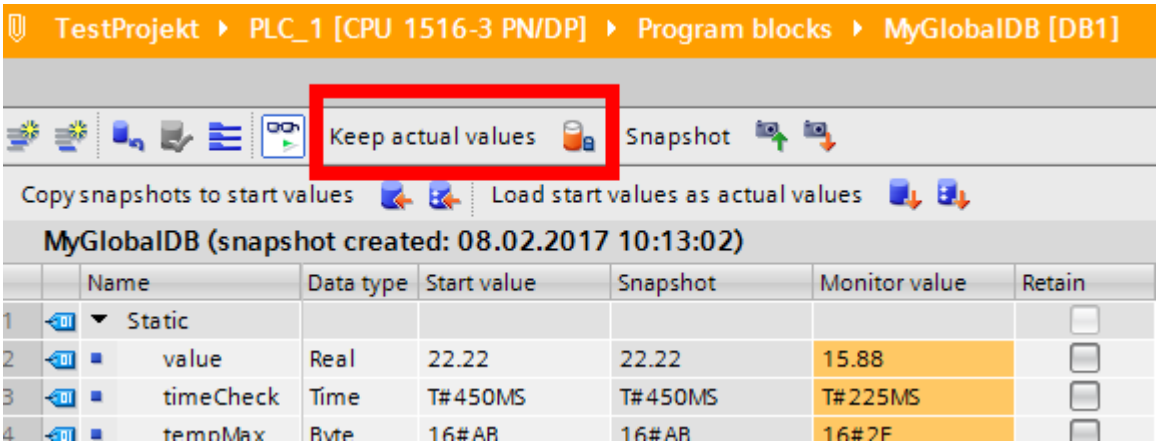


Since “Bx Persist” rights data blocks on the memory card with their current values, this means that these data blocks will appear as “Different” or “Changed” when going online with your project. The reason for this is that you are technically changing the data block starting values on your memory card 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.



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

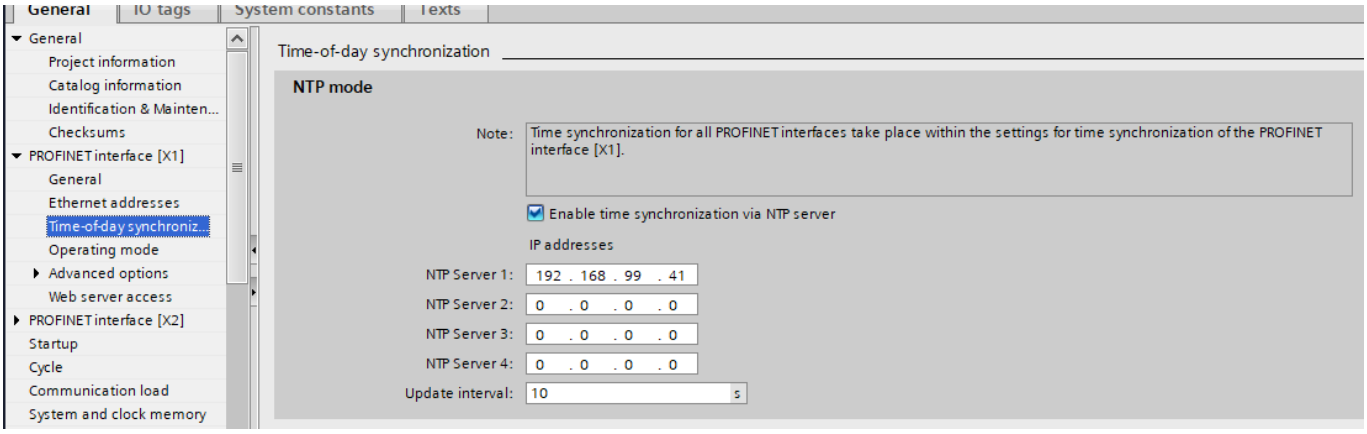
Due to the mechanism that BatchXpert registers its historical data, the Time Zone of the PLC is not relevant, and all-time stamps are always considered as local time of the operating stations. This is done to make the time stamping in the PLC more robust, since most of the time, the PLC time zones are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

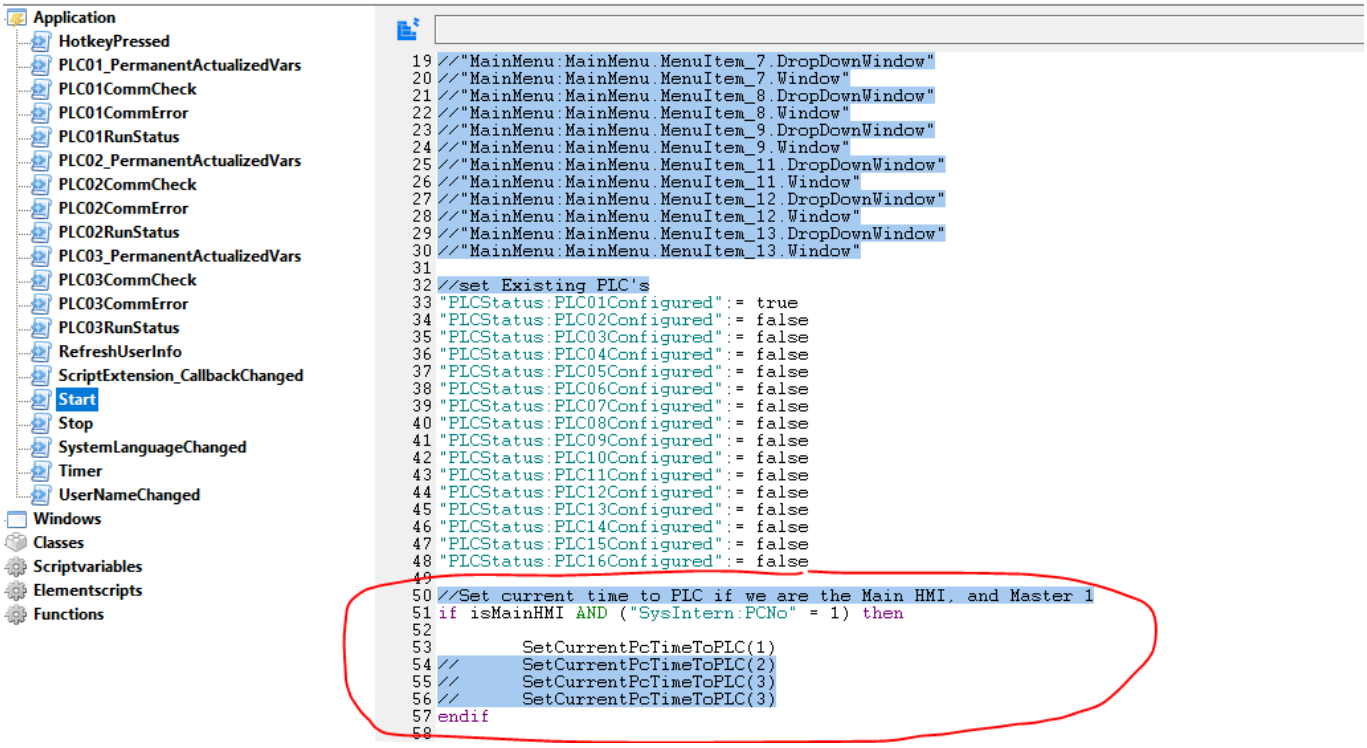
The BatchXpert PLC framework includes a mechanism to set the local module system time of your PLC from an HMI application. The HMI application provides several scripts to automatically send the current time to the PLC and request setting this time as system time on the controller. Usually this is only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



PLC Framework Structure

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.

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.

The “Trans xx” blocks for IO signal transfer

Many BatchXpert control modules provide an abstraction layer between the PLC program and the physical I/O (for example *Actuators*, *Digital Inputs*, and *Analog Inputs*). Rather than reading inputs and writing outputs directly in application code, BatchXpert expects field signals to be mapped to the control modules’ external (**x**-) signals. This approach standardizes scaling, filtering, simulation, diagnostics, and alarm handling across the project and improves maintainability by keeping device-specific behavior inside the corresponding module.

The **Transxx** transfer blocks implement this mapping in a consistent, auditable way. They copy physical input values (e.g., actuator feedback contacts, discrete sensor states, analog raw values) into the corresponding module **x**-signals and, in the opposite direction, copy the modules’ calculated output commands (e.g., **Out** or analog setpoints) to the PLC output image. In addition, transfer blocks are typically the correct location to apply simple signal adaptation that is tied to wiring (such as inversion or selection of an alternative source), provided that project-specific logic is kept in the **User** blocks as described below.

Taglist

I/O assignment is typically maintained in the **Taglist** (often edited in Excel). After updating the Taglist, import it into the BatchXpert database and regenerate the corresponding **Trans IO** blocks. Regenerate whenever you change I/O addresses, add or remove devices, change module types, or modify hardware-related configuration (for example, adding feedback signals or changing signal directions).

Important: Generation, import, compilation, and download of the generated transfer blocks are manual engineering steps and are not performed automatically.

Import, compile and download

The Project Engineering tool generates transfer block source files (typically STL/AWL). Import these sources into the PLC project, compile them, and download the updated blocks to the controller. In many cases, you can overwrite the existing transfer blocks with the regenerated versions; however, you should confirm that the block interfaces and symbol names match the versions currently used by the PLC framework and any user code.

If the regenerated blocks are not compatible with the versions currently in the project, merge the changes manually. Overwriting is safe only if you have not modified the generated **System** blocks. As a rule, do **not** edit “System” transfer blocks; place all project-specific adaptations in the corresponding **User** blocks. This separation ensures you can regenerate and redeploy the System code at any time without losing manual adjustments.

Modifications in generated code

Some projects require deviations from the standard generated mapping (for example, special simulation behavior, pulse shaping, signal inversion, redundancy selection, or devices controlled via communication rather than hardwired I/O). Because the **System** transfer blocks are regenerated, any direct edits would be overwritten during the next generation cycle. Therefore, each transfer block is provided as a pair: a generated **Trans IO System** block and a non-generated **Trans IO User** block intended for customization.

The PLC framework calls the blocks in sequence: first the generated **System** block, then the corresponding **User** block. The User block can overwrite assignments made by the System block (for example, replacing a source signal) and can add additional logic that prepares signals for the control modules before the modules execute. Because User blocks are not generated, their content is preserved across regeneration cycles.

- Redirect a module output to a different physical output (e.g., temporary rerouting during commissioning).

- Derive an input from alternative sources (e.g., inverted wiring, combined permissives, or simulated signals) or intentionally leave an input “floating” so a module generates default feedback behavior.
- Implement project-specific signal conditioning (filtering, debouncing, pulse stretching), simulation logic, or communication interfaces to third-party equipment.

The “Trans IO User” block is the correct place for customized I/O-to-control-module transfer logic.

Special Considerations for Analog Modules

Analog I/O channels (used by **Analog Input** and **PID Regulator** control modules) often require project-specific scaling and engineering-unit conventions (e.g., raw counts vs. physical units, 4–20 mA ranges, or drive telegram scaling). These settings are described in the corresponding module chapters. Please refer to the Analog Input Scaling and PID Regulator Output Scaling chapters for more information.

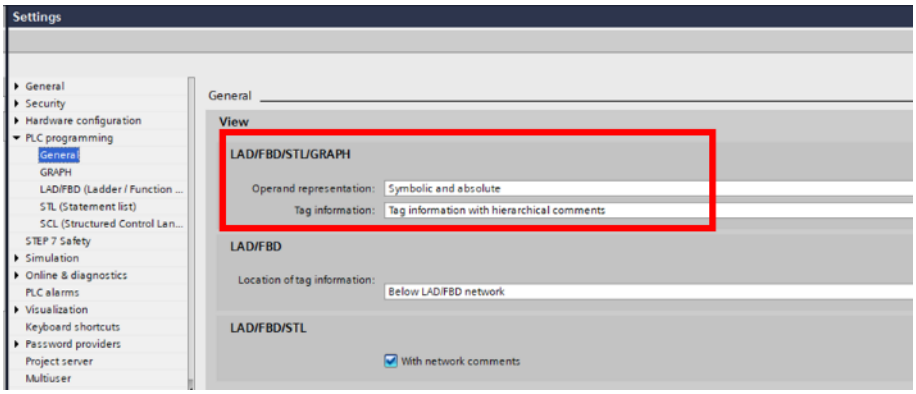
Using TIA-Portal

While the older SIMATIC Manger 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.

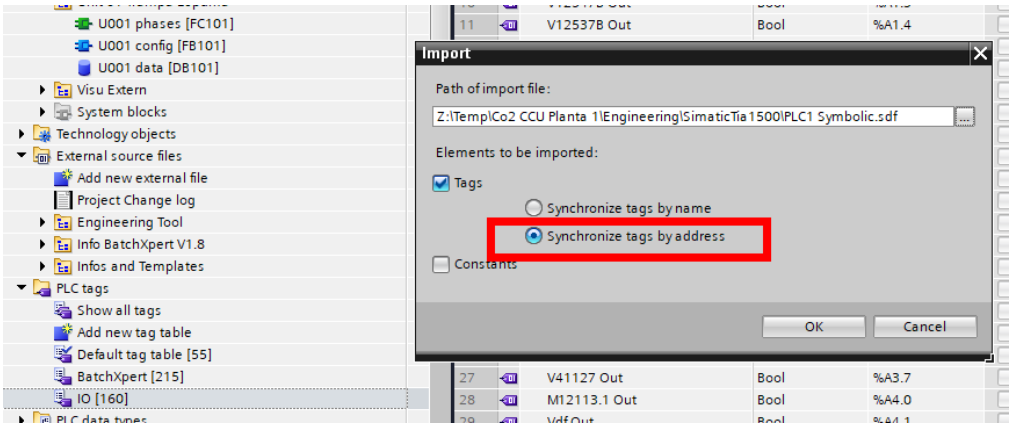
Use Hierarchical Data block comments

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.



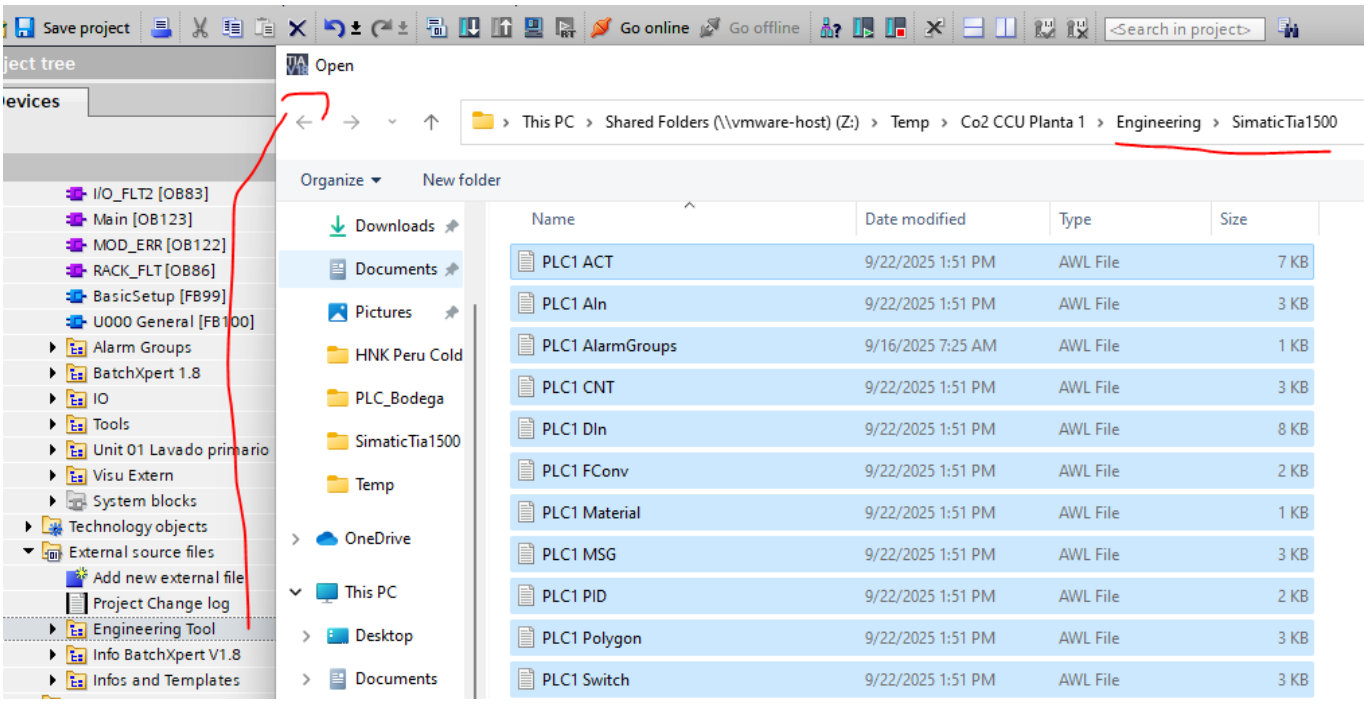
Import Symbols always “By Adress”

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.



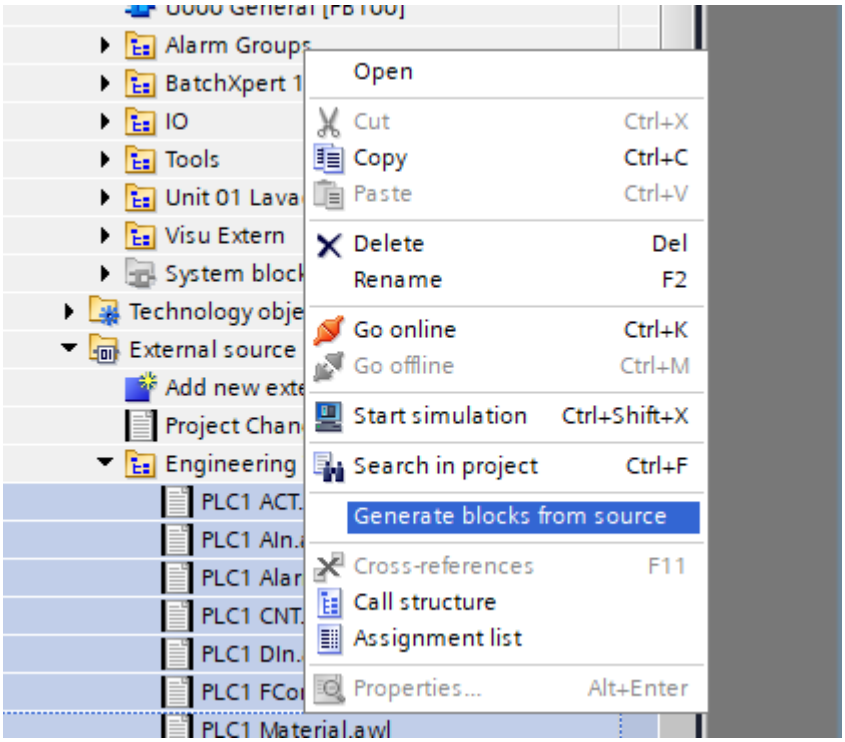
Importing and compilation of Generated block sources

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 overrides 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.

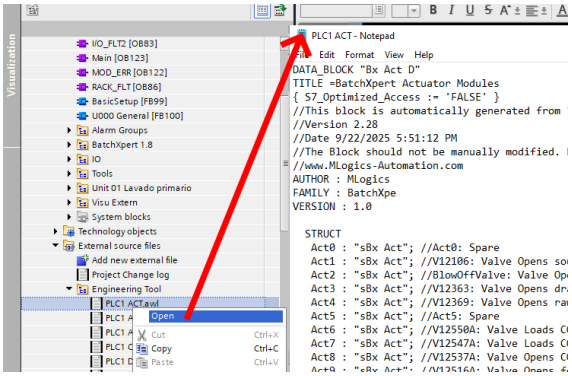
The first thing to note is that you should never compile an already existing data block, since this will override the existing data block, and requires a download with default values into your PLC.

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.

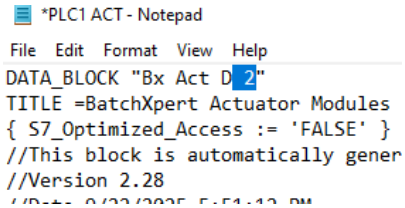
1. Open the data block source file

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



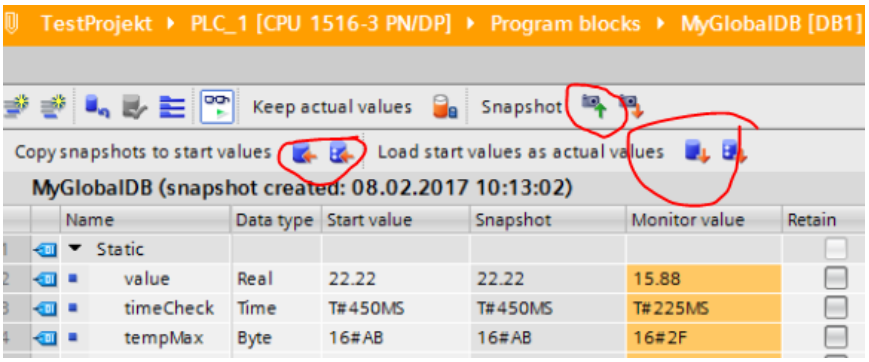
2. Change the Blocks symbolic name

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.



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)

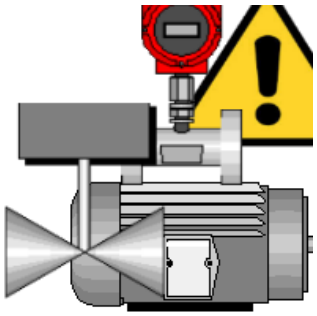
Time values

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, control module data (e.g., Actuator, Analog Input, PID) is stored in dedicated data blocks as arrays of structures. For readability, these arrays can be generated in an “unrolled” form where each element is exposed as a separate structure instance, allowing meaningful comments to be applied per module instance. Regardless of the chosen representation, it is essential that the internal structure and ordering of the System objects remains unchanged so the PLC framework can access the data reliably. BatchXpert Engineering tools can generate these unrolled data blocks with descriptive comments derived from the database; this is the recommended approach because it significantly improves code readability and commissioning efficiency.

Execution of the Control Modules

In the BatchXpert PLC framework, control modules are executed automatically by the system runtime. The programmer does not need to call module function blocks explicitly in user code; instead, the PLC framework iterates through the configured module data blocks and processes every instantiated module each PLC cycle.

You can reduce the execution load and memory footprint by limiting the number of configured modules of a given type (for example, by generating a smaller array size for that module type in the corresponding data block).

In addition, you can cap how many modules are processed per cycle by adjusting the processing limits in **Bx SysBegin**. This is useful for performance tuning on smaller CPUs or when commissioning a subset of a project.

```
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

For each control module type, BatchXpert provides one data block that contains all instances of that module type. Internally, the data block is an ordered list (array) of module structures, where the array index corresponds to the module number. Each module structure contains the module’s **Commands**, **Status**, **Parameters**, and internal helper values used for timing, alarm handling, and simulation.

Commands

This section describes the **command signals** that the user program writes to request actions from the BatchXpert control modules (e.g., start a status check, request automatic mode, enable an alarm). Command signals are subject to module-specific rules and should be written only as described in the corresponding module chapter.

Most commands are **write-only** from the perspective of the user program. The PLC framework evaluates these commands during the cycle and then resets them automatically to prevent unintended “latched” commands caused by programming errors.
Because commands reset each PLC cycle, they must be asserted continuously (or set each cycle) while the requested action is needed. This design prevents commands from remaining active indefinitely. In LAD/FBD/STL, it is therefore common to use a one-cycle set pattern (e.g., **S** instruction) where appropriate, while ensuring the command is reasserted as required by the sequence logic.

Status

The **Status** section contains signals provided by the BatchXpert framework to inform the user program about the current state of each control module (e.g., On/Off state, alarm state, operating mode, filtered signal values). Status signals are generally **read-only** and must not be written by the user program. Use these signals to drive sequence logic, interlocks, and diagnostics. Refer to the specific module chapter for the recommended status signals to use (for example, prefer **.On**/**.Off** over raw feedback inputs).

Parameters

The **Parameters** section contains configuration values for the control modules (e.g., delay times, limit thresholds, scaling constants). In normal operation, these parameters are set and maintained through the module faceplates on the BatchXpert HMI, not by PLC logic.
For a limited set of parameters, projects may choose to overwrite values from PLC code to enforce fixed limits (for example, to clear simulations on emergency stop). This approach should be used sparingly because it contradicts the standard BatchXpert philosophy of operator-adjustable module configuration and can lead to confusion if the HMI displays a value that is immediately overwritten by the PLC. If parameter overriding is required, document it clearly and apply it consistently (preferably in one central location).

Alarms

Many control modules support alarm supervision. An active alarm condition is indicated by the status signal **GAl** (General Alarm). **GAl** remains TRUE as long as the underlying alarm condition is present.
The status signal **GAlS** (General Alarm Saved) is the latched alarm indicator. It becomes TRUE when the alarm is triggered and stays TRUE until an operator acknowledges it—either on the individual module faceplate or globally via **QuittAll** (M878.7), depending on project configuration.

Alarm evaluation is module-specific and typically includes a configurable delay timer to prevent nuisance alarms during expected transitions (e.g., actuator movement, filter settling, startup conditions). Refer to the corresponding module chapter for the exact alarm criteria and timing behavior.

Ignore

Most control modules include an **Ignore** mode (parameter **Ign**). When Ignore is active, the module suppresses alarm generation (i.e., **GAl**/**GAlS** are not set) even if an alarm condition is detected. Depending on the module, the underlying condition may still be visible through other status signals; therefore, Ignore should be used only under controlled circumstances (e.g., commissioning) and in accordance with site procedures.

Simulation

Most control modules also provide a **Simulation** mode (parameter **Sim**). When Simulation is active, the module disconnects from physical PLC I/O (or external signals) and instead uses simulated values provided via the HMI faceplate. This enables functional testing of sequence logic, alarming, and operator interfaces without

manipulating field wiring. The exact simulation behavior is module-dependent (e.g., simulated feedback for actuators, simulated process value for analog inputs).

External signals

Many control modules require values that are supplied from outside the module (for example, physical feedback contacts or raw I/O values). These are labeled with an **x** prefix to indicate an **external** signal (e.g., **xSig**, **xFBa1**). External signals are typically written by the corresponding **Transxx** transfer blocks and should not be written directly by application logic unless explicitly intended.

For example, an actuator receives physical feedback via external signals such as **xFBa1** and **xFBa2**. The actuator module processes these inputs and provides derived, application-ready status signals such as **On** and **Off**. Because the derived status signals are validated, filtered, and consistent across simulation/ignore modes, they should be preferred in application logic over raw external signals.

Actuators (Act)

Actuators represent controllable field devices that can be energized or de-energized by the automation system. Typical examples include on/off valves, pumps, motors, solenoid-driven devices, indicator lamps, and other equipment controlled via a single digital output. Depending on the device design, an actuator can be configured with optional feedback signals (e.g., open and/or closed limit switches) to confirm the commanded position.
An actuator may also be implemented as a virtual (logical) actuator with no direct hardwired I/O (no physical output and no feedback inputs). Virtual actuators are useful when the underlying equipment is controlled through analog signals (e.g., 4–20 mA) or communication interfaces (e.g., fieldbus/drive telegrams), but the application benefits from a discrete on/off command, interlocking, alarming, and standard HMI faceplate behavior. The association between an actuator and its physical I/O (where applicable) is typically realized in the transfer blocks (e.g., FC501 “TransAct System” and the corresponding “Transxx User” block).

Operating Principle

An actuator is driven by one digital output (command) and may use up to two digital inputs as feedback to confirm the resulting device state (typically “Open/On” and “Closed/Off”). Feedback is optional; when configured, the actuator continuously compares the commanded state to the feedback signals and supervises the transition using an alarm delay timer.

- If the output command is TRUE (energized) and the expected ON feedback is not received, the alarm delay timer starts.
- If the output command is FALSE (de-energized) and the expected OFF feedback is not received, the alarm delay timer starts.
- If both ON and OFF feedback signals are TRUE simultaneously (invalid state), the alarm delay timer starts.
- If the alarm delay timer expires without the feedback returning to a valid, expected state, the actuator raises an alarm.

When the feedback signals match the expected state, the actuator sets its internal status signals (e.g., **On** and **Off**) accordingly. These internal status signals are the recommended interface for application logic and should be used to determine whether the actuator has reached the commanded “Activated” (On) or “Deactivated” (Off) state.

Simulation

When an actuator is in **Simulation** mode, the module generates consistent, internally simulated feedback and status information (On/Off) independent of the physical feedback inputs. As a result, the actuator will not raise feedback-related alarms, and the module behaves as if the device always reaches the commanded state.

Ignore

In **Ignore** mode, the actuator suppresses the generation of the alarm signals (e.g., **GAl** and **GAlS**) even if a feedback mismatch occurs. However, the actuator continues to evaluate and expose the actual feedback-derived status (On/Off) where feedback is wired, meaning that incorrect field feedback may still be visible to the application and operator but will not trigger alarms.
For most commissioning and testing scenarios, **Simulation** mode is preferred over **Ignore** mode, because Ignore can conceal fault conditions without providing simulated feedback behavior, which may lead to misleading system states during troubleshooting.

Interlocks

Actuators support two independent enabling signals ("releases") that must both be TRUE to permit output activation:

Release 1 (Production Release) is intended for process or operational interlocks that do not present an immediate safety hazard (e.g., sequencing constraints, permissives, quality-related conditions). If required, Release 1 can be temporarily overridden by an operator using the actuator's *Emergency Release* function on the HMI faceplate, subject to project safety rules and site procedures.

Release 2 (Safety Release) is intended for safety-critical interlocks and must **not** be bypassed through the HMI. Use Release 2 for conditions that protect personnel, equipment, or product integrity (e.g., emergency stop chains, manhole/key-switch circuits, access doors, tank entry detection, or other certified safety devices).

You can find some examples about general Actuator interlocks that we recommend implementing in Actuator Interlocks. You can find some examples here.

Automatic Activation

The "ACo" (Automatic Control) signal is the signal that should be used in your application to indicate to the Actuator that it should be activated, if possible. This signal is reset automatically each Plc cycle, so that you can use the "S" (Set) instruction in your application to activate this signal.

External Activation

The "ExCo" (External Control) is a signal that you can use if your actuator includes local switching equipment, for turning the actuator on from the "field". This is the case for example for some types of pumps that need a local switch for maintenance, to turn them on during some maintenance operations.

Keep in mind that each activation generates a Manual operations Event in the Historical data registry. Do not

Feedback

The feedback are written by the TransAct blocks, and processed internally. In your user application you can always use the "On", "Off" and "Mov" status signals to check if the actuator is energized or deenergized with its corresponding feedback. In the case the actuator does not include A feedback, it is automatically generated so that you can always rely on the "On" and "Off" signals to have the corresponding state off your actuator.

- If your actuator includes feedback sensors, these signals corresponds to the physical feedback in the field.
- If your actuator does not have feedback sensors in the field, these signals are generated internally to what the status of the actuator should be.

Status Check

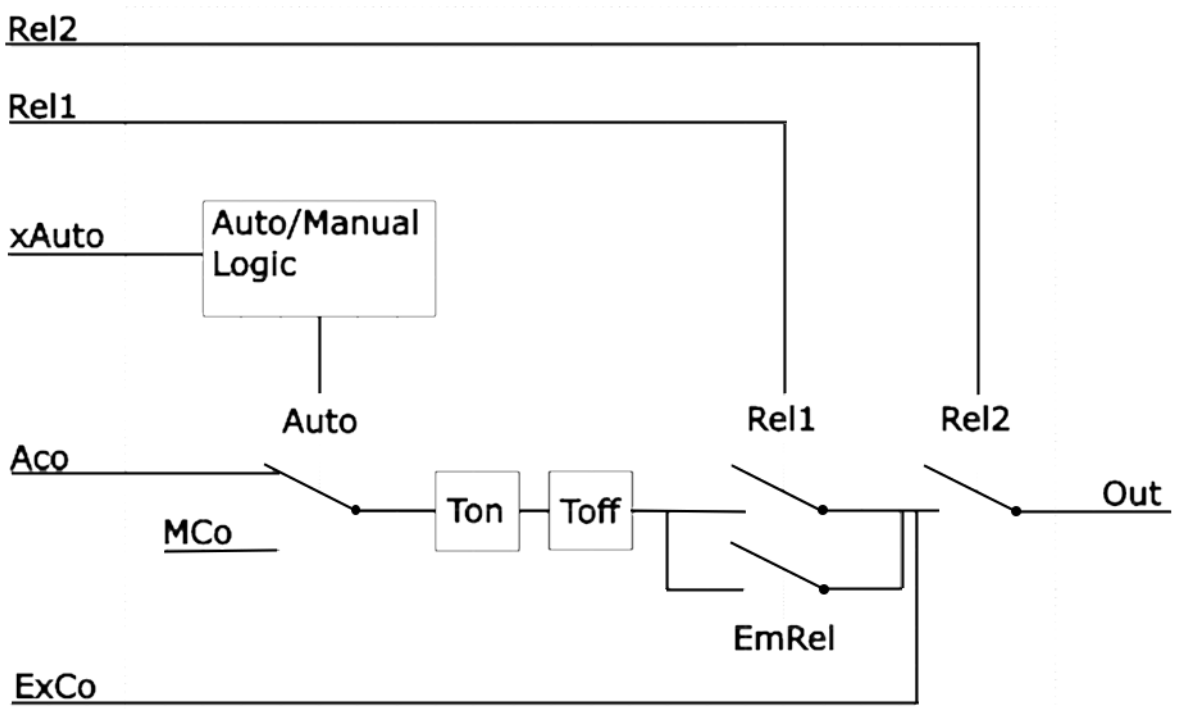
If you set the status check signal, the actuator automatically checks if it is deactivated and has the correct feedback and no error. If the actuator is activated, does not have the off feedback or has even an error, the actuator responds by setting the “SCE” (Status Check error) signal, and by marking the actuator in the HDMI application to make the user aware off the faulty device. You should set this signal during a special step called “Start Check” that should be executed at the beginning of every recipe.

Manual Automatic

Each actuator can be operated in manual or automatic mode. You can retrieve the current operating status from the “Auto” status signal of your actuator. The automatic control signal (Aco) is only considered if the actuator is currently in auto mode.

To request automatic operating mode from your user application, you should set the “xAuto” command in your actuator module. This is usually the case when the corresponding unit is in “run” operating mode.

Function diagram



Activation and Release Logic

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	GAlQuitt	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	InterlockGAl	BOOL	interlock by alarm
2.7	Maint	BOOL	maintenance
3.0	GAl	BOOL	general alarm
3.1	GAlS	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 This is the Automatic activation that you should use in your application logic. This signal is reset for each Plc Cycle. We recommend that you use the “S” (Set) instruction to activate an actuator.
ExCo	External control (e.g., by a switch), effective in automatic and manual. Production-related interlocks are ignored. Safety interlocks are still considered. *Keep in mind that reach activation generates a Manual operation Event in the Historical database.*
SCS	status check start Set SCS during the recipe ‘Start Check’ to verify the actuator is OFF, has OFF feedback, and no errors; otherwise, it raises SCE and flags the device in the HMI.
xFBaOff	feedback Off Corresponds to the physical Feedback sensor signal of your actuator. Written from “TransAct”
xFBaOn	feedback On Corresponds to the physical Feedback sensor signal of your actuator. Written from “TransAct”
Rel	Release Corresponds to the “Production” release. More information in Interlocks
Rel2	release 2 Corresponds to the “Safety” release. More information in Interlocks
xAuto	External automatic This signal is usually connected to the “Run” status of its corresponding unit.

Status

Symbol	Remark
FBaOn	feedback ON intern Internal use only. Corresponds to the Feedback sensor signal. If you want to check the Actuator state, please use the “On” or “Off” signals.
FBaOff	feedback OFF intern Internal use only. Corresponds to the Feedback sensor signal. If you want to check the Actuator state, please use the “On” or “Off” signals.
GAl	**Latched general alarm** flag; remains TRUE until the alarm is acknowledged, even if the alarm condition clears.
GAlS	general alarm saves The module is currently on Alarm, or the alarm is gone, but not yet confirmed.
SCE	status check error Set SCS during the recipe ‘Start Check’ to verify the actuator is OFF, has OFF feedback, and no errors; otherwise, it raises SCE and flags the device in the HMI. Use this signal to exit the “Start Check” phase of your Unit.
Mov	actuator is moving The actuator is currently moving, but has not yet reached its end position.

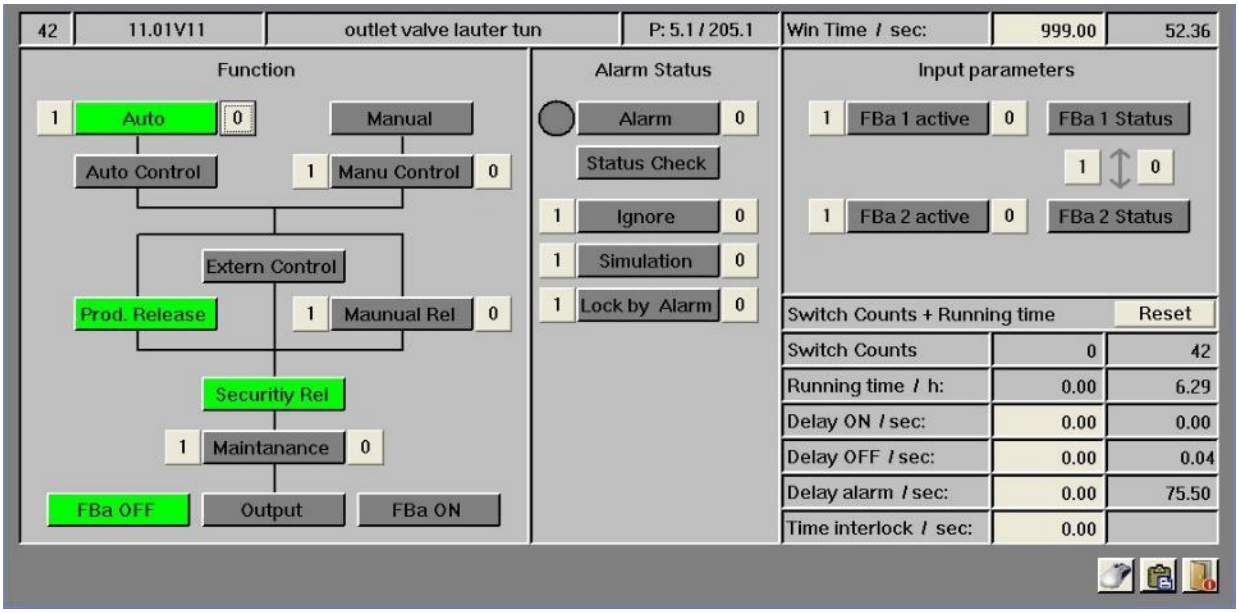
On	actuator is ON The actuator is currently turned on, and the feedback sensor have confirmed this position. If no Feeback sensor is present, this signal is generated internally. Use this signal to verify if an Actuator is On or Off.
Off	actuator is OFF The actuator is currently turned off, and the feedback sensor have confirmed this position. If no Feeback sensor is present, this signal is generated internally. Use this signal to verify if an Actuator is On or Off.
Out	Output The Actuator currently activates its output signal. You can use this to check if its output is turned on, but be aware that this does NOT indicate an actuators real position, use the “On” or “Off” signals for that.

Parameters

Symbol	Remark
FBaChange	change extern feedback (0 FBa1=OFF FBa2=ON / 1 FBa1=ON FBa2=OFF) Obsolete, do not use anymore. Indicates if FBa1 and FBa2 should be switched.
FBa1Active	feedback 1 active Obsolete, do not use anymore . Set automatically by “TransAct”. Indicates if the feedback exists.
FBa2Active	feedback 2 active Obsolete, do not use anymore . Set automatically by “TransAct”. Indicates if the feedback exists.
GAlQuitt	General Alarm Acknowledge
Ign	Ignore alarm Put module into Ignore Mode
Sim	Simulation Put module into Simulation mode
Auto	automatic mode Indicates the current operating mode of the actuator. Can be changed from the Faceplate.
MCo	manual control Activated from the faceplate and indicates if the User has activated manual On. This signal only works if the Actuator is in Manual Mode (“Auto == false”).
EmRel	emergency release Allows you to ignore the “Production Release” signal. Can be changed from the Faceplate
InterlockGAl	interlock by alarm If the actuator has an alarm, it automatically turns off its Output. This is the usual default behavior of Motors, so that they turn off, when tripped.
Maint	Maintenance If turned on, the actuator cannot be turned on by “Aco”, “Mco” nor “ExCo”. Can be activated from the Faceplate.

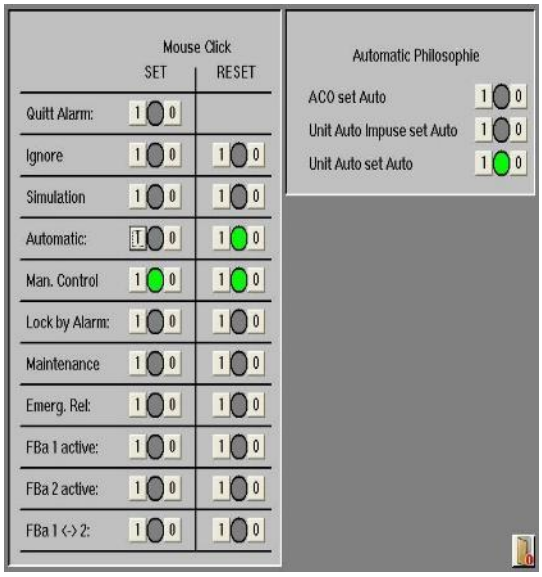
	The user can use this to temporarily block an actuator from accidentally being turned on, either manually or automatically.
TOnSp	turn on delay setpoint If the “Aco” signal is present, the delay must first complete, before the Signal is passed on to the output.
TOfSp	turn off delay setpoint If the “Aco” signal is turned off, the delay maintains the output until the time is complete. The Interlocks are always considered, and turn off the output without delay.
ADSp	alarm delay setpoint Defines the time that the Actuator allows the feedback signals to arrive, before activating an alarm “GAl”.
TInterlock	time interlock before restart If the Actuator was turned on, this timer must first complete, before it can be turned on again. This can be used to block “Turn On” of machines with significant inertia, that first must come to an standstill before turning on again.

Faceplate



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:



- 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 moule 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.

Assigning 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].GAls         //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

This section outlines recommended actuator interlocking and release concepts for common process scenarios. Many projects require additional, equipment-specific interlocks that are not covered here; therefore, the examples below should be used as guidance and adapted to local safety rules, operating philosophy, and applicable standards.

Emergency Stop

When an emergency stop is activated, all affected energized equipment must be de-energized immediately (e.g., pumps, valves, motors, and other driven devices). In application logic, use the actuator **.Rel** (Safety Release) signal for emergency-stop permissives, because this release cannot be overridden from the actuator faceplate. In addition, emergency-stop handling should typically rely on the digital input **.GAlS** (latched alarm) so the condition remains effective until the operator explicitly acknowledges the alarm in the system.

A special case of emergency stop logic involves vessel- or equipment-specific safety devices (e.g., manhole switches, access-door interlocks, trapped-key systems). If any of these devices indicate an unsafe condition, all equipment that can introduce hazards to the vessel or connected machinery must be shut down immediately. For example, agitators and internal spray devices should be blocked as soon as vessel access is detected.

Pumps

Pump interlocks are often among the most comprehensive interlocking scenarios because they combine multiple permissives (valve positions, level conditions, flow confirmation, motor protection, etc.). Implement *process* permissives using **.Rel** (Production Release), because Production Release may be temporarily overridden via the faceplate if the project allows an operator emergency release. Reserve **.Rel2** (Safety Release) for safety-critical conditions that must not be bypassed.

A practical approach is to separate pump permissives into **suction-side** and **discharge-side** checks. For each side, define the required valve lineup (e.g., at least one valid suction path open and at least one valid discharge path open) to prevent operation against closed valves or dead-end piping. This reduces logic complexity and makes commissioning and troubleshooting significantly easier.

If a flow meter is installed, use it for flow confirmation by enabling an appropriate low-flow alarm (e.g., LLA/LLA"). Enable this supervision only when the pump is commanded to run, and stop the pump if the flow alarm occurs after the configured delay. This provides reliable “no-flow” protection and helps detect closed valves, air binding, or dry suction conditions.

Another common pump permissive is **dry-run protection**. If the pump has a dedicated dry-run switch on the suction side, use it to block operation and/or stop the pump when no liquid is detected. For transfer pumps from a tank or vessel, an empty-level switch (LSL) can serve as dry-run protection. In practice, a short delay (typically 20–30 seconds) is often applied to prevent nuisance trips during transient conditions (e.g., brief aeration or sloshing), but the delay must be aligned with equipment protection requirements.

Full Signals (LSH)

If a vessel provides a “Full” indication (LSH), block all inflow paths (valves and pumps) that could overfill the vessel. Depending on the process, an alarm or interlock delay may be appropriate to avoid trips from short spikes or wave motion. Full signals may be discrete level switches or derived from analog level/volume measurement; ensure the signal used for interlocking has adequate reliability for the intended purpose.

Where recirculation is part of the normal operating strategy, evaluate whether recirculation should remain permitted at high level. In some designs, recirculation is still allowed because it does not increase vessel inventory; in others, it must be blocked to avoid pressurization, overflow at vent points, or unintended transfers.

Conductive level switches can produce false “Full” indications during CIP, because cleaning solutions (particularly caustic soda/NaOH) may cling to the probe and remain highly conductive. If this behavior is expected, consider suppressing or adapting the LSH interlock during CIP (e.g., use an alternative measurement, apply filtering, or condition the interlock with a validated CIP state). Any such adaptation must be risk-assessed to ensure overfill protection remains adequate.

Dry Run Protection (LSL)

Dry-run protection interlocks prevent pump damage when no liquid is available at the suction. Use a dedicated dry-run sensor where provided; otherwise, use the vessel empty switch (LSL) as a permissive for transfer pumps. Apply an appropriate delay (often 20–30 seconds) to balance equipment protection with process transients, and stop the pump if the condition persists beyond the configured delay.

Maintenance Switches

Maintenance (local isolator) switches are installed near energized equipment to allow maintenance personnel to disconnect devices locally. Where possible, wire and monitor feedback contacts from these switches so the control system can block actuator activation and generate a clear alarm indicating that local isolation is active. This reduces troubleshooting time and helps prevent unintended start attempts during maintenance.

Manholes and Key switches

Many vessels include access-detection devices such as manhole switches or trapped-key systems to indicate that a person may be inside the vessel or that protective barriers are open. If any of these devices indicate an unsafe state, immediately block and/or shut down all equipment that can create hazardous motion or energy transfer into the vessel (especially agitators, rotating devices, and internal spray systems). These conditions must be implemented as **Safety Release** interlocks and must not be bypassable through the HMI.

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

//Actuators that are not subject to any of the safety interlocks, come here.
//This however usually only applies to "Lamps", "Acoustic 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
- Ad 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 valve must be closed
      =     #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 (DIn) represent single discrete input points used to acquire binary field signals such as empty/full level switches, limit switches, pressure switches, safety contacts, and pushbuttons. The module supports configurable on-delay and off-delay filtering, as well as alarm supervision with an independent alarm delay to avoid nuisance alarms from short signal fluctuations.

General Principle

Each DIn processes one external (field) signal (typically **xSig**, provided via the transfer block) and generates an internal, conditioned status signal **Sig** for use in application logic. The transition timing and alarm behavior are determined by the configured on/off delays and alarm enable parameters.

- **On-delay filtering:** When the external signal transitions to TRUE, the on-delay timer starts. Once the timer expires, the internal **Sig** status is set to TRUE.
- **Off-delay filtering:** When the external signal transitions to FALSE, the off-delay timer starts. Once the timer expires, the internal **Sig** status is set to FALSE.
- **Alarm on FALSE (EA0):** If alarm-on-false is enabled and **Sig** becomes FALSE, the alarm delay timer starts.
- **Alarm on TRUE (EA1):** If alarm-on-true is enabled and **Sig** becomes TRUE, the alarm delay timer starts.
- **Alarm activation:** If the alarm delay timer expires while the alarm condition is still present, the module raises an alarm (e.g., **GAl**/**GAlS** depending on acknowledgement state).

Simulation

When a DIn is in **Simulation** mode, the physical field input is ignored and the internal **Sig** state is driven by the simulated value set by the operator (typically via the HMI faceplate). Alarm evaluation and status outputs are then based on this simulated **Sig** state, allowing functional testing without manipulating field wiring or sensors.

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	AlHM	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	GAlQuitt	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	GAl	BOOL	general alarm
3.1	GAlS	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 Enables alarms when the signal (Sig) is False. Use this signal in your code. This signal will be reset automatically for each plc Cycle. We recommend you use the set (“S”) instruction to activate this signal.
EA1	0	enable alarm by 1-signal Enables alarms when the signal (Sig) is True. Use this signal in your code. This signal will be reset automatically for each plc Cycle. We recommend you use the set (“S”) instruction to activate this signal.
SCS0	0	status check alarm by 0-signal Similar to Enable Alarm signals, but instead of alarms it generates de “Status Check Error” (SCE) signal. Use during the recipe ‘Start Check’ to verify the Modules status
SCS1	0	status check alarm by 1-signal Similar to Enable Alarm signals, but instead of alarms it generates de “Status Check Error” (SCE) signal. Use during the recipe ‘Start Check’ to verify the Modules status
xSig	x	signal extern Represents the “Field” signal of the connected sensor. This is written in the “TransDIn” block, and should not be used in your user application. Use the “Sig” signal instead.
User	x	free for user

Status

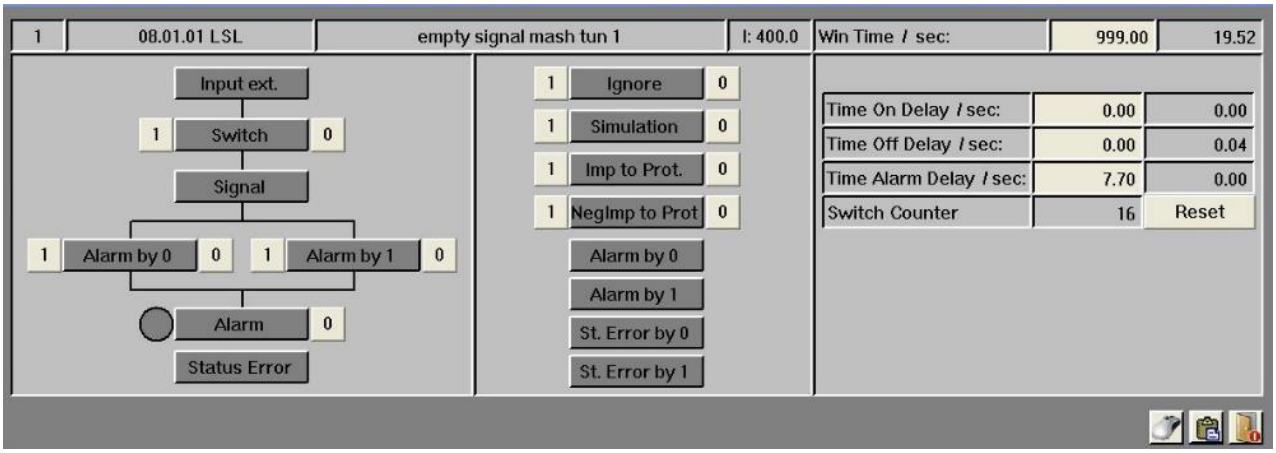
Symbol	Remark
GAI	general alarm The module currently has an incoming alarm. The current sensor status does not coincide with the “EA0” or “EA1” signals.
GAIS	**Latched general alarm** flag; remains TRUE until the alarm is acknowledged, even if the alarm condition clears.
SCE	status check error Set SCS during the recipe ‘Start Check’ to verify the current signal status, and no errors; otherwise, it raises SCE and flags the device in the HMI. Use this signal to exit the “Start Check” phase of your Unit.
Sig	signal state Represents the current signal status of the Module. Use this signal in your User application.
Imp	impulse flank When the sensor signal changes from “False” to “True”, this signal turns “TRUE” for one single Plc cycle. A Positive “Flank” or “One Shot”.
ImpNeg	negative impulse flank

	When the sensor signal changes from “TRUE” to “FALSE”, this signal turns “TRUE” for one single Plc cycle. A Negative “Flank” or “One Shot”.

Parameters

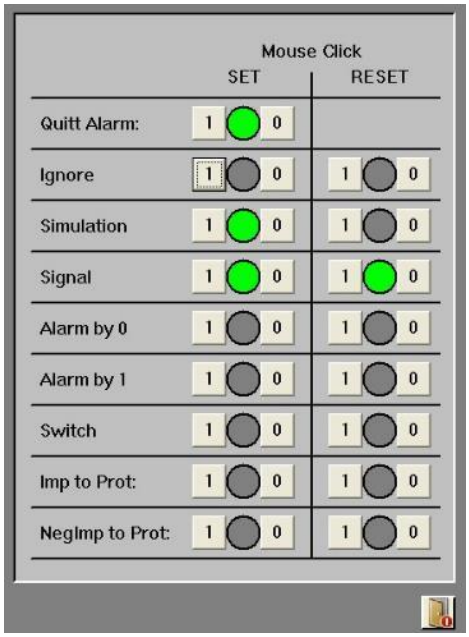
Symbol	Remark
Ign	Ignore alarm Activate Ignore mode
Sim	Simulation Activate Simulation Mode
iEA0	intern alarm by 0 Alarm enabling from the face plate. Do not use it in your User Application.
iEA1	intern alarm by 1 Alarm enabling from the face plate. Do not use it in your User Application.
ImpProt	write impulse flank to protocol It can be activated from the Faceplate. If a Positive impulse is detected, a Manual event is registered in the Historical data. This can be used to detect sporadic events in field devices.
ImpNegProt	write negative impulse flank to protocol It can be activated from the Faceplate. If a Negative impulse is detected, a Manual event is registered in the Historical data. This can be used to detect sporadic events in field devices.
Switch	convert as switch output Changes the working principle into an “Toggle” switch. This can be used if the Digital input is connected to a physical button. The input signal is not directly connected to the “Sig” signal, but rather Toggles the state of “Sig” each time a positive Impulse is detected.
TOnDSp	turn on delay setpoint If the input signal is TRUE, the delay must complete, before the “Sig” signal becomes TRUE
TOfDSp	turn off delay setpoint If the input signal is FALSE, the delay maintains the “Sig” signal until the timer completes
ADSp	alarm delay setpoint If the “Sig” signal does not coincide with the “EA0” or “EA1” signal an alarm active after this delay has completed
SwCntVal	switch counter value Every time the signal changes state, this counter increments by one.

Faceplate



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.



- 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 "Start Position"
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 (AIn)

An Analog Input (AIn) represents a single continuous measurement acquired from the field. Typical signal types include 4–20 mA current loops, 0–10 V voltage signals, or RTD sensor circuits (via appropriate input modules). The AIn module supports optional scaling of the raw signal to engineering units and can apply a configurable polygon (linearization) table to perform non-linear conversions or compensate sensor/installation characteristics.
Scaling can be disabled for signals that are already provided in the desired engineering units (for example, values delivered by intelligent transmitters or communication interfaces). When scaling is disabled, the module passes the incoming value through as the process value.

General Principle

Each AIn processes one external (field) value and produces an internal process value **PVal**. The conversion pipeline depends on the configuration of **NPA** (No Periphery Adaptation) and optional polygon linearization.

- If **NPA** (No Periphery Adaptation) is **FALSE**, the module applies the configured scaling to the raw input and transfers the scaled result to **PVal**.
- If **NPA** is **TRUE**, the module bypasses scaling and transfers the external value (**xPVal**) directly to **PVal**.
- If a polygon table is configured, it is applied after the scaling/bypass step to produce the final **PVal**.
- If **PVal** exceeds the configured **HHVal** or falls below **LLVal**, a warning condition is set.
- If **PVal** exceeds the configured **HHAVal** or falls below **LLAVal**, the alarm delay timer starts.
- If the alarm condition remains present until the alarm delay expires, the module raises an alarm.

The AIn module provides multiple limit and alarm signals with configurable **hysteresis**. Hysteresis introduces a dead band around each threshold to prevent rapid toggling when the process value fluctuates near a limit. In general, a limit becomes active when **PVal** crosses the setpoint and becomes inactive only after **PVal** returns past the setpoint by the hysteresis value.
Example: If a high limit is set to 90 with a hysteresis of 30, the limit becomes active when **PVal** rises to 90 or above.
The limit clears only when **PVal** falls below 90 − 30 = 60.
After it has cleared, the limit becomes active again only when **PVal** rises back to 90 + 30 = 120.

Simulation

When an AIn is in **Simulation** mode, the physical input signal is ignored, and the internal process value (**PVal**) is driven by a simulated value set by the operator (typically via the HMI faceplate). All limit checks, warnings, alarms, and derived signals are evaluated based on this simulated value, enabling functional testing without manipulating field wiring or sensors.

Filters

The AIn module supports up to three independent low-pass filters that can be enabled simultaneously. Each filter adds additional smoothing to reduce noise and short-term fluctuations. Higher filter percentages provide stronger smoothing but also increase signal lag, which should be considered for fast control loops.
If the measured value is too noisy for reliable control or alarming, enable one or more filters and tune the strength until the signal is stable while still meeting the required response time for the process.

Structure

Address	Symbol	Type	Remark
0.0	ELLA	BOOL	**Enable LLA** (Low-Low Alarm). When TRUE, the module supervises **PVal** against **LLAVal** and triggers the low-low alarm after the configured alarm delay.
0.1	EHHA	BOOL	**Enable HHA** (High-High Alarm). When TRUE, the module supervises **PVal** against **HHAVal** and triggers the high-high alarm after the configured alarm delay.
0.2	xAI	BOOL	**External alarm input.** Optional external fault indication from hardware/field; when TRUE, the module sets alarm status according to the configured alarm handling.
0.3	NPA	BOOL	**No Periphery Adaptation (NPA).** When TRUE, bypasses scaling and uses **xPVal** directly as **PVal**; when FALSE, applies the configured scaling before any polygon conversion.
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	GAlQuitt	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	GAl	BOOL	general alarm
3.1	GAlS	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 hgih 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 Unscaled measurement from the transfer block / I/O (e.g., 4–20 mA converted to a raw engineering representation). Used as the source for scaling and filtering.
88.0	iPVal	REAL	process value intern (without polygon)

Command

Symbol	Data type	Remark
ELLA	BOOL	Enables the **Low-Low Alarm (LLA)** supervision of *PVal* against *LLAVal* (with alarm delay).
EHHA	BOOL	Enables the **High-High Alarm (HHA)** supervision of *PVal* against *HHAVal* (with alarm delay).
xAI	BOOL	External alarm/fault indication from the field or hardware that drives the module's alarm status when active.
NPA	BOOL	**No Periphery Adaptation**: when TRUE, bypasses scaling and uses *xPVal* directly as the internal value; when FALSE, applies configured scaling.
User	BOOL	Reserved user memory/flag for project-specific logic and not used by the standard module function.
xPVal	REAL	Raw external process value from I/O or transfer blocks that serves as the source for scaling, filtering, and limit/alarm evaluation.

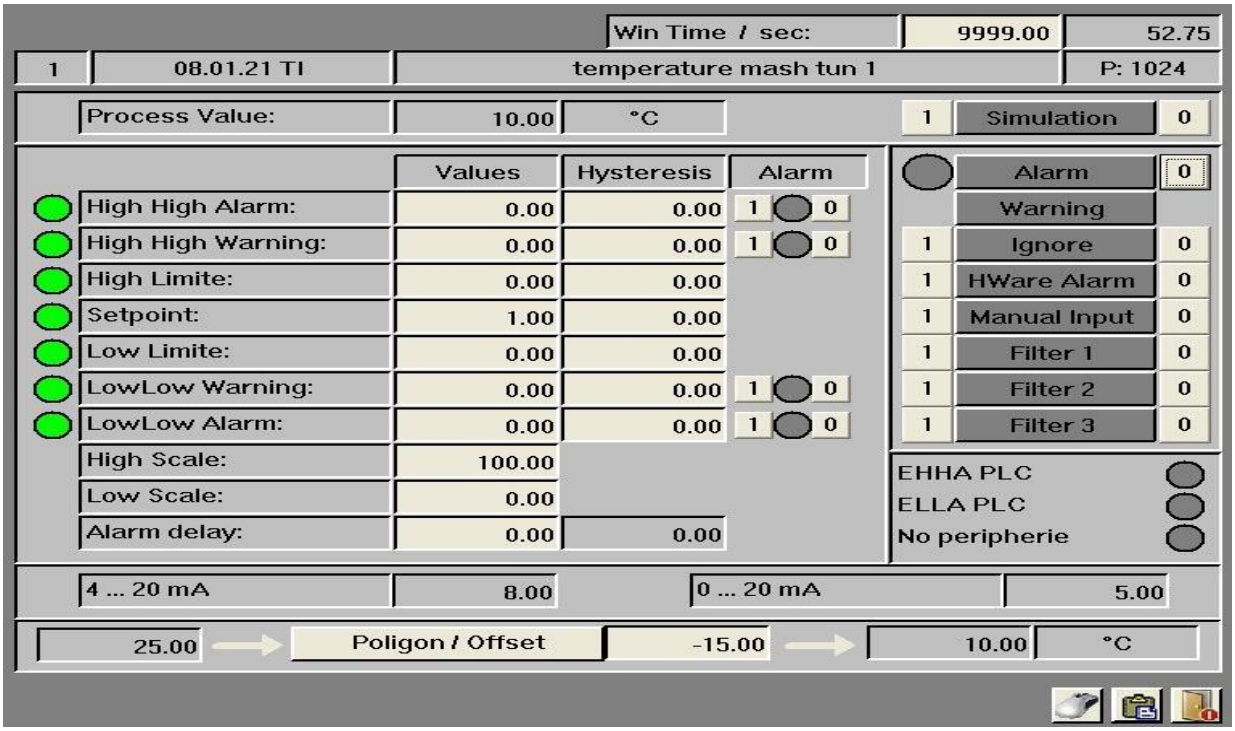
Status

Symbol	Data Type	Remark
MLLA	BOOL	Indicates a **Low-Low alarm condition** when low-low alarming is enabled and the value falls below the configured alarm threshold.
MLL	BOOL	Indicates a **Low-Low limit warning** when enabled, typically used for early warning before a low-low alarm.
ML	BOOL	Indicates the process value is **below the Low limit** threshold.
MSp	BOOL	Indicates the module is currently using or supervising the **configured setpoint (SP)** value.
MH	BOOL	Indicates the process value is **above the High limit** threshold.
MHH	BOOL	Indicates a **High-High limit warning** when enabled, typically used for early warning before a high-high alarm.
MHHA	BOOL	Indicates a **High-High alarm condition** when high-high alarming is enabled and the value exceeds the configured alarm threshold.
MHWA	BOOL	Indicates an **alarm/fault reported by hardware or an external source**, independent of calculated limits.
GAI	BOOL	**General alarm active** flag; TRUE while any configured alarm condition is currently present.
GAIS	BOOL	**Latched general alarm** flag; remains TRUE until the alarm is acknowledged, even if the condition clears.
Warn	BOOL	**General warning** flag; TRUE when any configured warning/limit condition is present.
User	BOOL	Reserved **user memory/flag** for project-specific logic and not used by the standard module function.
PVal	Real	The module's **processed process value** (after scaling/linearization/filtering, depending on configuration).

Parameters

Symbol	Data Type	Remark
Ign	Bool	Enables **Ignore mode**, suppressing alarm generation even if an alarm condition is detected.
Sim	Bool	Enables **Simulation mode**, using simulated values instead of physical I/O for testing.
iEHWA	Bool	Enables the **hardware/external alarm input** evaluation (external fault supervision).
iELLA	Bool	Enables the **Low-Low Alarm (LLA)** supervision against the configured low-low alarm threshold.
iEHHA	Bool	Enables the **High-High Alarm (HHA)** supervision against the configured high-high alarm threshold.
iELLW	Bool	Enables the **Low-Low warning** supervision against the configured low-low warning limit.
iEHHW	Bool	Enables the **High-High warning** supervision against the configured high-high warning limit.
Filter1	Bool	Enables **Filter 1** (≈75% smoothing) to reduce noise on the measured value.
Filter2	Bool	Enables **Filter 2** (≈88% smoothing) to provide stronger noise reduction with more lag.
Filter3	Bool	Enables **Filter 3** (≈94% smoothing) for maximum smoothing and highest response delay.
ManuInp	Bool	Enables **manual input / no-periphery mode**, allowing a user-defined value to be used instead of the field signal.
Sp	Real	Setpoint value used as the **target reference** for supervision or control functions.
LScal	Real	Low scaling point defining the **engineering-unit value** corresponding to the low raw input range.
HScal	Real	High scaling point defining the **engineering-unit value** corresponding to the high raw input range.
LLAVal	Real	Threshold value for the **Low-Low Alarm (LLA)** condition.
LLVal	Real	Threshold value for the **Low-Low warning/limit** condition.
LVal	Real	Threshold value for the **Low** limit condition.
HVal	Real	Threshold value for the **High** limit condition.
HHVal	Real	Threshold value for the **High-High warning/limit** condition.
HHAVal	Real	Threshold value for the **High-High Alarm (HHA)** condition.
LLAHys	Real	Hysteresis applied to the **Low-Low Alarm** to prevent chattering near the LLA threshold.
LLHys	Real	Hysteresis applied to the **Low-Low warning** to prevent toggling near the LL limit.
LHys	Real	Hysteresis applied to the **Low** limit to stabilize the signal near the L threshold.
SpHys	Real	Hysteresis band around the **setpoint** used for deviation/hold or setpoint-based supervision.
HHys	Real	Hysteresis applied to the **High** limit to prevent oscillation near the H threshold.
HHHys	Real	Hysteresis applied to the **High-High warning** to prevent chattering near the HH limit.
HHAHys	Real	Hysteresis applied to the **High-High Alarm** to prevent repeated alarms near the HHA threshold.
ADSp	Real	Alarm delay setpoint defining how long an alarm condition must persist before an alarm is raised.
PoTNo	Dint	Polygon/offset selector: **positive** selects a polygon table number, **negative** applies a fixed offset.

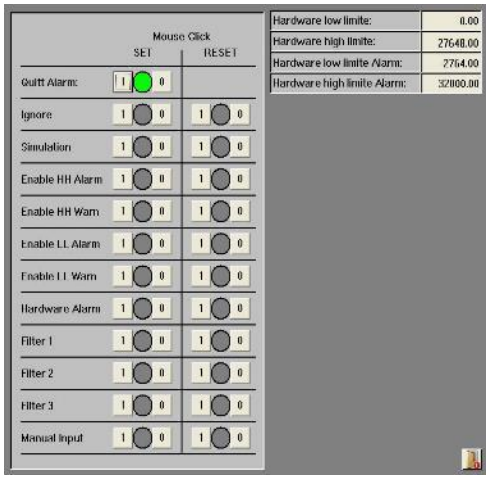
Faceplate



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:



- 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
```

AIn 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

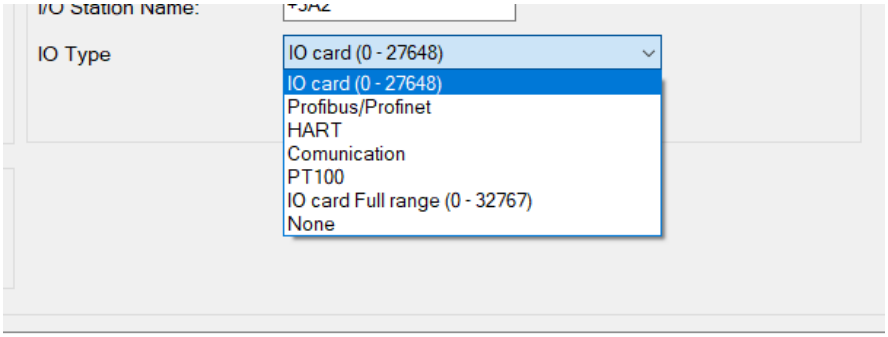
Analog Input control Modules can be assigned from different hardware sources which may have different Scaling values, depending on the Type of Value you are supplying to the Analog Input Module.

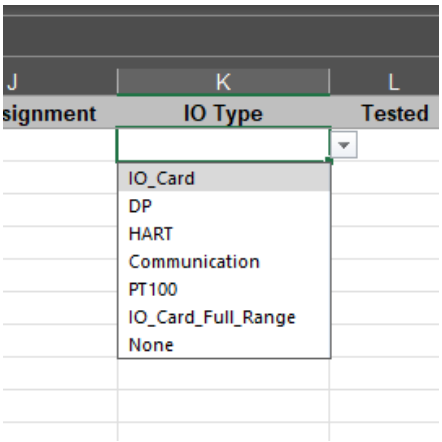
Typical Analog input sources

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.

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.

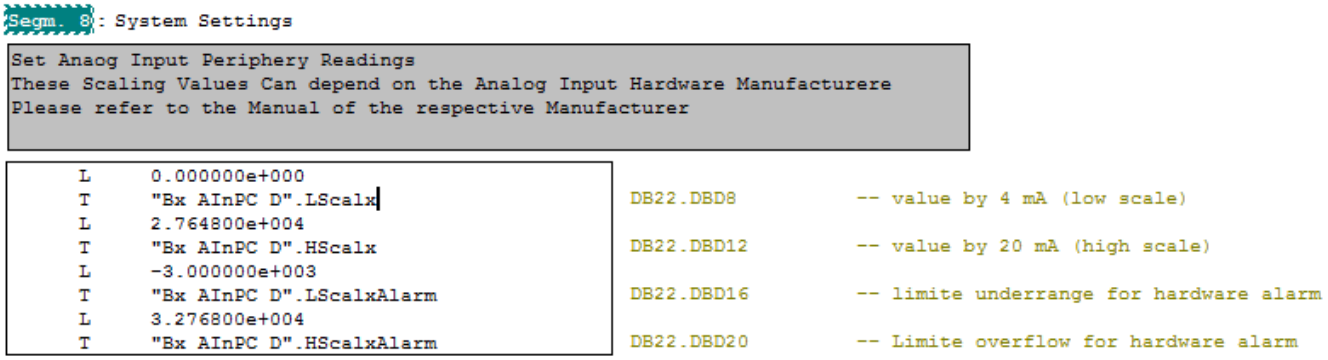


Figure 2 OB1

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:

- NAN > 0.0 = FALSE
- NAN < 0.0 = False
- NAN == 0.0 = 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 potentially 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     #AIn.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     #AIn.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

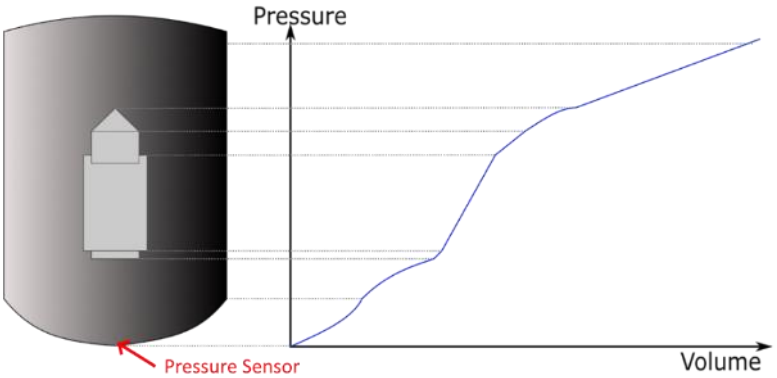
```
L "Bx AIn D".AIn[1].PVal    //Pressure Below
L "Bx AIn D".AIn[2].PVal    //Pressure Above
/R                          //Since Pressure Above may be 0.0 there is a chance of NAN
T #PotentialNANValue

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 measured value that has a **non-linear relationship** to the desired engineering value. The table defines a set of coordinate pairs and the PLC performs **piecewise linear interpolation** between the nearest points. This provides a practical way to approximate an arbitrary curve with good accuracy while keeping the implementation simple and deterministic.

Example premise

A common application is calculating **tank volume** from a **bottom pressure transmitter**. The pressure transmitter measures the hydrostatic head (water column) and therefore represents **liquid level**. However, the relationship between level and volume is not always linear because tanks often include cones, dished bottoms, internal coils, agitators, or other geometry that reduces usable volume.

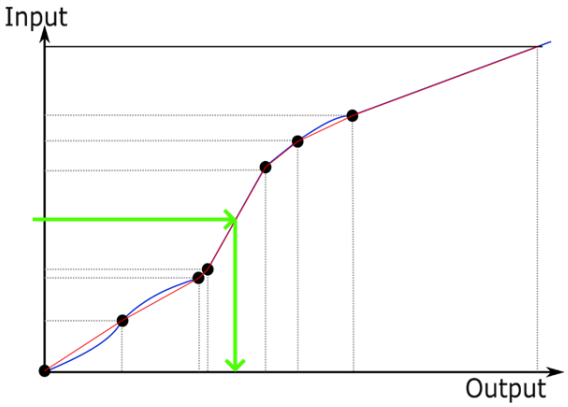


In functional form, the volume depends on level:

$$V = f(h)$$

Given the measured level (derived from pressure), we want to calculate the corresponding tank volume.

In simple vessels, volume can be calculated analytically (cylinder + cone, etc.). In real installations, geometry is frequently too complex or not documented sufficiently for reliable calculation inside the PLC. A polygon table provides a robust alternative that can be commissioned using measured reference points.

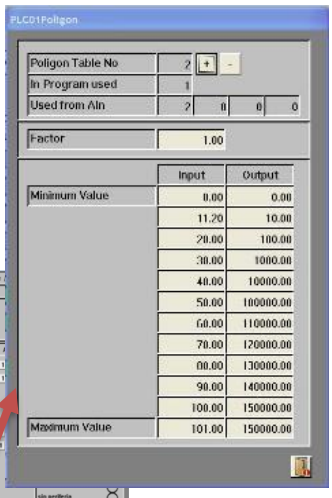


Polygon table principle (interpolation)

For the tank example, define the table so that:

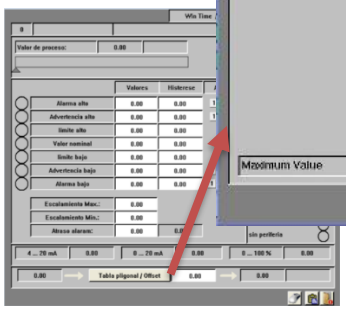
- **Input (Y)** = measured value (e.g., pressure in mbar or level in mm)
- **Output (X)** = required value (e.g., volume in liters)

During runtime, the PLC finds the two table points surrounding the current input value and linearly interpolates the output. The interpolated result approximates the real curve; adding more points increases accuracy, especially in regions where the slope changes significantly (e.g., at the transition from cone to cylinder).



Creating and assigning a polygon table

In BatchXpert, a polygon table is assigned to an **Analog Input module** by entering the polygon table number in the corresponding faceplate field. Polygon tables do not need to be programmed manually in the PLC — a default set is provided (e.g., 16 tables). If more tables are required, the polygon table data block can be expanded to provide additional instances.



The polygon table editor allows up to **16 points** per table. Points should be ordered monotonically by input value (increasing) and should cover the complete operating range to avoid extrapolation.

Commissioning example (tank level → volume)

A practical way to build the table during commissioning is to record real operating reference points:

1. Start with an empty tank (0 volume reference).
2. Fill the vessel in steps using a reliable reference (typically a **flowmeter totalizer**, or in small systems a calibrated transfer).
3. After each fill step, stop flow, allow the signal to settle, then record:
    - the **pressure/level input** (from the transmitter)
    - the **actual volume** (from the reference totalizer)
4. Take measurements at ~10% increments, and add extra points where geometry changes (e.g., cone-to-cylinder transition).

Your resulting data set should look like the table below (enter these pairs into the polygon faceplate table):

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

Example non-linear valve characteristic (CV mapping)

Polygon tables are also useful when an actuator or device has a **non-linear response curve**, and you need a more linear “engineering behavior” from the control output.

For example, many control valves have an **equal-percentage characteristic**: a change from 20% to 30% opening does not increase flow by the same amount as a change from 70% to 80%. If you want the operator or controller output to behave more linearly with respect to flow, you can build a polygon table:

- **Input** = valve command (% output)
- **Output** = expected flow (% or engineering units)

The PLC can then convert “requested linear flow” into the correct valve output command (or vice-versa), improving control performance and tuning stability.

PID Regulator (PID)

Analog Output / PID Regulator (PID) represents equipment that is driven by a continuous control signal (typically 0–10 V or 4–20 mA) and, optionally, includes an integrated PID control algorithm. Typical applications include control valves, variable-speed pumps, flow/pressure/temperature loops, and setpoint outputs to drives or third-party controllers.

Operating Principle

In **closed-loop** mode, the module calculates the output value based on the difference between the measured process value (*PV*) and the target setpoint (*SP*). The resulting controller output (*CV*) is then written to the analog output channel. If regulation is not required, the PID function can be disabled and the module operates in **open-loop** mode, where a user-defined output value is forwarded directly to the analog output (for example, a speed reference to a variable frequency drive or a remote setpoint to an external controller).

Key Tuning Parameters

- **Controller direction (Reverse/Direct action):** Defines whether the output increases when the process value is below the setpoint (reverse action, typical for heating/valves) or decreases when the process value is below the setpoint (direct action, typical for cooling). Selecting the wrong direction will cause unstable control.
- **Proportional gain (P):** Determines how strongly the output reacts to the current error $$e=SP-PV$$. Higher gain generally increases responsiveness but can introduce oscillation.
- **Integral time (I):** Eliminates steady-state error by integrating the error over time. Too aggressive integral action can cause overshoot and slow recovery after disturbances.
- **Derivative time (D):** Predicts error trend and can improve stability by damping rapid changes. Derivative action is sensitive to noise and is typically reduced or disabled for noisy measurements.

Deadband and Output Ramp

Deadband (sometimes referred to as “Banda Muerta”) defines a neutral zone around the setpoint in which small deviations do not change the controller output. This can reduce actuator wear and output chatter when the process naturally fluctuates near the target. **Output ramp** limits the rate of change of the output (typically expressed in %/s). If the calculated output changes faster than the configured ramp, the module slews the output at the maximum allowed rate. Set the ramp to **0.0** to disable ramp limiting and allow instantaneous output changes. Ramp limiting is recommended for valves and pumps to reduce mechanical stress and avoid process shocks.

Alarms and Monitoring

The module supports limit and deviation monitoring to detect abnormal process conditions and insufficient control performance. A configurable **Check Delay** (sometimes labeled “Backlog/Checkup Delay” in legacy texts) delays alarm evaluation after the controller starts, allowing the loop time to stabilize before supervision begins. After the delay expires, the following monitoring functions can be enabled as required:

- **Absolute limits:** High/low boundaries that the process value must not exceed. If PV violates these limits (after Check Delay), the module raises an alarm/fault to protect equipment and product.
- **Deviation (hysteresis) alarm:** Monitors the absolute deviation $$|SP-PV|$$. If the deviation remains above the configured threshold after Check Delay, an alarm is generated, indicating the controller cannot achieve the target (e.g., insufficient capacity, valve stuck, sensor fault).
- **Deviation warning:** Same principle as the deviation alarm but generates a warning instead of an alarm, enabling early detection before a fault condition occurs.

Simulation

When the module is in **Simulation** mode, the physical input measurement is ignored and the internal process value (PV) can be set by the operator via the HMI. The controller then calculates its output based on the simulated PV, enabling loop logic testing, alarm verification, and faceplate checks without manipulating field wiring or process instrumentation.

Structure

Address	Symbol	Type	Remark
0.0	EAl	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	OVOn	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	AlHM	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	GAlQuitt	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	GAl	BOOL	general alarm
3.1	GAlS	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	OVMax	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	Data Type	Improved Remark (one sentence)
EAl	Bool	Enables alarm supervision for the module according to the configured alarm conditions and delays.
SCS	Bool	Starts a status check to verify the module is in a valid expected state (raises a status-check error if it fails).
MStC	Bool	Activates a fixed (static) output mode, forcing the output to a predefined value instead of regulating.
MStrt	Bool	Defines the output value used as the initial start value when the module transitions into active control.
MOVMin	Bool	Minimum output limit (lower clamp) applied to the manipulated/output value.
MOVMax	Bool	Maximum output limit (upper clamp) applied to the manipulated/output value.
OVOn	Bool	Output-on preset value used when the module is commanded ON (e.g., default open/speed reference).
DisOut	Bool	Disables writing to the physical output channel when TRUE (FALSE = output periphery enabled).
User	Bool	Reserved user memory/flag for project-specific logic and not used by the standard module function.
xSp	Real	External setpoint input provided by the user program or transfer logic (used as the target value).

xPVal	Real	External process value input from the user program (used as the measured PV for monitoring/control).
StC	Real	Static output value in percent used when static/manual output mode is active.
Fuzzy1V	Real	Input variable for fuzzy control function 1 (used to adapt the controller/output behavior).
Fuzzy2V	Real	Input variable for fuzzy control function 2 (used to adapt the controller/output behavior).

Status

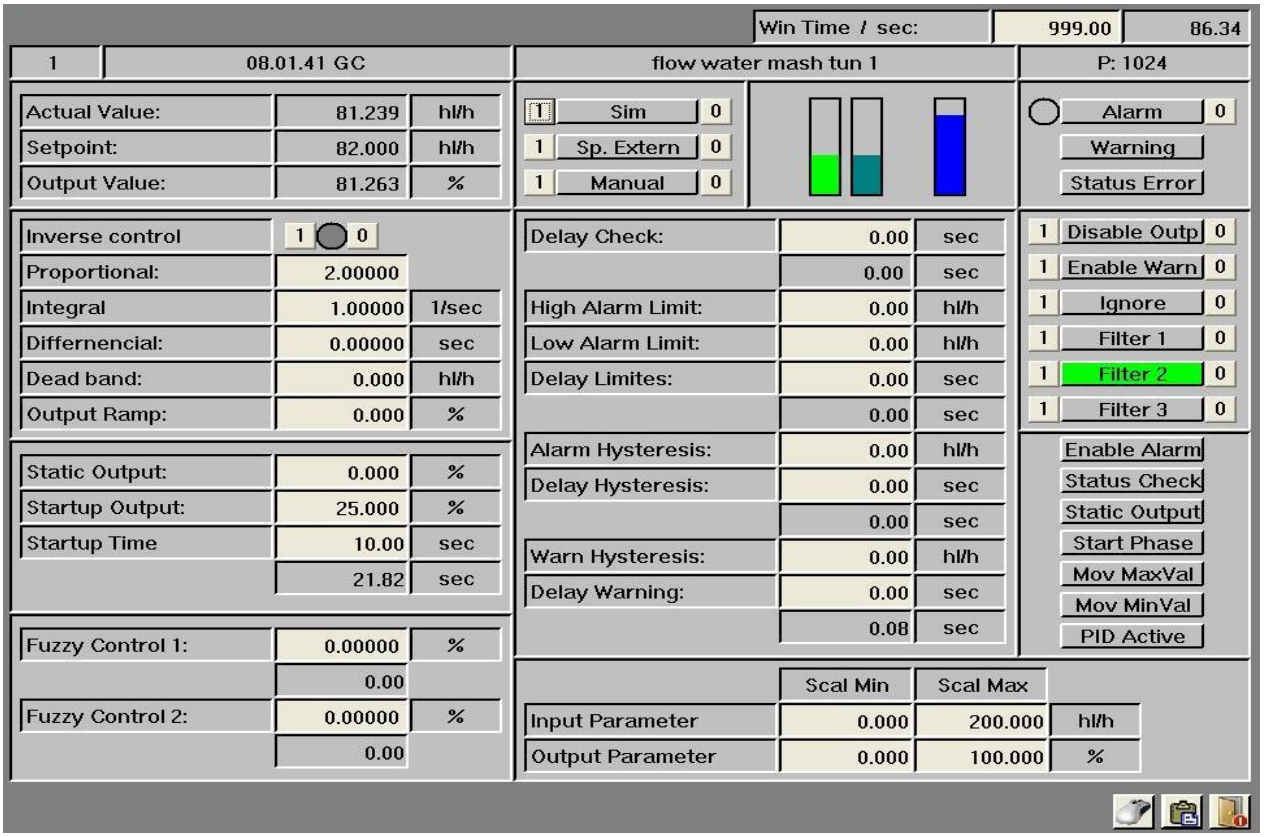
Symbol	Data Type	Improved Remark (one sentence)
GAl	**Bool**	**General alarm active** flag; TRUE while any alarm condition is currently present.
GAlS	**Bool**	**Latched general alarm** flag; remains TRUE until the alarm is acknowledged, even if the alarm condition clears.
SCE	**Bool**	**Status check error** flag; TRUE when a configured status check fails (e.g., during Start Check / SCS evaluation).

Parameters

Symbol	Data Type	Improved Remark (one sentence)
Ign	**Bool**	Enables **Ignore mode**, suppressing alarm generation even if alarm conditions are present.
Sim	**Bool**	Enables **Simulation mode**, using simulated PV/SP values instead of physical I/O for testing.
MCOn	**Bool**	Enables the **controller function** (closed-loop control active); FALSE forces the controller off.
MSpExt	**Bool**	Selects **external setpoint** when TRUE; FALSE uses the internal setpoint source.
DisOut	**Bool**	Disables writing to the **physical output channel** when TRUE (FALSE = output enabled).
EW	**Bool**	Enables **warning supervision** (limit warnings) according to configured warning thresholds and delays.
Filter1	**Bool**	Enables low-pass **Filter 1** (≈75% smoothing) to reduce measurement noise.
Filter2	**Bool**	Enables low-pass **Filter 2** (≈88% smoothing) for stronger noise reduction with more lag.
Filter3	**Bool**	Enables low-pass **Filter 3** (≈94% smoothing) for maximum smoothing and highest response delay.
CA	**Bool**	Sets the **control action** (TRUE = inverse/reverse acting; FALSE = direct acting).
OVal	**Real**	Current controller **output value** (manipulated variable) used to drive the analog output.
Sp	**Real**	Controller **setpoint (SP)** used as the target reference for regulation and supervision.
PVal	**Real**	Current **process value (PV)** used as the measured input for control and monitoring.
LScal	**Real**	Low scaling point defining the engineering value corresponding to the low raw input range.
HScal	**Real**	High scaling point defining the engineering value corresponding to the high raw input range.
OVMin	**Real**	Minimum clamp for the output value, limiting the controller to a defined lower bound.

OVMax	**Real**	Maximum clamp for the output value, limiting the controller to a defined upper bound.
StC	**Real**	Static output value (%) used when the controller is configured for fixed/manual output mode.
Strt	**Real**	Start output value (%) applied when the controller starts to ensure a defined initial CV.
StrTSp	**Real**	Start-up time setpoint defining how long start conditions (e.g., initialization/transition) apply.
LLAVal	**Real**	Low threshold for **alarm** supervision (low alarm limit).
HHAVal	**Real**	High threshold for **alarm** supervision (high alarm limit).
AHys	**Real**	Hysteresis band for alarms to prevent chattering around alarm thresholds.
CheckDSp	**Real**	Check delay setpoint defining the time before supervision/alarms become active after start.
ADHLSp	**Real**	Alarm delay setpoint for high/low limit supervision (time condition must persist to alarm).
ADSp	**Real**	Alarm delay setpoint for deviation/hysteresis supervision (time deviation must persist to alarm).
WHys	**Real**	Hysteresis band for warnings to prevent repeated toggling near warning thresholds.
WDSp	**Real**	Warning delay setpoint defining how long a warning condition must persist before being raised.
KP	**Real**	Proportional gain determining the immediate output response to the current control error.
KI	**Real**	Integral gain determining how strongly accumulated error drives the output over time (1/s).
KD	**Real**	Derivative gain determining how strongly error rate-of-change influences the output (s).
RampV	**Real**	Output ramp limit defining the maximum allowed change rate of **OVal** (per second).
DeadB	**Real**	Deadband around zero error where small deviations do not change the output.
Fuzzy1G	**Real**	Gain factor for fuzzy function 1 used to adapt controller behavior based on the fuzzy input.
Fuzzy2G	**Real**	Gain factor for fuzzy function 2 used to adapt controller behavior based on the fuzzy input.

Faceplate



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

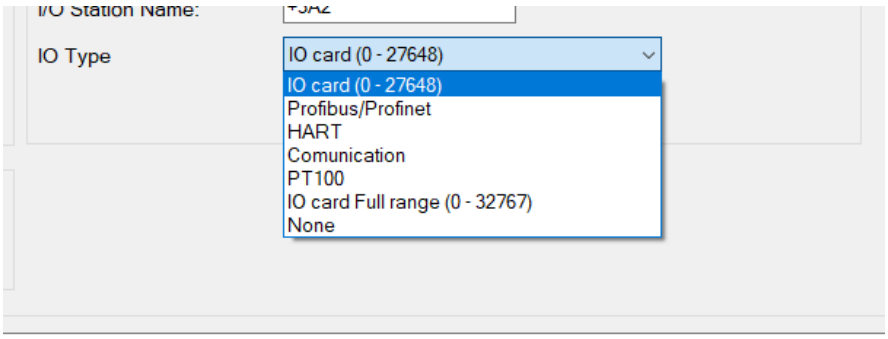
PID Regulator control Modules can be assigned to different hardware destination which may have different Scaling values, depending on the Type of Value you are supplying to the Analog Input module.

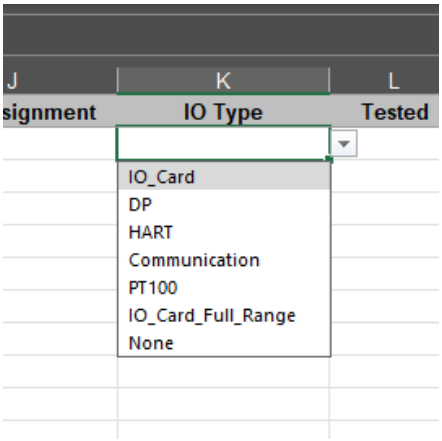
Typical Analog input sources

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.

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

AIn 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)

The Counter Module (Cnt) is used to count discrete pulses from a digital input and convert them into a totalized quantity. Typical applications include flow totalization, packaging counts, dosing pulses, or any device that provides a pulse output where each pulse represents a fixed quantity (e.g., volume, mass, pieces, or cycles). A common use case is a flow meter with a pulse output that generates one pulse per defined amount of product passing through the sensor (for example, 1 pulse per 0.1 L). By counting pulses and applying the configured pulse value, the module provides a running total in engineering units.

General Principle

The Counter Module processes the external signal xSig and increments the internal counter value CVal on each valid pulse (typically on a rising edge transition from FALSE to TRUE). The calculated process total PVal is derived from:

- CVal = number of detected pulses
- ImpVal = engineering-unit value represented by one pulse (e.g., L/pulse, kg/pulse)
- PVal = CVal × ImpVal

To ensure reliable counting, the input signal should be free of contact bounce or noise. If the pulse source is mechanical or susceptible to chatter, upstream filtering (debounce logic, minimum pulse width, or signal conditioning) should be applied so that each physical pulse is counted only once.

Simulation

When the Counter Module is in Simulation mode, the physical input xSig is ignored and the module allows the operator to set a simulated process total (or simulated pulse behavior, depending on implementation). This enables functional testing of sequence logic, reporting, and HMI behavior without requiring real pulse hardware or live process conditions.

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	xAl	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	GAlQuitt	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	GAl	BOOL	general alarm
3.1	GAlS	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 If no Impulse signal was observed during the Alarm time, an alarm is generated. You must activate this signal when impulses are expected to arrive, for example an pump is running, and set the Alarm Time appropiatly
ELLA	enable low low alarm Enables the Low Low Alarm (LLA) supervision against the configured low low alarm threshold.
EHHA	enable high high alarm nables the High High Alarm (HHA) supervision against the configured high high alarm threshold.
xAl	alarm from extern external alarm signal to activate an module alarm from axiliary external sources
ResetBlock	interlock counter reset While true, the reset can not be performed through the faceplate or by setting the Reset signal
xSig	impulse input This is the main couning impulse signal. You must connect this signal to our counting signal source, such as counting input or tother
User	memory free for user

Status

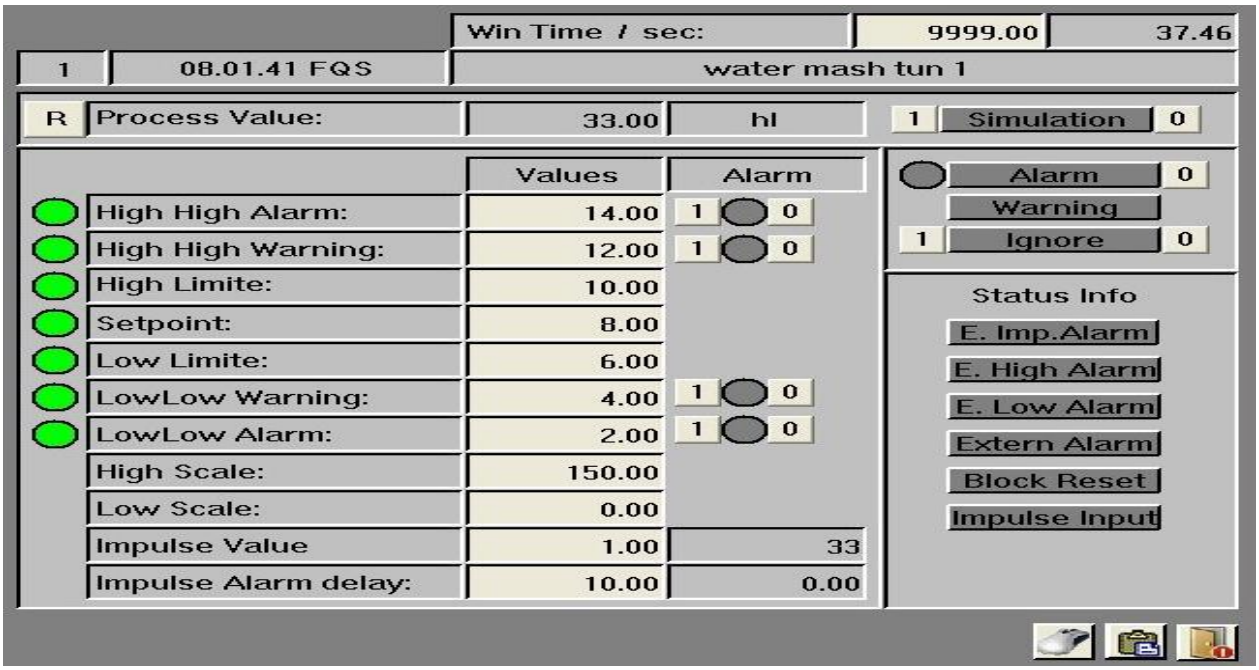
Symbol	Data Type	Remark
MLLA	BOOL	Indicates a **Low-Low alarm condition** when low-low alarming is enabled and the value falls below the configured alarm threshold.
MLL	BOOL	Indicates a **Low-Low limit warning** when enabled, typically used for early warning before a low-low alarm.
ML	BOOL	Indicates the process value is **below the Low limit** threshold.
MSp	BOOL	Indicates the module is currently using or supervising the **configured setpoint (SP)** value.
MH	BOOL	Indicates the process value is **above the High limit** threshold.
MHH	BOOL	Indicates a **High-High limit warning** when enabled, typically used for early warning before a high-high alarm.
MHHA	BOOL	Indicates a **High-High alarm condition** when high-high alarming is enabled and the value exceeds the configured alarm threshold.
MHWA	BOOL	Indicates an **alarm/fault reported by hardware or an external source**, independent of calculated limits.
GAI	BOOL	**General alarm active** flag; TRUE while any configured alarm condition is currently present.
GAIS	BOOL	**Latched general alarm** flag; remains TRUE until the alarm is acknowledged, even if the condition clears.
Warn	BOOL	**General warning** flag; TRUE when any configured warning/limit condition is present.
User	BOOL	Reserved **user memory/flag** for project-specific logic and not used by the standard module function.
PVal	Real	The module's **processed process value** (after scaling/linearization/filtering, depending on configuration).

Parameters

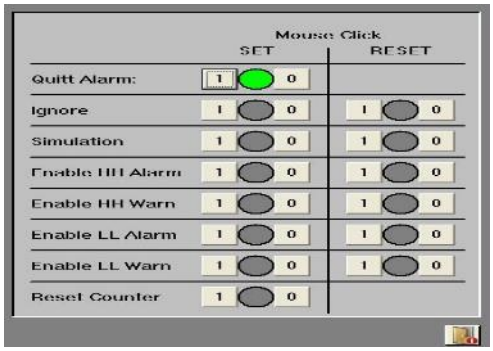
Symbol	Data Type	Remark
Ign	Bool	Enables **Ignore mode**, suppressing alarm generation even if an alarm condition is detected.
Sim	Bool	Enables **Simulation mode**, using simulated values instead of physical I/O for testing.
iEHWA	Bool	Enables the **hardware/external alarm input** evaluation (external fault supervision).
iELLA	Bool	Enables the **Low-Low Alarm (LLA)** supervision against the configured low-low alarm threshold.
iEHHA	Bool	Enables the **High-High Alarm (HHA)** supervision against the configured high-high alarm threshold.
iELLW	Bool	Enables the **Low-Low warning** supervision against the configured low-low warning limit.
iEHHW	Bool	Enables the **High-High warning** supervision against the configured high-high warning limit.
Filter1	Bool	Enables **Filter 1** (≈75% smoothing) to reduce noise on the measured value.
Filter2	Bool	Enables **Filter 2** (≈88% smoothing) to provide stronger noise reduction with more lag.
Filter3	Bool	Enables **Filter 3** (≈94% smoothing) for maximum smoothing and highest response delay.
Reset	Bool	Reset Counter If true, resets the accumulated counting value in PVal to 0.0
Sp	Real	Setpoint value used as the **target reference** for supervision or control functions.
LScal	Real	Low scaling point defining the **engineering-unit value** corresponding to the low raw input range.

HScal	Real	High scaling point defining the **engineering-unit value** corresponding to the high raw input range.
LLAVal	Real	Threshold value for the **Low-Low Alarm (LLA)** condition.
LLVal	Real	Threshold value for the **Low-Low warning/limit** condition.
LVal	Real	Threshold value for the **Low** limit condition.
HVal	Real	Threshold value for the **High** limit condition.
HHVal	Real	Threshold value for the **High-High warning/limit** condition.
HHAVal	Real	Threshold value for the **High-High Alarm (HHA)** condition.
LLAHys	Real	Hysteresis applied to the **Low-Low Alarm** to prevent chattering near the LLA threshold.
LLHys	Real	Hysteresis applied to the **Low-Low warning** to prevent toggling near the LL limit.
LHys	Real	Hysteresis applied to the **Low** limit to stabilize the signal near the L threshold.
SpHys	Real	Hysteresis band around the **setpoint** used for deviation/hold or setpoint-based supervision.
HHys	Real	Hysteresis applied to the **High** limit to prevent oscillation near the H threshold.
HHHys	Real	Hysteresis applied to the **High-High warning** to prevent chattering near the HH limit.
HHAHys	Real	Hysteresis applied to the **High-High Alarm** to prevent repeated alarms near the HHA threshold.
ADSp	Real	Alarm delay setpoint defining how long an alarm condition must persist before an alarm is raised.
ImpVal	Real	value per impulse Every time an Impule is observed, this value is added to PVal

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

Signal Transfer

```
U "Act". Act[18]. Out          //Water Valve
U "DIn". Din[233]. Imp         //Positive Digital One-shot 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 other system modules—such as actuators, digital inputs, and related components—already incorporate their own messaging and alarming capabilities, the use of the dedicated message module is relatively limited. Its primary function is to generate alarms or operator instructions that are not directly associated with specific peripheral devices. This ensures that system-level notifications, warnings, or guidance messages can still be communicated effectively, even when they fall outside the scope of standard hardware-driven events.

General Operating Principle

The behavior of the message module is based on the state of an external trigger signal and configurable timing parameters:

- **Signal Activation (TRUE State):**
  When the external signal transitions to a TRUE state, the system initiates an On-Delay timer. Once this delay period has elapsed, the module generates a message based on the configured parameters. Depending on the setup, this output may be:
  - An Operating Message (OpMsg), intended to provide informational or instructional feedback to the operator, or
  - A General Alarm (Gal), intended to alert the operator to abnormal or critical conditions requiring attention.
- **Signal Deactivation (FALSE State):**
  When the external signal returns to a FALSE state, an Off-Delay timer is triggered. This delay ensures that the generated message or alarm is not immediately cleared, allowing sufficient time for operator acknowledgment or system stabilization before the message is reset.

Additional Notes

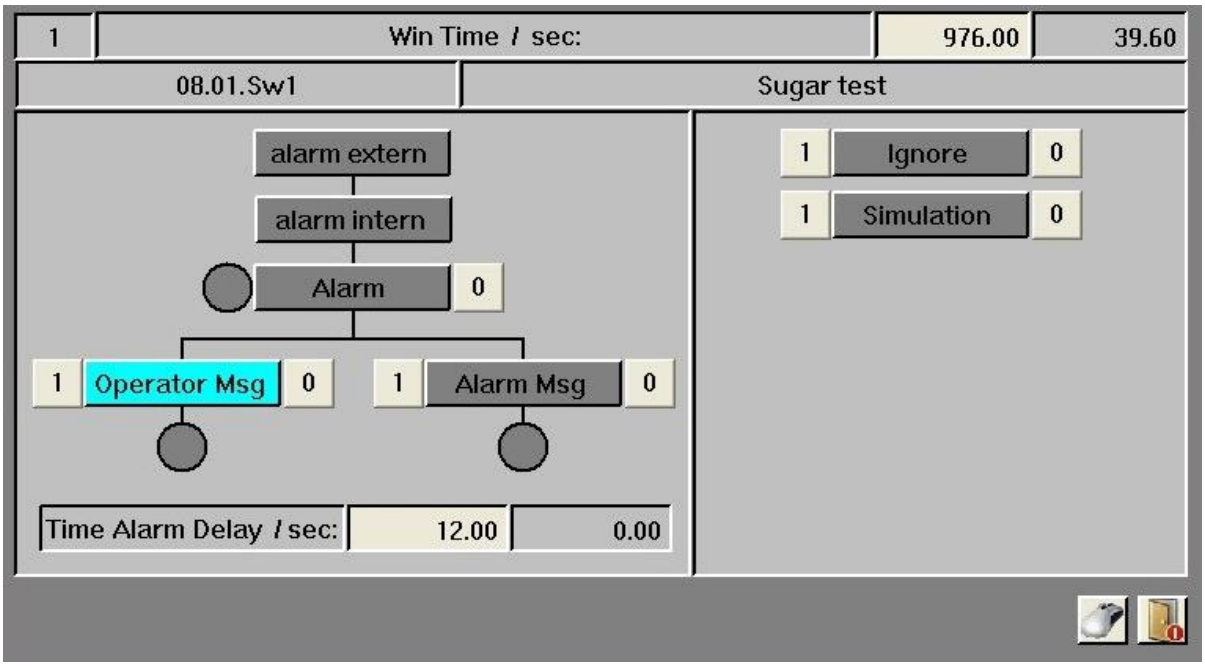
The use of configurable On-Delay and Off-Delay timers helps prevent rapid message toggling caused by transient signal fluctuations, improving system reliability and operator clarity.
Proper parameterization of message types (alarm vs. operational message) allows the system to distinguish between critical events and general informational prompts, enhancing usability and response prioritization.

Structure

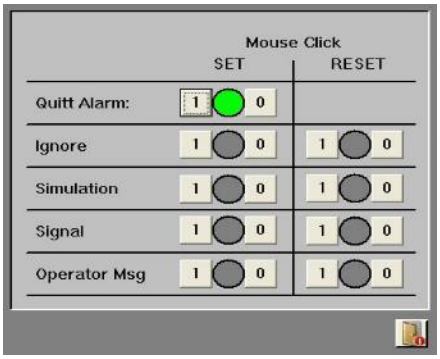
Address	Symbol	Type	Remark
0.0	B24	BOOL	spare
0.1	B25	BOOL	spare
0.2	B26	BOOL	spare
0.3	B27	BOOL	spare
0.4	xAlarm	BOOL	signal extern for alarm condition
0.5	B29	BOOL	spare
0.6	B30	BOOL	spare
0.7	B31	BOOL	spare
1.0	B16	BOOL	spare
1.1	B17	BOOL	spare
1.2	B18	BOOL	spare
1.3	B19	BOOL	spare
1.4	B20	BOOL	spare
1.5	B21	BOOL	spare
1.6	B22	BOOL	spare
1.7	B23	BOOL	spare

2.0	GAlQuitt	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	GAl	BOOL	general alarm
3.1	GAlS	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



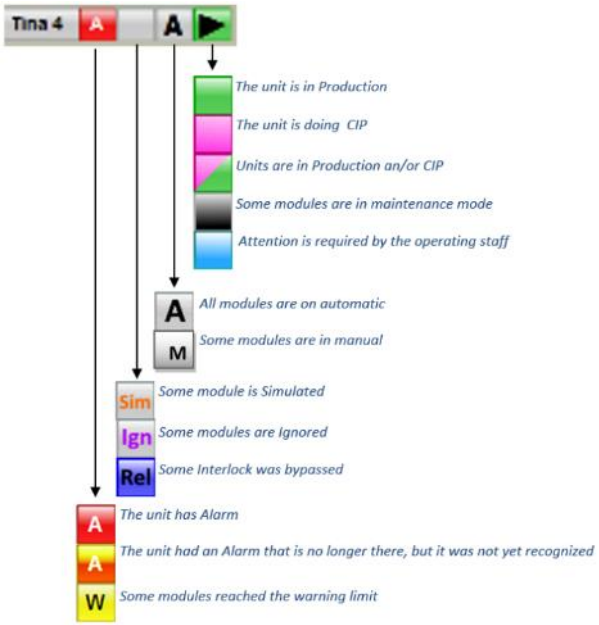
In addition to the system window for the counter for default parameter settings, there is also a window for mouse settings. This determines what should happen when you click the mouse over the item.

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 provide a structured method to organize and monitor alarms by grouping related control modules into logical units. This approach enables operators and engineers to view the collective status of multiple modules through a single, consolidated representation. Typically, alarm groups are configured to reflect specific production areas, functional sections, or complete production units, allowing for a clear and efficient overview of system health and operational status.



Within BatchXpert, alarm grouping plays a key role in simplifying complex systems by aggregating information and presenting it in an intuitive, hierarchical format. This improves situational awareness and supports faster decision-making during normal operation as well as fault conditions.

Configuration in BatchXpert

BatchXpert provides dedicated batch configuration tools that allow users to define unit structures and assign control modules accordingly. Based on this configuration:

- Alarm group function blocks are automatically generated by the system.
- These function blocks can then be downloaded to the PLC, ensuring that the alarm grouping defined in the engineering database is consistently represented at the control level.
- Each production unit is typically associated with a corresponding alarm group that includes all subordinate control modules assigned to that unit.
- This automated approach ensures consistency between the engineering environment and the runtime system while reducing the risk of manual configuration errors.

Visualization and HMI Integration

Alarm groups are extensively utilized within the BatchXpert HMI (Human-Machine Interface), particularly in the standard menu structure of operator stations. The interface provides:

- A hierarchical view of production areas and units, reflecting the configured plant structure.
- Alarm group indicators (symbols) displayed alongside each unit or area. These symbols dynamically represent the current alarm condition and overall process status of all control modules assigned to the group, including any nested subgroups.
- This visualization enables operators to quickly identify problem areas and navigate efficiently to detailed views for further analysis or corrective action.

Execution of Alarm Groups

The execution of alarm groups is managed automatically through system-generated functions:

- The project engineering tool creates a central system function responsible for calling and updating all alarm groups.
- Each unit is associated with a dedicated alarm group, which consolidates the status of all its assigned submodules as defined in the batch configuration.

Important Engineering Considerations

- The generated system function and alarm group function blocks are automatically generated by the engineering tool but must be manually downloaded.
- Manual modifications are strongly discouraged, as any changes made directly to these files will be overwritten the next time the system regenerates the alarm configuration.
- BatchXpert provides blocks, where customizations can be made safely
- To ensure persistence and consistency, all modifications should be made exclusively through the BatchXpert configuration tools.

Basic Functions for Alarm groups

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 DIn	FC 553	Add a Control Module to the Alarm group, to report its error, simulation, and automatic status
Bx Alarm Group AIn	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	GAl	Bool	general alarm
1.1	GAlS	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 ensure simple, consistent, and user-friendly operation across the control system, the Switch module (software-based switch) is provided as a standardized interface for operator interaction. This module enables operators to generate discrete control signals directly to the PLC, independently of other control modules such as actuators or input modules.

The software switch is particularly valuable for operations that are not inherently tied to physical I/O signals but still require operator intervention or confirmation. By centralizing such interactions, the module contributes to a uniform control philosophy and improves overall system usability.

Typical Applications

The software switch is commonly used for a variety of operator-driven actions, including but not limited to:

- Alarm acknowledgment or confirmation, allowing operators to confirm that a fault condition has been recognized.
- Manual operation confirmation, where operator approval is required before executing certain actions.
- Activation of optional process features, such as enabling or disabling auxiliary functions within the process.
- These functions are typically presented on the HMI as buttons or command elements, providing an intuitive interface for the operator.

Operational Behavior

The software switch generates a signal that is transmitted to the PLC when activated by the operator via the HMI. This signal can be used within the control logic to trigger specific operations, transitions, or acknowledgments. Because the switch operates independently of hardware inputs, it offers flexibility in implementing process control features that rely on user interaction rather than physical events.

Forced States and Interlocking

The Switch module also supports forced states at the PLC level:

- The switch can be forced to a SET or RESET state directly within the PLC program.
- When a forced condition is active, the corresponding control element on the HMI is automatically disabled (blocked), preventing operator interaction.
- This functionality is typically used for maintenance, commissioning, or safety purposes to ensure that specific states remain fixed and cannot be altered unintentionally by the operator.

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

Variable speed control of pumps and motors has become a standard practice in modern industrial automation. These systems are typically implemented using Variable Frequency Drives (VFDs), commonly referred to as Frequency Converters (FConv) within the control environment.

Unlike conventional motor control, frequency converters require both a start/stop command and a speed setpoint (nominal value) to operate effectively. This enables precise control of motor speed, improving process efficiency, energy consumption, and mechanical performance.

Communication Methods

Historically, frequency converters have been integrated into control systems using analog signals, most commonly through a 4–20 mA current loop. In such implementations:

- The start/stop command is provided via digital signals.
- The speed reference is transmitted as an analog output signal, often managed by Analog Output or PID modules.

In recent years, however, there has been a significant shift toward digital communication technologies. Many modern drives now support industrial fieldbus systems such as:

- PROFIBUS
- PROFINET
- Modbus
- Proprietary Networks, such as SEW-Bus and others

These communication networks may implement standardized protocols such as PROFIdrive, or alternatively, manufacturer-specific proprietary protocols.

Protocol Considerations

Despite the existence of standards like PROFIdrive, there is no universally adopted communication protocol across all frequency drive manufacturers. As a result:

- Different drives may use incompatible communication structures and data formats.
- Many vendors implement custom or proprietary protocols, even when using standard fieldbus infrastructure.

Due to this variability, integration of frequency converters into a control system requires the use of protocol-specific function blocks.

Implementation in BatchXpert

In BatchXpert, each frequency converter must be handled through a communication function tailored to the specific protocol of the drive being used. These protocol-specific functions:

- Interface with the drive using its native communication structure.
- Internally map and adapt data to the standardized BatchXpert FConv module.
- Ensure consistent behavior and integration within the overall control system, regardless of the underlying protocol differences.

This modular approach allows BatchXpert to maintain a uniform control interface while supporting a wide range of drive technologies.

General Communication Principle

Although communication protocols may differ significantly in implementation, most frequency converters follow a similar conceptual data exchange model. Typically, the following types of data are communicated between the PLC and the drive:

- Status Word – Indicates the current operating state of the drive (e.g., ready, running, faulted).
- Control Word – Commands sent to the drive to control its operation (e.g., start, stop, reset).
- Alarm Word – Provides detailed information about faults or warnings generated by the drive.
- Speed Setpoint (Nominal Value) – Defines the desired operating speed of the motor.
- Actual Speed Value – Reports the current measured speed of the motor.
- Additional Process Values – May include current, torque, voltage, or other diagnostic parameters depending on the drive.

These data elements are typically grouped into structured communication messages and cyclically exchanged between the PLC and the frequency converter.

Key Differences Between Protocols

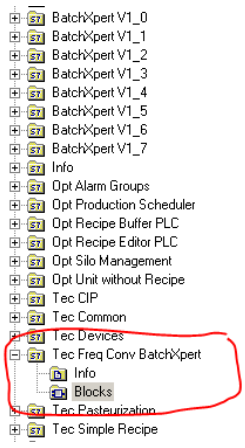
While the general communication principle remains consistent, the primary differences between protocols lie in:

- Encoding of data structures, particularly the Status, Control, and Alarm words.
- State models, defining how the drive transitions between operating modes (e.g., ready, enabled, fault).
- Data organization and mapping, which can vary significantly between implementations.

These differences are often substantial enough that protocols are not interchangeable, requiring dedicated handling for each type of drive.

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

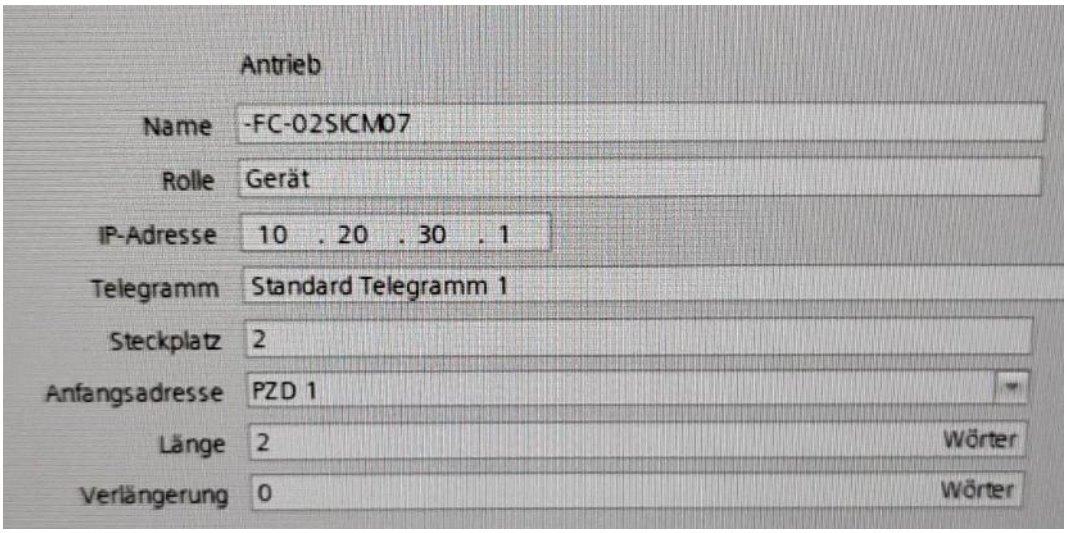
Most Frequency drives require you to set certain parameters to make the communication protocol work. This configuration depends completely on the device manufacturer, and you must consult the manufacturer's manual to check how and which settings to adjust.

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



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.

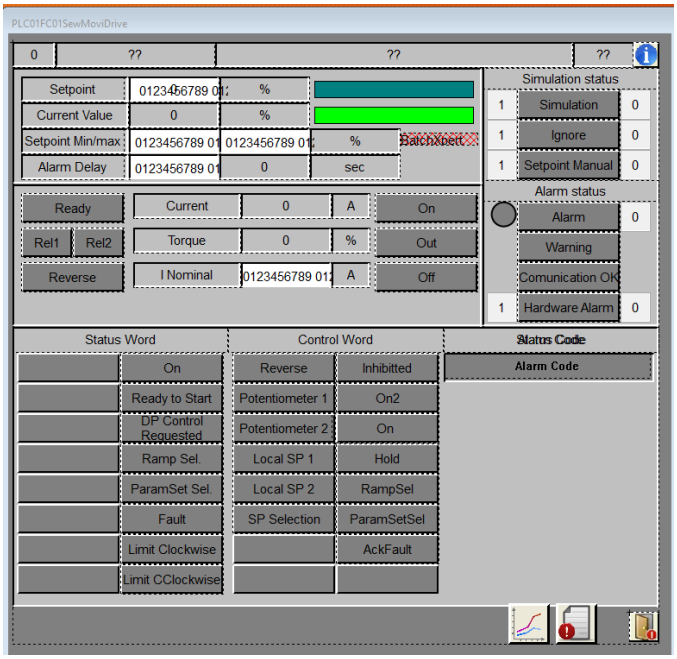
Simulation

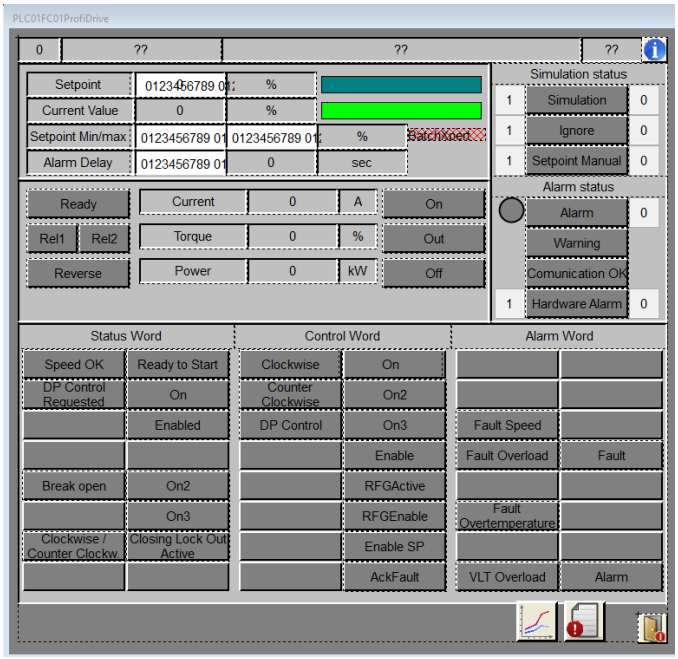
If the Frequency drive is simulated, it will never receive a communication error, always return the correct feedback, and return the requested nominal speed as current speed.

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	Potentionmeter1	BOOL	Potentionmeter control. Keep at FALSE
10	0.2	Potentionmeter2	BOOL	Potentionmeter 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 Movidrive 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 MoviDrive 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

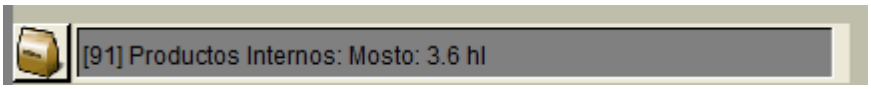
A material module allows you to easily create Material transfers either for Raw materials, produced materials or Transfers between two batches. This module allows you to specify a material that is being transferred, and origin, destination batch and an amount. When the Transfer is marked as “Finished”,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

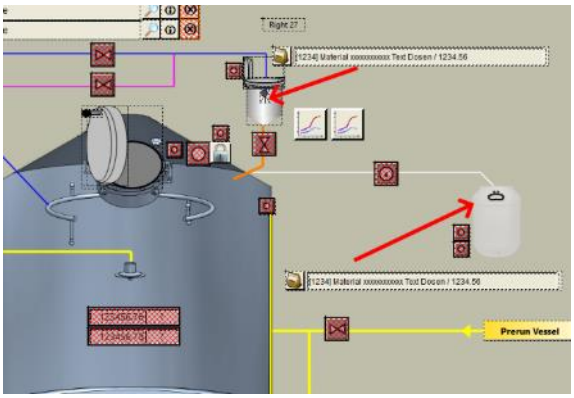
In summary the Material module has two objectives:

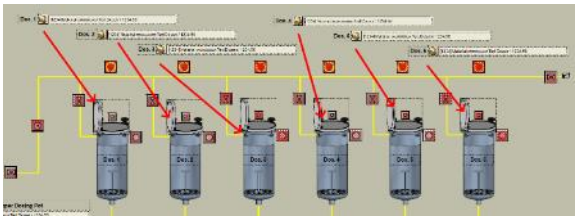
- Record Material intakes.
- Create relations between two or more batches to form a Batch Trace



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”



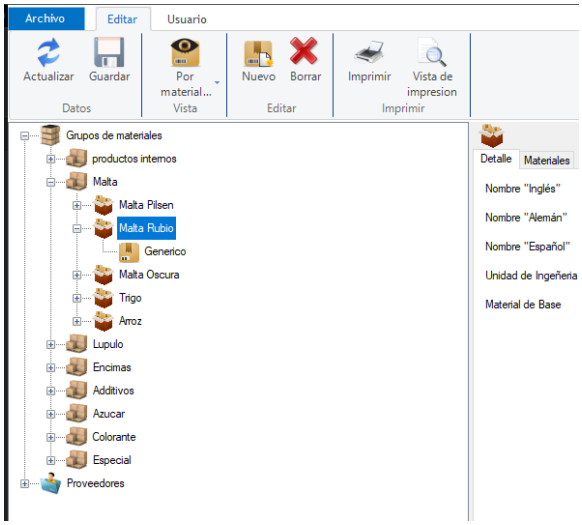


If you have an automatic dosing pump, you should implement a Material Module for each of the Dosing Pumps, or automatic dosing mechanisms.

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

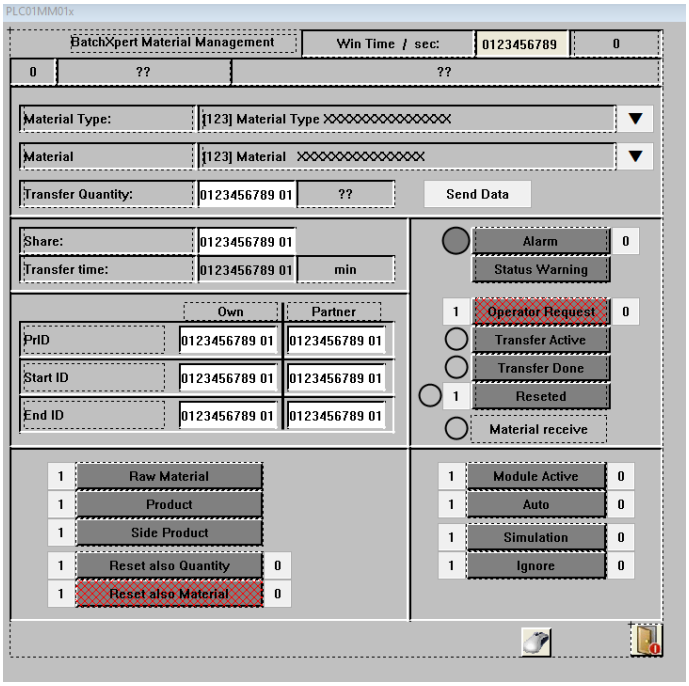
A Material is a specific “Product” that can be added into a Batch or is produced by a batch. Material type can be for example: Wheat Malt, Aroma Hops, Honey, Cold Wort etc.

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 Bew, 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



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 Dosing

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 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 entity such as a fermentation tank, pasteurizer, or filter. In BatchXpert projects, most application-specific PLC programming is implemented at the unit level. Units contain the phases and actions required to execute a recipe and are the primary consumers of the control modules (Actuators, Inputs, PID regulators, etc.). To support fast, consistent engineering, BatchXpert provides helper functions and standardized templates that simplify common unit behaviors.
Units execute the phases and steps defined in a recipe. For each active recipe step, a unit typically performs the following tasks:

- Transfer parameters from the current recipe step into the unit's active parameter set (Unit DB).
- Execute the unit logic by running the appropriate phase for the current step.
- Record events in the historical/event log when phases or steps start/finish, including relevant state transitions.

General Structure of a Unit in the PLC

In the PLC, a unit is typically composed of the following building blocks:

Unit Data Block (Unit DB): Holds the unit's recipe interface data, runtime status, and unit-specific user data.
Unit Function Block (Unit FB): Contains general unit logic such as actuator release handling, parameter transfers, and other infrastructure logic that is not phase specific. This block is usually called from a central unit dispatcher (commonly FC100) which then invokes the unit implementation.
Unit Function (Unit FC): Contains the unit's recipe-driven actions and phase logic (i.e., where most unit-specific behavior is programmed).

Unit Numbering Scheme

Units follow a fixed numbering convention where DB/FB/FC = Unit Number + 100, for example:

Unit 1 → DB101, FB101, FC101
Unit 2 → DB102, FB102, FC102
Unit 120 → DB220, FB220, FC220

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.

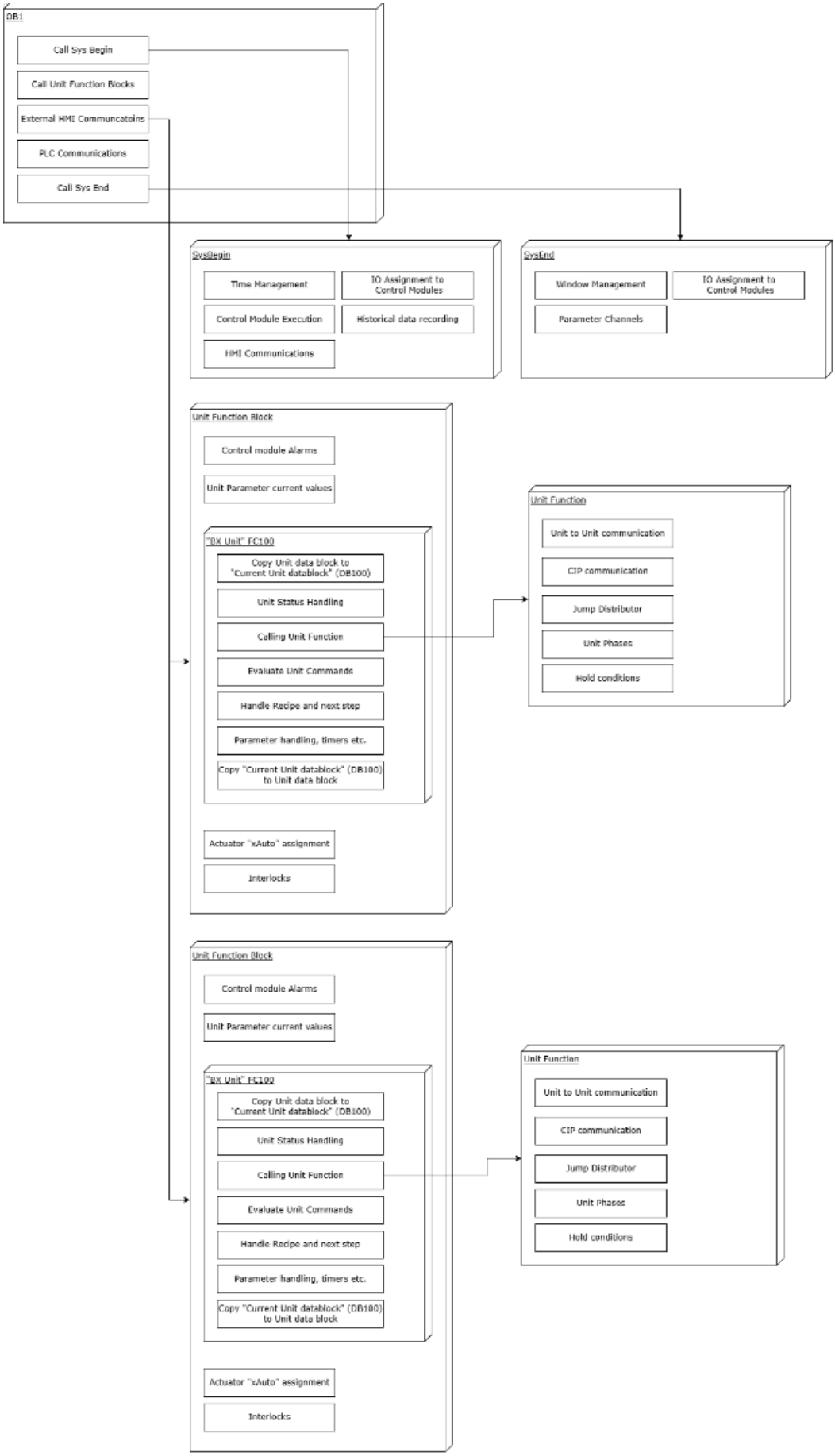


Figure 3 Unit Execution flow in the Plc

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	GAlS	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 Switch

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 limits.
//
//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 faceplate, since the PLC will always overwrite the value set by
//the faceplate.
//
//This means, that we should only set alarms that are 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 time 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 Wrong panel is 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 around 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 Counter "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
//is, 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 milliseconds.
//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 monitors that we receive impulses earlier than the
//Alarm delay on the Counter 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 an "Virtual Actuator", that has
//no physical output but nevertheless can pe 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 form 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

//Actuators that are not subject to any of the safety interlocks, come here.
//This however usually only applies to "Lamps", "Acoustic 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.

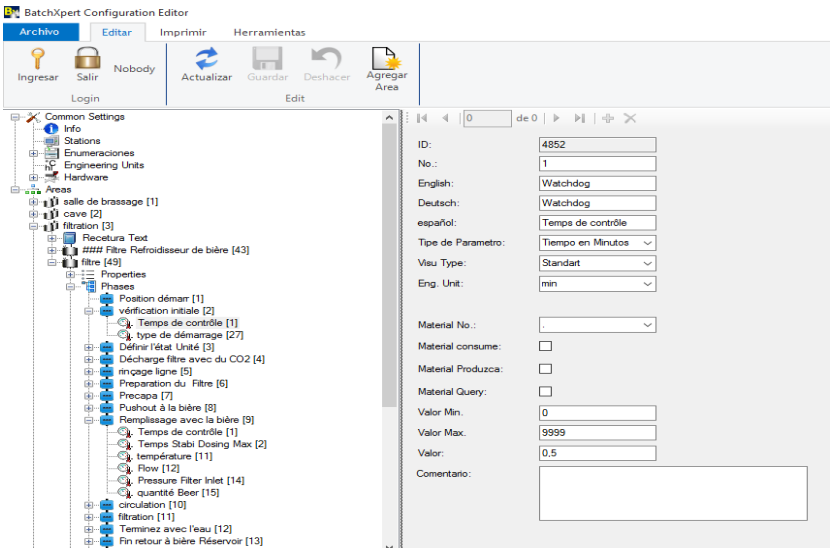


Figure 4 Example of a Parameter module configuration in the Batch Configurator

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.

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.

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	

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.

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:

$$D\ (Done) = Current\ Value \geq Nominal\ Value$$

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 form 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

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:

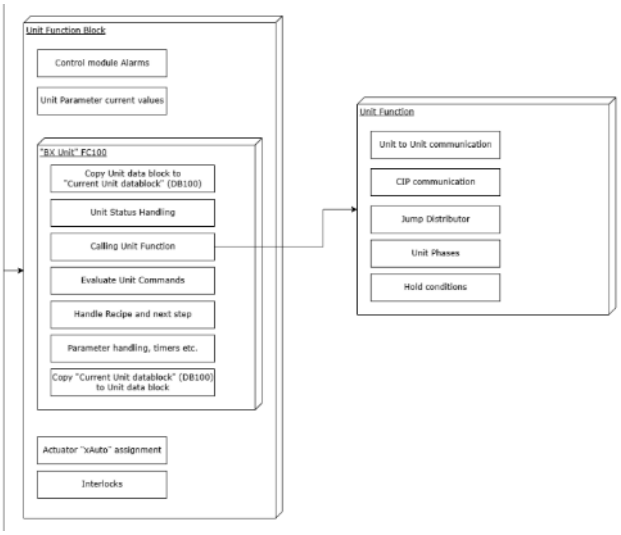
- Heating Up
- Transfer
- Filling
- Mixing
- Homogenization
- Boiling

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 mor 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. Se 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 batch 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
SatusNotClean	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
GAlQuitt	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.

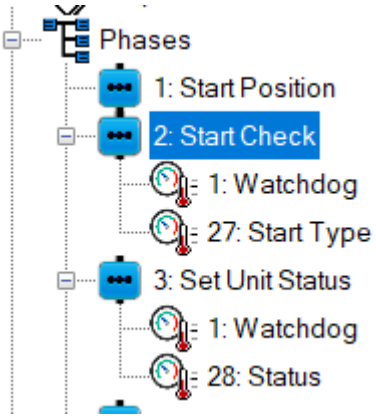
```
      L     "Phase"
      SPL   X000  //For invalid phase number we jump to "X000"
      SPA   X000  //Phase 0, is invalid
      SPA   X001  //if Phase = 1
      SPA   X002  //if Phase = 2
      SPA   X003  //if Phase = 3
      SPA   X000  //if Phase = 4, but it does not exist, we use "X000" as padding
      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.Slave1.Start;

// phase end condition
      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 Dosing

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 modules
      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 my implement a function where this phase active signal goes to false when “PhaseEnd” gets signaled. **“BatchXpert” Does not put face active to force 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.Slave1.Start;

// phase end condition
      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 FALS", 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" function 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)

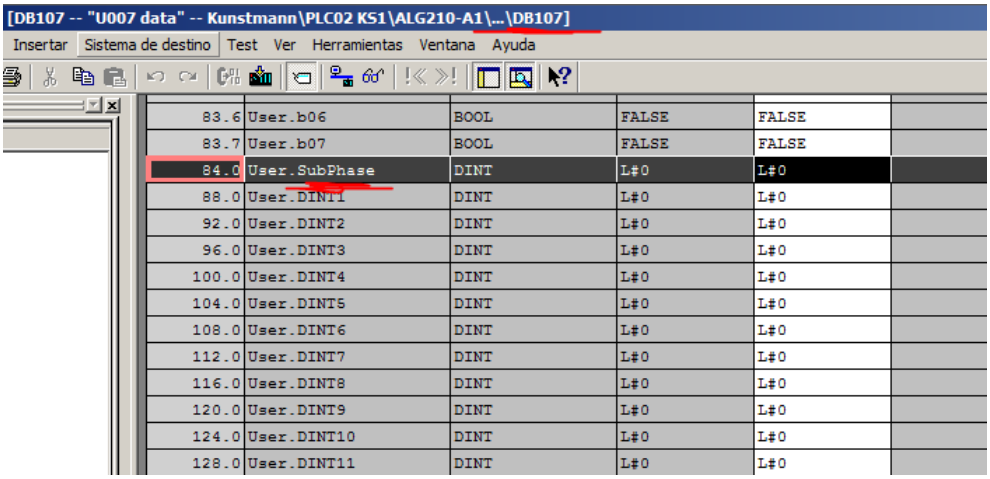
BatchXpert in and of itself does not support Subphases, which are Phases that are running inside of Phases. These are most used in phases such as “rinsing”, where you could have sub-phases for each of the different rinsing intervals, such as spray bowl, lateral inlet, lower inlet etcetera. Since BatchXpert 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.



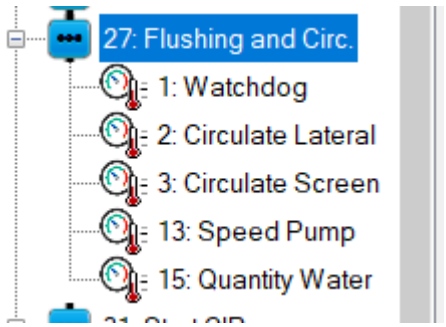
83.6	User.b06	BOOL	FALSE	FALSE
83.7	User.b07	BOOL	FALSE	FALSE
84.0	User.SubPhase	DINT	L#0	L#0
88.0	User.DINT1	DINT	L#0	L#0
92.0	User.DINT2	DINT	L#0	L#0
96.0	User.DINT3	DINT	L#0	L#0
100.0	User.DINT4	DINT	L#0	L#0
104.0	User.DINT5	DINT	L#0	L#0
108.0	User.DINT6	DINT	L#0	L#0
112.0	User.DINT7	DINT	L#0	L#0
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.

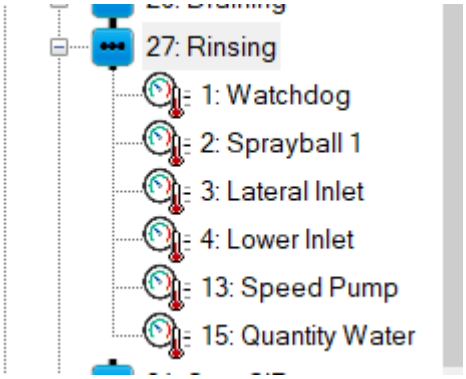



```
NETWORK
TITLE =phase 27:  Rinsing and Circulate

//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 we 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 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 Amount reached
      =     "PhaseEnd"

//Here we put our sub-phase logic
//_____________________________________________________
//Subphase Handling

//here we preset our "SubPhase" memory to point to the
//first subphase when we 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 to 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” buts, especially in the “UxxD”.User section, you must be aware what value these variables may heave 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).

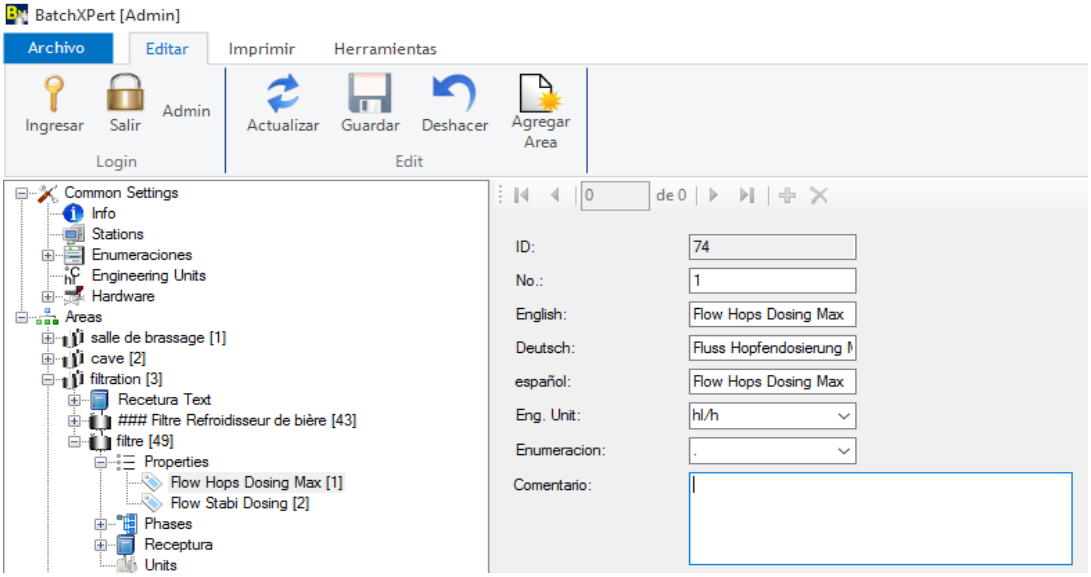
```
//Preset Help signal when starting phase and also when exiting
      U "PFCycle"
      O “PLCycle”
      R “UxxD”.User.HelpBitLevelReached

//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.

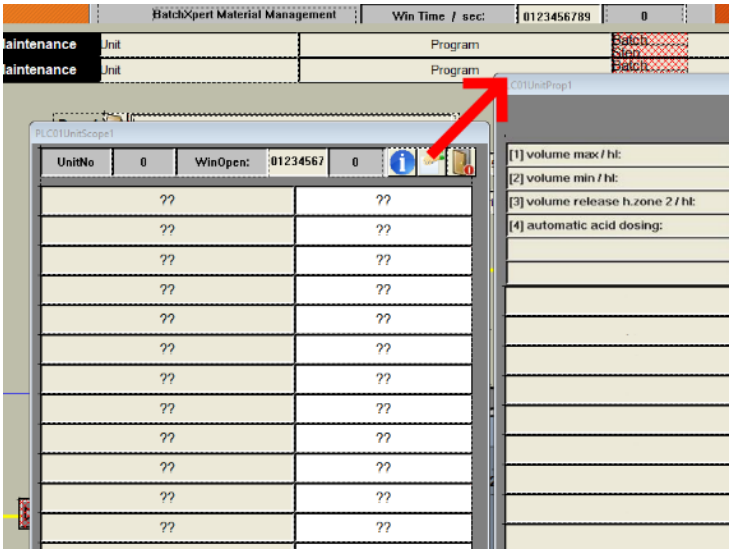
Configuration

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

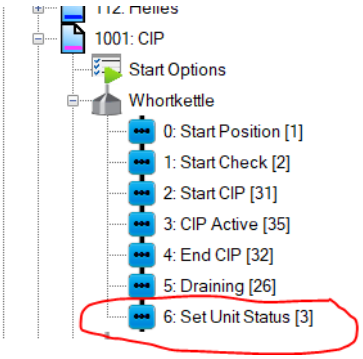
Each unit in BatchXpert automatically incorporates Hygienic state management. It can have the following statuses:

- 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.

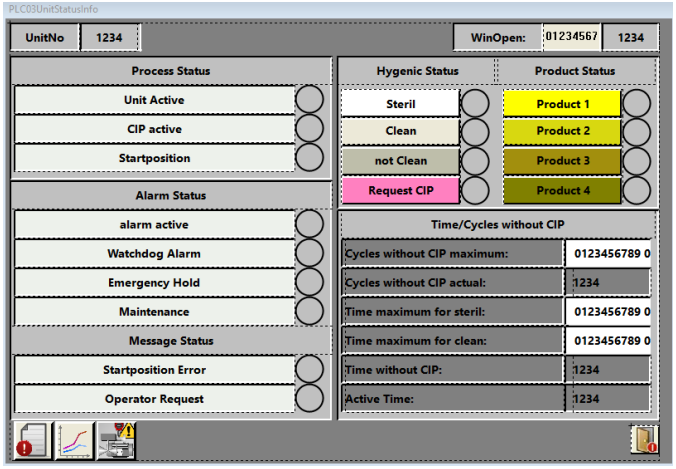
Implementation in a Unit

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



Starting a unit with an existing or new Batch

For any units to execute and recipe, you must start corresponding united 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.

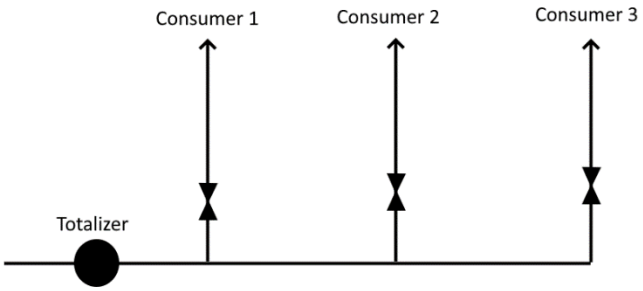
```
NETWORK
TITLE =phase 01:  start position

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.

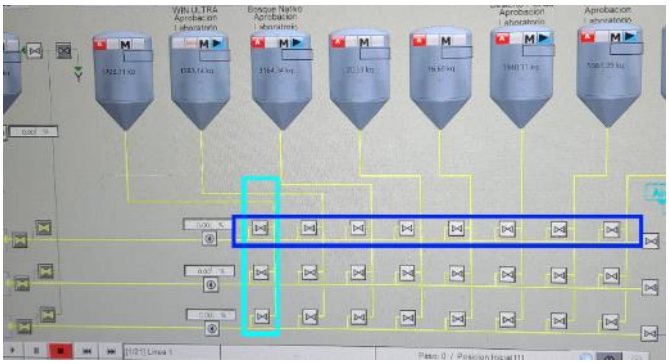
```
//Each requesting user must provide an "ID" to identify each of the requests and be
//able to assign the resource to one of them. The ID must be unique between all
//the requests. In this example we simply use the "Unit Number" as ID

//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
```

Mutex Interlock example

In this example we implement Interlocks in a Matrix valve configuration for a tank battery.

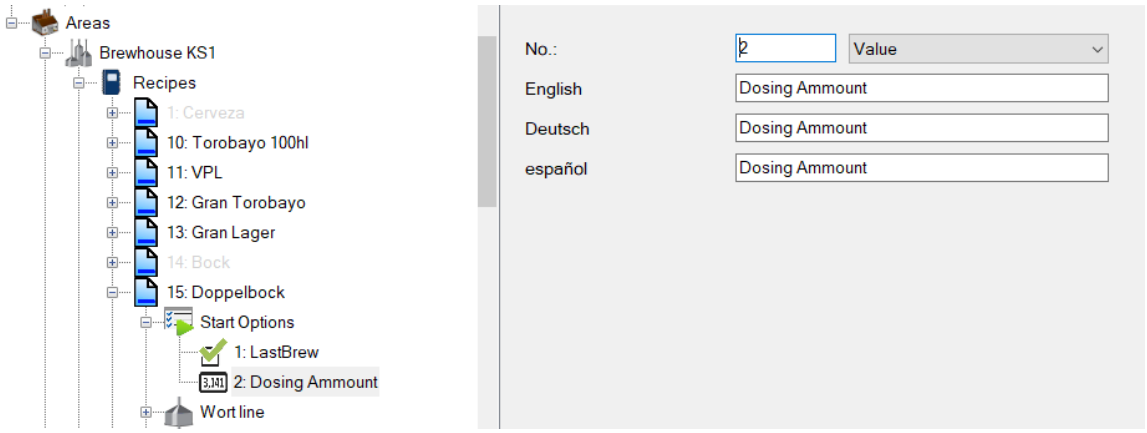


```
//This code is in the FB of a tank class.
//In this case, we have a matrix of valves, where we need to block
//valves both horizontally and vertically.
//
//A Mutex (Mutual Exclusion) ensures that a resource (in this case a valve) can be
//used by only one user at a time. Users are identified by the number
//"ID".
//
//For vertical valves, the "ID" is the line number of each valve.
//For horizontal ones, it is the tank unit number.
//
//For this, we use two Mutexes. One for vertical and the other for horizontal.
//When Alloc = true, it tries to "take" the Mutex. If the mutex is free,
//the "ID" is written as the user of the Mutex. When "Alloc" = false, the mutex is
released, setting
//0 as the user.
//
//The "Release" indicates whether the mutex is free or used by this user. That is,
//it shows whether it can be opened according to the mutex.

// This resource is shared among all vertical Mutexes
CALL  "Mutex Interlock"
         Alloc           :="Class TkP CM D".OutletLine1.Out
         ID              :=1     //1 for Line 1, 2 for line 2 etc.
         Release         :=#Vertical
         ProcessResource :=#User.MutexOutlet

// This resource is shared among all Horizontal Mutexes
CALL  "Mutex Interlock"
         Alloc           :="Class TkP CM D".OutletLine1.Out
         ID              :="Uxx".U.UnitNo
         Release         :=#Horizontal
         ProcessResource :="TK Producto Mutex".Linea1

      U     #Vertical
      U     #Horizontal
      U     #EmStop
      =     "Class TkP CM D".OutletLine1.Rel2
```

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
```

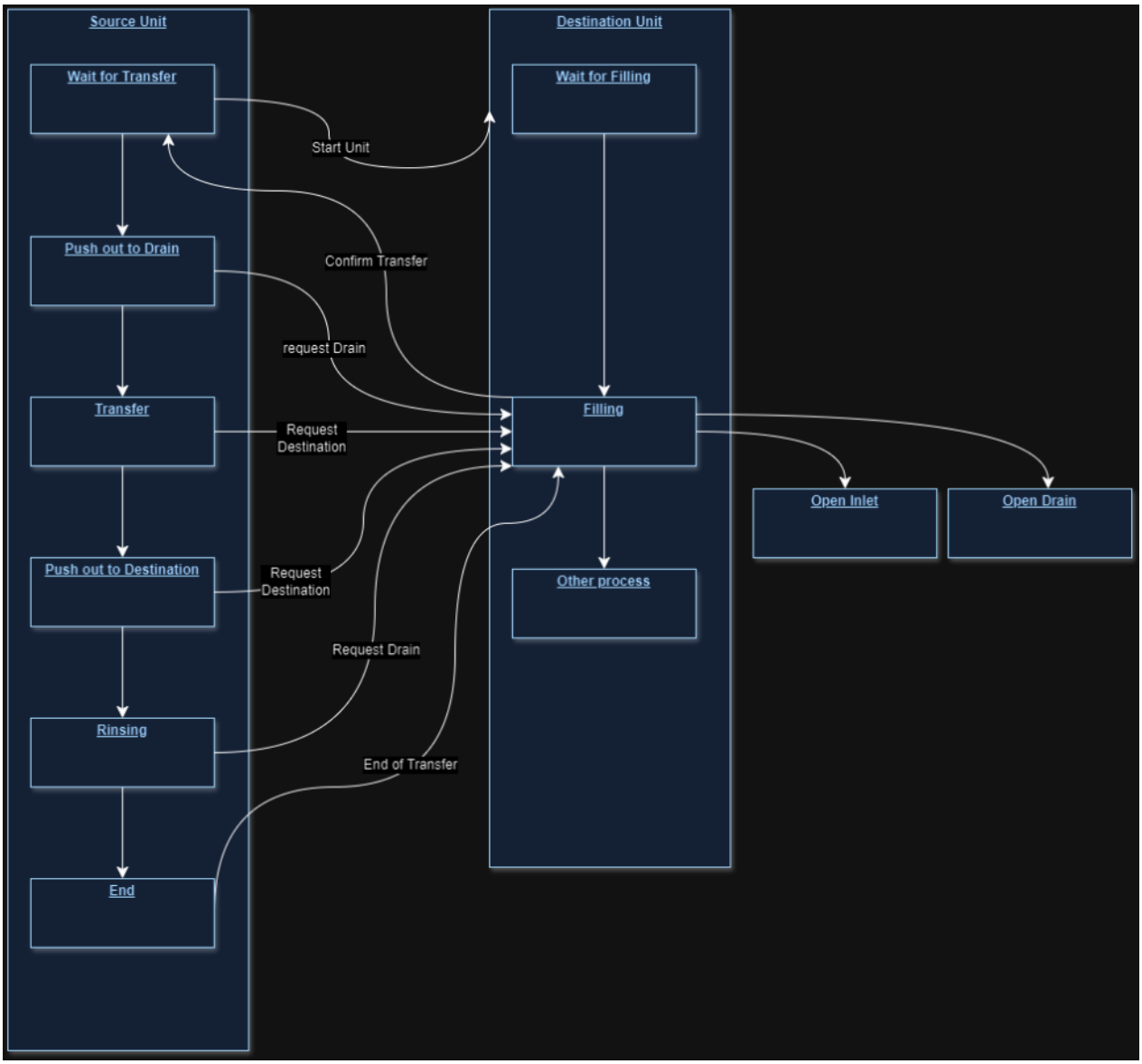
Unit-to-Unit Communication

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.

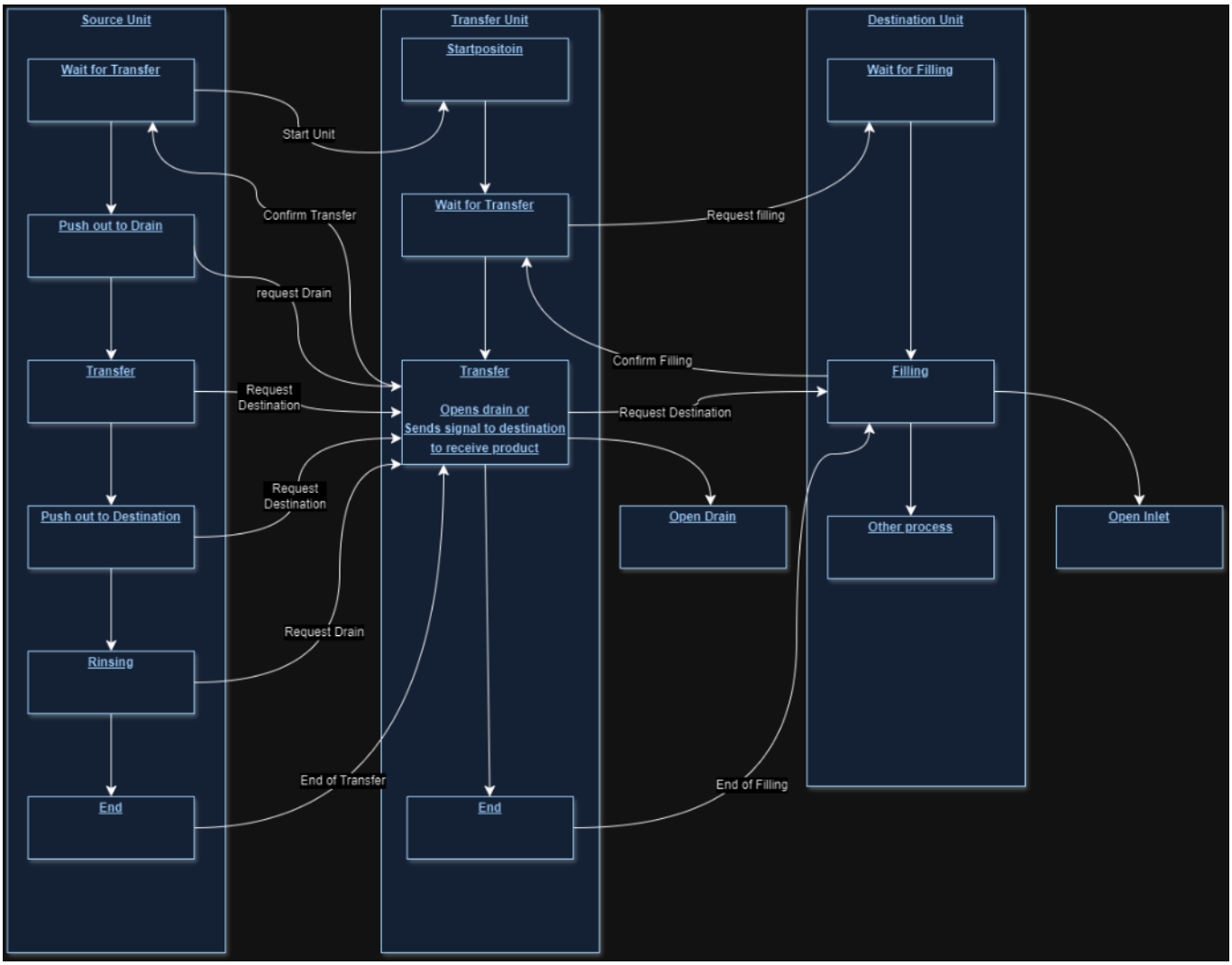
Simple Process Example

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 “Mater/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.

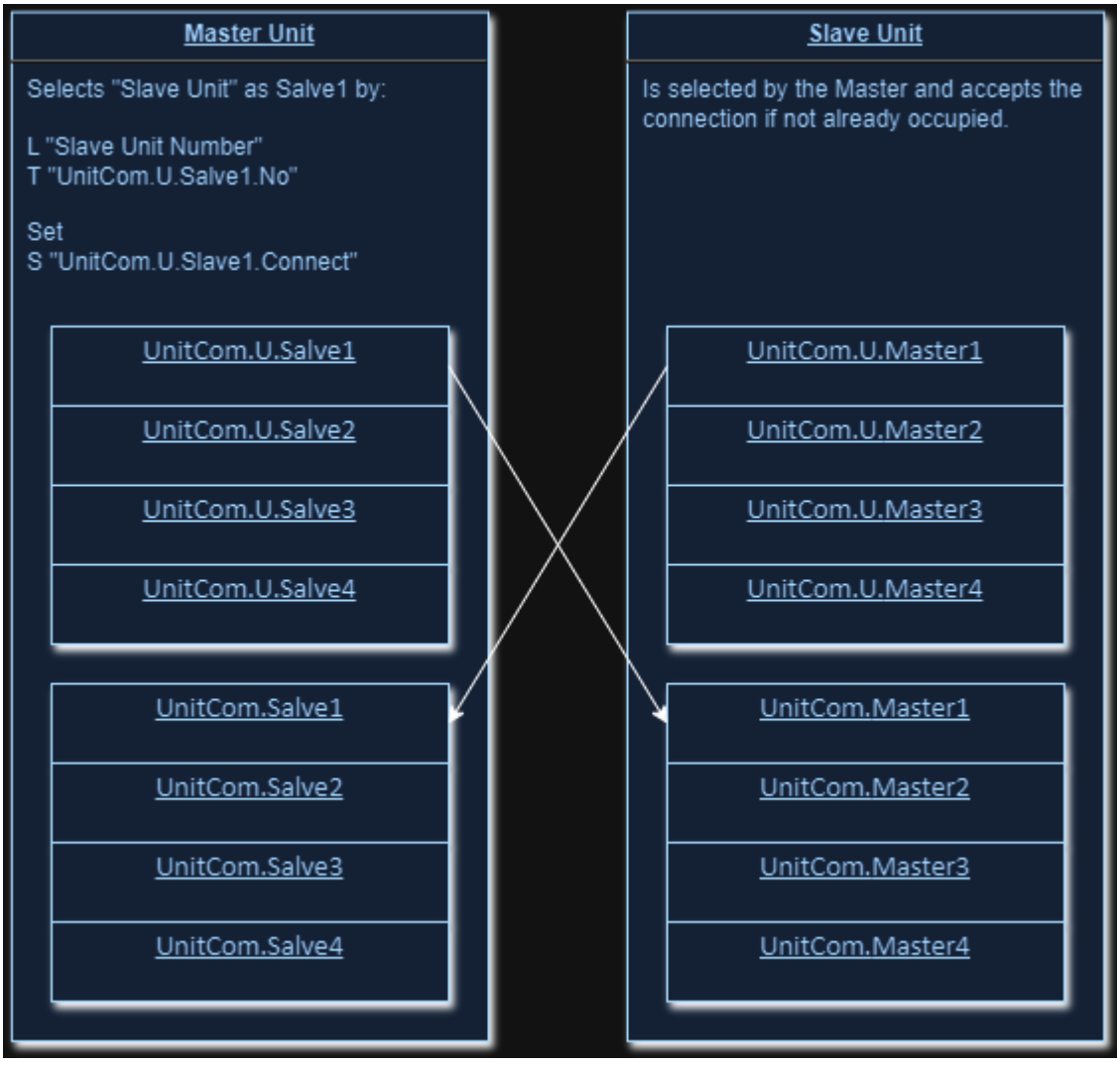
- Mash tun = Master
- Lauter tun = Slave

The mast 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”

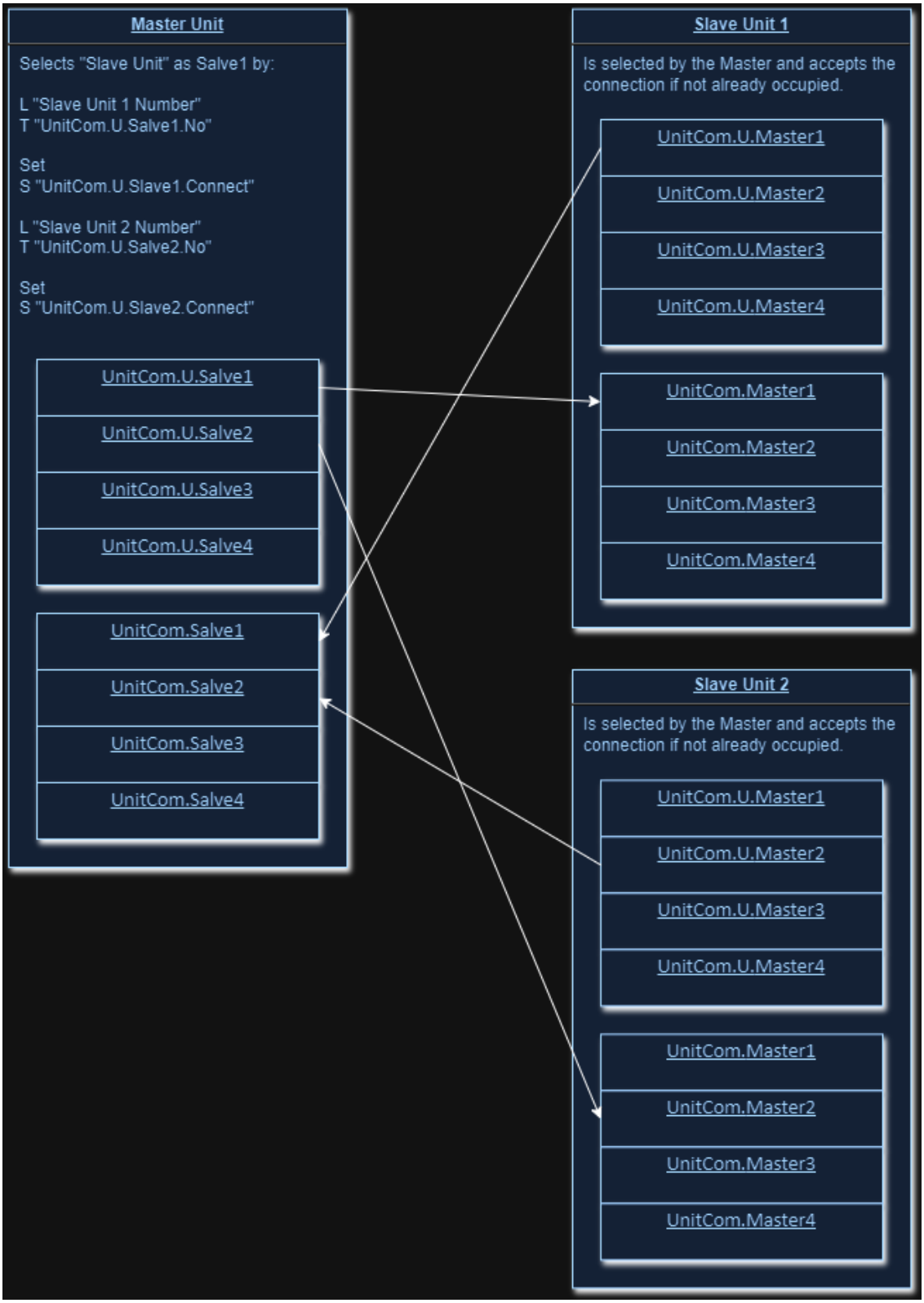
Data Transfer diagram with between two Units

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.PrId	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 Salve 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.Salve1.Connect       //Initiate Communication

      L 13                                //Partner Unit Number
      T "UnitCom".U.Salve1.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 memory from any data block you have available
//There are function 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

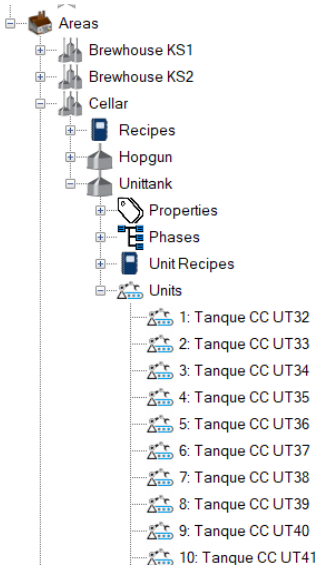
```
//These functions allow you to "Integrate" or "Totalize" a current flow to a volume. There
//are two versions of this. One generates an "Amount" as a Totalized number, and the other
//one generates impulses that can be used to connect to Counter control modules

     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.Real1       //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 amount
```

Value to Bit array (Value to Enumeration)

```
//This function 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= Enumeration 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"
```

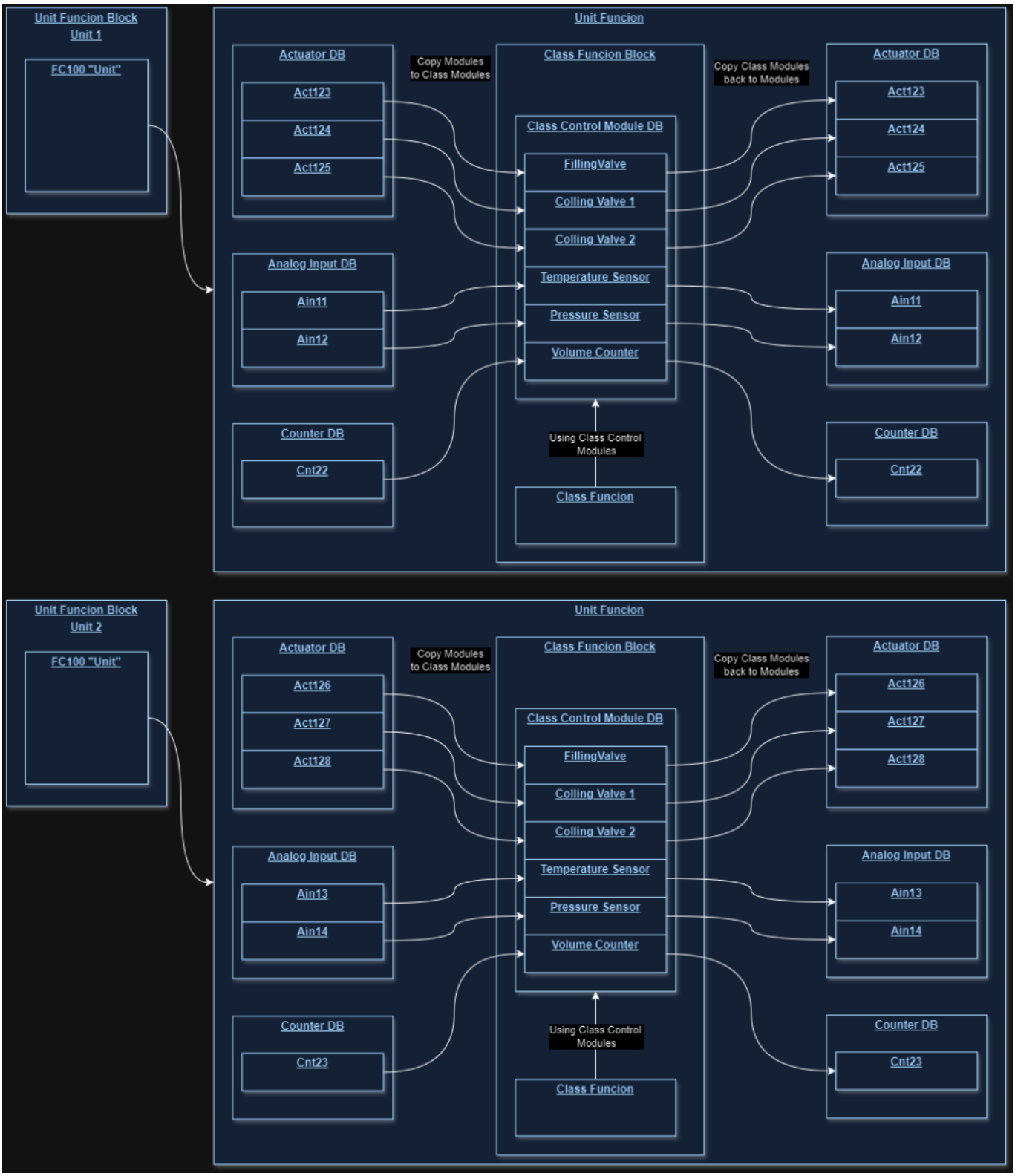
“Class” Programming

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.



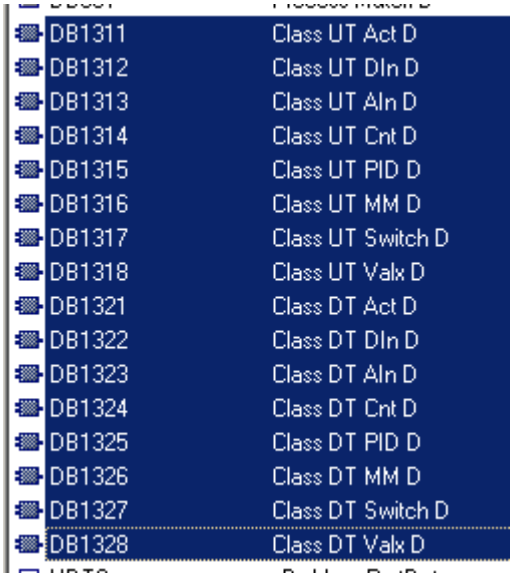
PLC Fow diagram

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	I
0.0		STRUCT	
+0.0	TempBottom	"sBx AIn"	
+92.0	TempTop	"sBx AIn"	
=184.0		END_STRUCT	

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" (
          IN0                 := 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 =De-supply 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
```

Unit Number 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   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
```

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's 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-Lean 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.

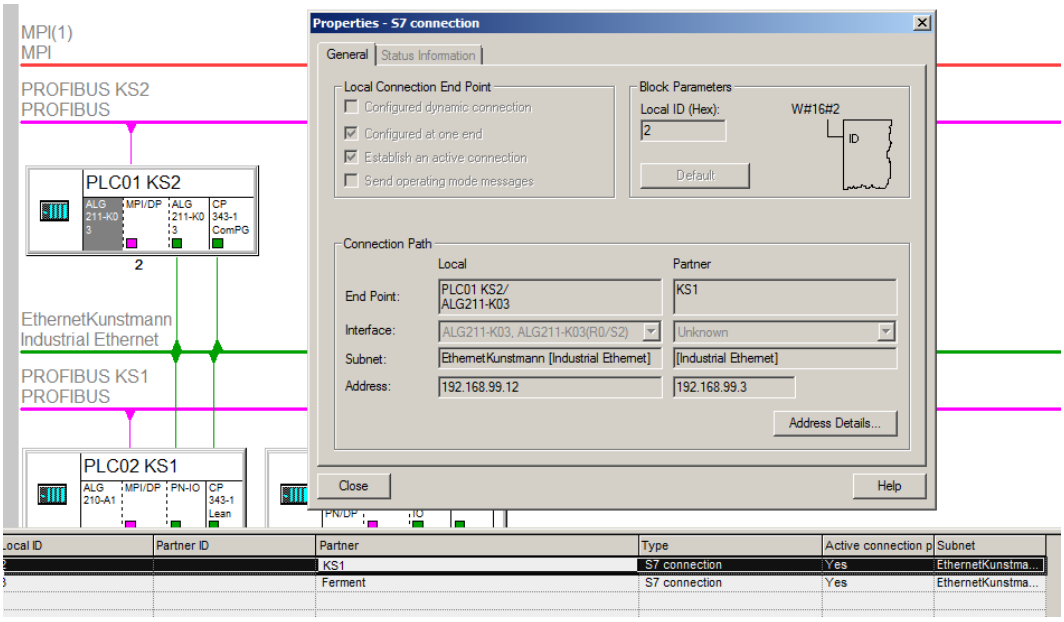
This way data will only ever get read by any PLC in their communication system, and will never be written by outside sources, such as communication channels.

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.



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.

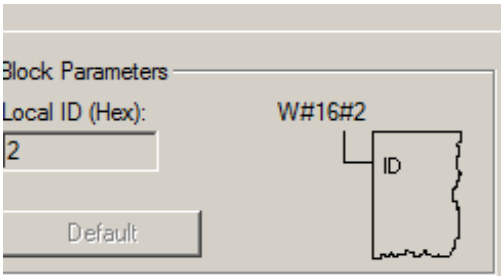
If more data needs to be exchanged between PLCs, than a single “GET” call can support, we recommend you use an “Chunking” or “Segmentation” Mechanism by extending the corresponding network in the communication function accordingly. See above for an example. You can extend the function to as many segments 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.

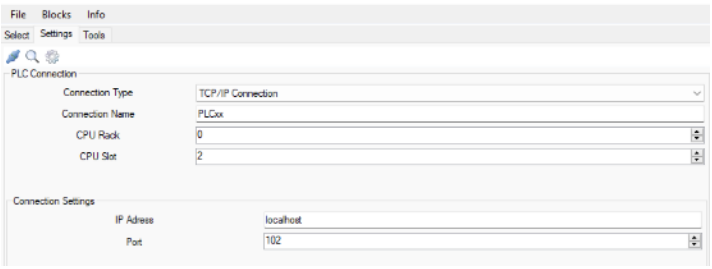
Extending the data block: S7-Backup

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.

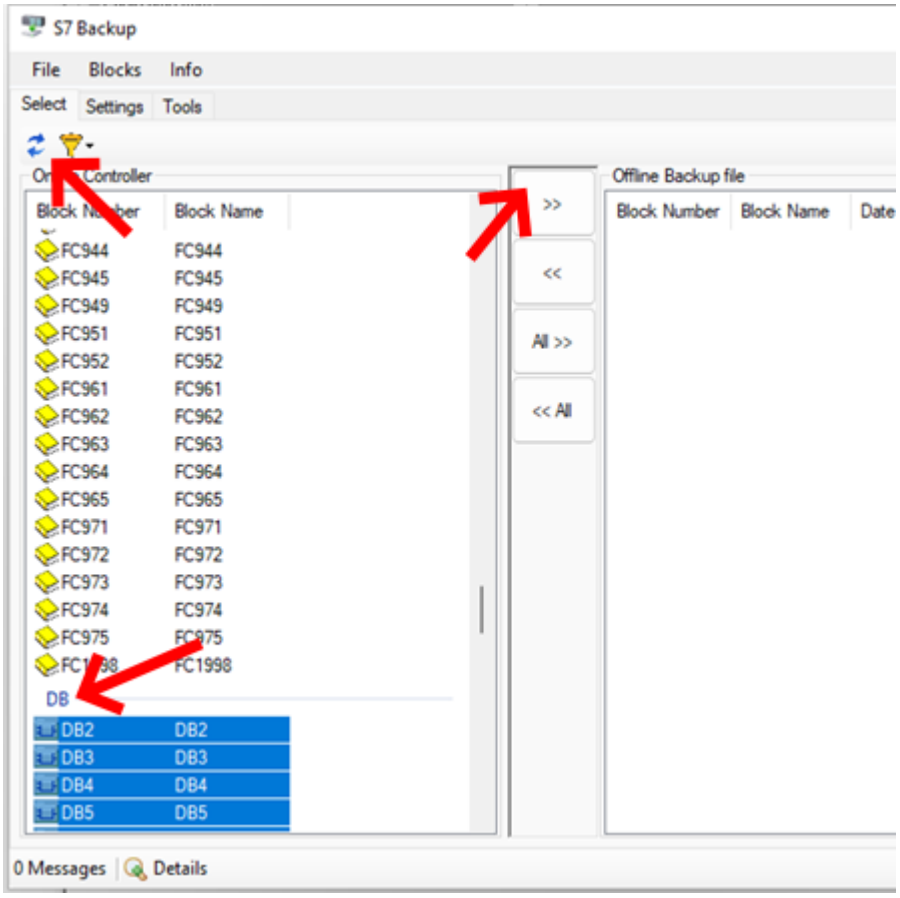
Extending the data block: How to upload and download

Input Connection parameters of your PLC



Discover all blocks and upload at least the data blocks

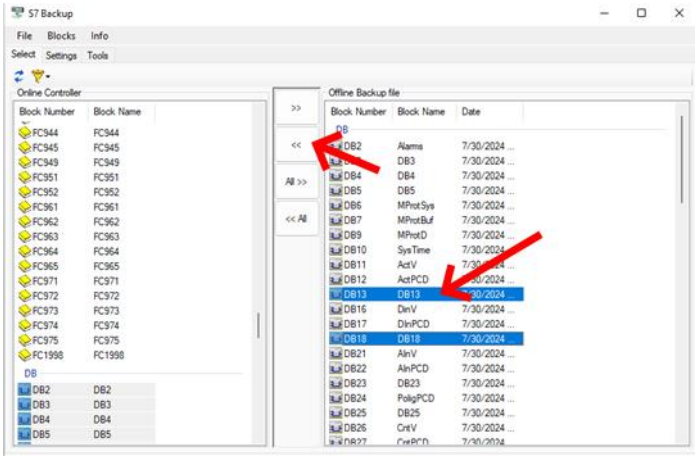
You can of course also upload all the blocks containing the controller, so you can already obtain an online backup of your current PLC. This upload can then be saved to and backup file, which can be reviewed or even restored if necessary.



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 **<u>select the "download current data only" option</u>**. 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.